Volume 30 Number 2 June 2006 ISSN 0350-5596

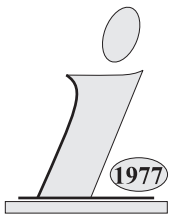
Informatica

An International Journal of Computing and Informatics

Learning in Web Search

Guest Editors:

Stephan Bloehdorn
Wray Buntine
Andreas Hotho



The Slovene Society Informatika, Ljubljana, Slovenia

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the list of referees. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

Editors' Introduction to the Special Issue "Learning in Web Search"

Introduction

The emerging world of search we see is one which makes increasing use of information extraction, gradually blends in semantic web technology and peer to peer systems, and employs grid style computing resources for information extraction and learning. This Informatica special issue explores the theory and application of machine learning in this context for the internet, intranets, the emerging semantic web and peer to peer search.

Search can also be viewed as a knowledge sharing service on the Web, as an interface to the Semantic Web. While some automation in building the Semantic Web has been achieved, it remains a labour intensive annotation process with problems in scaling up to the full free-text Web. A partial implementation of semantic-based search is possible where hierarchical concept spaces rather than full ontologies are used, and where information extraction and learning tools in the search engine perform approximate tagging of concepts. This partial semantic-based search could be viewed as a key infrastructure for more complete Semantic Web development, and arguably, as a safety net for it.

Overview of the issue

The articles in this special issue originate from two different backgrounds. Two articles are from the EU IST project ALVIS[1], which aims to bring web search infrastructure closer to the vision of the semantic web by automating some of the labor intensive annotation processes. The article *Semantic Search in Tabular Structures* by *Aleksander Pivk, Matjaz Gams and Mitja Lustrek* explores techniques for making tables and their content the subject of search while also considering the implicit semantics in the tabular structure. In the contribution *Beyond term indexing: A P2P framework for Web information retrieval*, the authors *Ivana Podnar, Martin Rajman, Toan Luu, Fabius Klemm and Karl Aberer* present a new framework for full-text information retrieval in P2P overlay networks and introduce a novel retrieval model based on highly discriminative keys.

Two further papers are selected papers from the workshop "Learning in Web Search" held at the International Conference on Machine Learning (ICML) in 2006 organized by the editors of this issue. The article *A Semantic Kernel to classify Texts with very few Training Examples* by *Roberto Basili, Marco Cammisa and Alessandro Moschitti* contributes to the field of using semantic background knowledge in the context of kernel methods. The article *Sailing the Web with Captain Nemo: a Personalized Metasearch Engine* by *Stefanos Souldatos, Theodore Dalamagas and Timos Sellis* presents the implementation of a metasearch engine that exploits personal user search spaces.

We thank the authors, reviewers and the Informatica editors for their efforts to ensure the quality of accepted papers and to make the reading as well as the editing of this special issue a rewarding activity.

Stephan Bloehdorn, Wray Buntine and Andreas Hotho

[1] http://www.alvis.info/

Semantic Search in Tabular Structures

Aleksander Pivk[1], Matjaž Gams[1] and Mitja Luštrek[1,2]

[1] Department of Intelligent Systems, Jozef Stefan Institute,
Jamova 39, SI-1000 Ljubljana, Slovenia

[2] Department of Computer Science, University of Alberta,
Edmonton, Alberta, Canada T6G 2E8

E-mail: *{aleksander.pivk, matjaz.gams, mitja.lustrek}@ijs.si*

Keywords: tabular structures, ontology learning, semantic web, query answering

Received: November 18, 2005

The Semantic Web search aims to overcome the bottleneck of finding relevant information using formal knowledge models, e.g. ontologies. The focus of this paper is to extend a typical search engine with semantic search over tabular structures. We categorize HTML documents into topics and genres. Using the TARTAR system, tabular structures in the documents are then automatically transformed into ontologies and annotated to build a knowledge base. When posting queries, users receive responses not just as lists of links and description extracts, but also enhanced with replies in the form of detailed structured data.

Povzetek: Razvili smo metode semantičnega spleta za iskanje informacij v tabelah.

1 Introduction

The World Wide Web has in the years of its exponential growth become a universal repository of human knowledge and culture, thus enabling an exchange of ideas and information on a global level. The tremendous success of the Internet is based on its usage simplicity, efficiency, and enormous market potential [4].

The success of the World Wide Web is countervailed by efforts needed to search and find relevant information. The search of interesting information turned out to be a difficult, time-consuming task, especially due to the size, poor structure and lack of organization of the Internet [3, 7, 31]. A number of approaches appeared in the last decade with a common objective to improve searching and gathering of information found on the Web.

One of the first solutions to cope with the information overload was search engines. In order for a search engine to function properly and to return relevant and satisfactory answers to user queries, it must conduct two important tasks in advance. The first task is to crawl the Web and gather, following the hyperlinks, as many documents as possible. The second task deals with document indexing [12, 35], hyperlink analysis, document relevance ranking [20, 23] and high dimensional similarity searches [13, 19].

Once these tasks are completed, a user may post queries to gain answers. User queries, unless they include highly selective keywords, tend to match a large number of documents, because they do not contain enough information to pinpoint most highly relevant resources [28]. They may sometimes even miss the most relevant responses, because of no direct keyword matches, but the most common disadvantage of this approach is that an engine might return thousands of potentially interesting links for a user to manually explore. The study described in [17] showed that the number of keywords in queries is typically smaller than three, which clearly cannot sufficiently narrow down the search space. In principle, this can be seen as the problem of users of search engines themselves. Reducing the ambiguity of search requests can be achieved by adding semantics, i.e. by making computers better 'understand' both the content of the web pages and the users' intentions, which can help to improve search results even with a request of a very limited size. In addition, search engines favor largest information providers due to name-branding and time optimization. To overcome this bottleneck, the Semantic Web search, where information is structured in a machine interpretable way, is a natural step forward, as originally envisioned by Tim Berners-Lee [4].

Moving to a Semantic Web, however, requires the semantic annotation of Web documents, which in turn crucially depends on some sort of automatic support to facilitate this task. Most information on the Web is presented in semi-structured and unstructured documents, i.e. loosely structured natural language text encoded in HTML, and only a small portion represents structured documents [2, 12]. A semi-structured document is a mixture of natural language text and templates [2, 12]. The lack of metadata that would precisely annotate the structure and semantics of documents and ambiguity of natural language in which these documents are encoded makes automatic computer processing very complex [9, 21].

Tabular structures (i.e. tables or lists) are incorporated into semi-structured documents and may have many different forms and can also differ substantially even if they represent the same content or data [14, 16].

Here we consider tabular structures as tables and lists which are described by a particular tag in HTML, i.e. `<table>` represents a table, where `<ul>`, `<ol>` and `<dl>` stand for different list types. A simple example of a table, found in a Web document, is represented in Figure 1.

Accommodation	Location	Type	Period	Cost
Bungalow GABER	Rogla	Apartment	Winter	1260,-
			Spring	1150,-
			Summer	1090,-
			Autumn	850,-
Hotel AREH	Pohorje	Single Room	Winter	590,-
			Rest	490,-
		Double Room	Winter	840,-
			Rest	690,-
Pension MARTIN	Pohorje	Apartment	Winter	1190,-
			Autumn	1090,-
			Rest	960,-
		Double Room	Winter	890,-
			Autumn	790,-
			Rest	730,-

Figure 1: A simple table example belonging to a tourism domain.

Understanding of their content is crucial when it comes to finding and providing explicit answers to user queries. The table understanding task demands efficient handling of tabular structures and automatic transformation of structures into explicit semantics, which enables further reasoning about the data within tables. But both, a table structure comprehension task and a semantic interpretation task exceed in complexity the corresponding linguistic task [16].

The paper is structured as follows. In Section 2 we first introduce a simplified view of the ALVIS search engine followed by a detailed description and an algorithm of our extensions. Section 3 describes the current state of our approach and its evaluations. After presenting the related work in Section 4 we give concluding remarks in Section 6.

2 ALVIS - Semantic search engine extensions

The basic idea for attaining a semantic search engine is to automatically enrich the indexed web pages with semantics, in our case in the form of ontologies. This is the key subject of the EU IST-project ALVIS (Superpeer Semantic Search Engine, IST-1-002068-STP), which represents the framework for our contribution. The general architecture of ALVIS is presented in Figure 2. Our major contribution is represented as two bold boxes on the right-hand side of the figure.

The ALVIS project (`www.alvis.info/alvis`) conducts research in the design, use and interoperability of topic-specific search engines with the goal of developing an open source prototype of a distributed, semantic-based search engine. ALVIS facilitates semantic retrieval and incorporates pre-existing domain ontologies using facilities for import and maintenance. The distributed design is based on exposing search objects as resources and on using implicit and automatically generated semantics (not just ontologies) to distribute queries and merge results. Because semantic expressivity and interoperability are competing goals, developing a system that is both distributed and semantic-based is the key challenge: research involves both the statistical and linguistic format of semantic internals, and determining the extent to which the semantic internals are exposed at the interface.

The problems that were identified and served as a motivation factor while developing the system and methodology are: (a) tabular structures present a general schema for data presentation due to their good visual comprehension, hence a great deal of data is hidden within them, (b) multiple table representations for the same content/data are very likely, (c) manual annotation does not scale in general, (e) tables and lists, due to their richer structure, support the discovery of interdependent information, whilst a typical keyword or link-analysis based search usually fails in this area [27], and (f) ontology learning from arbitrary text has so far had limited success [6, 9].

The whole process of semantic search over tabular structures consists of two subprocess: (a) transformation of tabular structures found on the Web into the knowledge base, described in subsection 2.1 and shown in Figure 3, and (b) user query processing, described in subsection 2.2 and shown in Figure 6.

2.1 Tabular structures transformation

The overall analysis, transformation, and formalization of tabular structures can be graphically observed in Figure 3. It roughly corresponds to the bold boxes in Figure 2 and directly to Figure 4. Here we will shortly present each of the five major steps with the obtained results given in section 3:

(1) Categorization of Web documents
  (a) into Topics
  (b) into Genres
(2) Filtering of proper tabular structures from documents
(3) Transformation of arbitrary tabular structures into formal representations
(4) Automatic ontology generation from formalized tabular structures
(5) Knowledge base construction

Figure 4: Tabular structure formalization procedure.

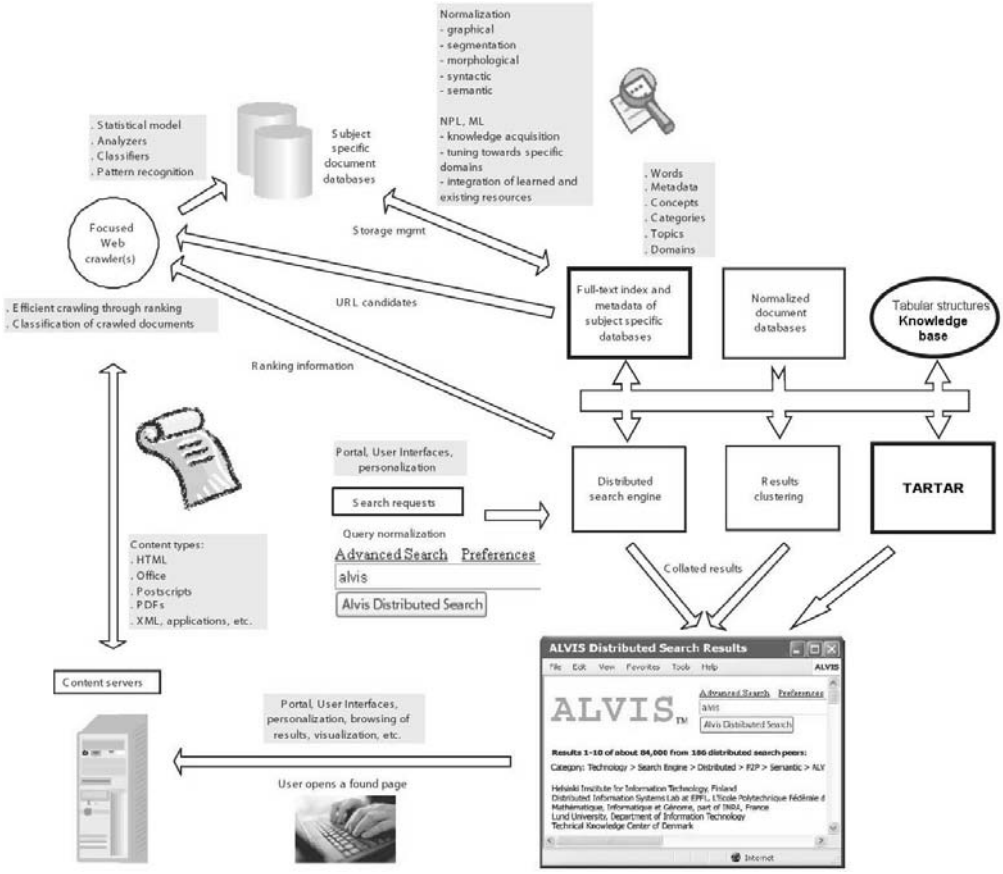


Figure 2: Performance architecture of ALVIS search engine.

1) Categorization of Web documents

Clustering is a technique for classification of Web documents into groups within which interdocument similarity is large compared to the similarity between documents chosen from different groups, and enables taxonomy design, e.g. topic hierarchy construction. The technique assists fast similarity search, while groups of similar documents significantly narrow the search space.

1.a) Topics

Work on topic classification has been utilized by our colleagues Mladenić et al. [22]. They used machine learning techniques on a document collection gathered from Yahoo to reconstruct Yahoo's manually built topic taxonomy. Documents are represented as feature-vectors that include word sequences, where the learning approach depends on feature subset selection and results in a combination of a set of independent classifiers. The whole categorization process belongs to the bold box (Full-text index...) in Figure 2.

1.b) Genres

Another type of categorization that we propose is categorization into genres. According to Merriam-Webster Online Dictionary, a genre is a category of artistic, musical, or literary composition characterized by a particular style, form, or content. Editorial and poem, for example, are genres characterized by style and form, while science fiction is a genre characterized by content. For our purpose, we will concentrate on the first kind of genres, because the second kind are already in large part covered by topical categorization. In information retrieval, genres should ideally be orthogonal to topics, because this best divides the search space and reduces the number of search hits. Traditional genres often do not satisfy this requirement, but in the selection of genres we use and in the preparation of training data for the genre classifier, we tried to be as close to the ideal as possible.

We chose 20 genres one can expect to find on the World Wide Web: some of them Web-specific, such as FAQ and community, others traditional, such as journalistic and poetry. The next step was the preparation of the training data for the genre classifier. We gathered Web pages returned by the most popular queries according to Google, random Web pages and Web pages specifically sought to belong to the less common genres. Each Web page was manually categorized by two annotators and in case of disagreement checked by the authors.

Finally, a suitable classification algorithm needed to be selected. Most algorithms require extraction of stylistic

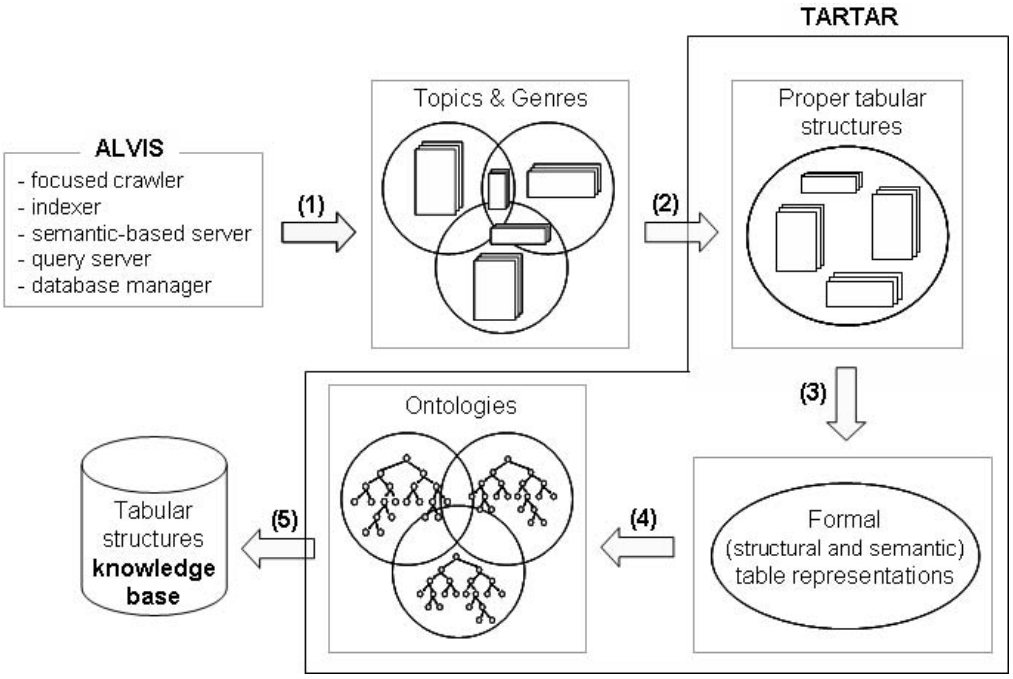


Figure 3: The transformation of tabular structures found on the Web into the knowledge base.

features from the Web pages. Due to the complexity and irregularity of HTML found on the Web, this is a time-consuming and error-prone procedure that can compromise the robustness of the classifier. Therefore we chose prediction by partial matching (PPM) classification algorithm, which can accept raw HTML as the input. This algorithm is derived from data compression techniques and essentially learns the way to most efficiently compress the Web pages belonging to each genre. An uncategorized page is then compressed with each genre-specific compressor in turn and categorized into the genre associated with the most effective compressor. Initial tests have been performed by Bratko and colleagues [5], proving successful in a related problem setting.

2) Filtering of proper tabular structures

Tabular structures appear in Web documents within specific tags, thus enabling their easy discovery. Unfortunately, it has been shown that at least 70% of such structures are not used for presentation of data but for better visual and layout purposes [8]. To solve this problem, we adopted a solution developed by Wang et al. [34]. It is based on machine learning techniques, in particular decision trees and support vector machines. In the learning process, several types of features are used, such as structural, content, and word/text features.

3) Formalization of tabular structures

This essential part concentrates on the analysis of tabular structures aiming to exploit their partial structure and cognitive modeling habits of humans. Understanding of table contents requires table-structure comprehension and semantic interpretation. The outcome of applying the method is a knowledge frame encoded in an F-Logic representation language [18].

The most comprehensive and complete model for the analysis and transformation of tables found in literature is Hurst's [15], which is also adopted here. The model analyzes tables along graphical, physical, structural, functional, and semantic dimensions. Our approach stepwise instantiates the last four dimensions.

In the first step, corresponding to the physical dimension, a table is extracted, cleaned, canonicalized and transformed into a regular matrix form.

In the second step, the goal is to detect the table structure. This is a very complex task, since there is a huge number of table layout variations. The three most important sub-tasks are: (a) to determine table reading orientation(s), which is discovered by measuring the distance/similarity among cells and hence among rows and columns (features given at the end of the paragraph); (b) to dismember a table into logical units and further into individual regions, in a way that regions consist of only attribute cells, e.g. 'Location' in Figure 1, or instance cells, e.g. 'Rogla'; (c) to resolve the table type, which must belong to the one of five pre-defined types (one- or two-dimensional, or three types of complex tables). For these purposes several heuristics, i.e. token type hierarchy, and measures are used, i.e. value similarity (regression), value features (character/numeric ratio, mean, variance, standard deviation), data frames (e.g. regular expressions for recognizing dates), and string patterns.

In the third step, the functional table model (FTM) is constructed. FTM is represented as a directed acyclic graph, which rearranges table regions in a way to exhibit

an access path for each individual cell. After finalizing the FTM construction, its recapitulation is carried out with a goal to minimize the model.

Finally, in the fourth step we deal with the discovery of semantic labels for table regions, where WordNet [11] lexical ontology and GoogleSets service are employed. These semantic labels serve as annotations of FTM nodes and are later also used within outgoing formal structures as can be observed in Figure 5.

A very detailed description of this particular part can be found in [30].

4) Automatic ontology generation

Tables with different structures that are categorized into the same topic/genre clusters and had been individually transformed into knowledge frames, could now be merged into a single cluster ontology. In this way, a generalized representation that semantically and structurally describes captured tables within each cluster would be created.

Our approach to frame matching and merging is multifaceted, which means that we use all evidence at our disposal to determine how to match concepts. In using this evidence we look not only for direct matches as is common in most schema matching techniques, but also indirect matches. The most relevant matching techniques are the following: (a) label matching which depends on WordNet features and string distance metrics; (b) value similarity, where matching is based on value characteristics; (c) expected values by using constant value recognizers in data frames; (d) constraints that include keys of the table, functional relationships, one-to-one correspondences, and subset/superset relationships; and (e) structural context, where features such as proximity, node importance measured by in/out-degree, and neighbor similarity help match object sets.

Once the mappings among frames have been discovered, the actual merging is executed. Sometimes two frames are directly fused by simply merging corresponding nodes and edges. Often, however, merging induces conflicts that must be resolved, where several different approaches for conflict resolution are used. Conflict resolution will not be discussed in this paper due to length limits. Finally, after all frames of a particular topic/genre cluster are incorporated, the outcome reflects an appropriate domain ontology, where the concepts are arranged into a directed acyclic graph, where the arcs represent the relations among concepts and are labeled with relation type.

Figure 5 illustrates a simple example of the ontology creation process by merging two same-cluster tables; the second table is shown in Figure 1. Both tables deal with hotel price information, but the naming convention and contents are different. The matching of the frame concepts and object sets are based on several techniques: Label Matching and Structure help determine the concept matching, while Value Similarity and Expected Values apply to price, room type, and period. The domain ontology is presented as a frame, but its concepts are further classified and the

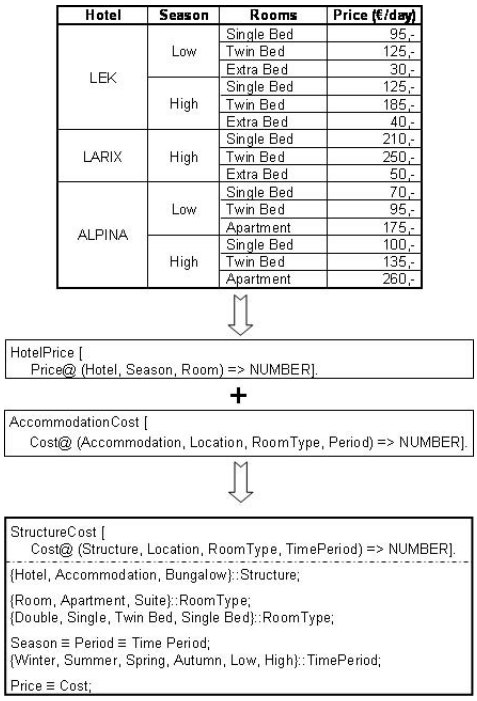


Figure 5: A simple example of the ontology creation process.

classification encoded in F-Logic notation. For example, concepts 'Hotel' and 'Accommodation', according to the WordNet, share a common hypernym 'Structure'; values 'Double/Single Room', 'Apartment', and 'Suite' become concepts and are hierarchically organized under a 'RoomType' concept; the same thing happens to season periods. Also note that concepts 'Price' and 'Cost' share the same synset in WordNet, which is in the ontology described by an equivalent operator (≡) among them. In this way a common ontology covering both frames is generated, which enables annotation and query answering, the later through the knowledge base.

We have explored issues of frame matching for ontology generation in [29], where further information can be found.

5) Knowledge base construction

By constructing multiple domain ontologies, where each one corresponds to a particular topic/genre cluster, the final step of our algorithm is evident in a construction of a knowledge base. The knowledge base is created by formalizing the table contents according to the newly generated formal structures. Together with the inference engine it will be used for providing replies to user queries.

Steps 2, 3, 4, and 5 are all implemented within a system named TARTAR (Transforming ARbitrary TAbles into fRames) as part of the PhD dissertation [25] and later extended for ALVIS. Further information regarding TARTAR can be found in [27, 30], while the open source code is freely available and can be downloaded at `http://dis.`

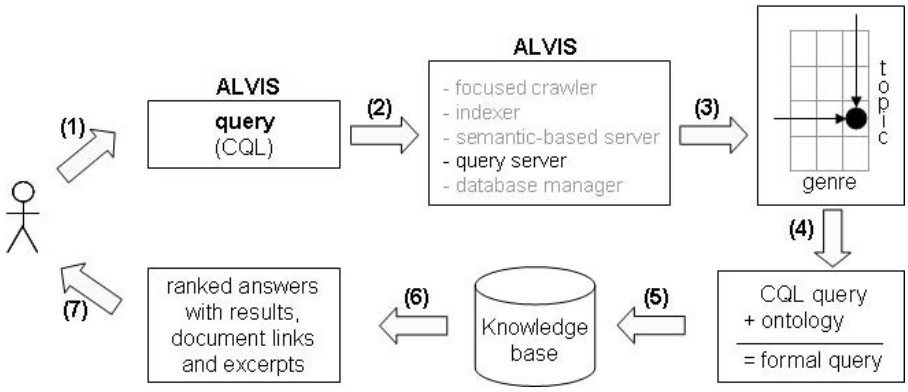


Figure 6: A seven-step user query processing schema.

ijs.si/sandi/work/tartar/.

Each of these algorithmic steps is important in the overall process and benefits search in terms of accuracy and reduced complexity.

2.2 Tabular structures querying

Many search systems permit the query to contain single words, phrases, and word inclusions and exclusions. Instead of exclusion, a safer way is to rate each document for how likely it is to satisfy the user's information need, to sort in decreasing order of this score, and present the results in the ranked list. Search engines differ in how they perform vector-space ranking.

Our motivation is to extend the ALVIS search engine with our tabular structure handling approach, where formal knowledge models and semantics decrease ambiguity and hence better determine similarity among a query and a document. In addition, they provide semantically relevant replies to user queries.

The execution of a user query is performed along the following seven steps, as shown in Figure 6 (note the corresponding numbers):

1. *Querying*: users provide queries using CQL [1] (Common Query Language). CQL is a formal language for representing queries to information retrieval systems such as Web indexes, bibliographic catalogs and museum collection information and has been adopted in ALVIS. The CQL design objective is that queries be human readable and human writable, and that the language be intuitive while maintaining the expressiveness of more complex languages such as SQL. Simplicity and intuitiveness of expression is achieved by combining with the richness of Z39.50's type-1 query [32]. This process corresponds to the 'Query requests' box in Figure 2.

2. *Resolution*: the ALVIS query server resolves the CQL query and extracts its key terms. This step corresponds to the 'Distributed search engine' box in Figure 2.

3. *Categorization*: extracted key terms are used to determine potentially interesting categories, topics and genres in particular, where detailed search should be executed. Categorization is perform as described in subsection 2.1 1) and is utilized by Mladenić and colleagues [22] and Bratko and colleagues [5]. The purpose of searching only within certain topics and genres is to significantly narrow the search space and consequently speed up the whole replying process, which is of particular importance when querying the knowledge base of tabular structures. This corresponds to the left-upper bold box (Full test index ...) in Figure 2.

4. *Mapping*: as described in subsection 2.1 (4), each cluster possesses its own automatically generated ontology. The ontology concepts and relations are used together with the extracted key terms in order to reformulate posted CQL queries into new, formal queries. These queries are encoded in the F-Logic formal language, same as the knowledge base. This corresponds to the TARTAR component in Figure 2.

5. *Inference*: The new formal queries, obtained in a previous step, are posted against the knowledge base using an appropriate inference engine, which returns plausible results. This part has been presented in the paper by Pivk and colleagues [30].

6. *Ranking*: ALVIS ranking component ranks, according to its ranking function, the results obtained from the knowledge base with the set of documents that match the CQL query. The ranked results are sent to the output creation module.

7. *Results*: ranked results and document excerpts are gathered and incorporated into a document, which is returned to the user. Note that this document involves also the results inferred from the knowledge base, which represent semantically aware answers to a posted query.

Here we present processing of two simple example queries that relate to the content of tables shown in

Figure 1 and 5. For simplicity, we will first present a query in a natural language, followed by individual step descriptions. Natural language interface is also being developed, but not directly as part of the ALVIS system. The two demonstrating examples are:

A) SHOW THE ACCOMMODATIONS THAT OFFER APARTMENTS.

1. Q="`accommodation=* and type=apartment`";
2. Key terms of Q, such as accommodation, type, and apartment would determine possible categories, i.e. "tourism:skiing:slovenia", and a genre, such as "commercial/shopping".
3. An example of an F-logic query where for mapping purposes the ontology shown in Figure 5 is used:

   ```
   FORALL S,L <- EXISTS A,T,P
   A:StructureCost[Cost@(S,L,
   apartment,T)->P].
   ```

4. The query Q posted against the knowledge base returns the following three results gained from the table examples:

   - S=Bungalow GABER, L=Rogla;
   - S=Pension MARTIN, L=Pohorje;
   - S=Hotel ALPINA, L=?;

5. - 7. The results are ranked and returned to the user as a Web document, as shown in Figure 7. Some results are upgraded with tables, where a table consists of all semantically relevant facts found in the particular document. These results are in the output document marked as 'Reply #' link, where # represents the relevance number of a document introducing the facts. Such examples can be observed from the first two results in Figure 7.

B) SHOW ALL PRICES OF DOUBLE ROOMS IN HOTELS ON POHORJE, SLOVENIA.

1. Q="`price=* and accommodation=hotel and hotel=* and place=pohorje and type='double room'`";
2. Holds a similar description as in the previous example;
3. ```
   FORALL S,T,P <- EXISTS A
   A:StructureCost[Cost@(S,pohorje,
   double room,T)->P and
   isa_(S,hotel)].
   ```
   and
   ```
   FORALL S,T,P <- EXISTS A
   A:StructureCost[Cost@(S,pohorje,
   twin bed,T)->P and isa_(S,hotel)].
   ```
4. - S=Hotel AREH, T=Winter, P=840;
   - S=Hotel AREH, T=Rest, P=690;
5. - 7. Similar as in the previous example.

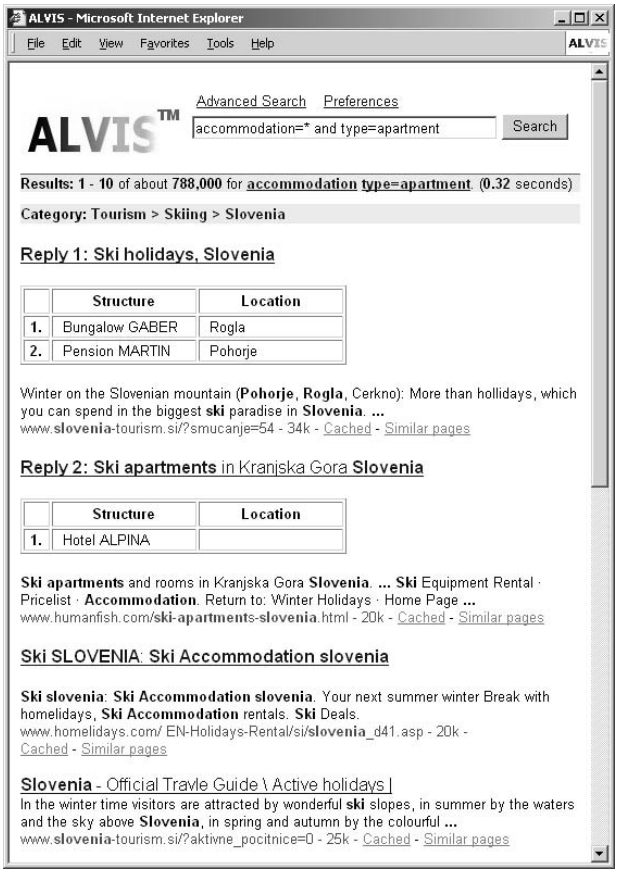


Figure 7: A simulated graphical representation of ranked results, requested in example A).

3 Current state and evaluations

Here we will shortly present the individual state-of-the-art of the described system components and some evaluations. More detailed descriptions are given in the referenced papers.

- *Topic categorization* is based on a hierarchical topic structure. For each topic, a naive Bayesian classifier is built. For an uncategorized document, each of the classifiers estimates the probability of it belonging to a given topic. Experimental evaluation on real world data shows that the proposed approach gives good results. The best performance was achieved with features selected based on a feature scoring measure known from information retrieval called Odds ratio using relatively small number of features [22].
- *Genre categorization*: For this purpose, we created a dataset of over 1500 Web pages manually categorized into 20 genres. Each genre has at least 70 examples. Work on classification is still in progress, but initial tests have been performed by Bratko and colleagues [5] using prediction by partial matching (PPM) classification algorithm [10], which proved very successful in related problem setting. Upon completion of the work, the dataset will be made publicly
   ```

available.

- *Filtering of proper tabular structures*: To solve this problem, we adopted a solution developed by Wang et al. [34], which is based on machine learning techniques, as presented in section 2.1 (2). The classification accuracy reaches almost 95%, meaning that nearly no human intervention is required [25, 26].
- *Formalization of tabular structures*: The empirical evaluation is performed from efficiency and applicability perspectives, where the dataset consisted of over 700 test tables, belonging to two different domains, $t$ourist and $g$eopolitical. First, the efficiency E of the approach is measured according to the portion of correctly transformed tables into knowledge frames reaching 84.52% ($E = \frac{1}{2}(E_t + E_g) = \frac{1}{2}(\frac{289}{369} + \frac{313}{345})$). The more detailed results description is omitted due to its complexity and the lack of space but can be found in [26, 25]. Second, the applicability is shown by querying the content of tables encoded in the knowledge base, where it is shown that returned answers are true and complete in 100% of cases [30].
- *Automatic ontology generation*: Initial tests of the ontology generation process have been shown in [29], indicating that the process is feasible but pointing out that the conflict resolution still needs some improvements. Further tests are still in progress.
- *Knowledge base construction*: This step cannot be evaluated separately since it depends on the results of previous two tasks and merely formalizes the data found within tables. Anyway, the study in [30] showed that by querying the content of tables encoded in the knowledge base, the returned answers are true and complete in all cases.

The system consisting of the described parts is not yet fully implemented. While individual parts have already been tested and achieved the expected performance, the functionality of the integrated system is yet to be verified.

4 Related work

In this paper we have covered several important topics of modern information handling, which can be basically split into three main areas: document categorization, ontology learning, and analysis of tabular structures. We will not present each of these areas thoroughly, since these are given in respective papers, but will rather show their main achievements.

An important portion of information retrieval [3, 31] is focused to the document categorization tasks, where several different techniques and methods have been used, ranging from machine learning techniques such as Bayesian learning, SVM, or (hierarchial) clustering, to natural language processing, graph theory, and combination of these approaches.

Genre categorization is most commonly performed by parsing the documents to extract a set of stylistic features, which vary significantly from author to author, and then using one of the numerous classification algorithms: regression, SVM, decision trees, naive Bayes, clustering. Less common are character-base methods, which use raw text as the input. The best classification accuracy in this group is shown by Peng et al. [24] reaching 86% when classifying documents into 10 different genres. Prediction by partial matching (PPM), the method which we are going to employ, also belongs to this group. Finally, there are visual methods, but these are typically used on scanned documents represented as bitmaps and are not well suited for use in a search engine.

A very recent systematic overview of related work on table recognition, transformation, and inferences can be found in [36]. Most of the work in this area has been done on table detection and recognition addressing several types of document encodings, mostly plain text files, images, and HTML documents. Work performed on textual tables and images was mainly oriented towards table detection, row labeling, and cell classification [36]. Work on HTML tables was extended to indexing relation detection, cell/row/table merging or splitting, classification [34], and other approaches aiming at the deep understanding of table structure [36]. Table extraction methods have also been applied in the context of question answering and ontology learning. A similar proposal to ours was introduced by Tijerino et al. [33]. They presented only a vision for a system that would be able to generate ontologies from arbitrary tables or table-equivalents. Their approach consists of a four-step methodology which includes table recognition and decomposition, construction of mini ontologies, discovery of inter-ontology mappings, and merging of mini-ontologies. For the purpose of semantics discovery the approach is multifaceted, meaning they use all evidence at their disposal (i.e. Wordnet, data frames, named entities, etc.). Since the paper, unlike ours, presents only a vision, no evaluation is provided.

5 Conclusion and Future Work

ALVIS represents a major attempt to create a semantic search engine. For this purpose, it introduces several novel approaches and mechanisms. Our approach is novel in the sense that it is the first to address the whole process of semantically aware information discovery and query answering by analyzing information presented in tabular structures. By putting together the components, such as two-aspect document categorization, detection, filtering, transformation and formalization of tabular structures, a search engine gets enhanced with new capabilities in a scalable way.

Each of the major formalization algorithmic steps is important in the overall process and benefits in terms of accuracy and reduced complexity. We anticipate that the system

will also benefit in terms of scalability and speed, but cannot show or discuss that in details since it has not been fully implemented and tested yet.

Regarding success rate of HTML input tabular transformations into formal semantic descriptions, the proposed approach reaches nearly 85%. Besides the transformation the approach also enables annotation of the original resources with generated formal descriptions.

Although tabular semantic approach by itself may not be the ultimate research goal, and although its success relies on specific properties of tables, i.e. implicit relationships, it might give some indications regarding semantic search from plain text. We are aware that our approach and the ontology learning text processing approaches tackle and cope with different problems, thus making direct comparison nearly impossible, we still assume that our results show great potential for wide acceptance.

Future research in the tabular structure analysis area is aiming into two main directions. Firstly, the focus will be to incorporate some machine learning techniques, in particular for: (a) the classification of tabular structures into the (predefined) table type classes, and (b) the improved and more scalable discovery and division of tables into logical units and regions. This will enable better grounds for the ongoing transformation process. Secondly, our methodology is domain and document type independent, but has been implemented to cover HTML tables only. Therefore two important extensions need to be provided in the future: (a) the extensions to cover other document types, i.e. PDF, excel, etc., and (b) to enable a framework for a simple inclusion of the other domain-dependent knowledge models. With these extensions comprehended, we anticipate that the approach will be used to semantically enrich a variety of legacy data and will support the conversion of the existing Web into a Semantic Web.

Acknowledgement

This work has been supported by the EU IST-projects ALVIS (Superpeer Semantic Search Engine, IST-1-002068-STP) and SEKT (Semantically Enabled Knowledge Technologies, IST-2004-506826). The research was also supported by Slovenian Ministry of Education, Science and Sport, and by Marie Curie Fellowship of the European Community program 'Host Training Sites', mediated by FZI and AIFB at University of Karlsruhe, Germany. Thanks to all our colleagues for participating in the evaluation of the system as well as to the reviewers for useful comments on the paper.

References

[1] Z39.50 International Maintenance Agency. CQL - Common Query Language, 2004. `http://www.loc.gov/z3950/agency/zing/cql/`.

[2] A. Antonacopoulos and J. Hu. *Web Document Analysis: Challenges and Opportunities*. World Scientific, 2004.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001(5), 2001.

[5] A. Bratko and B. Filipic. Exploiting structural information for semi-structured document categorization. *Information Processing & Management*, 42(3):679–694, 2006.

[6] P. Buitelaar, P. Cimiano, and B. Magnini, editors. *Ontology Learning from Text: Methods, Applications and Evaluation*, volume Frontiers in Artificial Intelligence and Applications. IOS Press, 2005.

[7] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[8] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale HTML texts. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 166–172, 2000.

[9] P. Cimiano, L. Schmidt-Thieme, A. Pivk, and S. Staab. Learning taxonomic relations from heterogeneous evidence. In P. Buitelaar, P. Cimiano, and B. Magnini, editors, *Ontology Learning from Text: Methods, Applications and Evaluation*, pages 56–71. IOS Press, 2005.

[10] John G. Cleary and Ian H. Witten. A comparison of enumerative and adaptive codes. *IEEE Transactions on Information Theory*, 30(2):306–315, 1984.

[11] C. Fellbaum. *WordNet, an electronic lexical database*. MIT Press, 1998.

[12] W.B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1992.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *VLDB*, pages 518–529, 1999.

[14] J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard? In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 129–133, 2001.

[15] M. Hurst. Layout and language: Beyond simple text for information interaction - modelling the table. In *Proceedings of the 2nd International Conference on Multimodal Interfaces*, 1999.

[16] M. Hurst. Layout and language: Challenges for table understanding on the web. In *Proceedings of the International Workshop on Web Document Analysis*, pages 27–30, 2001.

[17] B. Jansen, A. Spink, and J. Bateman. Searchers, the subjects they search, and sufficiency: A study of a large sample of excite searchers. In *Proceedings of the World Conference on the WWW, Internet and Intranet (WebNet-98)*, pages 913–928. AACE Press, 1998.

[18] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[19] J.M. Kleinberg. Two Algorithms for Nearest-neighbor Search in High Dimension. In *ACM Symposium on Theory of Computing*, pages 599–608, 1997.

[20] J.M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[21] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002.

[22] D. Mladenič and M. Grobelnik. Feature Selection on Hierarchy of Web Documents. *Decision Support Systems*, 35:45–87, 2003.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998. `http://www-db.stanford.edu/~backrub/pageranksub.ps`.

[24] F. Peng, S. Dale, and W. Shaojun. Language and Task Independent Text Categorization with Simple Language Models. In *Proceedings of the Human Language Technology Conference of the ACL*, pages 189–196, Edmonton, Canada, 2003.

[25] A. Pivk. *Automatic Generation of Ontologies from Web Tabular Structures*. PhD Thesis (in Slovene), University of Maribor, Slovenia, 2005.

[26] A. Pivk. Automatic Ontology Generation from Web Tabular Structures. *AI Communications*, 19(1):83–85, 2006.

[27] A. Pivk, P. Cimiano, and Y. Sure. From Tables to Frames. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2–3):132–146, 2005.

[28] A. Pivk and M. Gams. Domain-dependant information gathering agent. *Expert Systems with Applications*, 23:207–218, 2002.

[29] A. Pivk and M. Gams. Construction of domain ontologies from tables. In *Proceedings of the 8th International Multi-Conference on Information Society (IS'05)*, pages 615–619, 2005.

[30] A. Pivk, Y. Sure, P. Cimiano, M. Gams, V. Rajkovic, and R. Studer. Transforming Arbitrary Tables into Logical Form with TARTAR. *Data & Knowledge Engineering*, accepted, 2006.

[31] F. Sebastiani, editor. *Advances in Information Retrieval*, Proceedings of the 25th European Conference on IR Research (ECIR 2003), Lecture Notes in Computer Science, Vol. 2633, Pisa, Italy, April 14-16, 2003. Springer.

[32] M. Taylor and M. Cromme. Searching very large bodies of data using a transparent peer-to-peer proxy. In *Proceedings of DEXA 2005 Workshop*, pages 1049–1053. IEEE Computer Society, 2005.

[33] Y.A. Tijerino, D.W. Embley, D.W. Lonsdale, and G. Nagy. Ontology generation from tables. In *Proceedings of 4th International Conference on Web Information Systems Engineering (WISE'03)*, pages 242–249, 2003.

[34] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web*, pages 242–250. ACM Press, 2002.

[35] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compression and Indexing Documents and Images*. Multimedia Information and Systems. Morgan Kaufmann, 1999.

[36] R. Zanibbi, D. Blostein, and J.R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. *International Journal of Document Analysis and Recognition*, 7(1):1–16, 2004.

Beyond Term Indexing: A P2P Framework for Web Information Retrieval

Ivana Podnar, Martin Rajman, Toan Luu, Fabius Klemm and Karl Aberer
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Keywords: information retrieval, peer-to-peer overlay networks

Received: December 05, 2005

Web search over peer-to-peer (P2P) networks shows promise to become an alternative to the state-of-the-art search engines since P2P overlays offer means for decentralized search across widely-distributed document collections. However, the design of effective techniques for P2P indexing and retrieval raises a number of technical challenges due to potentially unscalable resource (e.g. bandwidth, storage) consumption.
The paper presents a framework for full-text information retrieval in structured P2P networks and introduces a novel retrieval model based on highly discriminative keys*—terms and term sets appearing in a restricted number of documents—that ensure efficient and scalable retrieval. Our goal is to design scalable techniques for building a* global key index *in structured P2P overlays for large document collections. We present experimental results that show acceptable indexing and retrieval costs while the retrieval quality is comparable to standard centralized solutions with BM25 relevance computation scheme.*

Povzetek: Razvito je P2P ogrodje za internetne iskalnike.

1 Introduction

Web search over P2P overlay networks has the potential to become an alternative to current Web search engines due to its decentralized nature and favorable scalability properties. Contemporary Web search engines are in essence centralized and require a central coordination service, which, even when replicated, has been identified as a major system bottleneck [5]. P2P overlays are self-organizing networks for resource sharing that require no central coordination. Moreover, they require minimal infrastructure and maintenance, and enable higher diversity in contents and search methods, which makes P2P Web search appealing from a business perspective [4].

However, there is an ongoing debate on the feasibility of P2P Web search for scalability reasons. In [9] it is shown that the naïve use of P2P networks is practically infeasible for Web search, since the generated traffic exceeds the available capacity of the Internet. Thus different schemes have been devised to make P2P Web search feasible. Several approaches target at a term-to-peer indexing strategy, where the unit of indexing are peers rather than individual documents [7, 3]. Hierarchical solutions for resource discovery have also been proposed where a backbone P2P network maintains a directory service which routes queries to peers with relevant resources, i.e. documents [2, 11]. At this point little evidence is available whether these approaches can scale to very-large scale P2P Web search engines.

In this paper we introduce a general framework for information retrieval (IR) over structured P2P networks which keeps indexing at document granularity while ensuring scalable retrieval costs. Our assumption is that peers are cooperative and provide documents for indexing that will become searchable through a global index. At the same time, they offer computing and storage resources to build and maintain the global index and the underlying P2P network. Peers choose independently the indexing features and related posting lists for their local document collection and, for this task, combine their local knowledge and the global knowledge maintained by the P2P overlay. A peer basically advertises its local document collection and collaborates with other peers by taking the responsibility for a part of the global indexing space. Note that a single peer cannot impose the rules used at a global level, but a malicious peer might impede the search performance by providing fictive descriptions of its local collection.

The framework assumes an underlying distributed hash table (DHT) which maintains a global *key-to-document index*. The approach is characterized by the following central idea: We carefully select the indexing keys so that they consist of terms and term sets that are *rare* and thus *discriminative* with respect to a global document collection. As such, they appear in a limited number of documents, thus limiting the size of the posting lists associated with the keys in the global inverted index. This directly addresses the main problem identified in [9] for structured P2P Web search, namely the processing and transmission of extremely large posting lists. The achieved key-based indexing procedure results in an extremely efficient retrieval because the posting lists associated with the keys in the global P2P index are short and correspond to precomputed and therefore readily available results for potential multiple term queries.

Many possible techniques can be considered for creating

a rare key vocabulary, but the major issue to cope with is the fact that such a vocabulary can easily become unmanageable in size because it theoretically grows with $2^{|V|}$, where V is the single-term vocabulary. Therefore, we present a particular instantiation of our key-indexing which creates keys by combining terms appearing in well-defined contexts in a limited number of documents. We hereafter refer to such keys as highly discriminative keys (HDKs). Our experiments show the following: The growth of the HDK vocabulary per peer and the size of the global index per peer are logarithmic when increasing the global document collection by adding new peers that contribute with their local documents to the global collection. More importantly, the average posting list size remains constant when increasing the global document collection size, which bounds the generated traffic during retrieval. In addition, the observed retrieval performance is comparable to the one achieved with a single-term centralized model using BM25 (Okapi) as relevance computation scheme. Our indexing scheme therefore represents a contribution toward scalable P2P Web search engines that opens the opportunity to index and search virtually unlimited document collections, well beyond the capacity of today's best centralized solutions.

The rest of the paper is structured as follows: Section 2 gives an overview of existing strategies for P2P information retrieval and introduces our concept of indexing with HDKs. Section 3 presents the P2P framework for building a global full-text index in structured P2P overlays with scalable retrieval costs and provide details about the HDK-based retrieval model. In Section 4 we analyze its performance through experimental evaluations using our truly-distributed P2P-IR prototype. The related work in the area of P2P information retrieval is revisited in Section 5, and we conclude the paper in Section 6.

2 Web information retrieval over P2P overlays

There are two main strategies for designing P2P search engines in the area of IR: The first strategy uses unstructured/hierarchical P2P networks where peers maintain local indexes of their document collections, and the second strategy builds a global index in structured P2P networks.

In a P2P network with N peers and a global document collection D, the first strategy divides D over the peer network, and each peer P_i maintains the index of its local document collection D_i. Such indexes are in principle independent, and a naïve solution broadcasts a query to all the peers generating an unscalable number of messages. To limit the query traffic, more advanced approaches use e.g. random walks [12] or answer the query at two levels, the peer and document level. At the first level a group of peers with relevant document collections is located, while at the second level the query is submitted to relevant peers that produce answers by querying their local indexes. The answers are subsequently merged to produce a single ranked answer set. A number of different approaches for the initial peer selection process have been proposed in the context of unstructured [7], hierarchical [2, 10], and structured P2P overlays [3].

The second strategy splits the global document index over a structured P2P network that maintains a global DHT mapping of each indexing feature to the related posting list [15, 18]. Structured P2P overlays offer efficient data inserts and searches [1]: $insert(key, data)$ routes the data to the peer responsible for the given key, and $search(key)$ retrieves the data for the given key. Such networks typically guarantee the routing latency of $O(logN)$ number of hops, while the routing information maintained by each peer is bounded by $O(logN)$.

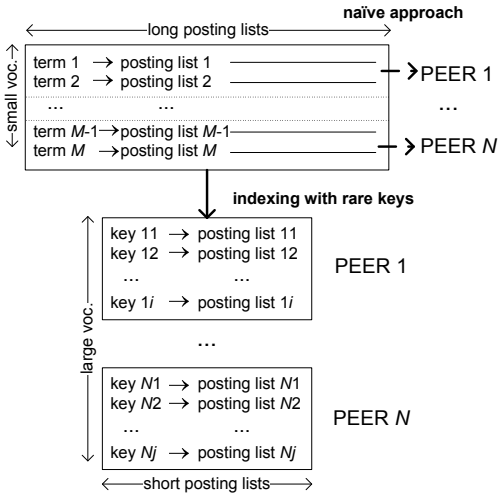


Figure 1: The basic idea of indexing using rare keys

The naïve approach splits the global index over a peer network as depicted in the upper part of Figure 1 and makes each peer responsible for a disjoint set of indexing terms from the global vocabulary. While solving the issue of storing very large indexes, such an approach is practically infeasible and unscalable because of extremely large posting lists for frequent terms. The *key-based indexing* approach directly addresses this issue by limiting the size of the posting lists as shown in the lower part of Figure 1. The key index only contains single terms and term sets that are *rare* and thus *discriminative* with respect to a document collection. Notice that the extension of the index entries to rare terms and term sets, while limiting the size of the posting lists, entails the expansion of the key vocabulary. However, the distributed nature of the global index can provide a virtually unlimited storage space if the number of peers is large enough, but only in case the growth of the key vocabulary size and the corresponding index size is scalable.

We define a key k from the set of keys K as a set of terms $\{t_1, t_2, \ldots, t_s\}$. These terms are elements of the document collection indexing vocabulary V appearing in a single document $d \in D$. The number of terms comprising a key is bounded, i.e. $1 \leq s \leq s_{max}$.

The quality of a key k for a given document d with respect to indexing adequacy is determined by its *discriminative power*. To be *discriminative*, a key k must be as specific as possible with respect to the document d it is associated with and the corresponding document collection D [17]. In this perspective, we categorize a key on the basis of its *global document frequency* (DF), and define a threshold DF_{max} to divide the set of keys K into two disjoint classes, rare and frequent keys. If a key k appears in more than DF_{max} documents, i.e. $DF(k) > DF_{max}$, the key is *frequent*, and has low discriminative power. Otherwise, k is *rare* and specific with respect to the documents it is associated with in the document collection.

The rareness property is in line with the capacity constraints of P2P networks that are capable of storing and transmitting posting lists of limited size. However, the set of rare keys, although bounded for a limited document collection size, might be extremely large. It is therefore important to select among the potential key candidates those that have favorable properties for the retrieval performance of the P2P search engine. Additionally, it is important to understand that keys can be interpreted as discriminative user queries that can be answered very efficiently because the global index already stores precomputed answers associated with such queries.

3 P2P framework for information retrieval with HDKs

In this section we present a general framework for full-text retrieval in a P2P environment using key-based indexing. The framework is designed for building a global key-based index in structured P2P networks and takes into account capacity constraints of P2P networks, such as available bandwidth and storage. Resource management is performed at two levels, *locally* as a peer builds an index of its local document collection to be subsequently inserted into the global index, and *globally* as the peer maintains a part of the global index. Note that a single peer can simultaneously perform tasks at both levels.

3.1 Indexing

Indexing can be divided into the following three steps:

1. local selection of the vocabulary,
2. local selection of postings, and
3. global selection of postings.

The two initial steps represent local efforts of a single peer to choose suitable indexing features for its local document collection since each peer is competing with others for a part of the global index space. The third step is performed by all the peers maintaining the global index.

As the example in Figure 2 shows, P_i builds the key vocabulary K_i for its local document collection D_i. In the next step it chooses a set of postings associated with each key according to the global resource constraints it is aware of, and subsequently inserts the $(key, posting\ list)$ pairs into the P2P overlay. In the given example, P_j is responsible for three such pairs from P_i. Concurrently, two other peers P_k and P_l have built a vocabulary for their document collections, and have started inserting $(key, posting\ list)$ pairs into the P2P overlay. In the third step P_j chooses postings to be stored in the global index, e.g., it builds a posting list for k_x using the postings received from P_i, P_k and P_l.

We will now describe in more details each of the three indexing steps.

3.1.1 Vocabulary selection

A peer chooses locally rare keys because it assumes they are good indexing candidates that are potentially globally rare. A peer can also put frequent keys into the key vocabulary if they are strongly representative of the documents they are associated with (as measured by standard IR weighting schemes). The goal of this phase is to select discriminative keys that meet the following criteria: the selected keys can achieve good retrieval quality, they are likely to appear in user queries, and the key vocabulary does not contain redundant keys.

Possible key filtering mechanisms for selecting a 'good' key vocabulary are the following:

Proximity. The *proximity filter* uses textual context to reduce the size of the rare key vocabulary, and retains keys only composed of terms appearing in the same textual context, e.g., a phrase, a paragraph, or a document window of a given maximal size. The main argumentation behind this approach is that words appearing close to each other in documents are good candidates to appear together in a query. The analysis presented in [16] reports the importance of text passages that are more responsive to particular user needs than full documents. A similar reasoning is used in a recently proposed method for static index pruning [8] which indexes 'significant sentences', i.e. phrases appearing in similar contexts.

Redundancy. A key k is *semantically adequate* for a document d if there is a high probability that a user produces k when searching for d. As the user is more likely to generate generic terms in a query, a semantically adequate key contains a specific combination of quite generic and frequent terms. In other words, the redundancy filter ensures that all term subsets of a key are frequent keys. Note that all supersets of rare keys are redundant, as they only increase the vocabulary without improving retrieval performance.

Top-k keys. As it is possible to compute relevance between a document and a key, we can choose for each document the top-k most relevant keys. The number of chosen keys can, for example, depend on document

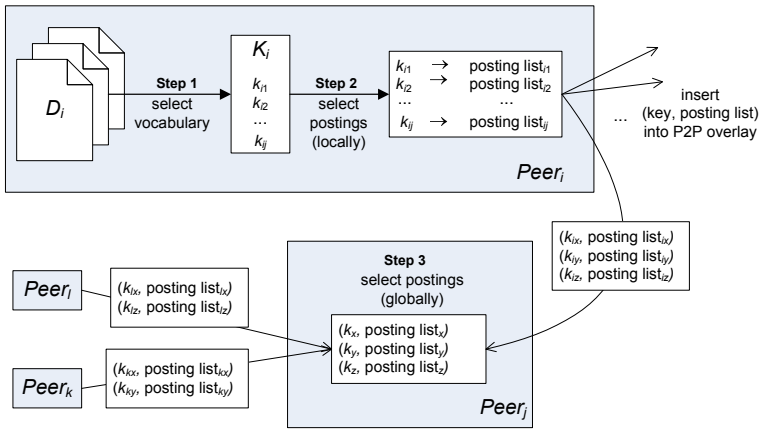


Figure 2: The framework for P2P indexing

length and be controlled by a relevance threshold. The key vocabulary size can also be limited to a predefined value.

Query-adaptive indexing. Keys are coined by combining terms appearing in past user queries.

Note that the above mentioned techniques can be combined to reduce the key vocabulary size. Our current prototype uses the proximity and redundancy filter, and computes HDKs by combining terms appearing within a document window of maximal size w while ensuring that all key subsets are globally frequent. We also extend the key vocabulary with *frequent keys* to improve retrieval performance. The top-k indexing of frequent keys is feasible because our analysis shows that the size of the frequent key vocabulary is less than 1% of the HDK vocabulary size.

Computing HDKs. We compute HDKs of increasing size because a peer needs to know s-level globally frequent keys to compute $(s+1)$-level keys since these are candidates for expansion with additional keys. An indexing peer first categorizes terms from V as locally rare and locally frequent single-term keys, and publishes their local DFs in the P2P network. In our current implementation the indexing peer needs to be informed if keys from its local document collection are *globally* frequent. Locally frequent keys are necessarily globally frequent, but locally rare keys may still be globally frequent. As the indexing peer has no local knowledge about the global key DFs, it relies on the P2P network which maintains the global statistics and notifies the indexing peer if a key considered as locally rare is globally frequent. With such a notification mechanism in place, each indexing peer will have a consistent knowledge about globally frequent keys for its local document collection. A globally frequent key is expended by another frequent key with a restriction that their terms occur within a predefined window as defined by the proximity filter. The indexing peer counts the number of documents in which this property holds to determine local DF of a newly created key, and categorizes it as locally rare or frequent. Then it publishes the key and its DF into the P2P network. The process continues until s_{max}-term keys are created.

3.1.2 Local selection of postings

In this phase, a peer decides which documents are indexed for its key vocabulary. Since there are capacity constrains that restrict the number of postings associated with a key, each peer is allowed to insert only a limited number of postings, e.g. controlled by a globally defined value DF_{max}. Firstly, a peer can insert all postings for locally rare keys, and the P2P overlay then decides globally about the set of postings that get stored. Secondly, a peer can be notified when its already published locally rare keys become globally frequent, and using this global knowledge provided by the P2P network, insert postings only for globally rare keys, which will reduce the indexing traffic compared to the first solution. Thirdly, for locally and globally frequent keys, a peer can publish top-DF_{max} representative postings by ranking documents based on their relevance with respect to a given key.

Our current prototype implements the second and third option: Indexing peers publish all local postings for globally rare keys. To improve retrieval performance and efficiently answer queries composed of frequent terms and term combinations, we have decided to index also globally frequent keys and maintain only DF_{max} postings in the global index. Indexing peers therefore also insert top-DF_{max} postings associated with their local keys that are globally frequent.

3.1.3 Global selection of postings

This phase takes place at the peer responsible for storing and maintaining the posting list associated with a given key. The total index size maintained by the peer is has to comply with its available storage space, which in turn also influences the maximum number of postings associated with a key. Furthermore, the posting list size is restricted to meet capacity constraints imposed by the retrieval phase.

For globally rare keys peers store all received posting lists because rare key are by definition associated with short posting lists. If there are more postings than the constraints allow, the responsible peer has to select a set of postings to keep as it is already done locally for frequent keys. The mechanisms for choosing DF_{max} postings can range from a simple random selection procedure, to sophisticated ranking of postings using available techniques for content and link-based ranking. Our prototype currently stores all received posting associated with globally rare keys and top-DF_{max} ranked postings associated with frequent keys, i.e. documents with highest relevance to a frequent keys according to TF-IDF scores.

3.2 Retrieval

The retrieval part of our framework deals with the problem of finding, given a query $Q = \{t_1, t_2, \ldots, t_q\}$, the corresponding relevant keys in the global P2P index, retrieving the postings associated with those keys, and ranking of the result set. It is performed in two steps:

1. mapping of a query to keys and subsequent retrieval of postings from the P2P overlay, and

2. merging of the received answers with a global ranking scheme in order to produce a single hit list.

Figure 3 shows the querying process when peer P_i receives the query Q. We assume Q is a simple set of terms, and it must be mapped to a set of keys present in the P2P global index. All term subsets of a query are possible key candidates that may be stored in the P2P index, and P_i needs to explore the lattice of term combinations and check which key candidates are indeed keys from the global index.

The optimistic strategy is to start with the largest possible term set (marked as level-1), limited either by the query size q (as it is assumed in Figure 3) or the maximal number of terms forming a key (s_{max}). P_i will try to retrieve the keys sequentially by exploring initially level-1 keys, and then depending on the result of the previous step continue with level-2 keys, ..., level-q keys. The perfect situation occurs when $k_{11} = \{t_1, t_2, \ldots, t_q\}$ is an HDK, in other words, a user has posed a good discriminative query for the indexed document collection: The posting list is readily available and is simply retrieved from the global index by issuing a single request $search(k_{11})$ to the P2P network. Indeed, this will not happen with all user queries. Therefore, level-2 set of potential key candidates of size $q - 1$ is explored. If the P2P network stores postings associated with level-2 keys that cover all terms in Q, the hit list is the union of the retrieved postings. However, it is possible that either (a) no level-2 keys exist in the index, or (b) that some $t_i \in Q$ are not covered by the retrieved keys. In case of (a), P_i explores all level-3 keys and subsequently higher level keys until finally reaching level-q. In case of (b), P_i explores level-3 keys but only those that contain non-covered terms from the previous levels. Depending on the retrieved set of keys, the procedure is extended to higher-level keys until all $t_i \in Q$ are covered by the retrieved key set. The resulting answer set is the union of postings associated with the retrieved keys.

However, in some cases the answer set may be empty because all key candidates are frequent. A valid option is to notify a user that the query is non-discriminative with respect to the document collection, and offer support for query refinement. Another possibility is to apply additional measures and, for example, index frequent keys or perform query expansion that maps terms to semantically related terms which enables the creation of many key-candidates and increases the probability of finding relevant keys in the global index. As our prototype currently indexes frequent keys, all single-term keys from V can eventually be retrieved, which solves the problem of queries that map to frequent keys.

Let us again consider the example in Figure 3. The P2P network does not contain k_{11}, and P_i searches for level-2 keys. Out of q possible level-2 keys, only k_{21} is an HDK and its posting list is retrieved from P_l. However, t_q is still not covered, and P_i unsuccessfully explores keys on higher levels containing t_q, until finally reaching level-q where a posting list is associated with k_{qq}.

Both posting lists are merged by a simple union procedure in the second step. Note that there is a possibility that t_q appears in some documents associated with k_{21}, but as it does not appear within the predefined window, k_{11} was not produced during the indexing procedure. Note also that documents associated with k_{qq} are those with highest relevance to t_q, but they may also contain other terms from Q, also outside the window. The ranking function produces the final ranked result set giving higher scores to documents with higher relevance to Q. The peer uses available ranking methods, either content based, and/or link-based, to produce the final ranking of the merged result set. The ranking procedure must be implemented in a distributed fashion using global document collection statistics such as global DFs available from the P2P index.

4 Experimental evaluation

Experiments have been performed using our prototype P2P search engine which is built on top of the P-Grid P2P layer [19]. Our implementation is a fully-functional P2P search engine that integrates a solution for distributed maintenance of global key vocabulary with associated document frequencies, calculates HDKs used for indexing in a completely distributed fashion, and stores posting lists in a global index for computed keys. The presented experiments analyze both rare and frequent keys associated with top-DF_{max} postings, and the corresponding index sizes.

Experimental setup. The experiments were carried out using a subset of news articles from the Reuters corpus[1].

[1] http://about.reuters.com/researchandstandards/corpus/

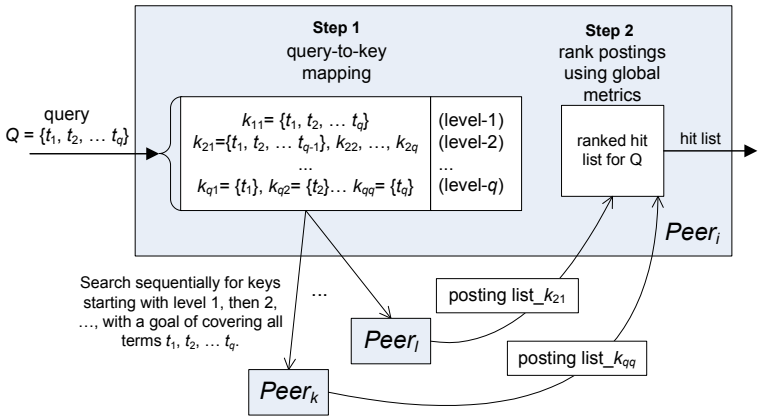


Figure 3: The framework for P2P retrieval

The documents in our test collection contain between 70 and 3000 words, while the average number of terms in a document is 170, and the average number of unique terms is 102. To simulate the evolution of a P2P system, i.e. peers joining the network and increasing the document collection, we started the experiment with 4 peers, and added additional 4 peers at each new experimental run. Each peer contributes with 5000 documents to the global collection, and computes HDKs for its local documents. Therefore, the initial global document collection for 4 peers is 20,000 documents, and it is augmented by the new 20,000 documents for each experimental run. The maximum number of peers is 24, hosting in total the global collection of 120,000 documents. The experiments were performed on our campus intranet. Each peer runs on a Linux RedHat PC with 1GB of main memory connected via 100 Mbit Ethernet. The prototype system is implemented in Java.

Performance analysis. Experiments investigate indexing and retrieval costs in terms of bandwidth and storage consumption, and compare retrieval performance of our P2P search engine to a centralized engine. All documents are pre-processed: First we removed 250 common English stop words and applied the Porter stemmer, and then we removed 100 extremely frequent terms (e.g. the term 'reuters' appears in all the news). DF_{max} is set to 250 and 500, and s_{max} is 3. We have decided to build keys with a maximum of 3 terms having observed that an increase of s_{max} substantially increases the computational load without significant improvement of retrieval performance. In particular this is beneficial for P2P Web search engines because short queries are the ones typically used for Web retrieval. We set the window size to 20 words for the proximity filter as this is the average length of phrases in the collection.

Figure 4 shows the growth of the HDK vocabulary per peer maintained in the global index when increasing the number of documents by adding new peers. Note that each peer contributes an equal number of documents into the global collection. As expected, an increased value of DF_{max} reduces the key vocabulary size because there are

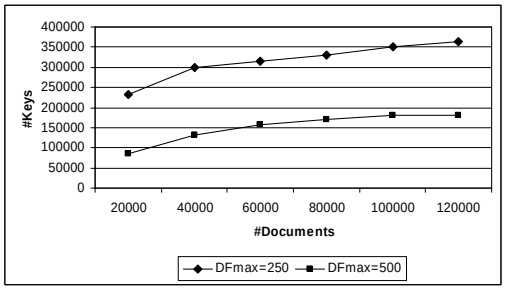


Figure 4: HDK vocabulary per peer

less frequent single-term keys to build 2 and 3-term rare keys. Both experimentally obtained result sequences show logarithmic growth and are expected to converge to a constant value for large document collection sizes. We can conclude that a peer will maintain a negligibly increasing number of HDKs when increasing the document collection size by adding new peers. Therefore, our experiments show that the size of the total key vocabulary maintained in the global P2P index will increase almost linearly with the number of documents for large document collections. The number of keys is significantly larger compared to the single-term vocabulary that grows with $\sqrt{x}$ (Heaps law), but we expect benefits in terms of retrieval efficiency and bandwidth consumption.

Figure 5 compares the average posting list size in case of HDK and single-term indexing. Surprisingly, the average posting list size of the HDK index remains constant and much smaller than DF_{max}, although we only limit the maximum size of posting lists. It is superior to single-term indexing that exhibits a linear increase of the average posting list size. This finding has a major impact on the bandwidth consumption during retrieval as it is shown in Figure 6 that depicts the average number of retrieved postings per query and compares single-term to the HDK approach with $DF_{max} = 250$ and $DF_{max} = 500$.

Since a query log is not available for the Reuters corpus,

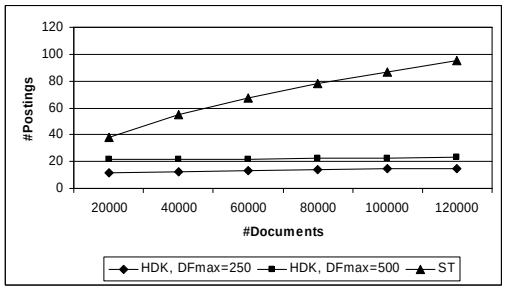


Figure 5: Average posting list size

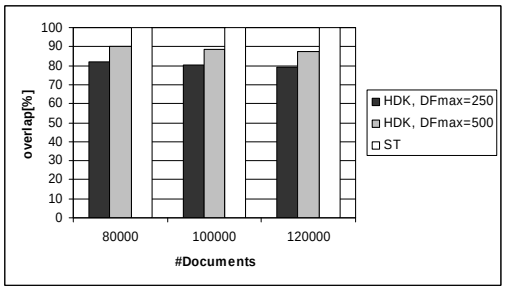


Figure 7: Overlap on top 20

we have preprocessed a Wikipedia query log[2] to generate 1,000 queries. From all available queries we have initially extracted 1,000,000 queries that have more than 20 hits from the Reuters corpus. The resulting 1,000 queries were selected randomly from this set. The generated queries contain on average 3.14 terms, with a minimum of 2 and maximum of 8 terms. Note that longer queries are favorable for our search engine because the probability of finding precomputed rare keys in the P2P index increases. Single term queries would generate bounded traffic since only DF_{max} postings are retrieved and are therefore not considered.

Figure 6 shows an enormous reduction of bandwidth consumption per query of the HDK-based approach compared to the naïve single term indexing: 92% for $DF_{max} = 500$ and even 96% for $DF_{max} = 250$. In addition, the reduction increases when increasing the number of documents in the global collection which practically demonstrates the effect of the bounded number of index postings to bandwidth consumption during retrieval.

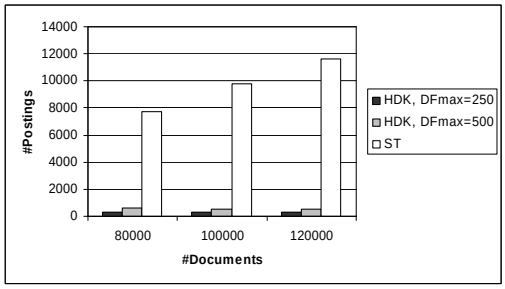


Figure 6: Number of retrieved postings per query

The essential question remains whether the retrieval performance of the HDK approach is satisfactory and comparable to centralized counterparts. Due to the lack of relevant judgment for the generated query set, we have compared the retrieval performance to a centralized engine[3] with BM25 relevance computation scheme which is currently considered as one of the top performing relevance schemes [13].

[2]www.wikipedia.org, query log 08/2004 and 09/2004

[3]Terrier search engine, http://ir.dcs.gla.ac.uk/terrier/

Figure 7 presents the overlap on top-20 retrieved documents retrieved by our system and the Terrier search engine. We are interested in the high-end ranking as typical users are often interested only in the top 20 results. The comparison shows significant and satisfactory overlap between the retrieved result sets. As expected, the retrieval performance is better for larger DF_{max} as we are approaching single-term indexing (when DF_{max} equals vocabulary size, all HDKs are single-term). There is obviously a trade-off between retrieval quality and bandwidth consumption of our indexing strategy because an increased value of DF_{max} results in an increased bandwidth consumption during retrieval, while on the other hand, offers retrieval performance that better mimics centralized engines. However, as required bandwidth is the major obstacle for retrieval, it is vital to choose an adequate value for DF_{max} taking into account available network capacity. There is another limiting factor influenced by this parameter: Note that the process of building the key vocabulary is less expensive in terms of computational costs and bandwidth consumption for an increased DF_{max}. This is intuitively clear because for small values of DF_{max}, the number of potential terms for building rare keys increases, and the whole process creates more precomputed keys, which is obviously very efficient during the retrieval phase if such term sets, i.e. keys, appear in user queries.

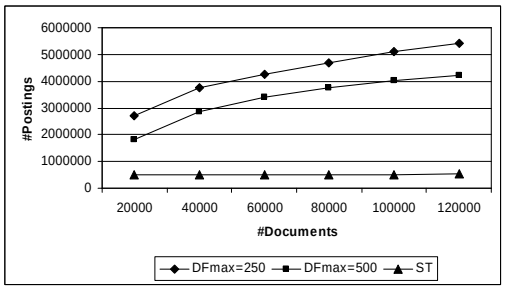


Figure 8: Number of postings per peer (index size)

To quantify indexing costs and the influence of DF_{max} on required storage and bandwidth consumption, Figure 8 shows the average number of postings stored per peer for $DF_{max} = 250$, $DF_{max} = 500$, and the single-term index.

This is the actual index size per peer. The curve is logarithmic for HDKs as it is mostly influenced by the key vocabulary size, and is linear for the single-term index (the decrease of the term vocabulary size per peer compensates for the increased posting list size). It is visible that a peer hosts significantly more postings for the HDK approach, and the index size increases when decreasing DF_{max}. The indexing costs are still feasible and profitable because the gains in terms of bandwidth consumption during the query phase are huge, and compensate for the increased indexing costs. Nevertheless, we plan to investigate further techniques to reduce the index size, and study the influence of DF_{max} on bandwidth consumption during indexing and retrieval, and on the resulting retrieval performance.

5 Related Work

To the best of our knowledge, [6] is the only published P2P framework for information retrieval that is based on distributed semantic indexing on top of structured P2P networks, and therefore highly related to our work. However, it uses a different indexing approach which relies on semantic locality to cluster documents with similar semantics, and to store them on nearby peers.

A number of solutions have been proposed to cope with the scalability problem of P2P information retrieval. Recent approaches combine global knowledge with local indexes: For example, PlanetP [7] gossips compressed information about peers' collections in an unstructured P2P network, while MINERVA [3] maintains a global index with peer collection statistics in a structured P2P overlay to facilitate the peer selection process, and implements a method which penalizes peers holding overlapping document collections. A similar approach for resource selection is presented in [10, 11] in the context of hierarchical P2P networks where special directory nodes route queries to appropriate peers having high chances of answering a query. A recent solution builds an index dynamically following user queries [2]. A super-peer backbone network maintains the information about good candidates for answering a query, while peers answer the queries based on their local document collections. Since large posting lists are the major concern for global single-term indexing, both [15] and [18] have proposed top-k posting list joins, Bloom filters, and caching as promising techniques to reduce search costs for multi-term queries.

The listed solutions are orthogonal to our approach since they use different assumptions to reduce network traffic. However, our approach is not the only indexing strategy that uses term sets as indexing features. The set-based model [14] indexes term sets occurring in queries, and exploits term correlations to reduce the number of indexed term sets. The authors report significant gains in terms of retrieval precision and average query processing time, while the increased index processing time is acceptable. In contrast to our indexing scheme, the set-based model has been used to index frequent term sets, and has been designed for a centralized setting.

6 Conclusion

We have presented a P2P framework for information retrieval that uses a novel retrieval model based on global indexing of rare keys in structured P2P overlay networks. Rare keys are terms and term sets occurring in a limited number of documents, limiting thus the size of posting lists stored in the global index. This approach directly address the unscalable network traffic cased by large posting lists for single-term indexing. The experimental results have shown the potential of our approach in preserving a retrieval quality (top-k precision) comparable to the standard single term with BM25 scoring scheme with enormous reduction of generated traffic during retrieval. Next, it gives evidence that the indexing process is feasible because it produces the key vocabulary and global index of manageable size. More importantly, the average posting list size remains constant when increasing the global document collection size. Therefore, the generated traffic during retrieval remains bounded, which solves one of the major obstacles for scalable IR in P2P networks.

Acknowledgements

The work presented in this paper was carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European FP 6 STREP project ALVIS (002068).

References

[1] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The Essence of P2P: A Reference Architecture for Overlay Networks. In *Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 11–20, 2005.

[2] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Dl meets p2p - distributed document retrieval based on classification and content. In *9th European Conference on Research and Advanced Technology for Digital Libraries, (ECDL)*, pages 379–390, 2005.

[3] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in P2P search engines. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '05)*, pages 67–74, 2005.

[4] W. Buntine, K. Aberer, I. Podnar, and M. Rajman. Opportunities from open source search. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 2–8, 2005.

[5] F. Cacheda, V. Plachouras, and I. Ounis. A case study of distributed information retrieval architectures to index one terabyte of text. *Inf. Process. Manage.*, 41(5):1141–1161, 2005.

[6] Y. Chen, Z. Xu, and C. Zhai. A scalable semantic indexing framework for peer-to-peer information retrieval. In *SIGIR 2005 workshop: Heterogeneous and Distributed Information Retrieval*, 2005.

[7] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[8] E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving Web search efficiency via a locality based static pruning method. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 235–244, 2005.

[9] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. The feasibility of peer-to-peer web indexing and search. In *Peer-to-Peer Systems II: 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 207Ű–215, 2003.

[10] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM '03)*, pages 199–206, 2003.

[11] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *Advances in Information Retrieval, 27th European Conference on IR Research (ECIR)*, pages 52–66, 2005.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *16th International Conference on Supercomputing*, June 2002.

[13] V. Plachouras, B. He, and I. Ounis. University of Glasgow at TREC2004: Experiments in Web, Robust and Terabyte tracks with Terrier. In *Proceeddings of the 13th Text REtrieval Conference (TREC 2004)*, 2004.

[14] B. Pôssas, N. Ziviani, J. Wagner Meira, and B. Ribeiro-Neto. Set-based vector model: An efficient approach for correlation-based ranking. *ACM Trans. Inf. Syst.*, 23(4):397–429, 2005.

[15] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. *MiddlewareŠ03*, 2003.

[16] G. Salton, J. Allan, and C. Buckley. Approaches to Passage Retrieval in Full Text Information Systems. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 49–58, 1993.

[17] G. Salton and C. Yang. On the specification of term values in automatic indexing. *Journal of Documentation*, 4(29):351–372, 1973.

[18] T. Suel, C. Mathur, J.-W. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. *Proc. Int'l Workshop Web and Databases (WebDB '03)*, pages 67Ű–73, 2003.

[19] The P-Grid Consortium. The P-Grid project, 2005. http://www.p-grid.org/.

A Semantic Kernel to Classify Texts with Very Few Training Examples

Roberto Basili, Marco Cammisa and Alessandro Moschitti
Department of Computer Science
University of Rome "Tor Vergata"
Rome,Italy
e-mail: {basili,cammisa,moschitti}@info.uniroma2.it

Keywords: kernel methods, similarity measures, support vector machines, WordNet

Received: May 12, 2005

Advanced techniques to access the information distributed on the Web often exploit automatic text categorization to filter out irrelevant data before activating specific searching procedures. The drawback of such approach is the need of a large number of training documents to train the target classifiers. One way to reduce such number relates to the use of more effective document similarities based on prior knowledge. Unfortunately, previous work has shown that such information (e.g. WordNet) causes the decrease of retrieval accuracy.
In this paper, we propose kernel functions to use prior knowledge in learning algorithms for document classification. Such kernels implement balanced and statistically coherent document similarities in a vector space by means of the term similarity based on the WordNet hierarchy. Cross-validation results show the benefit of the approach for Support Vector Machines when few training examples are available.

Povzetek: Predstavljena je kategorizacija besedil na osnovi malo primerov.

1 Introduction

The access to Web distributed information often requires the detection and filtering of the target objects (e.g. texts) before specialized automatic retrieval processes can be applied. In this perspective, text categorization (TC) is a useful approach to filter out irrelevant (or equivalently accept relevant) data.

As the Web is a dynamic source of information, a flexible and fast design of categorization systems is required. One of the most important aspects to achieve the above properties is to limit the time and effort needed to manually annotate large training data. Unfortunately, when few data is available the classification accuracy is rather unsatisfactory. The main reason for this outcome is the document representation based on *bag-of-words* along with the term matching document similarity. If few documents are available for training there is a high probability that terms in test documents will not be matched. Consequently, the document similarity results inadequate for an effective classification.

This problem has been tackled by enriching the document representation with term clustering (*term generalization*) or adding compound terms (*term specification*). Such approaches are based on the assumption that the similarity between two documents can be expressed as the similarity between pairs of complex terms or term clusters. The latter are built based on corpus term distributions, e.g. [4], or prior knowledge external to the target corpus (e.g. provided by WordNet [9]).

The main problem of term cluster representation is the unclear relationship with the one based on simple words. Although (semantic) clusters tend to improve the system recall, simple terms are, on a large scale, more accurate (e.g. [17]). To overcome this problem, hybrid spaces containing terms and clusters were experimented (e.g. [19]) but, again, the results showed that the mixed statistical distributions of clusters and terms impact either marginally or even negatively on the overall accuracy. Hence, the successful introduction of prior/external knowledge relies on the solution of this problem.

In this paper, we propose a model to introduce the semantic lexical knowledge encoded in the WN hierarchy in automatic text classification. The idea is to compute document similarity between two documents d_1 and d_2 by summing the term similarity contributions of all term pairs $\langle t_1, t_2 \rangle$ where $t_1 \in d_1$ and $t_2 \in d_2$. Each pair contribution is evaluated by considering the spatial and topological properties that the two compounding terms have in WN. Such approach has two advantages: (a) we obtain a well defined space which supports the similarity between terms of different surface forms based on external knowledge and (b) we avoid to explicitly define term or sense clusters which inevitably introduce noise.

The above document similarity is a valid kernel that can be used with kernel-based learning machine methods such as Support Vector Machines (SVMs) [25]. Moreover, as we believe that the external knowledge in TC is not very useful when a sufficient amount of training documents is available, we experimented our model in poor training conditions (e.g. 10 documents for each category). The improvement in the accuracy, observed on the classification

of the well known Reuters and 20 NewsGroups corpora, shows that our document similarity model is very promising for general IR tasks: unlike previous attempts (e.g. [26, 22, 17], it makes sense of the adoption of semantic external resources (i.e. WN) in IR.

Section 2 introduces the WordNet-based term similarity whereas Section 3 defines the new document similarity measure, the kernel function and its use within SVMs. Section 4 discusses the computational aspects of such kernel. Section 5 presents the comparative results between the traditional linear and the WN-based kernels within SVMs. In Section 6 comparative discussion against the related IR literature is carried out. Finally Section 6 derives the conclusions.

2 Term similarity based on general knowledge

In IR, similarity metrics in vector space models are usually driven by lexical matching. When small training material is available, few words can be effectively used and the resulting document similarity may be inaccurate. Semantic generalizations overcome data sparseness problems as contributions from different but semantically similar words can be derived.

Methods for the induction of semantic word clusters have been widely used in language modeling and lexical acquisition tasks (e.g. [7]). The linguistic resource employed in most previous work is WordNet [9] which contains three subhierarchies for nouns, verbs and adjectives. Each hierarchy represents lexicalized concepts (or senses) organized according to an "*is-a-kind-of*" relation, where a concept s is described by a set of words $syn(s)$, called $synset$, and the words $w \in syn(s)$ are synonyms according to the sense s.

For example, the words *line*, *argumentation*, *logical argument* and *line of reasoning* describe a synset which expresses the methodical process of logical reasoning (e.g. "*I can't follow your line of reasoning*"). Each word/term may be lexically related to more than one synset depending on its senses. The word *line* is also a member of the synset *line*, *dividing line*, *demarcation* and *contrast*, as a *line* denotes also a conceptual separation (e.g. "*there is a narrow line between sanity and insanity*"). The Wordnet noun hierarchy is a direct acyclic graph[1] in which the edges establish the *direct_isa* relations between two synsets.

2.1 Problems with WordNet similarities

The automatic use of WordNet for NLP and IR tasks has shown to be very complex:

First, how the topological distance among senses is related to their corresponding conceptual distance is unclear. The pervasive lexical ambiguity is also problematic as it impacts on the measure of conceptual distances between word pairs.

Second, the approximation of a set of concepts by means of their generalization in the hierarchy implies a conceptual loss that affects the target IR (or NLP) tasks. For example, *black* and *white* are *colors* but also *chess pieces* and this impacts on the similarity score that should be used in IR applications.

Finally, similar words play different roles in IR tasks and in other NLP-based systems, e.g. machine translation, so that the equivalence between them cannot be imposed in general. It is thus difficult to decide the degree of generalization (which allows us to reduce a set of senses into single features) effective for IR.

To solve the above problems, some methods attempt to map (a priori) terms to specific generalization levels, i.e. they *cut* the hierarchy at some levels (e.g. [16, 18]) and use corpus statistics to assign weights to the resulting generalizations. For several tasks (e.g. in TC) this is unsatisfactory: different contexts of the same corpus (e.g. documents) may require different levels of generalization of the same word since they have a different impact on the document similarity.

On the contrary, the *Conceptual Density* (CD) [1, 3] is a flexible semantic similarity measure which depends on the generalizations of word senses not referring to any fixed level of the hierarchy.

2.2 The Conceptual Density

CD defines a metric according to the topological structure of WN. Intuitively, given two words, their lowest common WN hypernym determines a sub-hierarchy. This latter will suggest maximum relatedness if only few levels are used to connect the two words. The CD of such words is expressed as the ratio between the size of the minimal (*ideal*) tree connecting such words and the sub-hierarchy.

To formally define CD, we introduce some basic concepts: let $\bar{g}$ be the set of nodes of the hierarchy rooted in the synset g, i.e. $\{c \in S | c \; isa \; g\}$, where S is the set of WN synsets. By definition $\forall g \in S, g \in \bar{g}$. CD makes a guess about the proximity of the senses, s_u and s_v, of two words u and v, according to the information expressed by the minimal subhierarchy, $\bar{g}$, that includes them. Let G_u be the set of generalizations for at least one sense of the word u, i.e. $G_u = \{g \in S | \exists s \in \bar{g}, u \in syn(s)\}$. The CD of u and v is:

$$CD(u,v) = \begin{cases} 0 & \textbf{iff } G_u \cap G_v = \emptyset \\ max_{g \in G_u \cap G_v} \frac{\sum_{i=0}^{h} (\mu(\bar{g}))^i}{|\bar{g}|} & \textbf{otherwise} \end{cases} \quad (1)$$

where:

- $G_u \cap G_v$ is the set of WN shared generalizations (i.e. the common hypernyms) of u and v.

[1] As only the 1% of its nodes own more than one parent in the graph, most of the techniques assume the hierarchy to be a tree, and treat the few exception heuristically.

– $\mu(\bar{g})$ is the average number of children per node (i.e. the branching factor) in the sub-hierarchy $\bar{g}$. $\mu(\bar{g})$ depends on WordNet and in some cases its value can approach 1.

– h is the depth of the *ideal*, i.e. maximally dense, *tree* with enough leaves to cover the two senses, s_u and s_v, according to an average branching factor of $\mu(\bar{g})$. This value is actually estimated by:

$$h = \begin{cases} \lfloor log_{\mu(\bar{g})} 2 \rfloor & \textbf{iff } \mu(\bar{g}) \neq 1 \\ 2 & \textbf{otherwise} \end{cases} \quad (2)$$

When $\mu(g)$=1, h ensures a tree with at least 2 nodes to cover s_u and s_v ($height = 2$).

– $|\bar{g}|$ is the number of nodes in the sub-hierarchy $\bar{g}$. This value is statically measured on WN and it is a negative bias for the higher generalization levels (i.e. larger $\bar{g}$).

CD models the semantic distance as the density of the generalizations $g \in S_u \cap S_v$. Such *density* is the ratio between the number of nodes of the *ideal tree* and $|\bar{g}|$. The ideal tree should (a) link the two senses/nodes s_u and s_v with the minimal number of edges (isa-relations) and (b) preserve the same branching factor (*bf*) observed in $\bar{g}$. In other words, this tree contains the minimal number of nodes (and isa-relations) sufficient to connect s_u and s_v according to the topological structure of $\bar{g}$. When *bf* is 1, Eq. 1 degenerates to the inverse of the number of nodes in the path between s_u and s_v, i.e. the simple proximity measure used in [21].

Figure 1 shows a subhierarchy $\bar{g}$ of two senses s_u and s_v and the associated ideal tree. Note that the bf is the average of the node branching factors (nbfs), i.e. 2. This suggests that, in this *zone*, the topological structure has a density quantifiable by a factor equal to 2. Ideally, if two senses are *very close* they should be linked by only one node, i.e. using the ideal tree on the right of the figure. Once the ideal tree is built, the $CD(s_u, s_v)$ is computed by dividing the number of its nodes by the number of nodes in the real hierarchy, e.g. 3/7 for the example.

The final $CD(u, v)$ between two words is the maximum CD among all the word sense pairs. This means that $CD(u, v)$ is determined by the *closest lexical senses*, s_u, $s_v \in \bar{g}$: the remaining senses of u and v are irrelevant, with a resulting semantic disambiguation side effect.

As the number of word pairs is in general very high, efficient approaches to compute the document similarity are needed. The next section describes how kernel methods can make practical the use of the Conceptual Density in Text Categorization.

3 A document similarity kernel based on WordNet

Term similarities are used to design document similarities which are the core functions of most TC algorithms. The

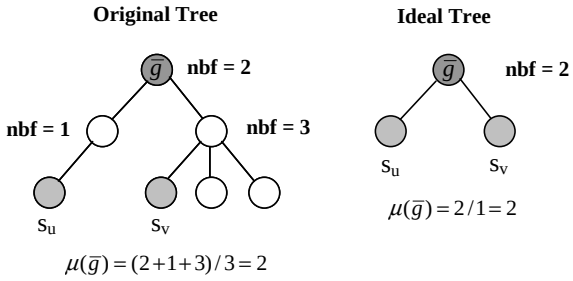


Figure 1: A subhierarchy $\bar{g}$, rooted in g, of two word senses, s_u and s_v and the corresponding ideal tree. The branching factor $\mu(\bar{g})$ is the average over the node branching factors (nbfs) of $\bar{g}$

one proposed in Eq. 1 is valid for all term pairs of a target vocabulary and has two main advantages:

1. the relatedness of each term occurring in the first document can be computed against *all* terms in the second document, i.e. all different pairs of similar (not just identical) tokens can contribute; and

2. if we use all term pair contributions in the document similarity, we obtain a measure consistent with the term probability distributions, i.e. the sum of all term contributions does not penalize or emphasize arbitrarily any subset of terms.

The positive aspects of the first point is quite clear and will solve data sparseness problems. Regarding the second point, we should consider that when document representation is enriched by means of some term clusters a simplification assumption about all the other possible clusters is made, i.e. a zero probability is assumed for them. As the literature on smoothing techniques has shown, this is not the best way to approach the problem. However, it should also be stated that the discriminative nature of Support Vector Machines makes them less sensitive to such smoothing aspects.

The next subsections present more formally the above ideas.

3.1 Document similarity Kernel

Given two documents d_1 and $d_2 \in D$ (the document set), we define their similarity as:

$$K(d_1, d_2) = \sum_{w_1 \in d_1, w_2 \in d_2} (\lambda_1 \lambda_2) \times \sigma(w_1, w_2) \quad (3)$$

where λ_1 and λ_2 are the weights of the words (features) w_1 and w_2 in the documents d_1 and d_2, respectively, and σ is a term similarity function, e.g. the conceptual density defined in Section 2.

The above document similarity could be used in kernel based machines if we prove that it is a valid kernel function, i.e. if it satisfies the Mercer's conditions [8]. Such conditions establish that the Gram matrix, $\boldsymbol{G} = K(d_i, d_j)$ $\forall i, j = 1, .., l$, where $d_1, .., d_l$ are the training documents,

must be positive semi-definite. In order to obtain such property in [21] was adopted as term similarity the matrix $\boldsymbol{M}' \cdot \boldsymbol{M}$, where $\boldsymbol{M}$ is defined by $\sigma(w_1, w_2)$ with $w_1, w_2 \in V$ and $\boldsymbol{M}'$ is its transposed. As shown in [8], $\boldsymbol{P} = \boldsymbol{M}' \cdot \boldsymbol{M}$ as well as $K(d_i, d_j) = \vec{\lambda}_i' \boldsymbol{P} \vec{\lambda}_j$ are positive semi-definite matrices. Unfortunately, this approach does not use the original similarity matrix $\boldsymbol{M}$, i.e. the term to term similarity defined in WN, since $\boldsymbol{P} = \boldsymbol{M}^2$. Although, we can see the application of such matrix as a feature expansion techniques, we loose the direct similarity semantics of two words encoded by the matrix $\boldsymbol{M}$.

With the aim to preserve an intuitive notion of document similarity, we adopt the simple similarity term matrix given by $\boldsymbol{P} = \sigma(w_1, w_2) = CD(w_1, w_2)$ without applying the square operation. This means that (a) we exactly use Eq. 3 as a kernel function and (b) we need to prove that P is positive semi-definite. To prove that $\boldsymbol{P}$ is positive semi-definite, a general way is to show that all its eigenvalues are non negative. Thus, we run the single value decomposition algorithm and verified that such condition holds.

Additionally, in [10], it is shown that when kernel functions are not positive semi-definite, SVMs still solve a data separation problem in pseudo Euclidean spaces. The drawback is that the solution may be only a local optimum. Therefore, we can experimentally observe if the empirical results are satisfactory. Our extensive experimentation (reported in Section 5) with different corpora and many training document subsets provides some evidence that Eq. 3 is a useful function as SVMs based on Eq. 3 always convergence to a significant accuracy.

The next section shows as a similarity measure can be used within Support Vector Machines.

3.2 Kernel methods and Support Vector Machines

Given a vector space in $\mathbb{R}^\eta$ and a set of positive and negative points, SVMs classify vectors according to a separating hyperplane, $H(\vec{x}) = \vec{\omega} \cdot \vec{x} + b = 0$, where $\vec{x}$ and $\vec{\omega} \in \mathbb{R}^\eta$ and $b \in \mathbb{R}$ are learned by applying the *Structural Risk Minimization principle* [25]. From the kernel theory we have that:

$$H(\vec{x}) = \Big(\sum_{h=1..l} y_h \alpha_h \vec{x_h} \Big) \cdot \vec{x} + b = \sum_{h=1..l} y_h \alpha_h \vec{x}_h \cdot \vec{x} + b =$$

$$= \sum_{h=1..l} y_h \alpha_h \phi(d_h) \cdot \phi(d) + b =$$

$$= \sum_{h=1..l} y_h \alpha_h K(d_h, d) + b. \quad (4)$$

where y_h and α_h are the class labels and the Lagrange multipliers associated with the l training documents d_h. d is the classifying document and ϕ is the mapping which projects it and d_h in the vectors $\vec{x}$ and $\vec{x}_h$, respectively. By choosing the right ϕ, the product $K(d, d_h) = \langle \phi(d) \cdot \phi(d_h) \rangle$ will correspond to the *Semantic WN-based Kernel* (SK).

Eq. 4 shows that to evaluate the separating hyperplane in $\mathbb{R}^\eta$, we do not need to evaluate the entire vector $\vec{x_h}$ or $\vec{x}$. As it is sufficient to compute $K(d, d_h)$, we can carry out the learning with Eq. 3 in $\mathbb{R}^n$, avoiding to use the explicit representation in the $\mathbb{R}^\eta$ space. The real advantage is that we can consider only the word pairs associated with non-zero weights, i.e. we can use a sparse vector computation. Additionally, to have a uniform score across different document size, the kernel function can be normalized as follows:

$$\frac{SK(d_1, d_2)}{\sqrt{SK(d_1, d_1) \cdot SK(d_2, d_2)}}$$

4 Computational Aspects

The previous section has shown that we can apply the kernel trick to train the SVMs in the dual space. In this way we avoid to compute the huge space of all WN word pairs. However, the computational complexity of the algorithm is higher than the usual approach based on the *bag-of-words* model. It depends on two main aspects:

1. The similarity measure between two documents d_1 and d_2 requires the evaluation of all the word pairs $\langle w_1, w_2 \rangle$. This leads to a complexity of $O(|d_1| \times |d_2|)$ which is remarkably higher than the usual complexity, $O(|d_1| + |d_2|)$, of traditional approaches.

2. The conceptual density evaluation requires to navigate the WN hierarchy which includes more than 10^5 nodes. On the contrary the traditional term similarity is carried out by a fast string matching function.

Since we use an implicit document representation, we need to test all document pairs during the kernel evaluation, thus, unless we apply a feature selection in the kernel space, the complexity of point 1 cannot be improved. On the contrary, we can improve the conceptual density evaluation by pre-computing it for all WN term pairs and store them in a hash table.

In the next section we described the technical approach that we adopted.

4.1 Technical approach

To evaluate the CD between two words u and v, for each sense pairs, s_u and s_v, we need to derive: (a) the minimal subhierarchy $\bar{g}$ (with its number of nodes) which includes both of them and (b) the ideal tree associated with $\bar{g}$.

Step (a) requires to evaluate the lowest hierarchy node g that dominates s_u and s_v, i.e. we need to consider all the ISA relation paths that links s_u and s_v. To optimize this step, we pre-computed all the *transitive closures* (about 2×10^5) of the ISA relation for all WN synsets along with the number of nodes dominated by g.

The ideal tree evaluation corresponds to derive its height and the branching factor, $\mu(\bar{g})$. The former is computed

by means of Eq. 2. The latter can be incrementally precomputed by navigating bottom-up the hierarchy.

The current version of WordNet package[2] makes available a set of built in libraries, written in *C* language, to navigate the hierarchy. These use an internal structure to store the WN information (e.g. glosses, relations,...). To make more efficient such data structures, we retain only (1) the relations between nouns and synsets, (2) the *"is-a-kind-of"* hierarchy and (3) we implemented special libraries to gather information efficiently.

Moreover, to speed-up the kernel computation, we designed hashed associative containers which store any word pair similarity requested during the learning or testing phase. Note that, although the term pairs are sparse, their similarity values are not. We observed that, for a set of about $32,000$ words (i.e. $1,024 \times 10^6$ pairs), the number of different values were about $6,500$, if we consider only similarity values higher than 2×10^{-5}. We exploit this property by replacing them with an integer index to access a dictionary of float numbers. This reduced the memory usage by a factor of 2.

Given the above optimized architecture, we carried out an extensive experimentation (as illustrated in the next section).

5 Experiments

The use of WordNet (WN) as a term similarity function introduces a prior knowledge whose impact on the Semantic Kernel (SK) should be experimentally assessed. The main goal is to compare the traditional Vector Space Model kernel against SK, both within the Support Vector learning algorithm.

The high complexity of the SK limits the size of the experiments that we can carry out in a feasible time. Moreover, we are not interested to large collections of training documents as in these training conditions the simple *bag-of-words* models are in general very effective, i.e. they seem to model well the document similarity needed by the learning algorithms. Thus, we carried out the experiments on small subsets of the 20NewsGroups[3] (20NG) and the *Reuters-21578*[4] corpora to simulate critical learning conditions.

5.1 Experimental set-up

For the experiments, we used the SVM-light software [12] (available at `svmlight.joachims.org`) with the default linear kernel on the token space (adopted for the baseline evaluations). For the SK evaluation we implemented Eq. 3 with $\sigma(\cdot,\cdot) = CD(\cdot,\cdot)$ (Eq. 1) inside SVM-light. As Eq. 1 is only defined for nouns, a part of speech (POS) tagger was applied. However, also verbs, adjectives and numerical features were included in the feature space. For these tokens a $CD = 0$ is assigned to pairs made by different strings. As the POS tagger could introduce errors, in a second experiment, any token with a successful look-up in the WN noun hierarchy was considered in the kernel. This approximation has the benefit to retrieve useful information even for verbs and capture the similarity between verbs and some nouns, e.g. *to drive* (via the noun *drive*) has a common synset with *parkway*.

For the evaluations, we applied a careful SVM parameterization: a preliminary investigation suggested that the trade off (between the training-set error and margin, i.e. c option in SVM-light) parameter optimizes the F_1 measure for values in the range [0.02,0.32][5]. We noted also that the cost-factor parameter (i.e. j option) is not critical, i.e. a value of 10 always optimizes the accuracy. Feature selection techniques and weighting schemes were not applied in our experiments as they cannot be accurately estimated from few training documents.

The classification performance was evaluated by means of the F_1 measure[6] for the single category and the MicroAverage F_1 for the final classifier pool [28]. Given the high computational complexity of SK, we selected 8 categories from the 20NG[7] and 8 from the Reuters corpus[8]

To derive statistically significant results from few training documents, we randomly selected 10 different samples from the 8 categories of each corpus. We trained the classifiers on one sample, parameterized on a second sample and derived the measures on the other 8. By rotating the training sample, we obtained 80 different measures for each model. The size of the samples ranged from 24 to 160 documents depending on the target experiment. The training of the SVMs adopting SK required about 10/20 minutes for each sample (depending on their size). Considering that the parameterization phase carries out the training of each classifier 16 times (one for each parameter values), we chose to use an 8 multiprocessor machine in which the classifiers can run independently. Even with this optimized strategy, we employed about 1 month to accomplish all the experiments.

5.2 Cross validation results

With the aim of showing the benefit of SK (Eq. 3) for text categorization, we compared it with the linear kernel which obtained the best F_1 measure in [12].

First, in Table 1, we report the results for 8 categories of 20NG on 40 training documents. They are expressed as the *Mean* and the *Std. Dev.* over 80 runs. Column 2, 3 and 4 show the F_1 for the linear kernel (bow), for SK

[2]Downloadable from `wordnet.princeton.edu`

[3]Available at `www.ai.mit.edu/people/jrennie/20Newsgroups/`.

[4]The Apté split available at `kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`.

[5]We used all the values from 0.02 to 0.32 with step 0.02.

[6]F_1 assigns equal importance to Precision P and Recall R, i.e. $F_1 = \frac{2P \cdot R}{P+R}$.

[7]We selected the 8 most different categories (in terms of their content) i.e. *Atheism*, *Computer Graphics*, *Misc Forsale*, *Autos*, *Sport Baseball*, *Medicine*, *Talk Religions* and *Talk Politics*.

[8]We selected the 8 largest categories, i.e. *Acquisition*, *Earn*, *Crude*, *Grain*, *Interest*, *Money-fx*, *Trade* and *Wheat*.

without applying POS information and for SK with the use of POS information (SK-POS), respectively. The last row of the table shows the MicroAverage performance for the above three models on all 8 categories. We note that SK improves bow of 3%, i.e. 34.3% vs. 31.5% and that the POS information reduces the improvement of SK, i.e. 33.5% vs. 34.3%.

Second, to verify that the above results are general, we repeated the evaluation over the 8 categories of Reuters with samples of 24 and 160 documents, respectively. Table 2 illustrates that (1) again SK improves bow (41.7% - 37.2% = 4.5%) and (2) as the number of documents increases the improvement decreases (77.9% - 75.9% = 2%).

Third, the complexity of the classification task across the samples varies remarkably thus the standard deviations assume high values. Nevertheless, the high number of samples should provide reliable results. To verify the hypothesis that SK improves bow, we evaluated the Std. Dev. of the difference, d, between the MicroAverage F_1 of SK and the MicroAverage F_1 of bow over the samples. For instance, in relation to the Table 2 experiment, we obtained that the mean and the Std. Dev. of d on the 80 test samples (of 24 documents) are 4.53 and 6.57, respectively. Using the Normal Distribution, we found that at a confidence level of 99% d is in the range [2.40,6.66], thus the probability that d is negative, i.e. bow is better than SK, is very small.

Category	*bow*	*SK*	*SK*-POS
Atheism	29.5±19.8	32.0±16.3	25.2±17.2
Comp.Graph	39.2±20.7	39.3±20.8	29.3±21.8
Misc.Forsale	61.3±17.7	51.3±18.7	49.5±20.4
Autos	26.2±22.7	26.0±20.6	33.5±26.8
Sport.Baseb.	32.7±20.1	36.9±22.5	41.8±19.2
Sci.Med	26.1±17.2	18.5±17.4	16.6±17.2
Talk.Relig.	23.5±11.6	28.4±19.0	27.6±17.0
Talk.Polit.	28.3±17.5	30.7±15.5	30.3±14.3
MicroAvg. F_1	31.5±4.8	34.3±5.8	33.5±6.4

Table 1: Performance of the linear and Semantic Kernel with 40 training documents over 8 categories of 20NewsGroups collection.

Category	24 docs		160 docs	
	bow	*SK*	*bow*	*SK*
Acq.	55.3±18.1	50.8±18.1	86.7±4.6	84.2±4.3
Crude	3.4±5.6	3.5±5.7	64.0±20.6	62.0±16.7
Earn	64.0±10.0	64.7±10.3	91.3±5.5	90.4±5.1
Grain	45.0±33.4	44.4±29.6	69.9±16.3	73.7±14.8
Interest	23.9±29.9	24.9±28.6	67.2±12.9	59.8±12.6
Money-fx	36.1±34.3	39.2±29.5	69.1±11.9	67.4±13.3
Trade	9.8±21.2	10.3±17.9	57.1±23.8	60.1±15.4
Wheat	8.6±19.7	13.3±26.3	23.9±24.8	31.2±23.0
Mic.Avg.	37.2±5.9	41.7±6.0	75.9±11.0	77.9±5.7

Table 2: Performance of the linear and Semantic Kernel with 24 and 160 training documents over 8 categories of the Reuters corpus.

Next, the above findings confirm that SK outperforms the *bag-of-words* kernel in critical learning conditions as the semantic contribution of the SK recovers useful information. To confirm this hypothesis we carried out experiments with samples of different size, i.e. 3, 5, 10, 15 and 20 documents for each category. Figures 2 and 3 show the learning curves for 20NG and Reuters corpora. Each point refers to the average on 80 samples.

As expected the improvement provided by SK decreases when more training data is available. However, the SK model without POS information on 160 training documents still outperforms the baseline of about 2-3%. This suggests that the matching between noun-verb pairs still provides semantic information which is useful for topic detection. In particular, during the similarity estimation, each word shows a non-null similarity with 60.05 words on average. This is useful to increase the amount of information available to the SVMs. To confirm such hypothesis, we removed the string matching contributions from SK such that only words having different surface forms participate to the evaluation of Eq. 3. The interesting result is that SK still converged to a MicroAverage F_1 measure of 56.4% (compare with Table 2). This shows that SVMs can discern between the correct and incorrect categories by using only the WN similarity.

Finally, to provide a comparison with literature models, we experimented with SK by training on 10 random samples of 40 documents and testing on the Reuters test set, i.e. on the 2,502 documents labeled with the 8 target categories. The SK obtained a MicroAverage F_1 (averaged on the 10 runs) of 67.4% which is higher than 65,0% of the baseline outcome. In a second experiment, we used for the *Acquisition* category all the available training data from the 8 categories (i.e. 6,367 documents) obtaining a F_1 of 94.5% for the SK vs. a F_1 of 96.0% of the baseline. This shows that when the number of training documents is large, the word distributions assume a statistical significance that is more reliable than the distribution of the term pairs weighted by WN. Indeed, these latter introduce necessarily some errors due to disambiguation mistakes or incorrect (for the specific target domain) term similarities.

In summary, WN allows the learning algorithm to carry out document similarity when few or no terms can be matched. When precise terms are available with a reliable statistical distribution, string matching is more precise since it is less affected by errors.

6 Related Work

Several IR studies focus on the term similarity models to embed prior knowledge in document similarity.

In [15] a *Latent Semantic Indexing* analysis was used for term clustering. Such approach assumes that values x_{ij} in the transformed term-term matrix represents the similarity (values ≤ 0) and anti-similarity (values < 0) between terms i and j. This enables both positive and negative clusters of terms. Evaluation of query expansion techniques

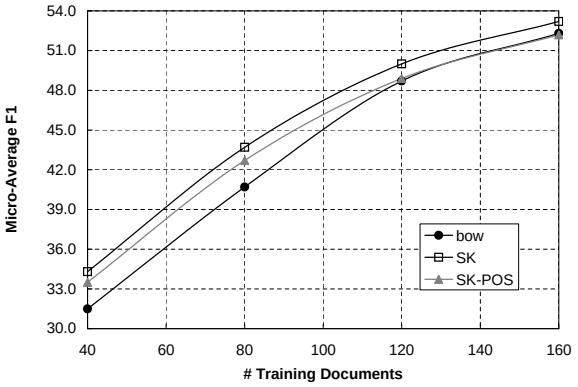


Figure 2: MicroAverage F_1 of SVMs using bow, SK and SK-POS kernels over the 8 categories of 20NewsGroups.

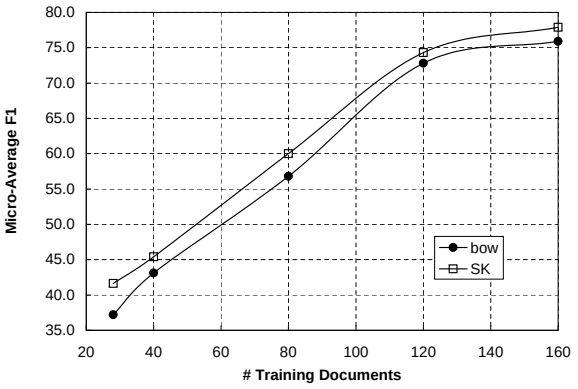


Figure 3: MicroAverage F_1 of SVMs using bow and SK over the 8 categories of the Reuters corpus.

showed that positive clusters can improve Recall of about 18% for the *CISI* collection, 2.9% for *MED* and 3.4% for *CRAN*. Furthermore, the negative clusters, when used to prune the result set, improved the precision.

In [4], a feature selection technique that clusters similar features/words, called the Information Bottleneck (IB), is applied to TC. Support Vector Machines trained over such clusters were experimented with three different corpora: *Reuters-21578*, WebKB and 20NewsGroups. Controversial results were obtained as the cluster based representation outperformed the simple *bag-of-words* only on the latter collection (>3%). This was explained as a consequence of the corpus "complexity". Reuters and WebKB corpora seem to require few features to reach optimal performance. IB can thus be adopted either to reduce the problem complexity as well as to increase accuracy by using a simpler representation space.

In [6], Latent Semantic Analysis (LSA) was applied to derive domain-specific concepts and to create semantic document representations over these concepts. Such representations (based on both terms and concepts) were used to design weak classifiers. Concepts were derived from different LSA models (i.e. LSA spaces of different dimensions). AdaBoost was applied to efficiently combine weak hypotheses and integrate term and concept based information. The experiments on two standard document collections show that conceptual features in addition to terms lead to consistent and quite substantial accuracy gains. In our own opinion such evidence is restricted to the used experimental set-up, i.e. classification models, parameterization, data pre-processing and so on. Indeed, the highest F_1, i.e. 85.82%, reached on the Reuters corpus using the extended representation is lower than the one achieved by means of the *bag-of-words* in other work, e.g. 87.8% using ADABOOST.MH (see the table in [20] about comparative TC results on Reuters corpus). Consequently, we cannot derive that LSA representations are better than the simple *bag-of-words* (when a sufficient amount of training data is used).

The use of *external* semantic knowledge for document retrieval has even been more problematic. In [22], a study on the impact of semantic ambiguity was carried out. A WN-based semantic similarity function between noun pairs was applied to improve indexing and document-query matching. However, the WSD algorithm had a performance ranging between 60-70%, and this made the overall semantic similarity not effective.

Other studies on semantic information for improving IR were carried out in [24] and [26, 27]. Word semantic information was used for text indexing and query expansion, respectively. In [27], it was shown that semantic information derived directly from WN with automatic WSD produces poor results. Nevertheless, recently, a revised approach to the use of word senses for document indexing was proposed in [23]. Only senses which are automatically determined with a high probability are utilized. This enabled the experimented retrieval system to improve the accuracy over the simple *bag-of-words*.

In TC word senses have a similar impact if not lower: when enough training data is available, the positive and negative examples of a category allow the learning algorithm to build implicit word clusters based on corpus statistics. These provide matching capabilities more accurate than the matching between concepts of different surface forms defined in external resources, e.g. WN term similarity. Moreover, different categories are better characterized by different words rather than different senses [17].

In [19], WN senses were used to replace words without any word sense disambiguation. The result was a small improvement on a poorly accurate *state-of-art* TC algorithm on a small corpus. When a more statistical reliable set of documents was used, the adopted representation produced a performance decrease. The scale and assessment provided in [17] (3 corpora using cross-validation techniques) showed that even an accurate disambiguation of WN senses (about 80% accuracy on nouns) did not improve TC.

In [14], an extensive experimentation with several algorithms has been carried out to compare the accuracy of classifiers based on words and senses. The target document collection was a subset of the Brown Corpus annotated with semantic concordance. The results indicate that the use of

senses does not produce any significant categorization improvement.

In [5], AdaBoost classifiers are trained with document represented by concepts extracted from WN. Experiments with Reuters and Ohsumed show an improvement on the *bag-of-words* representation. Again this results cannot be generalized as the absolute value of the highest achieved F_1, i.e. 85.89%, on the Reuters corpus, is lower than the best literature results, i.e. 87.8%. Nevertheless, it is worth to note that other relevant improvements were obtained on the Ohsumed corpus for which the results of the best TC models are not available. Thus, on corpora different from Reuters we do not know if the conceptual representation improves the *bag-of-words*. According to the analysis carried out in [4], we may assume that the Reuters corpus is not representative in general and the *bag-of-words* approach is superior only on some corpora.

In [21] an approach similar to the one presented in this article was proposed. A term proximity function was used to design a kernel able to semantically smooth the similarity between two document terms. Such semantic kernel was designed as a combination of the Radial Basis Function (RBF) kernel with the term proximity matrix. Entries in this matrix are inversely proportional to the length of the WN hierarchy path linking the two terms. The performance, measured over the 20NewsGroups corpus, showed an improvement of 2% over the *bag-of-words*. The main differences with our approach are:

First, the term proximity is not fully sensitive to the information of the WN hierarchy. For example, if we consider pairs of equidistant terms, the nearer to the WN top level a pair is the lower similarity it should receive, e.g. *sky* and *location* (hyponyms of *entity*) should not accumulate similarity like *knife* and *gun* (hyponyms of *weapon*). Measures, like CD, that deal with this problem have been widely proposed in literature (e.g. [18]) and should be always applied.

Second, the kernel-based CD similarity is an elegant combination of lexicalized and semantic information. In [21] the combination of weighting schemes, the RBF kernel and the proximity matrix has a less clear interpretation.

Finally, the experiments were carried out by using only 200 features (selected via Mutual Information statistics). In this way the contribution of rare or non statistically significant terms is neglected. In our view, such features may give, instead, a relevant contribution once we move in the SK space generated by the WN similarities.

Other important work on semantic kernel for retrieval has been developed in [8, 13]. Two methods for inferring semantic similarity from a corpus were proposed:

In the first a system of equations were derived from the dual relation between word-similarity based on document-similarity and vice versa. The equilibrium point was used to derive the semantic similarity measure.

The second method models semantic relations by means of a diffusion process on a graph defined by lexicon and co-occurrence information. The major difference with our approach is the use of a different source of prior knowledge, i.e. WN. Similar techniques were also applied in [11] to derive a Fisher kernel based on a latent class decomposition of the term-document matrix.

In summary, a careful analysis of literature work shows that prior knowledge (derived directly from the corpus or extracted by external resources) is not able to improve the best TC model learned with an adequate number of training data. On the contrary, the experiments shown in this paper suggest the following reasonable hypothesis: when the statistical word distributions derivable from training data are not reliable, we can use external resources to provide an effective semantic smoothing. In other words, two documents containing different terms have zero match probability in the *bag-of-words* model. Using term similarity we associate them with a probability different from zero designing a more accurate model. Of course, this approximation is less accurate than the probability distributions derived from a statistically representative sample of documents.

7 Conclusions

The way to use semantic prior knowledge in IR has always been an interesting subject as confirmed by the examined literature work.

In this paper, we applied the conceptual density function on the WordNet (WN) hierarchy to define a document similarity metric. Accordingly, we defined a semantic kernel (SK) to train Support Vector Machine classifiers. Cross-validation experiments over 8 categories of 20NewsGroups and Reuters corpora over multiple samples have shown that:

- in poor training data conditions, the WN prior knowledge can be effectively used to improve the TC accuracy (up to 4.5 absolute percent points, i.e. 10%);
- the CD is an effective way to capture the topological WN properties; and
- the higher is the number of training documents, the lower is the improvement produced by SK. This suggests that the general prior knowledge embedded in WN is useful to increase accuracy until the category statistical information (e.g. its word probability distributions) is not *completely* reliable.

These promising results enable a number of future researches: (1) larger scale experiments with different measures and semantic similarity models (e.g. [18]); (2) improvement of the overall efficiency by exploring feature selection methods over the SK; and (3) the extension of the semantic similarity by a general (i.e. non binary) application of the conceptual density model as proposed in [2] for semantic tagging.

References

[1] E. Agirre and G. Rigau. Word sense disambiguation using conceptual density. In *Proceedings of COLING'96, pages 16–22, Copenhagen, Danmark.*, 1996.

[2] R. Basili and M. Cammisa. Unsupervised semantic disambiguation. In *In Proceedings of LREC Workshop on "Beyond Named Entity Recognition - Semantic Labelling for Natural Language Processing Tasks"*, Lisbon, Portugal, 2004.

[3] R. Basili, M. Cammisa, and F. M. Zanzotto. A similarity measure for unsupervised semantic disambiguation. In *In Proceedings of Language Resources and Evaluation Conference*, 2004.

[4] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. On feature distributional clustering for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 146–153, New Orleans, Louisiana, United States, 2001. ACM Press.

[5] S. Bloehdorn and A. Hotho. Text classification by boosting weak learners based on terms and concepts. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK*, pages 331–334. IEEE Computer Society, NOV 2004.

[6] L. Cai and T. Hofmann. Text categorization by boosting automatically extracted concepts. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 182–189, New York, NY, USA, 2003. ACM Press.

[7] S. Clark and D. Weir. Class-based probability estimation using a semantic hierarchy. *Comput. Linguist.*, 28(2):187–206, 2002.

[8] N. Cristianini, J. Shawe-Taylor, and H. Lodhi. Latent semantic kernels. *J. Intell. Inf. Syst.*, 18(2-3):127–152, 2002.

[9] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press., 1998.

[10] B. Haasdonk. Feature space interpretation of SVMs with indefinite kernels. *IEEE Trans Pattern Anal Mach Intell*, 27(4):482–92, Apr 2005.

[11] T. Hofmann. Learning probabilistic models of the web. In *Research and Development in Information Retrieval*, pages 369–371, 2000.

[12] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, 1999.

[13] J. Kandola, J. Shawe-Taylor, and N. Cristianini. Learning semantic similarity. In *in Neural Information Processing Systems (NIPS 15) - MIT Press.*, 2002.

[14] A. Kehagias, V. Petridis, V. G. Kaburlasos, and P. Fragkou. A comparison of word- and sense-based text categorization using several classification algorithms. *J. Intell. Inf. Syst.*, 21(3):227–247, 2003.

[15] A. Kontostathis and W. Pottenger. Improving retrieval performance with positive and negative equivalence classes of terms, 2002.

[16] H. Li and N. Abe. Generalizing case frames using a thesaurus and the mdl principle. *Computational Linguistics*, 23(3), 1998.

[17] A. Moschitti and R. Basili. Complex linguistic features for text classification: a comprehensive study. In S. McDonald and J. Tait, editors, *Proceedings of ECIR-04, 26th European Conference on Information Retrieval*, Sunderland, UK, 2004. Springer Verlag.

[18] P. Resnik. Selectional preference and sense disambiguation. In *Proceedings of ACL Siglex Workshop on Tagging Text with Lexical Semantics, Why, What and How?, Washington, April 4-5, 1997.*, 1997.

[19] S. Scott and S. Matwin. Feature engineering for text classification. In I. Bratko and S. Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 379–388, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[20] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[21] G. Siolas and F. d'Alché Buc. Support vector machines based on a semantic kernel for text categorization. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 5*, page 5205. IEEE Computer Society, 2000.

[22] A. F. Smeaton. Using NLP or NLP resources for information retrieval tasks. In T. Strzalkowski, editor, *Natural language information retrieval*, pages 99–111. Kluwer Academic Publishers, Dordrecht, NL, 1999.

[23] C. Stokoe, M. P. Oakes, and J. Tait. Word sense disambiguation in information retrieval revisited. In *Proceedings of SIGIR03, Canada*, 2003.

[24] M. Sussna. Word sense disambiguation for free-text indexing using a massive semantic network. In A. P. New York, editor, *The Second International Conference on Information and Knowledge Management (CKIM 93)*, pages 67–74, 1993.

[25] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[26] E. M. Voorhees. Using wordnet to disambiguate word senses for text retrieval. In R. Korfhage, E. M. Rasmussen, and P. Willett, editors, *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 171–180. ACM, 1993.

[27] E. M. Voorhees. Query expansion using lexical-semantic relations. In W. B. Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, pages 61–69. ACM/Springer, 1994.

[28] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval Journal*, 1999.

Captain Nemo: A Metasearch Engine with Personalized Hierarchical Search Space

Stefanos Souldatos, Theodore Dalamagas and Timos Sellis
School of Electrical and Computer Engineering,
National Technical University of Athens, 157 73, Athens, GR
E-mail: {stef, dalamag, timos}@dblab.ntua.gr
http://www.dblab.ntua.gr/∼ {stef, dalamag, timos}

Keywords: web search, personalization, metasearch engine, classification, hierarchy

Received: November 18, 2005

Personalization of search has gained a lot of publicity the last years. Personalization features in search and metasearch engines are a follow-up to the research done. On the other hand, text categorization methods have been successfully applied to document collections. Specifically, text categorization methods can support the task of classifying Web content in thematic hierarchies. Combining these two research fields, we have developed Captain Nemo, *a fully-functional metasearch engine with personalized hierarchical search spaces.* Captain Nemo, *retrieves and presents search results according to personalized retrieval models and presentation styles. Here, we present the hierarchical Web page classification approach newly adopted.* Captain Nemo *lets users define a hierarchy of topics of interest. Search results are automatically classified into the hierarchy, exploiting hierarchical k-Nearest Neighbor classification techniques. The user study conducted demonstrates the effectiveness of our metasearch engine.*

Povzetek: Opisan je metaiskalnik Captain Nemo.

1 Introduction

Searching for Web content can be extremely hard. Web content can be found in a variety of information sources. The number of these sources keeps increasing, while at the same time sources continually enrich their content. Not only should users identify these sources, but they should also determine those containing the most relevant information to satisfy their information need.

Search and metasearch engines are tools that help the user identify such relevant information. Search engines retrieve Web pages that contain information relevant to a specific subject described with a set of keywords given by the user. Metasearch engines work at a higher level. They retrieve Web pages relevant to a set of keywords, exploiting other already existing search engines.

Personalization on the Web is an issue that has gained a lot of interest lately. Web sites have already started providing services such as preferences for the interface, the layout and the functionality of the applications. Personalization services have also been introduced in Web search and metasearch engines. However, those services deal mostly with the presentation style and ignore issues like the retrieval model, the ranking algorithm and topic preferences.

On the other hand, text classification methods, including k-Nearest Neighbor (k-NN) [30, 18], Support Vector Machines (SVM) [15, 8], Naive Bayes (NB) [20, 2], Neural Networks [21], decision trees and regression models, have been successfully applied to document collections (see [31] for a full examination of text classification methods).

Such methods can support the task of classifying Web content in thematic hierarchies. Organizing Web content in thematic categories can be useful in Web search, since it helps users easily identify relevant information while navigating in their personal search space.

There are two main approaches for classifying documents in thematic hierarchies:

- *Flat Model* (Flatten the hierarchy): Every topic of the hierarchy corresponds to a separate category having its own training data. A classifier, based on text categorization techniques determines the right category for a new incoming Web document.

- *Hierarchical Model* (Exploit the hierarchy): A hierarchy of classifiers is built such that every classifier decides each time to classify a document in the appropriate category among the categories of the same level in the hierarchy, following a path from the root down to the leaves of the hierarchy tree. For example, an incoming document might be added to *Arts* category (between *Arts*, *Science* and *Sports*), then to *Dance* category inside *Arts* (between *Poetry*, *Photography* and *Painting*), then to *Spanish_Dances* inside *Arts/Dance*. The assignment scores for all these decisions can determine the final category for the incoming document.

Combining these two research fields, namely personalization of search and Web content hierarchical classification, we have created *Captain Nemo*, a fully-functional

metasearch engine with personalized hierarchical search spaces. *Captain Nemo*, initially presented in [26], retrieves and presents search results according to personalized retrieval models and presentation styles. In this paper, we present the hierarchical Web page classification approach, recently adopted in *Captain Nemo*. Users define a hierarchy of topics of interest. Search results are automatically classified into the hierarchy, exploiting Nearest-Neighbour classification techniques.

Our classification approach is a hybrid one. Every topic of the hierarchy is considered to be a separate category having its own training data, as in the flat model. However, the training data set of a topic is enriched by data from its subtopics. As a result, the decision of whether a Web page belongs to a category strongly depends on its descendants.

A typical application scenario for *Captain Nemo* starts with a set of keywords given by the user. *Captain Nemo* exploits several popular Web search engines to retrieve Web pages relevant to those keywords. The resulting pages are presented according to the user-defined presentation style and retrieval model. We note that users can maintain more than one different *sets of preferences*, which result to different presentation styles and retrieval models. For every retrieved Web page, *Captain Nemo* recommends the most relevant topic of user's personal interest. Users can optionally save the retrieved pages to certain folders that correspond to topics of interest for future use.

Contribution. The main contributions of our work are:
(a) We expand personalization techniques for metasearch engines, initially presented in [26].
(b) We suggest semi-automatic hierarchical classification techniques in order to recommend relevant topics of interest to classify the retrieved Web pages. The thematic hierarchy is user-defined.
(c) We present a fully-functional metasearch engine, called *Captain Nemo*[1], that implements the above framework.
(d) We carry out a user study to evaluate the hierarchical classification process and its effect on searching. The experiments demonstrate the effectiveness of our approach.

Related Work. The need for Web information personalization has been discussed in [25] and [24]. Following this, several Web search and metasearch engines[2] offer primitive personalization services.

Concerning the topics of interest, topic-based search will be necessary for the next generation of information retrieval tools [4]. Inquirus2 [11] uses a classifier to recognize Web pages of a specific category. Northern Light[3] has an approach called *custom folders* that organizes search results into categories. However, these categories are created dynamically by the search results and do not reflect the users' personal interest. A similar approach is presented in [6], but the thematic hierarchy is the same for all users.

Recently, many researchers have looked into the problem of classifying Web content into thematic hierarchies, using either the flat or the hierarchical model. The former approach has shown poor results, since flat classifiers cannot cope with large amounts of information including many classes and content descriptors. In [17], an n-gram classifier was used to classify Web pages in Yahoo categories. Probabilistic methods to automatically categorize Web documents are presented in [12, 10], while statistical models for hypertext categorization are presented in [5]. The hierarchical approach has been explored initially in [16]. Experiments with bayesian classification models showed the superiority of the hierarchical model over the flat. Experiments on two-level classification using SVMs were conducted in [7], while a kernel-based algorithm for hierarchical text classification was presented in [23]. Finally, [28] exploits the structure of the hierarchy, by grouping the topics into meta-topics.

Outline. The rest of this paper is organized as follows. The personalization features of *Captain Nemo* are discussed in Section 2. The architecture of *Captain Nemo* and several implementation issues are discussed in Section 3. A user study is presented in Section 4. Section 6 concludes the paper.

2 Personal Search Spaces

Personal search spaces are maintained for users of *Captain Nemo*. Each *personal search space* includes user preferences able to support the available personalization features. In fact, more than one *sets of preferences* can be maintained for each user, which result to different retrieval models and presentation styles. A *personal search space* is implemented through three respective personalization filters.

We next discuss the available personalization features regarding the retrieval model, the presentation style and the topics of interest. The hierarchical Web page classification approach is presented in the following section.

2.1 Personal Retrieval Model

As seen before, most of the existing metasearch engines employ a standard retrieval model. In *Captain Nemo*, this restriction is eliminated and users can create their *personal retrieval model*, by setting certain parameters in the system. Default values of the parameters are preset for users that do not want to spend time on this. These parameters are described below:

Participating Search Engines. Users can declare the search engines they trust, so that only these search engines are used by the metasearch engine.

[1] http://www.dblab.ntua.gr/∼ stef/nemo/
[2] Google, Alltheweb, Yahoo, AltaVista, WebCrawler, MetaCrawler, Dogpile, etc.
[3] http://www.northernlight.com/index.html

Search Engine Weights. In a metasearch engine, retrieved Web pages may be ranked according to their ranking in every individual search engine that is exploited. In *Captain Nemo*, as shown in Section 3.1, the search engines can participate in the ranking algorithm with different weights. These weights are set by the user. A lower weight for a search engine indicates low reliability and importance for that particular engine. The results retrieved by this search engine will appear lower in the output of *Captain Nemo*.

Number of Results. A recent research [14] has shown that the majority of search engine users (81.7%) rarely read beyond the third page of search results. Users can define the number of retrieved Web pages per search engine.

Search Engine Timeout. Delays in the retrieval task of a search engine can dramatically deteriorate the response time of any metasearch engine that exploits the particular search engine. In *Captain Nemo*, users can set a timeout option, i.e. time to wait for Web pages to be retrieved for each search engine. Results from delaying search engines are ignored.

2.2 Personal Presentation Style

Captain Nemo results are presented through a customizable interface, called *personal presentation style*. Again, default values of the parameters are preset for users that do not want to spend time on this. The following options exist:

Grouping. In a typical metasearch engine, results returned by search engines are merged, ranked and presented in a list. Beside this typical presentation style, *Captain Nemo* can group the retrieved Web pages (a) by search engine or (b) by topic of interest. The latter is based on a hierarchical classification technique, described in Section 3.2. An example of search results grouped by topic of interest is shown in Figure 1.

Content. The results retrieved by *Captain Nemo* include three parts, title, description and URL. Users can declare which of these parts should be displayed. The available options are (a) title, description and URL, (b) title and URL and (c) title.

Look and Feel. Users can customize the general look and feel of *Captain Nemo*. Selecting among the available color themes and page layouts, they can define preferable ways of presenting results. There are six color themes and three page layouts.

2.3 Topics of Personal Interest

Captain Nemo users can define *topics of personal interest*, i.e. thematic categories where search results can be kept for future reference. The retrieved Web pages can be saved in folders that correspond to these topics. These folders have a role similar to *Favorites* or *Bookmarks* in Web browsers.

Figure 1: Results grouped by topic of interest.

For every retrieved Web page, *Captain Nemo* recommends the most relevant *topic of personal interest*. Users can optionally save the retrieved pages to the recommended or other folder for future use.

The *topics of personal interest* are organized in a hierarchy. The hierarchy can be thought of as a tree structure having a *root* and a set of *nodes* which refer to topics of the thematic hierarchy. For every topic node, there is (a) a label that describes its concept and (b) a stricter description of the concept (a set of keywords).

Figure 2 shows such a hierarchy of *topics of personal interest*. The hierarchical classification technique is discussed in more detail in Section 3.2.

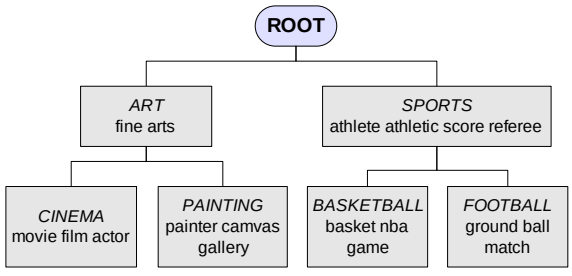


Figure 2: Hierarchy of topics of personal interest.

3 System Implementation

This section presents the architecture of our application and discusses various interesting implementation issues. Figure

3 illustrates the main modules of *Captain Nemo*.

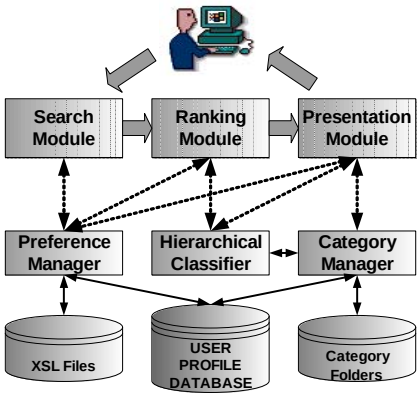


Figure 3: System architecture.

Search Module. It implements the main functionality of the metasearch engine, providing connection to the search engines selected by the user. It retrieves the relevant Web pages according to the retrieval model defined by the user. The results are sent to the ranking module for further processing. The module is implemented in Perl, using the search engine wrappers WWW::Search[4], parameterized by user preferences.

Ranking Module. The retrieved Web pages are ranked and grouped according to the personal retrieval model of the user. For every retrieved Web page, a corresponding topic of personal interest is determined. The ranking process, implemented in Perl, is discussed in further detail in Section 3.1.

Presentation Module. It presents the search results provided by the ranking module. It is implemented in Perl CGI generating XML output. The latter is passed through the appropriate XSL filter representing the look and feel settings of the specific user.

Preference Manager. It provides the connection between the three aforementioned modules (i.e. search module, ranking module, presentation module) and the information stored in user profiles. It is also responsible for updating user profiles and the corresponding XSL files. It is implemented in Perl on top of the PostgreSQL database system[5].

Hierarchical Classifier. It implements the hierarchical classification of results to the thematic hierarchy of the user, as described in Section 3.2. It is implemented in Perl.

[4]http://search.cpan.org/dist/WWW-Search/lib /WWW/Search.pm
[5]http://www.postgresql.org/

Category Manager. It manages the topics of interests and keeps the appropriate folders on disk in accordance with the user profiles. It provides all the necessary information to the hierarchical classifier. It cooperates with the presentation module, when grouping by topics of interest is selected by the user. Thematic hierarchies are represented by XML indexes, which are parsed by Perl.

The next sections discuss in detail the ranking and classification mechanisms used in our application.

3.1 Ranking

Given a query, a typical metasearch engine sends it to several search engines, ranks the retrieved Web pages and merges them in a single list. After the merge, the most relevant retrieved pages should be on top. There are two approaches used to implement such a ranking task. The first one assumes that the initial scores assigned to the retrieved pages by each one of the search engines are known. The other one does not presupposes any information about these scores.

In [22], it is pointed out that the scale used in the similarity measure in several search engines may be different. Therefore, normalization is required to achieve a common measure of comparison. Moreover, the reliability of each search engine must be incorporated in the ranking algorithm through a weight factor. This factor is calculated separately during each search. Search engines that return more Web pages should receive higher weight. This is due of the perception that the number of relevant Web pages retrieved is proportional to the total number of Web pages retrieved as relevant for all search engines exploited by the metasearch engine.

On the other hand, [9, 13, 27] stress that the scores of various search engines are not compatible and comparable even when normalized. For example, [27] notes that the same document receives different scores in various search engines and [9] concludes that the score depends on the document collection used by a search engine. In addition, [13] points out that the comparison is not feasible not even among engines using the same ranking algorithm and claims that search engines should provide statistical elements together with the results.

In [1], ranking algorithms are proposed which completely ignore the scores assigned by the search engines to the retrieved Web pages: bayes-fuse uses probabilistic theory to calculate the probability of a result to be relevant to the query, while borda-fuse is based on democratic voting. The latter considers that each search engine gives votes in the results it returns, giving N votes in the first result, $N-1$ in the second, etc. The metasearch engine gathers the votes for the retrieved Web pages from all search engines and the ranking is determined democratically by summing up the votes.

Weighted Borda-Fuse. The algorithm adopted by *Captain Nemo* is the weighted alternative of Borda-fuse. In this

algorithm, search engines are not treated equally, but their votes are considered with weights depending on the reliability of each search engine. These weights are set by the users in their profiles. Thus, the votes that the i result of the j search engine receives are:

$$V(r_{i,j}) = w_j * (max_k(r_k) - i + 1) \quad (1)$$

where w_j is the weight of the j search engine and r_k is the number of results rendered by search engine k. Retrieved pages that appear in more than one search engines receive the sum of their votes.

Example. A user has defined the personal retrieval model of Table 1.

Search Engine	Results	Weight	Timeout
SE1	20	7	6
SE2	30	10	8
SE3	10	5	4

Table 1: Personal retrieval model.

The user runs a query and gets 4, 3 and 5 results respectively from the three search engines specified. According to Weighted Borda-Fuse, the search engines have given votes to the results. The first result of each search engine receives 5 votes, as the largest number of results returned is 5. Table 2 shows the votes received by the search engines.

Search Engine	1st	2nd	3rd	4th	5th
SE1	5	4	3	2	-
SE2	5	4	3	-	-
SE3	5	4	3	2	1

Table 2: Result votes by search engines.

Captain Nemo multiplies these votes by the weight of each search engine to push upward results of search engines trusted most by user. The final votes of each result of each search engine is shown in Table 3.

Search Engine	1st	2nd	3rd	4th	5th
SE1	35	28	21	14	-
SE2	*50*	40	30	-	-
SE3	25	20	15	10	5

Table 3: Result votes by *Captain Nemo*.

So, the first result to appear in the rank is the first result of search engine SE2.

3.2 Hierarchical Classification of Retrieved Web Pages

As we have already mentioned, *Captain Nemo* recommends relevant topics of interest to classify the retrieved pages, exploiting k-Nearest Neighbor classification techniques. Other classification algorithms can be easily adopted as well. However, our efforts were focused on providing the appropriate framework and not on testing various classification algorithms, which has been widely addressed by many researchers (see Related Work in Section 1). Thus, we selected the simple yet effective [31] k-Nearest Neighbor classification technique.

Retrieved Web pages are processed by k-NN and classified in the thematic hierarchy. The part of a Web page that is used for classification includes its title and the part of its content extracted by search engines. The latter is usually strongly relevant to the imposed query. The whole content of Web pages could be used for higher accuracy, but this would deteriorate the response time [7].

k-NN Classification. The k-NN classification method presumes that a group of categories is defined for a data set and a set of training documents corresponds to each category. Given an incoming document, the method ranks all training documents according to the similarity value between those documents and the incoming document. Then, the method uses the categories of the k top-ranked documents to decide the right category for the incoming document by adding the per-neighbour similarity values for each one of those categories [30, 31]:

$$y(\mathbf{x}, c_j) = \sum_{\mathbf{d}_i \epsilon kNN} sim(\mathbf{x}, \mathbf{d}_i) \times y(\mathbf{d}_i, c_j) \quad (2)$$

where:

1. $\mathbf{x}$ is an incoming document, $\mathbf{d}_i$ is a training document, c_j is a category,
2. $y(\mathbf{d}_i, c_j) = 1$ if $\mathbf{d}_i$ belongs to c_j or 0 otherwise,
3. $sim(\mathbf{x}, \mathbf{d}_i)$ is the similarity value between the incoming document $\mathbf{x}$ and the training document $\mathbf{d}_i$,

Using thresholds on these scores, k-NN obtains binary category assignments and allows the system to assign a document to more than one categories. Instead it can just use the category with the highest score as the right one for the incoming document. *Captain Nemo* follows the second approach.

Hierarchical k-NN Classification. Hierarchical k-NN classification algorithms are usually implemented in a top-down approach. The document under consideration is first classified to one of the first-level categories. Recursively, the classification continues in the subtree rooted to the category selected in the previous step. The process stops when the selected category is either a leaf or more similar to the document than its subcategories. In this approach, all categories in the hierarchy should be defined in detail to attract documents that belong to one of their subcategories. To avoid this difficulty, in *Captain Nemo*, where the descriptions of the categories are given by users, a hybrid approach is employed.

Our Hybrid Approach. Our classification approach is a hybrid one. The topics of interest are organized in a thematic hierarchy. Every topic of the hierarchy is considered to be a separate category having its own training data (its keyword description), as in the flat model. However, the training data set of a topic is enriched by data from its subtopics. For example, the categories of the hierarchy of Figure 2 are enriched as shown in Figure 4. As a result, the decision of whether a Web page belongs to a category strongly depends on its descendants.

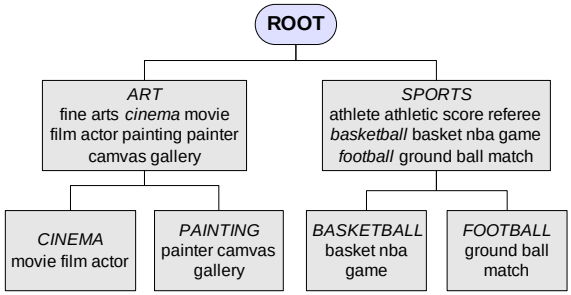


Figure 4: Enriched hierarchy.

In *Captain Nemo*, the topic descriptions set by the user are used instead of training documents in k-NN. To be more specific, *Captain Nemo* needs to calculate similarity measures between the description of each retrieved Web page and the description of every *topic of personal interest*. The similarity measure employed is a $tf - idf$ one [29]. Let D be the description of a topic of interest and R the description of a retrieved Web page. The similarity between the topic of interest and the retrieved Web page, $sim(R, D)$, is defined as follows:

$$Sim(R,D) = \frac{\sum_{t \epsilon R \cap D} w_{R,t} \times w_{D,t}}{\sqrt{\sum_{t \epsilon R \cap D} w_{R,t}^2} \times \sqrt{\sum_{t \epsilon R \cap D} w_{D,t}^2}} \quad (3)$$

where t is a term, $w_{R,t}$ and $w_{D,t}$ are the weights of term t in R and D respectively. These weights are:

$$w_{R,t} = \log\left(1 + \frac{C}{C_t}\right) \quad (4)$$

$$w_{D,t} = 1 + \log f_{D,t} \quad (5)$$

where C is the total number of topics of interest, C_t is the number of topics of interest including term t in their description and $f_{D,t}$ is the frequency of occurence of t in description D.

Having a new, retrieved Web page, we rank the topics of interest according to their similarity with the page (the topic of interest with the highest similarity will be on the top). Then, the top-ranked topic of interest is selected as the most appropriate for the retrieved page.

Example. We have created a user with the hierarchy of *topics of personal interest* presented in Figure 2. For this user, we have run the query "michael jordan", asking for just a few results. A screenshot of the results grouped by topic of interest is shown in Figure 1. Totally, there are:

- 0 results in category *1. ART*,
- 3 results in category *1.1. CINEMA*,
- 2 results in category *1.2. PAINTING*,
- 3 results in category *2. SPORTS*,
- 8 results in category *2.1. BASKETBALL*,
- 0 results in category *2.2. FOOTBALL*.

As expected, the majority of results are matched to topic *BASKETBALL*. However, there are results matching to other topics as well. Results matching to topic *CINEMA* deal with Michael Jordan as an actor. Results in topic *PAINTING* refer to photo galleries with photos of Michael Jordan. Finally, results matching to topic *SPORTS* refer to the athletic background of Michael Jordan in general.

4 User Study

A user study was conducted to evaluate the hierarchical classification process and its effect on searching. Twelve persons of various backgrounds participated in the experiments. We divided users into two teams, *users* and *testers*.

Experiment 1. The first experiment evaluated the performance of the hierarchical classification process. The six *users* were assigned the task to create a hierarchy of four to six topics of personal interest in *Captain Nemo*. However, the users were advised to restrict in one domain, so that we can test classification among similar categories, e.g. apples vs oranges. Testing among totally different categories, for example oranges vs shoes, would be easy; hence it was avoided. The user hierarchies are shown in Table 4.

After the six hierarchies (category names and descriptions) were defined in the system, we asked the users to recognize categories of the Dmoz directory[6] that correspond to their thematic categories. Then, we fed the Dmoz pages into the Hierarchical Classifier and counted the percentage of pages that were classified correctly in the appropriate category. These percentages are shown in Table 4. For instance, 75% of the pages found under the Dmoz category corresponding to category *audio* were classified to category *audio* as well. The average percentage of pages correctly classified for each user is noted next to the user label.

On average, 73% of pages found in the Dmoz hierarchy are classified in the correct category by the Hierarchical Classifier of *Captain Nemo*. The reader should keep in mind that the categories created by users in this experiment

[6] http://www.dmoz.org/

User 1 (avg: 64%)	**User 2** (avg: 67%)	**User 3** (avg: 73%)
electronics (55%)	databases (67%)	jazz musicians (75%)
⊢audio (75%)	⊢data mining (94%)	⊢bassists (92%)
⊢photography (43%)	⊢warehousing (62%)	⊢trumpeters (50%)
⊢digital camera (83%)	⊢olap (44%)	⊢trombonists (76%)
User 4 (avg: 80%)	**User 5** (avg: 81%)	**User 6** (avg: 70%)
cooking (61%)	furniture (49%)	music festivals (82%)
⊢potatoes (67%)	⊢leather (93%)	⊢folk (59%)
⊢onions (100%)	⊢bamboo (100%)	⊢electronic (56%)
⊢pizza (93%)	⊢bedroom (81%)	⊢dance (83%)

Table 4: User-defined thematic hierarchies and percentage of correctly classified query results.

were forced to belong in the same domain. In real cases, users define categories of various domains, making categories more distinguishable and classification percentages even higher.

Experiment 2. The second experiment was conducted to evaluate the effect of presenting the results classified in user-defined categories. We measured the time users need to identify Web pages relevant to their information need (in the spirit of [6]). We conducted the experiments with *users* that created the hierarchy themselves, and *testers* who were not previously aware of the user-defined categories.

Each *user* was given the results of a query in the domain of the self-defined hierarchy and was asked to identify Web pages for a query more detailed than the given one (called *target query*), as in [6]. For example, *user3* was given the results of query 'brian' and was asked to identify a Web page regarding the famous bassist Brian Bromberg. The time the user spent using the *classified-by-category* interface and the *classified-in-a-list* interface is shown in Table 5. Next, each *tester* was asked to do exactly the same using the user-defined hierarchy of their corresponding *user* (see Table 5). For *users* that have defined their own thematic hierarchy, searching was more than 60% faster than searching of *testers* that have not defined the categories themselves.

5 Conclusion

Getting this idea from two research fields, namely personalization of search and Web content classification, we have created *Captain Nemo*, a fully-functional metasearch engine with personalized hierarchical search spaces. *Captain Nemo*, initially presented in [26], retrieves and presents search results according to personalized retrieval models and presentation styles. In this paper, we presented the hierarchical Web page classification approach, recently adopted in *Captain Nemo*. Users define a hierarchy of topics of interest. Search results are automatically classified into the hierarchy, exploiting k-Nearest Neighbor classification techniques.

For future work, we are going to improve the hierarchical classification process, exploiting background knowledge in the form of ontologies [3]. Next, we will incorporate a Word Sense Disambiguation (WSD) technique in the spirit of [19].

References

[1] J. A. Aslam and M. Montague. Models for metasearch. In *Proceedings of the 24th ACM SIGIR Conference*, pages 276–284. ACM Press, 2001.

[2] L. D. Baker and A. K. McCallum. Distributional clustering of words for text classification. In *Proceedings of the 21st ACM SIGIR Conference*, pages 96–103. ACM Press, 1998.

[3] S. Bloehdorn and A. Hotho. Text classification by boosting weak learners based on terms and concepts. In *Proceedings of the 4th ICDM Conference*, pages 331–334, 2004.

[4] W. L. Buntine, J. Löfström, J. Perkiö, S. Perttu, V. Poroshin, T. Silander, H. Tirri, A. J. Tuominen, and V. H. Tuulos. A scalable topic-based open source search engine. In *Proceedings of the ACM WI Conference*, pages 228–234. ACM Press, 2004.

[5] S. Chakrabarti, B. E. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of the 17th ACM SIGMOD Conference*, pages 307–318. ACM Press, 1998.

[6] Hao Chen and Susan Dumais. Bringing order to the web: automatically categorizing search results. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–152, New York, NY, USA, 2000. ACM Press.

[7] S. Dumais and H. Chen. Hierarchical classification of web content. In *Proceedings of the 23rd ACM SIGIR Conference*, pages 256–263. Athens, Greece, ACM Press, 2000.

USERS		QUERY		BY CATEGORY		IN A LIST	
User	**Tester**	**Given**	**Target**	**User**	**Tester**	**User**	**Tester**
User 1	Tester 1	car	car audio system	7 sec	22 sec	92 sec	104 sec
User 2	Tester 2	select	SQL tutorial	8 sec	19 sec	45 sec	42 sec
User 3	Tester 3	brian	bassist Brian Bromberg	4 sec	14 sec	31 sec	28 sec
User 4	Tester 4	italian	Italian pizza recipe	5 sec	35 sec	85 sec	69 sec
User 5	Tester 5	outdoor	outdoor furniture	3 sec	17 sec	29 sec	36 sec
User 6	Tester 6	rainbow	Rainbow band	4 sec	12 sec	31 sec	28 sec

Table 5: Time to identify relevant Web pages for given queries.

[8] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and represenations for text categorization. In *Proceedings of the 7th ACM CIKM Conference*, pages 148–155. ACM Press, 1998.

[9] S. T. Dumais. Latent semantic indexing (lsi) and trec-2. In *Proceedings of the 2nd TREC Conference*, 1994.

[10] N. Fuhr and C.-P. Klas. A new effective approach for categorizing web documents. In *Proceedings of the 22nd IRSG Conference*, 1998.

[11] E. Glover, G. Flake, S. Lawrence, W. P. Birmingham, A. Kruger, C. Lee Giles, and D. Pennock. Improving category specific web search by learning query modifications. In *Proceedings of the SAINT Symposium*, pages 23–31. IEEE Computer Society, January 8–12 2001.

[12] N. Govert, M. Lalmas, and N. Fuhr. A probabilistic description-oriented approach for categorizing web documents. In *Proceedings of the 8th ACM CIKM Conference*, pages 475–482. ACM Press, 1999.

[13] L. Gravano and Y. Papakonstantinou. Mediating and metasearching on the internet. *IEEE Data Engineering Bulletin*, 21(2), 1998.

[14] iProspect. iProspect search engine user attitudes. http://www. iprospect.com /premiumPDFs/iProspectSurveyComplete.pdf, 2004.

[15] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th ECML Conference*, 1998.

[16] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the 14th ICML Conference*, 1997.

[17] Y. Labrou and T. Finin. Yahoo! as an ontology - using Yahoo! categories to describe documents. In *Proceedings of the 7th ACM CIKM Conference*, pages 180–187. ACM Press, 1998.

[18] B. Masand, G. Linoff, and D. Waltz. Classifying news stories using memory based reasoning. In *Proceedings of the 15th ACM SIGIR Conference*, pages 59–65. ACM Press, 1992.

[19] D. Mavroeidis, G. Tsatsaronis, M. Vazirgiannis, M. Theobald, and G. Weikum. Word sense disambiguation for exploiting hierarchical thesauri in text classification. In *Proceedings of the 16th ECML/9th PKDD Conference*, pages 181–192, 2005.

[20] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of the Learning for Text Categorization Workshop, at the 15th AAAI Conference*, 1998.

[21] H.-T. Ng, W.-B. Goh, and K.-L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the 20th ACM SIGIR Conference*, pages 67–73. ACM Press, 1997.

[22] Y. Rasolofo, F. Abbaci, and J. Savoy. Approaches to collection selection and results merging for distributed information retrieval. In *Proceedings of the 10th ACM CIMK Conference*. ACM Press, 2001.

[23] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Learning hierarchical multi-category text classification models. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 744–751, New York, NY, USA, 2005. ACM Press.

[24] M. Sahami, V. O. Mittal, S. Baluja, and H. A. Rowley. The happy searcher: Challenges in web information retrieval. In *Proceedings of the 8th PRICAI Conference*, pages 3–12, 2004.

[25] C. Shahabi and Y-S. Chen. Web information personalization: Challenges and approaches. In *Proceedings of the 3rd DNIS Workshop*, 2003.

[26] S. Souldatos, T. Dalamagas, and T. Sellis. Sailing the web with captain nemo: a personalized metasearch engine. In *Proceedings of the Learning in Web Search Workshop, at the 22nd ICML Conference*, 2005.

[27] G. Towell, E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies for information retrieval. In *Proceedings of the 12th ICML Conference*, 1995.

[28] Andreas S. Weigend, Erik D. Wiener, and Jan O. Pedersen. Exploiting hierarchy in text categorization. *Inf. Retr.*, 1(3):193–216, 1999.

[29] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, 2nd edition, 1999.

[30] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of the 17th ACM SIGIR Conference*, pages 13–22. ACM Press, 1994.

[31] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd ACM SIGIR Conference*, pages 42–49. ACM Press, 1999.

Plan Sharing: Showcasing Coordinated UAV Formation Flight

Henry Hexmoor, Swetha Eluru and Hadi Sabaa
Department of Computer Engineering and Computer Science
University of Arkansas
Fayetteville, AR 72701, USA.
E-mail: {Hexmoor, seluru, hsabaa}@uark.edu

Keywords: agents, collaboration, UAV, benevolence, help

Received: December 7, 2005

Agent teaming and autonomy are foundational themes in multi-agent systems. Agents may work as singletons or they may work in environments where other agents exist. In multi-agent systems, agents may form teams by sharing common goals with other agents. Cooperation is essential for any collaborative, group activity. Beyond coordination and judicious role assignment, cooperation enables members of a team to be aware and account for collection of their goals as well as the performance of agents on individual goals. This paper presents a general model of cooperation and illustrates how it may enhance group performance. In this paper, we present results of an application of the concept of cooperation in a simulated swarm of reconnaissance urban UAVs that are tracking vehicles in an urban environment.

Povzetek: Opisan je splošni model sodelovanja agentov.

1 Introduction

An agent is defined as an autonomous, problem-solving computational entity capable of effective operation in dynamic and open environments (Luck and Griffiths, 2003). A multi-agent system is a system of agents that exhibit social rationality, normative patterns, and values, among themselves within an environment (Hexmoor, 2003).

Typically, each agent in a multi-agent system possesses incomplete information for solving a problem with limited global knowledge. Therefore, agents interact with one another to gather information, act upon that information, and hence collectively solve a problem. *Collaborative*, behavior-coordinated activity involves participants to work jointly with each other to satisfy a shared goal that often yields more than the sum of individual actions (Grosz and Sidner, 1990). The mentioned type of activity may be distinguished from both interaction and simple coordination in terms of the commitments agents make to each other (Grosz and Kraus, 1996).

A theory of collaboration must therefore account for not only intentions, abilities, and knowledge about actions of individual agents, but also for their coordination in group planning and group acting. Furthermore, it must account for the manner in which plans may be incrementally formed and executed by the participants.

Agents may have different beliefs concerning the methods for performing an action or those for achieving a desired state. Pollack argued for a view of plans as purely data-structures (Pollack, 1990) i.e., a plan is more appropriately viewed as a set of partially ordered actions that, when performed under appropriate conditions, lead to a specified new state of the world. She has argued for a view of plans as mental states that are necessary for plan interference. Having a plan does not merely require the know-how to perform a behavior, but it also includes possessing the intention to perform the actions entailed.

To adequately model cooperation, it is necessary to accommodate differences among beliefs of individual participants as well as to distinguish between *knowledge about* action performance and the *intention to act*. Agents may differ not only in their beliefs about the strategies to perform an action and the state of the world, but also in their assessments of the ability and willingness of an individual to perform an action.

The shared plan formalization provides mental state specifications of both shared plans and individual plans. Shared plans are constructed by groups of cooperative agents and include subsidiary shared plans formed by subgroups as well as subsidiary individual plans formed by individual participants in the group (Lochbaum, 1994). The formalization distinguishes between complete plans in which all the requisite beliefs and intentions have been established and partial plans. In addition to the propositional attitude of intending to perform an action, it introduces the attitude of intending that a proposition be held.

Agents can enhance their fitness by mutual help rather than by competition, as is observed in nature (Benton, 2001). This assumes that resources adequate for both agents exist, or are created by interacting and sharing their information. This enhances both, the process of working together toward a common goal as

well as the process of sharing effort, expertise and resources to achieve mutually desirable outcome.

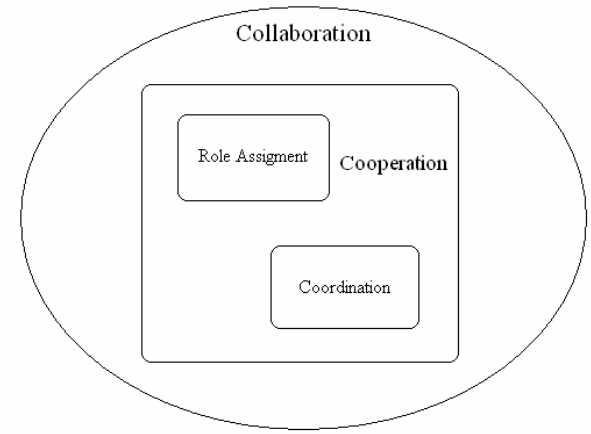


Figure 1. Collaboration subsumes Cooperation and Cooperation subsumes both Coordination and Role assignment. [Inspired by (Tuomela, 2000)]

Figure 1 illustrates that the coordination process and role assignment are subsets of the process of cooperation. The later, in turn, is a part of the collaboration process.

Agents need to organize themselves in a manner that permits them to perform their tasks efficiently. A malformed organization will affect the entire multi-agent system. When multi-agent systems change state, the agents in the system should be able to organize themselves accordingly by sharing information amongst them. When this is not accomplished, cooperation should be adapted in order to avoid disruption in the multi-agent system.

We have applied the just mentioned concept to a simulated swarm of reconnaissance Unmanned Arial Vehicles (UAVs) that are tracking vehicles in an urban environment with details discussed in Section 3.

This paper offers an approach to adapt cooperation in multi-agent systems. The main focus will be particularly on the cooperation among agents who are working together for a particular task while using the plan sharing techniques to enhance the cooperation. Rudimentary metrics are developed to gauge the effect of collaboration on system performance.

The novelties of this paper are in the following areas:

- Developing superior strategies for a given set of agents to work together
- Devising a process by which the agents are integrated into a team, regulated to achieve team goals
- Increasing agents' performance to contribute to a high functioning system.
- Evaluating agents during the process of cooperation.
- Quantifying the effect of cooperation on the goals and the system performance.

In the remainder of this paper, Section 2 contains related work in the area of cooperation between the agents and also the application of the cooperation in multi UAV interactions. In this section, issues related to cooperation, necessity of cooperation, and plan sharing among the agents are discussed. Application of cooperation and collaboration in our implemented UAV swarm is presented in section 3. Section 4 outlines a novel, generalized cooperation algorithm. We describe the importance of cooperation by illustrating how agent plan sharing enhances the cooperation process. Section 5 briefly describes incorporation of collaboration in our UAV swarm. We then present experiments and results in section 6. Section 7 provides concluding remarks and suggestions for future work.

2 Related Work

Since there is a growing demand for robust and intelligent multi-agent systems, a vast body of work is available in the area of group activity (Hexmoor, 2003).

According to Alonso, two agents may depend on one another in one of sixteen ways (Alonso, et. al., 1999). Sharing a goal achievement is central to forming agent teams (Cohen, et.al., 1997). Agents that want to maximize their gain may consider cooperation in order to lower their workload and temporal penalties (Beer et. al, 1999). Hexmoor extended the explorations for teaming (Hexmoor and Duchscherer, 2001).

An understanding of collaboration is essential to modeling the intentional context of discourse and its structure (Grosz and Sidner, 1990; Lochbaum, et.al., 1990 and Lochbaum, 1995, 1998). As a theoretical framework for modeling collaboration (Grosz and Kraus, 1996), it is evident that collaborative activities require a complex set of parameters that must be taken into account. The primary focus while attempting to achieve a goal is on understanding the states of mind of the individuals who participate in collaboration, and on properties of the group. An overview of the model designed by Grosz provides a setting in which to examine the roles of parameters of agents on collaborative plans and activities (Grosz, 1999).

In this process, the *mutual* beliefs of the discourse participants, the amount of knowledge that each individual participant or all participants should have, which was discussed in (Clark and Richard, 1981; Cohen, 1981) as well as differences in beliefs among participants in a discourse are important (Pollack, 1990).

Partial, individual plans are expanded to more complete plans through means-ends reasoning about intended goals. Cooperation mirrors this reasoning process, i.e., plan-collaboration process. However, their expansion requires communication and negotiation as well as means-ends reasoning about the way in which to perform the group action (Grosz and Kraus, 1999).

By and large, communication and collaboration are disjoint; yet interdependent activities. Communication is inherently a collaborative activity (Grosz and Sidner, 1990; Korta, 1995 and Arrazola, 1996). An agent communicates to achieve a purpose. The motivations underlying communication provide structure for an agent's discourse (Mataric, 1993). Collaboration, in turn, requires communication. Both communication and

collaboration can be used parametrically in agent design. As a result, theories and models of collaboration are essential to understand and model intentional states and the intentional attributes of discourse (Grosz and Sidner, 1990; Lochbaum, Grosz and Sidner, 1990 and Lochbaum, 1995, 1998), which use the Shared Plans formalization of collaboration, as the basis of a computational model for recognizing the intentional structure of disclosure.

Building directly on Lochbaum's use of Shared plans, others have constructed a collaborative graphical interface for a travel planning system. These applications use the logical specification provided by shared plans to constrain utterance generation and interpretation, e.g., (Rich and Sidner, 1994) and (Sidner and Rich, 1997).

Furthermore, in a collaborative activity, collaboration commonly occurs within the process of planning. Each agent may have incomplete or incorrect beliefs. Furthermore, their beliefs about each other's beliefs and capabilities to act may be incorrect. As a result, a collaborative act cannot be modeled simply by aggregating plans of individual agents.

Therefore, rather than modeling plan recognition, what must be modeled is the augmentation of beliefs about the actions of multiple agents and their intentions. Thus, Grosz and Sidner modified and expanded the Shared plan model of collaborative behavior originally proposed in (Grosz and Sidner, 1990), to present an algorithm for updating an agent's beliefs about a partial, shared plan, and describe an initial implementation of this algorithm in the domain of network management for augmenting an evolving jointly-held plan.

For a multi-agent collaborative control, Chandler and Pachter conclude that decision making through planning and management are the essence of the autonomous control problem (Chandler and Patcher, 1998).

To improve teamwork, we need to better understand the nature of coordination and its ramifications. This is explained with in-depth analysis of the coordination that is required to carry on a conversation.

For investigation of cooperative control of multiple UAVs, a simulator is offered (Chandler and Rasmussen, 2001). This is implemented in a hierarchal manner where inter-vehicle communication is explicitly modeled. During the construction of a UAV swarm, issues concerning memory usage and functional encapsulation were also considered. This simulation has been instrumental in evaluating cooperative control strategies for UAVs.

Control Automation and Task Allocation (CATA), which is a multiple-vehicle/multi-agent simulation was developed at the Boeing Corporation. This simulation has been used in several early cooperative control studies, such as in (Chandler and Rasmussen, 2001). Since CATA was relatively large and written in C++, it proved to be difficult for widespread use.

A number of other UAV simulations exist. Their payload weight carrying capability, their accommodations (volume, environment), their mission profile (altitude, range, duration) and their command, control and data acquisition capabilities vary significantly; for a brief survey, See (Lua, et. al., 2003).

Recent military operations have showcased the abilities of UAVs where they provide intelligence, surveillance, reconnaissance, command, and control information to commanders in real-time or near real-time format. The success of UAVs has raised questions about future roles for UAVs in the military operation. These roles include arming UAVs and using UAVs for target designation; these missions are commonly grouped under the title of Unmanned Combat Aerial Vehicles (UCAVs), see (Raymond, 2000).

The ability to control many remote entities with minimal user intervention has many military and commercial applications. Current techniques for controlling UAVs, which rely on centralized control and on the availability of global information, are not suited to the control of UAV swarms owing to the complexity that arises from the interactions between swarm elements (Stover and Gibson, 1997).

Traditional, centralized approaches, frequently lead to exponential increases in communication bandwidth requirements and in the size of the controlling swarm. In contrast, swarms of simple biological or artificial organisms exhibit rich, emergent behaviors, without the need for centralized control or global communication (Boanabeau, et.al., 1999). Controlling UAV swarms via human supervision is of great interest to the US military.

For coordination, as explained earlier, allocation of tasks to the UAV during their flight is one of the criteria for achieving the joint activity. Work for optimizing the task allocation problem for a fleet of Unmanned Aerial Vehicles with tightly coupled tasks and rigid relative timing constraints is available in (Alighanbari, et.al., 2003). The overall objective is to minimize the mission completion time for the fleet, and the task assignment must account for differing UAV capabilities and no-fly zones.

For many vehicles, obstacles, and targets, fleet coordination is a complicated optimization problem and the computation time increases rapidly with the problem size (Pachter, et.al, 2002 and Richards, et.al., 2001).

Work on particle swarms (Parker, 1993), cultural algorithms (Reynolds and Chung, 1996), and bacterial chemo taxis algorithms (Muller, et. al., 2002) has generalized the idea for abstract, n-dimensional cognitive spaces that make up self-organizing particle systems.

Interactions between particles result in complex global behavior which emerges from the joint actions and relatively simple behaviors of the individual particles, thereby exhibiting self-organization. These properties have been used in applications in computer graphics (Reynolds, 1987, 1999), multi-robot team control (Balch and Arkin, 1998; Fredslund and Matartic, 2002; Winder and Reggia, 2004; Vail and Veloso, 2003), and numerical optimization (Parker, 1993). We have implemented an Urban UAV test bed, described in the following section.

3 An Urban UAV surveillance system

UAVs in our simulation are modeled as powered aerial vehicles sustained in flight by aerodynamic lift and guided without an onboard crew. In general, a UAV may be expendable or recoverable, and can fly autonomously or be piloted remotely. When working together as a group, UAVs resemble a multi-agent system. UAVs interact with other UAVs and perform their tasks collectively.

Figure 2. Snapshot of our UAV simulation screen

Figure 2 depicts a typical flight pattern of a swarm of UAVs tracking terrestrial vehicles. The "white" circles depict cloud cover. Vehicles may temporarily disappear from UAV view when they are traveling below the randomly appearing clouds. This makes tracking them more challenging. UAVs need to interact to improve their collective tracking capability. Vehicles as well as cloud patches appear randomly in our simulation for a measure of realism. The system maintains the track quality, i.e., the number of cycles tracked, for each target, which is described in more detail as the Performance list and denoted as Perf [] in Section 4. Upon entry of a UAV in the theater, it determines a number of targets, i.e., vehicles, to track largely based on proximity. This is called a UAV Preference list denoted as Pref [], also described in Section 4.

Elsewhere, (Hexmoor, et. al., 2005) we have described how a human supervisor may guide and alter interactions among UAVs to improve system tracking. This is achieved primarily via parameters that affect a UAV social personality. These consist of four parameters.

- *Dedication* parameter determines how committed the UAVs are to reacquire lost targets.
- *Sociability* parameter determines how gregarious UAVs will be. A UAV with a positive sociability will tend to operate in proximity to other UAVs. Conversely, a UAV with negative sociability (i.e., anti-social) will make the UAV shun others and operate independently.
- *Conformity* parameter determines how quickly the UAV reacts to operator suggestions.
- Finally, *Disposition* parameter determines how quickly a UAV will become frustrated with the negotiation process.

In contrast, herein we focus on a single parameter of *Cooperation level*, denoted as CL, further described in Section 4. This parameter is used to adjust the level of collaboration among UAVs. We have explored setting CL at four levels. The results of a set of experiments are presented in Section 6.

4 A Proposed Cooperation Model

Although we used the same collaboration model, in this section we describe our collaboration model in abstract terms and we will not refer to UAVs or target tracking. Our model chiefly concentrates on how agents team up to form a collaborative pattern to achieve their goals.

Upon entry, an agent determines its own intentions for a plan to achieve its goal. Each agent has its own individual plan for achieving its goals. The common goal refers to the collective set of agent goals to be achieved.

Each agent has knowledge of its environment in the form of beliefs. An agent will desire to perform its individual tasks by assessing its knowledge of the environment. After environmental assessment and determination of a course of action (i.e., a plan), it will form intentions to achieve its selected goals.

Next, we outline our model in general terms. Let there be n agents in a given environment. Assume m goals are to be achieved at any instant in time.

After updating beliefs, an agent compiles a list of goals to be achieved. By default, an agent will wish to follow the order of goals in its list. Each agent forms a course of action.

We consider all agents to be identical in every respect. Furthermore, agents are assumed to possess identical capabilities and limitations. To recapitulate, in an environment with a team with m goals, each agent has its own courses of actions gleaned from its own personal observations.

Each agent may independently pursue its individual intentions. In some circumstances, agents might be successful in reaching their goals independently. However, this lack of interdependence might adversely affect the achievement of common goals.

Next, we outline a plan sharing process. Each agent shares its intentions, desires, and also the course of action in which it wants to proceed. In other words, each agent has global knowledge about other agents' intentions and desires, and also about all the goals which have been already achieved. This includes updating the changed intentions of all agents. If an agent changes its desire and thus changes its intentions, this is shared with other agents in the environment.

Cooperation is uniformly introduced in the process of achieving goals. Agents are considered to have the same cooperation level. The system level performance is quantified as the overall cumulative performance of all

the agents working in the system; that is, the number of goals processed in the duration of time. The rate of goal performance is broadcasted to other agents in the system, hence making it available for them to access.

During the cooperation process, each agent in each cycle considers its preferred intentions, the performance of each goal in the environment, and the predetermined cooperation level in the system. Tracking information (i.e., goal performance) is maintained locally on targets. This phenomenon provides a way for communication between UAV's where each UAV can access the tracking information on the targets. Cooperation level in our model is a system parameter for cooperativeness, shared by all agents. In each cycle, when agent decision making process considers the cooperation level in the environment, it generates a new *bid order* list which will be considered as the new intention list for each agent. enters

This new intention list is generated starting from the preference list of the particular agent and reflects changes to this preference list to favor the goals with low performance levels biased with the cooperation level.

For example, if a particular agent has a preference list of goals, say 3, 6, and 7. Assume the relative cumulative performance levels of these goals are 9, 21, and 11 respectively. Then without cooperation, agents may revise their goal list by comparing their preference list with the goal performance list. The revised list will be 6, 7 and 3. That means, here, the goal with less performance is given the least priority.

When cooperation is introduced in this example and with some global value for the cooperation level, the new, revised intention list will not only depend on the performance of the goals but also on the cooperation level among agents. We assume cooperation level to encapsulate an implicit notion of benevolence where agents tend to help one another achieve low performing goals. With cooperation, the revised example goal list will be 3, 7, 6. That means, here, the goal with less performance is given the highest priority.

Assume an agent has a goal g to be achieved and it has been trying it for a long period of time. Meanwhile, the agent concentrates only on the present goal, and by the time it plans to achieve the goal, which is the next one in its intention goal list, that goal might be unavailable for the agent due to unforeseen reasons. Here, benevolent agents might come forward to achieve this particular goal. Agents come forward even if goal *g* is less appealing due to its lack of performance.

Cooperative agents will act out their benevolence by striving to achieve poorly performing goals in order to exhibit cooperation. With the largest cooperation level values, agents will consider achieving the poorest performing goals. The summary of our model is thus as follows.

Consider an agent A in a given environment. It possesses its own beliefs, desires and intentions for achieving goals. Each agent constructs its own preference list of goals it wants to follow along with its individual plan. Let us denote the former as Pref[A]. Each goal in the list of goals of the environment is continually assessed and ascribed a performance level. This performance level is given depending on the number of times the goal has been attempted by agents in the environment. To summarize, each goal G has its instantaneous performance value codified with Perf [G].

In each cycle, each agent A considers its Pref [A], Perf [G], and Coop level. As shown in the Figure 3, a new *bid order list* for each agent is generated using performance and preference lists as input. If the cooperation level value is large the agent will prefer the poorest performing goals. The reordering of the agent preference list reflects the degree of the cooperation level.

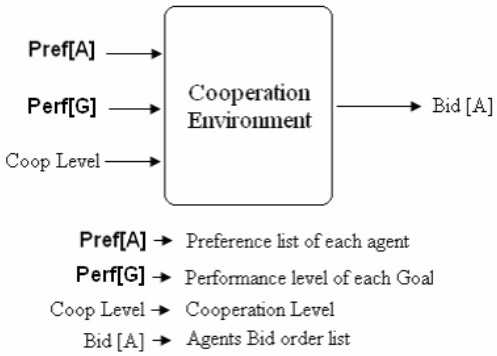


Figure 3. Parametric Cooperation

Figure 4 presents a pseudocode for our cooperation algorithm, which demonstrates how an agent cooperates. An agent A's bid list, B[ ], is initialized to its preference list. CL is the cooperation level parameter set to a constant. The reorder list is set to the size of the preference list Pref [A]. During each iteration, *reorder list* is computed using the given equation. The new reorder list is sorted and is assigned as the agent's new bid list. Intuitively, lack of performance is amplified by the cooperation level constant CL. Capability of agent [A] is the capability to perform the goal *i*.

```
1. B [ ] : = Pref[A] // initialize bid list to preference list
2. CL = constant  // cooperativeness level
3. reorder list = [ | Pref [A] | ]  // the same size as Pref [A]
4.  for ( i = 0 ; i < [reorder] ; i ++ ) {
          reorder of [ i ] = 1 - Perf [B [ i ] ] * CL * Capability of [A, i]
                                        // capability is toward the goal
5. sort  reorder [ i ]
6. B [ i ] = reorder [ i ]
```

Figure 4. Cooperation Algorithm

Suppose an agent's original preference list is [1, 2, 3] and the respective performance list of goals are [0.2, 0.7, 0.9]. Assume the cooperation level CL to be set at 0.75. Reorder list is computed:

For 1, reorder = 1 – 0.2 * 0.75 = 1 – 0.15 = 0.75
2, reorder = 1 – 0.7 * 0.75 = 1 – 0.525 = 0.475
3, reorder = 1 – 0.9 * 0.75 = 1 – 0.675 = 0.325

After sorting, we have [0.325, 0.475, 0.75], thus the new bid list is [3, 2, 1]. In this example we assume all agents are equally capable toward all goals, i.e., Capability[A, i] = 1.0.

5 Implementation and Experiments

Our cooperation algorithm (Figure 4) was applied to the UAV swarm system described in section 3. Here UAVs are agents and the goals are targets. An environment with different paradigms and parameters is used to achieve the simulation and experiments. The Java programming language is used to code the algorithms and the Borland JBuilder was used as our IDE.

Initializing all the UAVs with similar capabilities created the multi-agent system. All targets are initialized with similar qualities as well. Each UAV has its own preference list of targets and **a** performance list of all the targets in the corresponding preference list as mentioned in the algorithms. The initial positions of all UAVs are randomly generated. The preference list for each UAV is generated by the number of targets it has been able to sense in the environment. All the targets are mobile and keep moving.

The system is initially simulated without plan sharing and without cooperation. Then plan sharing process is introduced and the system is simulated at different levels of cooperation i.e., the cooperation level (CL) shown in the algorithm is assigned different values and then the system performance is captured by the simulation.

6 Results and Discussions

6.1 UAVs with no Plan-Sharing and no Cooperation

We used 200 UAVs for tracking, initially placed randomly, but then they move and values change. We first examined the system performance behavior when agents were not sharing plans nor cooperating with each other as shown in Figures 5 and 6. UAVs proceed with tasks in their preference list as they enter the system. As there is no cooperation or plan-sharing among the UAVs, the targets untracked during the given time cycle are not substantially decreased. As shown in the Figure 5, the targets untracked reach a steady state of about 194, given 225 as the total number of targets in the system.

Figure 6 shows the cumulative number of traces achieved, which is climaxed at about 332.

These results motivated us to introduce plan-Sharing and Cooperation as discussed in Section 5.

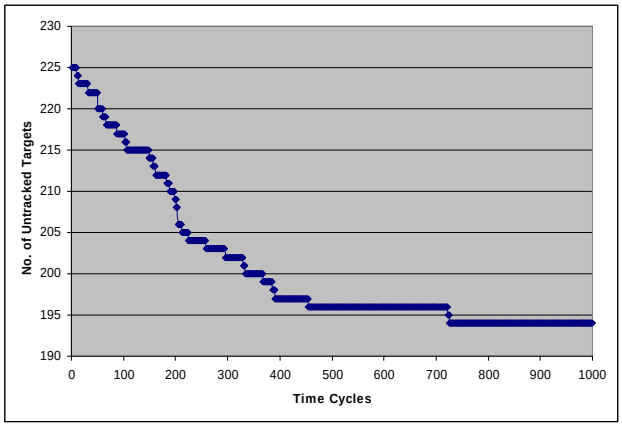


Figure 5. Untracked targets over time with no Plan Sharing and no cooperation

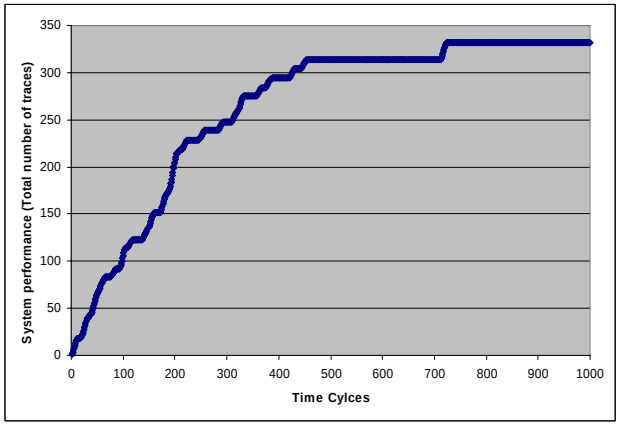


Figure 6. Total number of traces over time with no Plan-Sharing and no cooperation

6.2 UAVs with Plan-Sharing but no cooperation

Next, we first introduced Plan-Sharing among UAVs where they share their preference list with every other UAV present in the system. Although there is no explicit cooperation, plan sharing helps UAVs account for a larger number of targets. An implicit style of cooperation takes place by targets that are tracked independently by multiple UAVs.

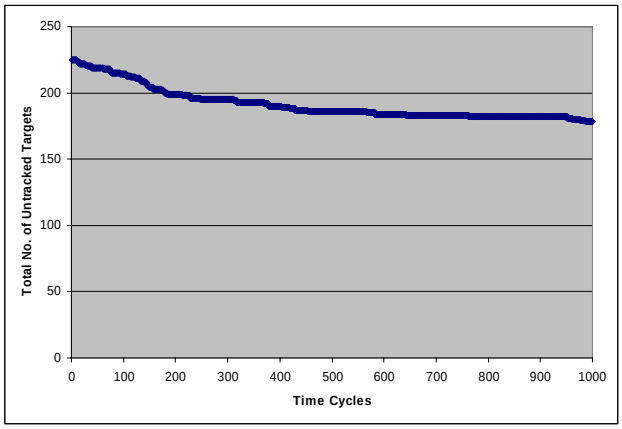


Figure 7. Change in untracked targets over time with Plan-Sharing but no cooperation

The decrease in the total number of untracked targets can be observed in Figure 7, at about 174, which is lower than that without plan-sharing. The enhancement in the system performance can also be observed as the total number of traces that increased up to 1248 in the Figure 8.

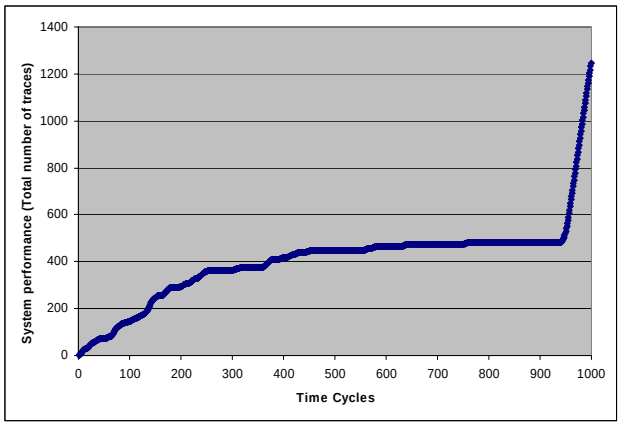


Figure 8. Total number of traces over time with Plan-Sharing but no cooperation (plan sharing is introduced at about cycle number 950, at "the knee of the curve")

6.3 UAVs with Plan-Sharing and with the lowest level Cooperation

Next, we introduced plan sharing as well as the lowest level cooperation. UAVs work together and generate their own bid list as explained in the algorithm in section 5. Here the cooperation level threshold is set to the lowest level. The results are depicted in Figures 9 and 10.

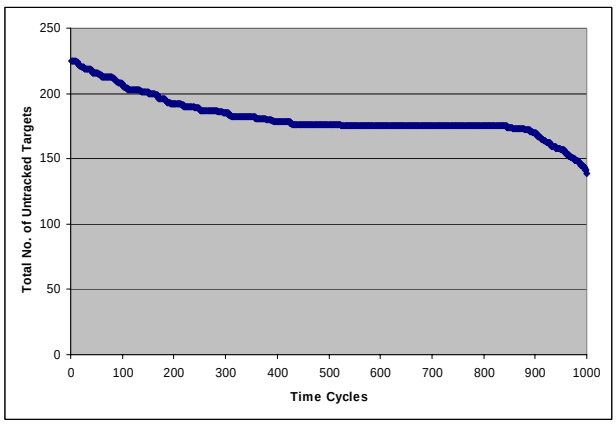


Figure 9. Untracked Targets over time with Plan-Sharing and with the lowest level cooperation

As shown in Figure 9, the total number of untracked targets is reduced to about 139. The system performance is shown in Figure 10 where the total number of collective system traces has increased to 2896.

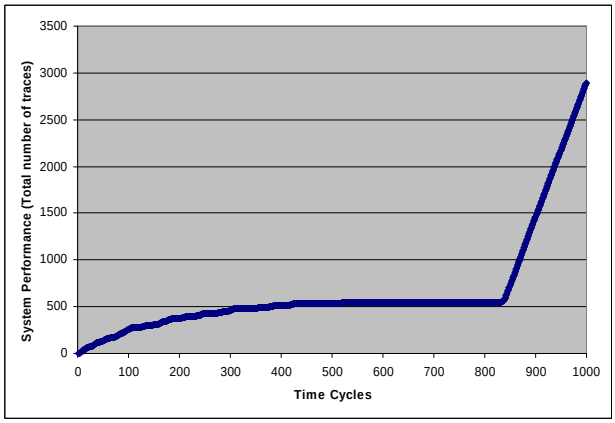


Figure 10. Total number of traces over time with Plan-Sharing and with the lowest cooperation level (low cooperation is introduced at about cycle number 850, at "the knee of the curve")

6.4 UAVs with Plan-Sharing and medium level cooperation

Here, the cooperation level is turned up to the medium level. With UAV bid list revised due to cooperation, performances are shown in Figures 11 and 12.

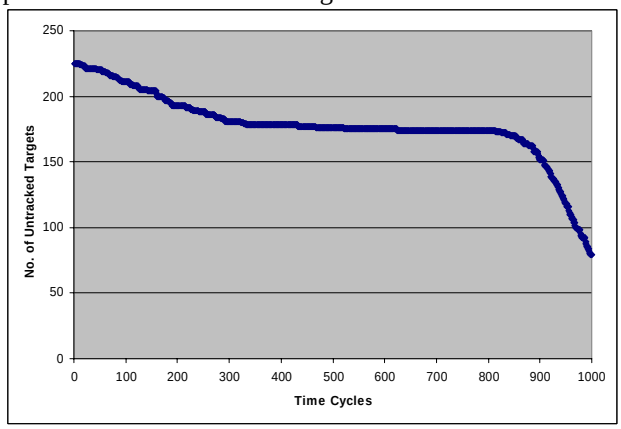


Figure 11. Untracked Targets over time with Plan-Sharing and medium level cooperation

Total number of untracked targets has further decreased to 79 as shown in figure 11. System performance as the total number of traces has increased to 3445.

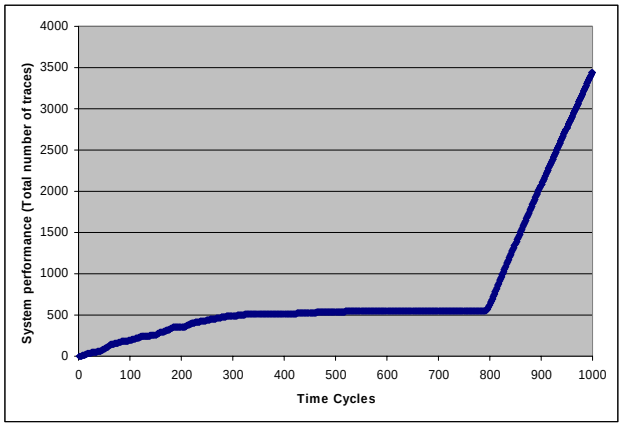


Figure 12. Total number of traces over time with Plan-Sharing and with medium level cooperation

6.5 UAVs with Plan-Sharing and medium high level of cooperation

At the medium high cooperation level, UAV bid lists were more seriously revised and the results are shown in Figures 13 and 14.

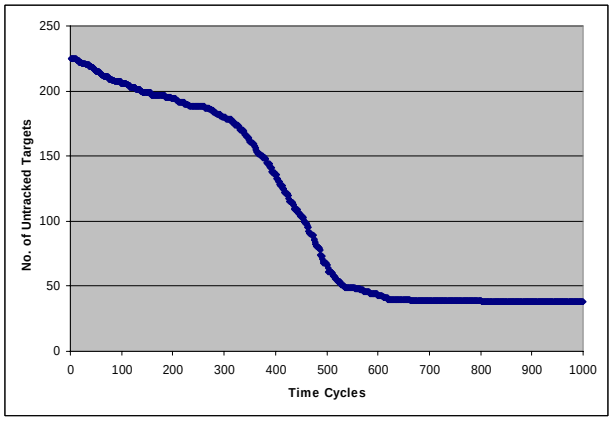


Figure 13. Untracked Targets over time with Plan-Sharing and at medium high cooperation level

The total number of untracked targets is further decreased to about 45 as shown in figure 13. Figure 14 shows an increase in the system performance to 4194.

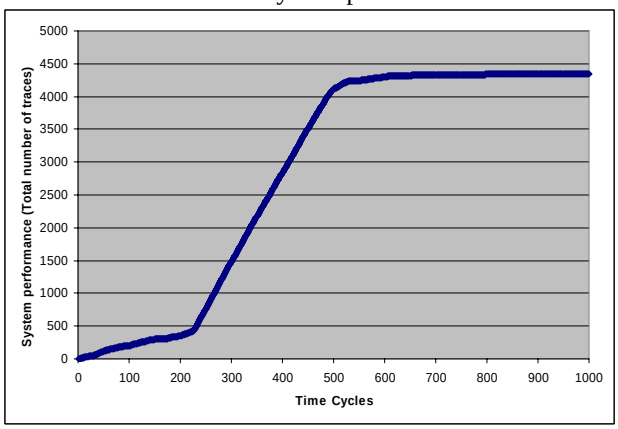


Figure 14. Total number of traces over time with Plan-Sharing and at medium high level cooperation

6.6 UAVs with Plan-Sharing and with the highest level cooperation

Finally, we increased the cooperation level to the highest level and the results are shown in Figures 15 and 16.

The total number of untracked targets, shown in Figure 15, is reduced to 14. The system performance, shown in Figure 16 reached 5353 traces.

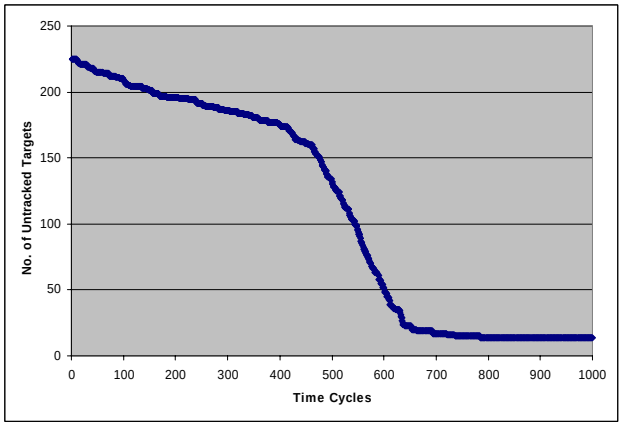


Figure 15. Untracked Targets over time with Plan-Sharing and at the highest cooperation level

It is observed that system performance increases by increasing level of cooperation.

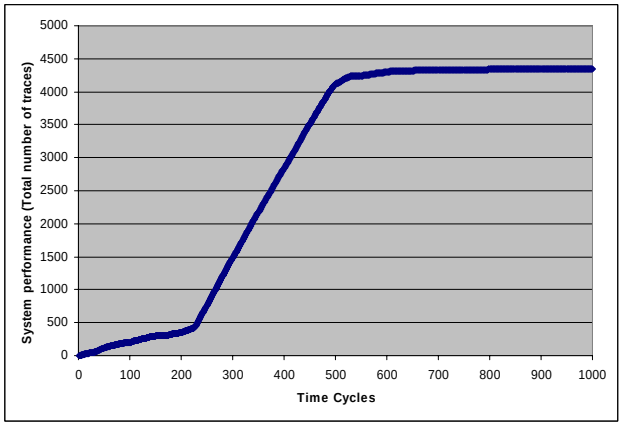


Figure 16. Total number of traces over time with Plan-Sharing and the highest cooperation level

7 Conclusions and Future work

The primary focus of this paper was on an implicit sense of plan sharing where agents modify their plans in light of other agents' plans. Communication is a key form of interaction in multi-agent systems, where multiple agents collaborate to attain a common goal.

The concept of collaboration was elaborated in a strategy for cooperation. Certain cooperation techniques are better suited for our experiments. Plan sharing and collaborative plan refinements clearly demonstrated improved performance.

Further work will consider agents with different capabilities as well as plan and goal priorities. Along with deontological notions of request and permission for collaboration, we will explore overlaps between collaboration, autonomy, and benevolence (Hexmoor, 2003).

8 References

[1] Alighanbari M., Kuwata Y., and How J. P., "*Coordination and Control of Multiple UAVs with Timing Constraints and Loitering*," Space Systems Laboratory, IEEE proceedings 2003.

[2] Alonso G., Schuldt H., Schek H.J., "*Transactional Coordination Agents for Composite Systems*," Proceedings of the International Database Engineering and Applications Symposium (IDEAS'99). Montreal, Canada, August 1999.

[3] Arrazola X., "*Collective Action: Logical Foundations for Interaction*," report no. ILCLI-96-FCSAI-1. Donostia – San Sebastian: ILCLI, 1996.

[4] Balch T. and Arkin R., "*Behavior-based Formation Control for Multi-robot teams*," IEEE Robotics and Automation, 14(6), 926-939. 1998.

[5] Beer J. M., Whatley J., "*Group Project Support Agents for helping students work online*," HCI International 1999, Munich, Germany, Lawrence Erlbaum, London.1999.

[6] Benton M.J., "*Biodiversity on land and in the sea*," Geological Journal, 36:211-230, 2001.

[7] Bonabeau E., Dorigo M. and Theraulaz G., "*Swarm Intelligence: From Natural to Artificial Systems*," New York, NY: Oxford University, 1999.

[8] Chandler R., and Pachter M., "*Research Issues in Autonomous Control of Tactical UAV's*," proceedings of the American Control Conference, 1998, Vol.: 1, 24-26, pp.394-398. June 1998

[9] Chandler P., Schumaker C., Rasmussen S., "*Task Allocation for Wide Area Search Munitions via Network Flow Optimization*," AIAA GNC, Aug 2001.

[10] Clark R., "*Uninhabited combat aerial vehicles: airpower by the people, for the people, but not with the people*," Maxwell air force base al: air university press CADRE paper, 2000.

[11] Cohen, D.M., Moridae. In W. Fischer, G. Bianchi and W.B. Scott (eds), "*FAO species identification sheets for fishery purposes*," Eastern Central Atlantic; fishing areas 34, 47 (in part). Vol. 3, FAO, Rome, 1981.

[12] Cohen, P. R., Levesque H. R., and Smith I., "*On team formation,*" Hintikka, J. and Tuomela, R. (Eds.) Contemporary Action Theory. Synthese, 1997.

[13] Fredslund J. and Mataric M., "*A General Algorithm for Robot Formations Using Local Sensing and Minimal Communication*," IEEE Transactions on Robotics and Automation, 18(5):837-846. 2002.

[14] Grosz. B. and Sidner. C., "*Plans of Discourse*," In P. Cohen et.al. eds., Intentions in Communication. MIT press. Forthcoming, 1990.

[15] Grosz B., Sidner C. and Loachbaum K. E., "*Models of plans of support communication: An initial report*," In proceedings of AAAI-90, Boston, MA, 1990.

[16] Grosz B. and Kraus S., "*Collaborative Plans for Complex Group Action*," In *Artificial Intelligence*. 86(2), pp. 269-357. 1996.

[17] Grosz B., and Kraus, S., "*The Evolution of SharedPlans*," In Foundations and Theories of Rational Agencies, A. Rao and M. Wooldridge, eds. pp. 227-262, 1999.

[18] Hexmoor, H., Duchscherer H., "*Efficiency as a Motivation to Team*,". FLAIRS Conference 2001: 49-52, 2001.

[19] Hexmoor, H., "Weekly Dependent Agents may rely on their luck or Collaboration: A Case for Adaptation," In Proceedings of The Society for the Study of Artificial Intelligence and the Simulation of Behavior (AISB), York, UK, 2003.

[20] Henry Hexmoor, Agent Autonomy in a group, A special issue of Journal of Connection Science, Taylor and Francis, 2003.

[21] Hexmoor, H., McLaughlin, B., and Baker, M., Swarm Control in Unmanned Aerial Vehicles, In Proceedings of International Conference on Artificial Intelligence (IC-AI), CSREA Press, 2005.

[22] Korta, K. and Larrazabal J. M. (eds.) "*Semantics and Pragmatics of Natural Language: Logical and Computational Aspects*," Proceedings of the Donostia-Toulouse 1993 Workshop. Donostia: ILCLI Series No. 1. (ISBN: 84-920104-3-6), 1995.

[23] Lochbaum K., "*Using Collaborative Plans to Model the Intentional Structure of Discourse*," PhD thesis, Harvard University, 1994.

[24] Lochbaum K., "*The Use of Knowledge Precondition in Language Processing*," In IJCAI95, Proceedings of the 14th International Joint Conference on Artificial Intelligence, pages 1260-1266, 1995.

[25] Lochbaum K., "*A collaborative planning model of intentional structure*," Computational Linguistics 24(4). 1998.

[26] Luck, M. and Griffiths, N., Coalition Formation Through Motivation and Trust. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2003)*, ACM Press, 2003.

[27] Lua, C.A., Altenburg, K., and Nygard, K.E., Synchronized Multi-Point Attack by Autonomous Reactive Vehicles with Simple Local Communication, Proceedings of IEEE Swarm Intelligence Symposium, 2003., 2003.

[28] Mataric, M.J., Kin recognition, similarity, *and group behavior*. In Fifteenth Annual Cognitive Science Society Conference, pages 705--710. Lawrence Erlbaum Associates, June 1993.

[29] Muller, S., Marchetto J., Airaghi, S. and Kourmoutsakos, P., "*Optimization Based on Bacterial Chemo taxis*," IEEE Transactions on Evolutionary Computation, 6:16-29. 2002.

[30] Parker, L., "*Designing Control Laws for Cooperative Agent Teams*," Proc. IEEE International Conference on Robotics and Automation, 3:582-587. 1993.

[31] Patcher, M., Chandler, P., Swaroop, D., Fowler, J., "*Complexity in UAV cooperative control*," ACC 2002. pp. 1831-1836. 2002.

[32] Pollack. M. E., "*Plans as complex mental attitudes*," in P. Cohen et. al., eds., Intentions in Communication. MIT press. Forthcoming, 1990.

[33] Raymond, R., Hill S., Dougherty, and Moore, J.T., "*Model Signal Latency Effects Using Arena*," Proceedings of the 2000 Winter Simulation Conference. 2000.

[34] Reynolds, C., Flocks, Herds and Schools, "*Computer Graphics*," 1987, 21:25-34. 1987.

[35] Reynolds, R., and Chung, C., "*A Test bed for Solving Optimization Problems Using Cultural Algorithms*," Evolutionary Programming, 225-236. 1996.

[36] Reynolds, C., "*Steering Behaviors for Autonomous Characters*," Proc. Game Developers Conf.1999, 763-782. 1999.

[37] Richards, A., Bellingham, J., Tillerson, M., and How, J., "*Multi-Task Allocation and Path Planning for Cooperative UAVs*," Second Annual Conference on Cooperative Control and Optimization, Nov 2001.

[38] Rich, C., Sidner, C. L**., ,** "*Adding a collaborative agent to graphical user interfaces*," Symposium on User Interface Software and Technology, Proceedings of the 9th annual ACM symposium on User interface software and technology, 1994.

[39] Sidner, C. L., Rich C., "*COLLAGEN: When Agents Collaborate with People*," Proceedings of the First International Conference on Autonomous Agents (Agents'97), 1997.

[40] Stover, J.A., and Gibson, R.E., "*Controller for Autonomous Device*," US patent # 5, 642,467, Issued June 1997.

[41] Tuomela, R., "*Cooperation*" , Kluwer Academic, 2000.

[42] Reggia, J. and Winder, R., "*Distributed Partial Memories to Improve Self-Organizing Collective Movements*," IEEE Transactions. SMC B, 2004, 34: 1967-1707. 2004.

[43] Vail D. and Veloso, "*Multi-Robot Dynamic Role Assignment and Coordination through Shared Potential Fields*," In Multi-Robot Systems. A. Shultz, L. Parker, and F. Schneider (editors), Kluwer, 2003.

An Integration Rule Processing Algorithm and Execution Environment for Distributed Component Integration

Ying Jin
California State University, Sacramento
Department of Computer Science
Sacramento, CA 95819, USA
E-mail: jiny@ecs.csus.edu

Susan D. Urban, Suzanne W. Dietrich and Amy Sundermier
Arizona State University
Department of Computer Science and Engineering
Tempe, AZ 85287-8809, USA
E-mail: s.urban@asu.edu, dietrich@asu.edu

Keywords: active databases, software component integration, rule processing algorithm, transaction, management

Received: July 25, 2005

The Integration Rules (IRules) Project[*] provides an active, rule-based approach for supporting event-driven activity in applications involving distributed software component integration. This paper presents the execution model, transaction model, and integration rule execution algorithm of the IRules environment. The paper begins with an overview of the IRules language framework to establish the context for the use of events and rules in the integration process, with Enterprise JavaBeans (EJBs) serving as a component model. The paper then elaborates on the integration rule processing algorithm and execution environment. The rule execution model supports traditional active rule coupling modes, and defines a new immediate asynchronous mode to support concurrent execution of triggered rules and transactions. The transaction model is based on the flexible transaction model, providing a means to coordinate global transaction execution with the transactional features of EJB containers. IRules component wrappers also provide support for the global transaction context as well as the synchronization of method execution with the nested execution of integration rules. The paper defines the semantics of coupling modes in terms of cycles and levels of rule execution, presenting the integration rule processing algorithm for coordinating the execution of events and methods on components with the nested execution of integration rules in the context of the transaction model. The details of the algorithm are presented using Unified Modeling Language (UML) activity diagrams, providing a generic approach that can be used as the foundation for rule processing in other distributed environments. An investment application is used to illustrate the concepts presented in this paper.

Povzetek: Predstavljen je algoritem za integracijo pravil.

1 Introduction

The development of advanced enterprise applications often requires the integration of distributed software components and services. Standard component models and distributed computing tools, such as the Common Object Request Broker Architecture (CORBA) [1] and Enterprise JavaBeans (EJBs) [2], have been developed to facilitate the integration process in distributed environments. However, component integration could be a difficult process in some cases, since application integrators must not only mediate the semantics of component interactions, but must also be skilled in low-level knowledge of middleware programming, event handling, and transaction management. This difficulty motivates the need for a more declarative approach to the integration process.

In response to this need, the Integration Rules (IRules) Project has developed an active rule-based approach to distributed component integration, using *integration rules* to provide a declarative approach to event-driven integration activity [3, 4, 5, and 6]. The IRules project is based on the concept of active database rules. Active database systems extend traditional databases by supporting mechanisms to automatically monitor and react to events that are taking place either inside or outside of the database system [7 and 8]. Active database rules, known as Event-Condition-Action (ECA) rules, are the core of any active system.

[*] This research was supported by National Science Foundation under Grant No. IIS-9978217.

Similar to an active rule, an integration rule consists of an event, a condition, and an action. The event of an integration rule is generated from distributed sources. The condition is expressed as a query over distributed components. If the condition evaluates to true, the action is invoked to execute methods on components or to invoke application transactions that capture integration logic. Integration rules can therefore be used to separate event processing from the main integration logic of an application. Furthermore, event handling and rule processing are managed within the transactional environment of the IRules system, shielding the low-level details of event handling and transaction management from integrators. Integrators can therefore focus on integration logic rather than low-level programming details.

The IRules approach to component integration consists of a language framework described in [4] and an execution environment for processing rules and transactions [6] over distributed components. The execution environment presented in this paper consists of a rule processing algorithm and transaction management system, illustrating a rule-based approach to the integration of components with well-defined interfaces based on the EJB component model [2]. The integration of black-box components introduces several challenges to the development of a rule and transaction processing framework for integration rules. First of all, components cannot be modified and they are typically not aware of their participation in the integration framework. As a result, components alone do not provide necessary behavior for participating in more global rule and transaction processing activities. Furthermore, the EJB component model has its own notion of transactional behavior, which is beyond the control of an external environment such as IRules. The execution of integration rules within a transactional context requires suitable control logic at the global IRules level to overcome the restrictions of the underlying EJB component model. Furthermore, active rules can trigger other active rules, thus forming a nested structure as a result of cascaded rule execution. Since the nested execution of rules and their transaction control in a distributed environment may span across several distributed locations, distributed rule processing is more challenging than that of centralized active rule environments.

The IRules execution model presented in this paper supports the traditional dimensions of active rule execution, with extensions for use in a distributed environment. The Integration Rule Processing (IRP) algorithm controls rule processing in a distributed environment, fully supporting immediate, deferred, and decoupling modes of execution. The immediate asynchronous mode is a new coupling mode defined in this research to support concurrent execution of triggering transactions and triggered rules, thus improving the performance of distributed rule processing. The IRP algorithm also provides support for the nested execution of immediate rules. Handling immediate rules in a distributed environment requires system control for synchronization between a triggering transaction and the triggered rules. The synchronization process requires suspension of the method execution of the EJB component, allowing the generation of events before and after the method execution, with the immediate execution of rules in response to the events. The IRP algorithm described in this paper contributes to the use of immediate coupling modes and nested rule execution within the IRules framework. These features for the nested execution of immediate rules in a distributed environment have not been addressed by previous research, especially in the context of component integration.

Our research uses Unified Modeling Language (UML) activity diagrams [9] to present the logic of the rule execution algorithm. The algorithm is generic so that it can be used in other environments for rule processing, although the specific implementation of the IRP algorithm within the IRules environment is supported by the IRules transaction management [6], wrapper design [10], and synchronization algorithms [11].

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 provides an overview of the IRules approach with a presentation of the language framework and system architecture. Section 4 presents the integration rule coupling modes as well as the transaction model and the transactional support found in wrappers for the synchronization of rule and method execution. Section 5 describes the rule processing algorithm and provides an example of rule execution using an investment application. The paper concludes in Section 6 with a summary and discussion of future research directions.

2 Related Work

There are several active database research projects that have influenced the development of the IRules environment, including relational active databases, such as POSTGRES [12] and Starburst [12], as well as object-oriented active databases, such as HiPAC [14], SAMOS [15], ADOOD RANCH [16], and REACH [17]. Active rules also exist in a limited form in commercial database systems as database triggers [18].

Active rule execution algorithms have been addressed in centralized environments, such as the research outlined in the introduction to active database systems in Widom and Ceri [8], as well as the work of Fraternali and Tanca [19] and Warshaw [20]. The algorithm in [8] provides a high level abstract view of rule processing, repeatedly retrieving a triggered rule, evaluating the condition, and performing the action if the condition evaluation is true. Similarly, the algorithm in [19] presents three phases of active rule processing. In the triggering phase, the algorithm builds a set of rules that are triggered. In the consideration phase, the algorithm first gets rules from

the set of rules constructed in the triggering phase, and then evaluates the condition part of each rule. In the action phase, if the condition evaluation of a rule returns non-empty bindings, the algorithm will perform the action of the rule. Compared to [8] and [19], this paper presents an active rule processing algorithm that was specifically designed to assist with component integration in a distributed environment. The IRules algorithm not only covers the three basic phases of active rule processing for a set of triggered rules, but also elaborates on the nested execution of integration rules for distributed components, which is a challenging extension to past work on centralized rule execution algorithms. The IRules algorithm also describes how to execute a rule according to four types of coupling modes and how to react to different types of events in the context of distributed component integration. Whereas the algorithm in [20] uses state transition to characterize rule execution according to different coupling modes, the IRules algorithm uses a coordinate system to describe the semantics of rule execution.

In addition to centralized active database systems, there are several research projects on active, rule-based distributed systems. In the system described in [21], ECA rules are used to provide distributed communication for the components that describe interfaces in the Object Management Group (OMG) Interface Definition Language (IDL) [22]. The project focuses on the specification, detection and management of composite events.

C^2offein [23] is a CORBA-based system with a comprehensive design of distributed event detection. The underlying data sources are wrapped to enable read access in a CORBA environment. If a database does not support an active mechanism, the wrapper queries the database at regular intervals to detect changes in the data. Clients can also call event detection before any update operation to the database. Rule processing is supported using a production rule expert system shell.

The FRAMBOISE (FRAMework using oBject OrIented technology for Supplying active mEchanisms) project [24] is an object-oriented framework formed by a toolbox to provided active database functionality, such as event definition, event detection, and rule execution. The set of architectural components that separates the active functionality from the underlying DBMS is called an activity service. A database event detection connector, condition evaluation connector, and database action execution connector regulate the interaction between the activity service and the underlying DBMS.

In the system described in [25], active rules are used to glue together existing applications in a distributed environment. Active rule processing is implemented through event, condition, and action services. The condition object of a rule subscribes to the event object of the rule, while the action object subscribes to the condition object of the rule. The system uses the publish/subscribe service implementation of X^2TS [26] as a notification mechanism between event, condition, and action, where X^2TS is based on the CORBA Notification Service. X^2TS can also provide additional transaction control mechanisms such as exception handling over the basic CORBA Transaction Service. In contrast, the IRules rule manager controls when to evaluate a condition and execute an action according to coupling modes, rather than through a publish/subscribe service.

The above distributed rule projects have all been based on the use of the CORBA standard. In contrast, the IRules project requires access to distributed resources that advertise services using the EJB component model. The use of distributed rules for the integration of EJB component technology has not been addressed by the existing research. The IRules project is also using Jini connection technology [27] as the primary means for distributed object computing, rather than CORBA as in other projects. Furthermore, the rule processing algorithm of IRules can handle nested rule execution in a distributed environment. Existing distributed rule projects have not addressed cascaded rule execution within distributed transactions.

Using active rules to control the flow between activities of a workflow system has been adopted by a number of projects, such as the project described in [28]. More recent work on workflow uses ECA rules both inside and outside of activities. The work in [29] is a centralized workflow management system, where ECA rules are used for constraint management inside tasks, as well as for the control of the execution order of tasks. In [30], the workflow system named CapBasED-AMS uses ECA rules to specify the security authorization requirements imposed on a task as well as the execution sequence. The $TriGS_{flow}$ system of [31] is introduced as a framework for workflow management, where ECA rules encapsulate and realize coordination policies. In the WIDE (Workflow on Intelligent Distributed Database Environment) project [32], active rules are used in a workflow management system for exception handling. Compared to component integration, the flow movements from task to task in a workflow environment are well-defined compared to the interconnection of software components. A workflow system has more control over the tasks that are executed, while access to component services, especially black-box components, may be more restrictive.

Active rules are also used in the composition of web services. The research in [33] proposed the SELF-SERV (compoSing wEb accessibLe inFormation & buSiness sERVices) system to compose services within a peer-to-peer paradigm. SELF-SERV includes a declarative language based on state charts and a peer-to-peer service execution model. A statechart consists of states and transitions. Transitions are labeled by ECA rules. Compared to IRules, SELF-SERV is for the composition of web services, while IRules is for the integration of software

components. ECA rules are used as the "glue" of the IRules integration to specify event-driven integration logic, while state charts are the "glue" for the composition of SELF-SERV. ECA rules are used in SELF-SERV for state transition, but the logic of execution is not controlled by ECA rules. In the IRules environment, integration logic is composed from the use of application transactions together with integration rules that respond to events.

3 The IRules Approach

There are two important aspects of the IRules environment: the IRules Definition Language (IRDL) for application specification and the IRules execution environment for integration rule and transaction execution. IRDL supports the definition of components, events, rules, and application transactions for distributed component integration. The execution environment consists of the execution model for integration rules, transaction management for rule execution, and the rule execution algorithm that coordinates the execution model with transaction management. The IRules execution environment is described in detail in Section 4. This section overviews the language framework and the architectural design of the IRules environment.

3.1 The IRules Definition Language

The IRDL consists of four sub-languages: the Component Definition Language (CDL) for defining IRules components, the IRules Scripting Language (ISL) for describing application transactions, the Event Definition Language (EDL) for defining events, and the Integration Rule Language (IRL) for defining active rules. The IRDL was initially reported in [3, 5] with refinements of the language presented in [10, 11, 34, 35]. The examples of the language presented in this section originally appeared in [4, 6] and are repeated here to make the paper self-contained.

3.1.1 Component Definition Language (CDL)

The CDL establishes an object model for application integration activity [3, 5, and 10]. The current implementation of IRules is based on the EJB component model, assuming that all of the components of the environment are EJB components. To support component interconnection with active rules, IRules adds a semantic layer on top of existing EJB components. This layer is the IRules wrapper layer. IRules wrappers are automatically generated after CDL is compiled. IRules wrappers provide additional functionality to black-box components, such as defining externalized relationships between distributed components, and specifying extents, derived attributes, and stored attributes for each component. The IRules wrapper layer also defines the events generated before and after method calls on components as well as the events that are internal to black-box components. The details of IRules wrappers can be found in [10].

As an example of CDL, Figure 1 defines two externalized relationships and one event for a pending order component within an investment application. The first line of the component definition indicates that the StockBroker_PendingOrder is an EntityBean component. The second line of the definition specifies an extent that can be used to query all pending order instances, a feature that is useful in the specification of integration rule conditions (see Section 3.1.4 for an example). Assuming that StockBroker_Stock, StockBrocker_Portfolio, and StockBroker_PendingOrder are implemented as separate distributed components, the first relationship definition in Figure 1 illustrates an externalized, bi-directional relationship between components: a StockBroker_PendingOrder is orderedBy a StockBroker_Portfolio, while in the inverse direction, a StockBroker_Portfolio orders a StockBroker_PendingOrder. The second relationship defines the relationship between StockBroker_PendingOrder and StockBroker_Stock: a StockBroker_PendingOrder actsUpon a StockBroker_Stock while a StockBroker_Stock has pendingTrades on the StockBroker_PendingOrder. The CDL also defines an event afterSetAction that is to be raised after the setAction operation on StockBroker_PendingOrder. Note that the CDL definition of StockBroker_PendingOrder does not repeat any of the method definitions of the original component definition. CDL is used to enhance the component definition with IRules functionality.

3.1.2 IRules Scripting Language (ISL)

In the IRules environment, ISL describes well-defined sequences of processing logic as application transactions [3, 5]. ISL is based on JACL [36], which is the Java version of the Tool Command Language (TCL) [37]. An ISL example in an investment application is shown in Figure 2. The clientWantsToSellStock application transaction consists of two steps: create a pending order component and print the information of this pending order. The newInstance command is a JACL extension that abstracts a sequence of statements into one command, thus making the script concise and easy to reuse.

```
Component StockBroker_PendingOrder implements
EntityBean
(extent pendingOrders)
{ relationship StockBroker_Portfolio orderedBy inverse
StockBroker_Portfolio::orders;
  relationship StockBroker_Stock actUpon inverse
StockBroker_Stock::pendingTrades;
  event afterSetAction (pnAction) {method after
setAction(string pnAction)};}
```

Figure 1: CDL of PendingOrder Component

```
application transaction clientWantsToSellStock(String pnId,
String portfolioId, String stockId, int numOfShares,float
```

```
desiredPrice, String action, Stock actUpon, Portfolio
orderedBy)
tcl newInstance
{
set pn [newInstance PendingOrder $pnId $portfolioId
$stockId $numOfShares $desiredPrice $action $actUpon
$orderedBy $irulesId];
 printPendingOrderInfo $pn $irulesId;
}
```

Figure 2: ISL Example for the clientWantsToSellStock transaction

3.1.3 Event Definition Language (EDL)

There are four different types of IRules events [3, 5, and 11]: *method events*, *application transaction events*, *internal events*, and *external events*. A method event is generated before or after the execution of a method on a component. An application transaction event is generated before or after the execution of an application transaction. An internal event is an event generated by a black-box component. An external event is generated by sources external to the IRules environment. EDL describes application transaction events and external events. Method events and internal events are defined in CDL following EDL syntax. Figure 1 illustrates the definition of a method event afterSetAction that is generated after the setAction operation in the Stock component.

Figure 3 shows an application transaction event definition in EDL. The specification has the syntax similar to the method event specification in Figure 1. The key word appTrans identifies that the event is an application transaction event. The event has an event name afterSellStockOnNewPO and five parameters. The event parameters are constructed by the projection of the parameters of the application transaction by parameter name.

```
event afterSellStockOnNewPO(stockId, price, portfolioId,
numOfShares, pn)
{
appTrans after sellStockOnNewPO(String stockId, float
price, String portfolioId, int numOfShares,
stockBroker.PendingOrderComponent.PendingOrder  pn);
}
```

Figure 3: EDL for the afterSellStockOnNewPO event

3.1.4 Integration Rule Language (IRL)

IRL is a language for defining integration rules [3, 5, 34, and 35]. IRL is based on the traditional ECA rule format in active database systems. An integration rule includes an event, a condition, and an action. A condition includes a Boolean clause and an optional query over the object model to define a binding structure for data that satisfies the condition. The action part consists of an optional from clause and a do clause. The from clause iterates through the binding structure passed from the condition. The do clause executes the action in the format of a method call or an application transaction.

```
create rule  clientWantsToSellStockRule
event     afterClientWantsToSellStock(pnId, portId, stockId,
numOfShares, desPrice, pnaction, actUpon, orderedBy)
condition immediate
          when pnaction = "sell"
          define stockAndPendingOrder as
                    select struct ( stk: s, newPo: pn )
                    from s in stocks, pn in pendingOrders
                    where pn.id=pnId and pn.actUpon=s
                    and desPrice<=s.price
action    immediate
          from sp in stockAndPendingOrder
          do  sellStockOnNewPO(stockId,  sp.stk.price,
portId, numOfShares, sp.newPo)
```

Figure 4: IRL Example of the clientWantsToSellStockRule rule

An example of IRL is shown in Figure 4. In this rule, the event is signaled after the clientWantsToSellStock application transaction. The condition checks whether the pendingOrder intends to sell stock.. If the rule condition is satisfied, a binding structure is defined for relevant instances of stock and pendingOrder. The binding structure definition uses the extents of the stock and pendingOrder components that are specified in the CDL and the parameters of the event to find relevant stocks and pending orders. The action part iterates through the stockAndPendingOrder structure and executes the sellStockOnNewPO application transaction to perform the functionality of selling stocks.

3.1.5 Putting It All Together

Figure 5 presents an example of how all of the sublanguages of IRDL work together. CDL defines wrapped components and the compilation of CDL generates wrappers. After wrappers are generated [10], distributed components in different containers may have externalized relationships. Four types of events are defined by EDL and CDL. Events can trigger integration rules defined by IRL. The action part of the rule can invoke an application transaction (defined by ISL) or a method on an EJB component. The execution of an application transaction or a method can raise application transaction events, method event, or internal events, which can trigger additional rules. As a result, integration rules can be triggered in a nested structure. We will present an execution scenario of rule nesting in Section 5.3.

3.2 The IRules Architecture

IRules has designed a distributed architecture to support the IRDL language framework. The

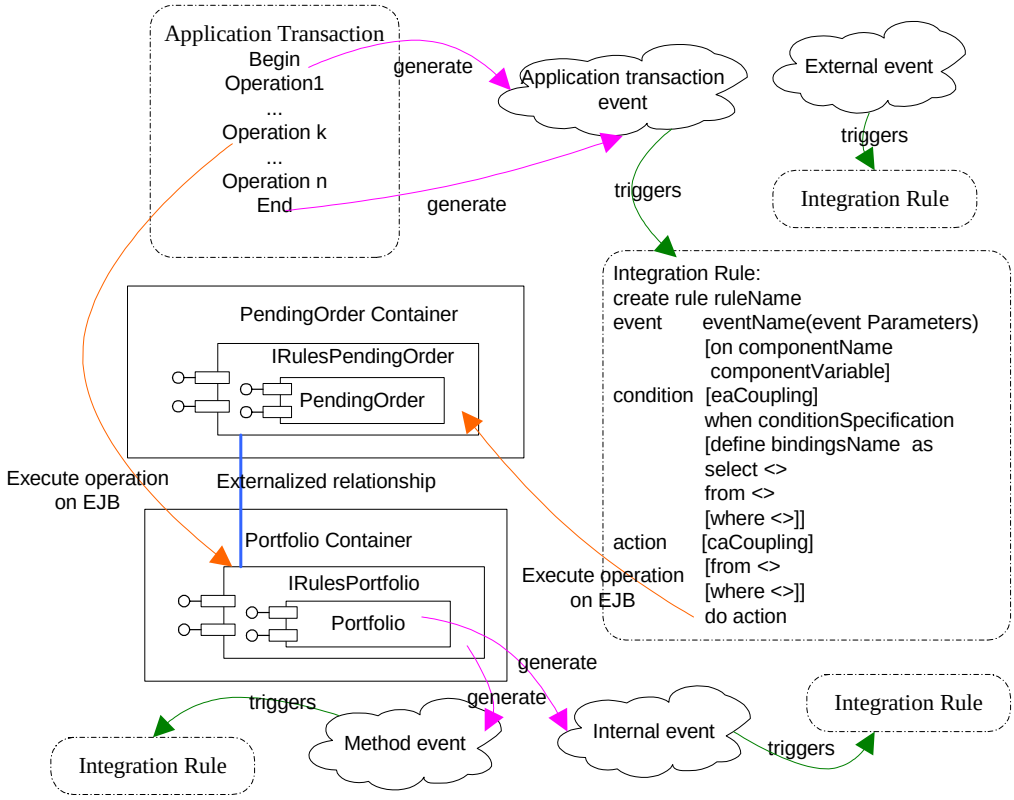


Figure 5 Interaction Between the IRDL Sublanguages

architecture can be abstracted into three layers, as shown in Figure 6. The top and middle layers are the interfaces. The bottom layer is the implementation of the integration system.

IRules provides interfaces for two types of integration users: integrators and end users. Integrators use IRDL to describe integration logic. The compilation of IRDL results in the population of metadata and the automatic generation of wrappers. The interface for end users consists of a list of application transactions, which have been expressed by integrators using ISL and compiled by the IRules compiler. An end user then selects an existing application transaction to express their integration request. For example, if a user wants to sell stock, the user can select the clientWantsToSellStock application transaction from Figure 2, providing values for each parameter of the application transaction. The user request is sent to the application transaction processor component of the IRules system.

The implementation layer consists of architectural components for the IRules system. Figure 6 presents the fundamental components of the architecture. The Jini distributed computing environment is used as the backbone of the system, with IRules architectural components implemented as Jini Services. Upon user request, the application transaction processor processes the ISL script. The processing may invoke wrapped EJB components. The processing of an application transaction or a method call on an EJB component can raise events. The event handler pushes the event to the rule manager, where the rule manager queries the metadata to retrieve rules triggered by this event. During rule processing, the rule manager interfaces with the transaction manager to establish the transaction context for rule execution. The rule manager also interfaces with the object manager for accessing components. The rule manager submits requests to the query processor for rule condition evaluation during rule processing.

Through the IRDL language framework and the architectural design, the IRules integration system allows application integrators to specify the integration logic in a declarative fashion, while the end users can use the defined integration logic to specify their request. In contrast to the traditional integration approach, the IRules approach does not require an integrator's low-level knowledge of distributed programming issues. Integrators can focus on integration logic, rather than the technical details of rule and transaction processing.

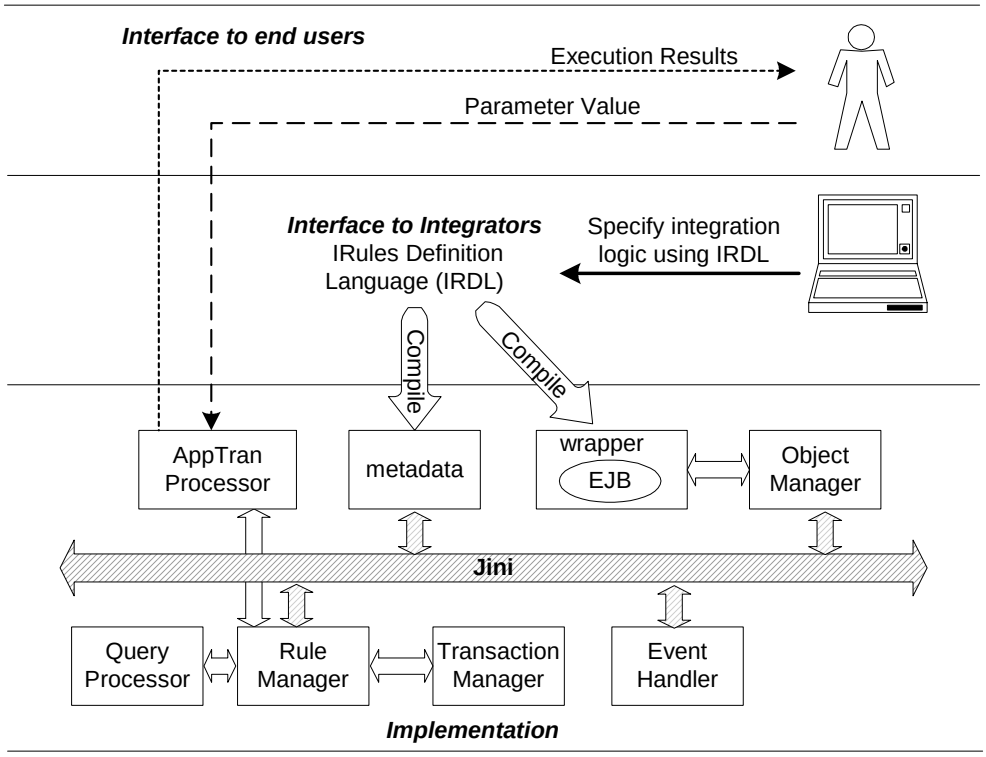


Figure 6: IRules Architecture

4 The IRules Execution and Transaction Model

This section presents the details of the IRules execution and transaction model. A preliminary discussion of the IRules transaction model appears in [6]. The full details of the transaction management system are presented in this section to establish the context for presentation of the rule processing algorithm in Section 5. Section 4.1 defines the coupling modes of the environment, with specific emphasis on the synchronous and asynchronous options of the immediate coupling mode. Section 4.2 elaborates on the transaction model and the manner in which it interacts with the transactional features of EJB containers. Section 4.3 discusses the support that IRules component wrappers provide for the global transaction context as well as the synchronization of rule and method execution.

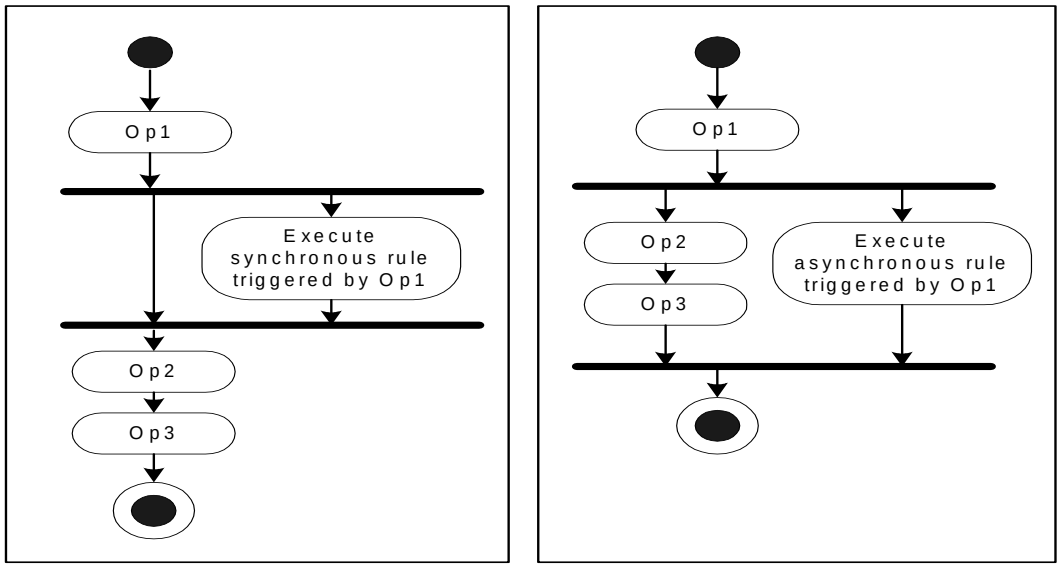
4.1 Integration Rule Coupling Modes

The execution model of an active rule system specifies how to coordinate a set of rules at runtime. The execution model is characterized by several features, such as coupling modes, transition granularity, net-effect policy, cycle policy, priority, and scheduling [7]. IRules follows the definition of the execution model features that are defined in [7]. In this paper, we address the coupling mode feature, since the use of coupling modes in a distributed environment is the primary focus of the IRules execution model.

The coupling modes of an active rule allow rule definers to specify how to execute the rule at run time. A coupling mode can be specified between the event and condition (E-C), between the condition and action (C-A), or between the event and action (E-A). Integration rules support four types of coupling modes: *immediate synchronous*, *immediate asynchronous*, *deferred*, and *decoupled*.

Using the E-C coupling mode as an example, the immediate synchronous E-C coupling mode indicates that the condition of a rule must be evaluated immediately after the event is raised. The immediate asynchronous mode is a new coupling mode that has been defined as part of this research. In an immediate asynchronous E-C coupling mode, the condition is evaluated immediately after the occurrence of an event, but the triggering transaction that raised the event will not be suspended. The execution of the integration rule and the triggering transaction are therefore concurrent.

Figure 7 illustrates the difference between the immediate synchronous and immediate asynchronous modes using a UML activity diagram [9]. In each box of Figure 7, the pair of synchronization bars (heavy black bars) represents the logic to fork and join processes, where the first bar is a fork and the second bar is a join. In Figure 7a, Op_1 is an event that triggers an immediate synchronous rule, so a subtransaction is started to process the rule. The triggering transaction suspends until the rule completes. After the rule joins the triggering transaction, Op_2 and Op_3 can be executed. In contrast, as shown in Figure 7b, Op_1 is an



a. Execution of Immediate Synchronous Rules

b. Execution of Immediate Asynchronous Rules

Figure 7: Synchronous vs. Asynchronous Rule Processing

event that triggers an immediate asynchronous rule. A new subtransaction is started to execute the asynchronous rule, but the triggering transaction does not suspend. As a result, Op_2 and Op_3 are concurrently executed with the asynchronous rules. At the end of the triggering transaction, the asynchronous rule joins the triggering transaction. The use of the immediate asynchronous mode can only be used when the rest the operations of the triggering transaction do not depend on the results of the immediate rule.

The deferred E-C coupling mode postpones rule condition evaluation to the end of the top-level transaction of execution (i.e., the outermost transaction within which the event was raised), based on the use of the deferred coupling mode as defined in [378]. The decoupled mode is only available for E-A and C-A coupling. Using the decoupled C-A coupling mode as an example, the decoupled action of a rule is executed immediately in a new top-level transaction, concurrent with the transaction that triggered the rule.

In addition to coupling modes, an integration rule can be triggered before an event happens or after an event happens. This feature is specified as before and after modifiers in the definition of a rule. Rule execution before an event is reasonable only when an event generator can trap the occurrence of the operation associated with the event.

4.2 Transaction Model of the IRules Environment

In an active system, the execution model relies heavily on the notion of transactions. For example, coupling modes are used to specify the transactional relationships between different parts of an active rule. Rules are also required to execute within appropriate transaction contexts for correct processing logic.

A fundamental issue with respect to transaction processing within IRules is the selection of a transaction model that is appropriate for the nested execution of rules over EJB components. In the nested transaction model [39], a subtransaction cannot release its results until its parent transaction commits. In contrast, the flexible transaction model [40] has a compensating mechanism that allows early commit of subtransactions. The flexible model avoids unnecessary blocking of subtransactions. The compensating mechanism ensures atomicity of a transaction when allowing unilateral commit of subtransactions. Although the flexible transaction model avoids unnecessary waiting time, the flexible transaction model can be more time-consuming than the nested transaction model when compensating work is required in the case of transaction failure.

The underlying component model also constrains the selection of a suitable transaction model. In transaction control for traditional databases, the release of locks and the update of permanent storage can be fully controlled by the transaction manager of the database. It is not possible to control black-box EJB components in such a manner. Each entity bean has an underlying relation in a relational database, and each instance of the bean corresponds to a tuple in that relation. Entity beans must be accessed with a *container-managed transaction*. That is, the transaction for method invocation of an entity bean is totally controlled by the EJB container.

As a more detailed explanation, each container uses ejbload() to refresh an entity bean's state from the database and ejbstore() to save the entity bean's state in the database. Before a method defined in the EJB remote interface is invoked from outside of the EJB component, the container will call ejbload(). After the method finishes execution, the container will call

ejbstore(). If the IRules transaction manager attempts to use the Two Phase Commit (2PC) protocol [41], the permanent storage can only be updated when all subtransactions are ready to commit. However, with no notion of the parent-child hierarchy of the outer transaction semantics for 2PC, the container is independently determining when to retrieve and update the database. In contrast, the flexible transaction model is more suitable for integration rules since it allows unilateral commit of subtransactions. This research has developed techniques for the use of the flexible transaction model to support the nested execution of integration rules. In the scope of this research, we assume a failure semantics where individual rules might abort without affecting the triggering transaction. The design of the compensating mechanism of the IRules system is a research issue that is currently under investigation.

In the IRules environment, transaction entities are execution objects that encapsulate the transactional control for rules and application transactions. All transaction classes inherit from an abstract class IRulesTransaction. The superclass IRulesTransaction encapsulates the generic logic of transaction execution. There are four sub-classes of IRulesTransaction: ISLTransaction, TopLevelTransactionForEvent, TopLevelTransactionForMethod, and NestedTransaction. These four sub-classes are responsible for capturing the execution-time behavior of transaction processing under different circumstances within the IRules system. ISLTransaction implements the transactional behavior of an application transaction. The TopLevelTransactionForEvent is used to offer transactional context to handle responses to internal and external events. The TopLevelTransactionForMethod applies to the specific case of a decoupled coupling mode when the action of the rule invokes a method of an EJB component. The NestedTransaction class encapsulates the execution of a subtransaction created as the context of a nested rule. Since the processing of rules is wrapped by transactions, rule nesting behavior can be controlled by the execution of parent and child transactions. For example, suppose rule R_x triggers immediate synchronous rule R_y, and the transaction contexts of R_x and R_y are T_x and T_y, respectively. Since we want to let R_x suspend until R_y finishes (according to the immediate synchronous E-C coupling mode), we control this behavior by suspending T_x until T_y commits.

4.3 Transactional Support for Wrappers and Rule/Method Synchronization

In the immediate synchronous coupling mode, the triggering transaction suspends while the triggered rule executes as a subtransaction, so this coupling mode results in rule nesting of rules. In this research, the suspension of transactions in a distributed environment is supported by the design of the IRules wrapper.

Figure 8 shows the structure of an IRules wrapper. A black-box bean may have a specific method, such as $m_1(param_1,param_2)$. IRules builds a property bean for each black-box bean to store external relationships between distributed components, as well as extents, derived attributes, and stored attributes for each component. A property bean has methods to provide the above functionality. A detail description of the property wrapper can be found in [10].

There is a proxy bean in the IRules wrapper structure that interfaces with clients. A proxy bean is responsible for generating method events, passing transaction contexts, and handling the suspension of current execution. A proxy bean has the same methods as the black-box bean, as well as a corresponding method for every method provided by the property bean and the black-box bean. For example, as shown in Figure 8, the proxy bean has a method $m_1(param_1,param_2)$ that is the same as the black-box bean. So any client that is unaware of the IRules system can still use the black-box API to access the purchased component. The proxy bean also has the $m_1(param_1,param_2,transactionId)$ method, which has the same name as the corresponding method in the black-box bean. In addition to the method parameters of the black-box bean, every method in the proxy bean has a parameter named transactionId. The transactionId parameter is used to pass transaction contexts during execution time in the IRules system. Similarly, the proxy bean has the $m_2(para_1,transactionId)$ method responding to $m_2(para_1)$ of the property bean to pass transaction contexts. When there is method invocation from clients to a proxy bean, the proxy bean will delegate the invocation to the corresponding method of the property bean or the black-box bean.

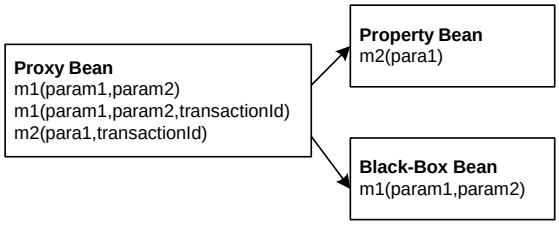


Figure 8: IRrules Wrapper Structure for EJB Components

A proxy bean has a three-step logic for event generation. The first step is to generate before method events, if any such events exist. When there is a method call to the proxy bean, the proxy bean will contact the metadata manager to determine whether this method call can generate a before method event. If it can, the proxy bean generates a method event through the Java Message Service (JMS) [42]. Then the proxy bean will try to read a semaphore object from the synchronization space in JavaSpaces [43]. JavaSpaces is a Jini Service that supports the storage and retrieval of objects in a distributed environment.

The blocking call mechanism of JavaSpaces is used to synchronize the execution of a transaction and its triggered immediate rules, releasing the suspension of the transaction upon the completion of the rule processing. Initially, the semaphore object does not exist. Since the read operation to JavaSpaces is blocking, the current transaction suspends at the proxy bean. During this suspension period, the event is propagated to the rule manager and the rule manager begins to process any rules triggered by the event. After processing rules triggered by a before method event, the rule manager will put the semaphore object into the synchronization space. Once the semaphore object is in the synchronization space, the proxy bean can successfully read the object to release the suspension of the wrapper.

The second step of the logic of the proxy bean is to call the property layer or the black-box bean to execute the method call. The third step is to generate after method events so that rules triggered after the execution of the method can be executed. The logic of generating an after event and the suspension for immediate synchronous rules is the same as in the first step. A more detailed description of the synchronization algorithm appears in [10].

5 Integration Rule Processing Algorithm

The rule processing algorithm is the logical circuit through which integration rules are processed. The algorithm instructs the rule manager in the processing of rules at execution time, depending on the transactional framework described in Section 4 for interaction with EJB components and coordination of rule execution with method execution. In Section 5.1, we specify the behavior of integration rule coupling modes in the context of cycles and levels of rule execution. In Section 5.2, we present the logic of the rule processing algorithm. A specific example of rule execution in the IRules environment is presented in Section 5.3. A brief summary of a performance analysis of the IRules environment appears in Section 5.4.

5.1 Specification of Coupling Mode Behavior

The Integration Rule Processing (IRP) algorithm is based on the algorithm of the ADOOD RANCH project [38], using *cycles* to control the nested execution of active rules. This research has re-designed the rule execution algorithm for a distributed environment, fully supporting the IRules coupling modes and transaction processing model.

Within the IRules environment, integration rules are processed according to coupling modes. A rule with an immediate E-C mode (either immediate synchronous mode or immediate asynchronous mode) is scheduled to execute as soon as it is triggered, while a rule with a deferred E-C mode is added to the deferred rule list that will be scheduled to execute at the commit time of the top-level transaction. A decoupled rule is executed immediately in a new top-level transaction, while the transaction of the triggering event execution resumes.

As shown in Figure 9, rule execution occurs in a coordinate system in two dimensions: the *Cycle* dimension and *Level* dimension. *Cycle* represents the logic of deferred rule processing, while *Level* represents the logic of nested rule execution. In Figure 9, dashed arrows represent immediate rule triggering in *Levels*, while solid arrows represent deferred rule scheduling in *Cycles*.

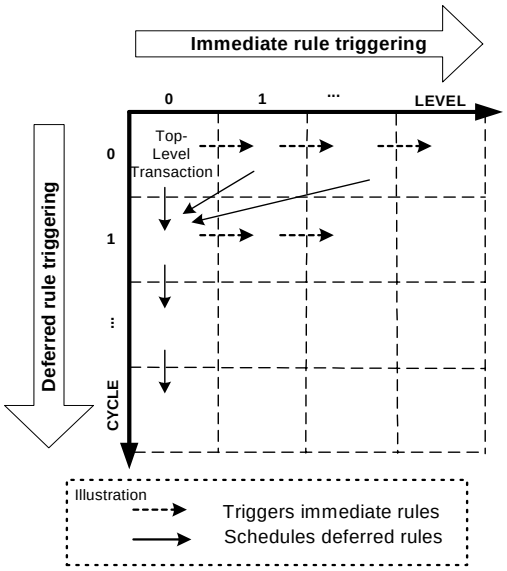


Figure 9: Cycles and Levels of Rule Execution

Each top-level transaction and its subtransactions are represented as a coordinate system, formed as a Cartesian product of *Cycle* and *Level*. A top-level transaction t^k, as the root of a transaction, is executed at $Cycle^k_0$ and $Level^k_0$ in coordinate system G^k. If an event in ($Cycle^k_0$, $Level^k_j$) triggers an immediate rule, the rule will be executed in the same cycle but in $Level_{j+1}$ as a new subtransaction. As an example, if an event e_1 in ($Cycle^k_0$, $Level^k_0$) triggers immediate rules r_1, r_2, and r_3, then r_1, r_2, and r_3 execute at ($Cycle^k_0$, $Level^k_1$). Within the same level, rules r_1, r_2, r_3 can execute sequentially or concurrently. The algorithm for determining sequential or concurrent rule execution is presented in [44].

If an operation of a top-level transaction ($Cycle^k_0$, $Level^k_0$) triggers a deferred rule, the rule will be scheduled to execute at the end of the top-level transaction in ($Cycle^k_1$, $Level^k_0$). In general, if an event in $Cycle^k_i$ triggers a deferred rule, the rule will be executed in $Cycle^k_{i+1}$. If there is more than one deferred rule at the end of the transaction, the rules are executed in sequence. If an event in $Level^k_j$ triggers a deferred rule, the rule will always be executed in $Level^k_0$ of the next cycle.

When an event in any ($Cycle^k_i$, $Level^k_j$) of a coordinate system G^k triggers a rule that contains a

decoupled action, the decoupled action is executed as a new top-level transaction at ($Cycle^n_0$, $Level^n_0$) of a new coordinate system G^n, where G^n and G^k are two distinct coordinate systems for rule execution.

Additional execution procedures apply for the execution of immediate asynchronous rules. If an event in ($Cycle^k_i$, $Level^k_j$) triggers a rule with an immediate asynchronous E-C mode, the rule will be executed in $Level^k_{j+1}$ without suspending the execution of the operations in $Level^k_j$. A *synchronization point* that requires the commit of asynchronous rule execution exists at this point. The *synchronization point* is after the last operation of $Level^k_j$ and before the commitment of a transaction in $Level^k_j$, which is called the *end-Proc* stage. This point allows the maximum time interval for the execution of the asynchronous rules without delaying the processing of the triggering transaction. In the *end-Proc* state of $Level^k_j$ within $Cycle^k_i$, all of the immediate asynchronous rules that executed at ($Cycle^k_i$, $Level^k_{j+1}$) are required to commit for the triggering transaction to continue. At the end of a top-level transaction, after all of the asynchronous rules commit, deferred rules can be processed.

The above presentation of IRP describes rule execution in a two dimensional coordinate system, focusing on the logic of rule execution. At run time, rules are executed in a distributed environment, which is related to a third dimension – *Location* of the objects accessed by a rule. An event is generated from one location, while the data accessed by a triggered rule can exist in multiple locations. The value of the *Location* depends on which software components are involved in the condition and action part of the rule. The IRP algorithm instructs the rule manager to invoke the IRules object manager to locate the position of a component. In a more complicated case, the objects accessed by a rule can require the use of multiple locations for evaluating the condition and performing the action of the rule.

5.2 Execution Logic of the IRP Algorithm

The IRP algorithm is the core of the rule manager. IRP instructs and regulates the execution behavior of the rule manager for the processing of application transactions and integration rules. In this section, the logic of the IRP algorithm is presented using Unified Modelling Language (UML) activity diagrams [9].

When a user makes a request to the rule manager to process an application transaction, the rule manager will start an application transaction processor to process the request. The processing logic is illustrated in Figure 10 for the PROCESS TOP-LEVEL TRANSACTION module. There are two sub-component modules of the processing: EXECUTE AN APPLICATION TRANSACTION and PROCESS DEFERRED RULES, which are detailed in Figures 11 and 13, respectively. As shown in Figure 10, after EXECUTE AN APPLICATION TRANSACTION, the execution arrives at the pre-commit state, which is the time for deferred rule processing. Then PROCESS DEFERRED RULES is executed and the transaction commits.

The EXECUTE AN APPLICATION TRANSACTION module is presented in Figure 11. Before execution of the application transaction, the algorithm checks for the existence of a before application transaction event. If a before event is raised, rules triggered by the before event are processed. Next, the algorithm will execute the application transaction. Since an application transaction consists of a set of operations, the algorithm calls the EXECUTE OPERATIONS module to execute all the operations of the application transaction. After all operations of the application transaction have been executed, the algorithm checks for an after application transaction event. If an after event exists, rules triggered by the event are executed according to different coupling modes. The EXECUTE AN APPLICATION TRANSACTION module uses the same algorithm as the EXECUTE OPERATIONS module (Figure 12) with respect to rule processing according to different coupling modes. The following paragraph provides an explanation of rule processing for different coupling modes in the context of the EXECUTE OPERATIONS module.

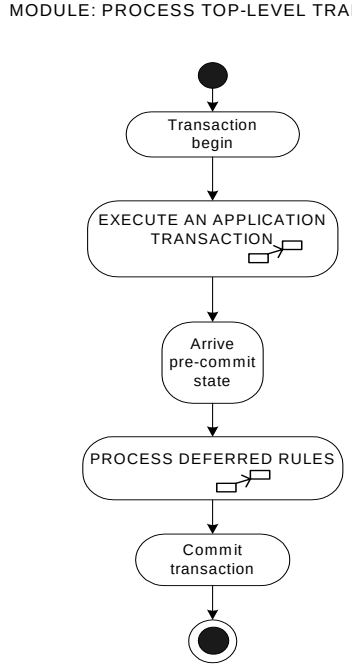


Figure 10: Process Top-Level Transaction

In Figure 12, the EXECUTE OPERATIONS module presents the logic of executing a sequence of operations for a transaction by iterating through all operations. Before execution of any operation, the algorithm will check for the existence of a before method event. If a before event has been raised, all rules triggered by this event will be obtained. The current transaction is suspended until the completion

of all triggered rules. The "*[For each rule]" notation indicates that the PROCESS A RULE module will be started for each rule. The PROCESS A RULE module is illustrated in Figure 14. These rules can be executed sequentially or concurrently.

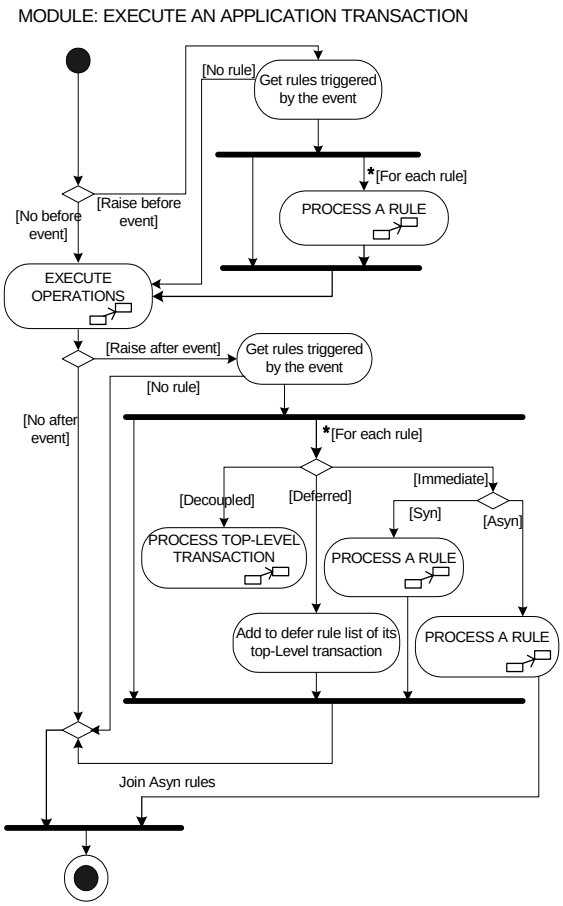


Figure 11: Execute An Application Transaction

Recall that only immediate synchronous rules are allowed for before events, so those rules will be processed immediately. After the rules with a before modifier are processed, the operation is executed as shown in Figure 12. The algorithm in Figure 12 checks for method events raised after the execution of the operation. Rules triggered by the after method event are obtained. If a rule is decoupled, a new top-level transaction is started that follows the logic of the EXECUTE TOP-LEVEL TRANSACTION module. If a rule is deferred, the rule is added to the deferred list of its top-level transaction. If the rule is immediate, the rule will be started immediately as a subtransaction by invoking the PROCESS A RULE module. In the case of an immediate synchronous rule, the current transaction cannot continue until the subtransaction joins the current transaction. In contrast, the current transaction can continue execution in parallel with the execution of its immediate asynchronous rules. At the *end-proc* state, the current transaction will suspend until all immediate asynchronous rules finish execution and join the current transaction.

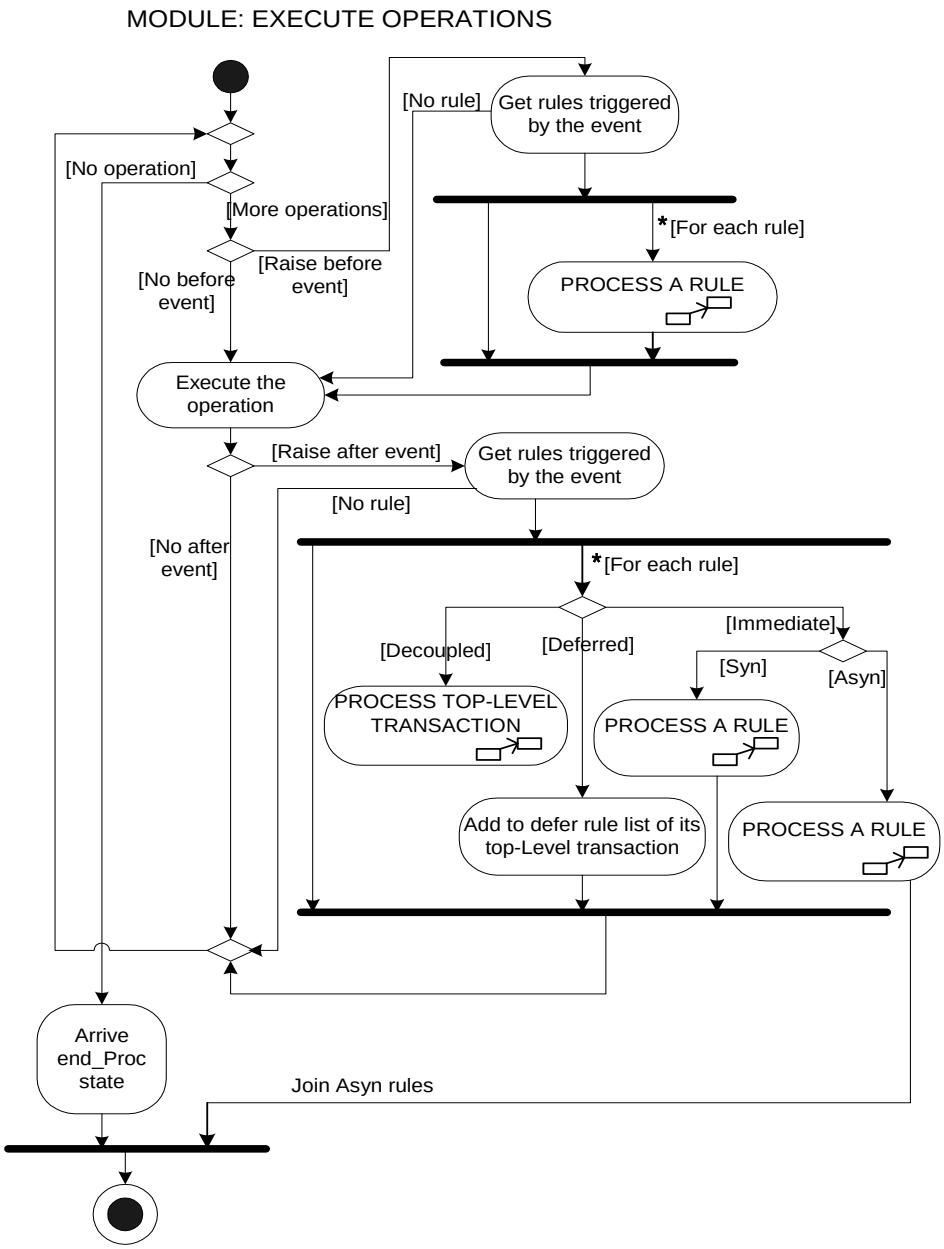


Figure 12: Execute Operations

The PROCESS DEFERRED RULES module is presented in Figure 13. Recall from Section 5.1 that deferred rules are executed in cycles. For each cycle, deferred rules are executed in sequence as described in the PROCESS A RULE module of Figure 14. A subtransaction will be created as the child of the triggering transaction. For an EA rule, the action is executed immediately. The execution of an action occurs in the EXECUTE ACTION module. For an ECA rule, the condition is evaluated first. If the C-A coupling mode is immediate synchronous, the action will be executed immediately. In the case of a decoupled C-A mode, a new top-level transaction is started immediately.

Recall that before calling the PROCESS A RULE module, the rule was already scheduled according to the E-C coupling mode for an ECA rule and the E-A coupling mode for an EA rule. So in the PROCESS A RULE module, the condition of an ECA rule and the action of an EA rule are always executed immediately.

Figure 15 illustrates the logic of the EXECUTE ACTION module. The execution of an action invokes the EXECUTE OPERATIONS module when the action is in the format of a method call. If the action is in the format of an application transaction, the algorithm will call the EXECUTE AN APPLICATION TRANSACTION module.

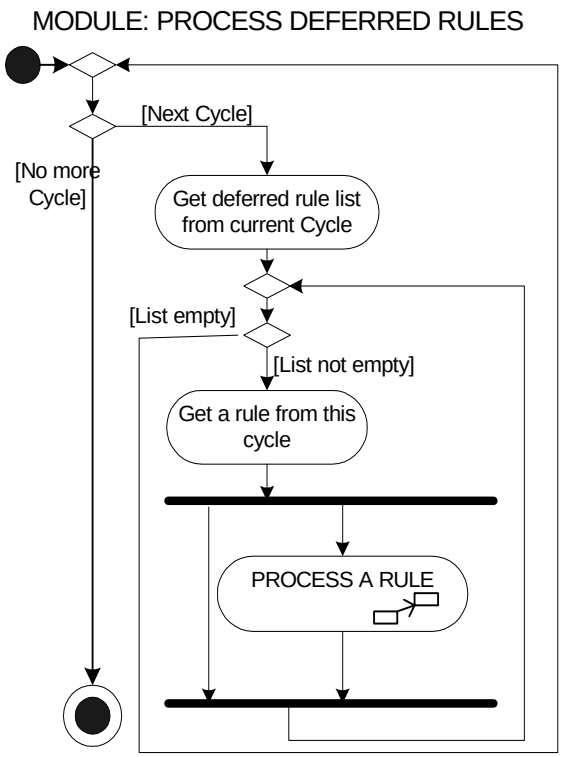


Figure 13: Process Deferred Rules

The algorithm has so far illustrated the rule processing logic for user requests as a top-level transaction. Recall that the other architectural component that can cause the rule manager to start top-level transaction processing is the event handler. For any internal or external event pushed by the event handler, the rule manager will handle the event according to the logic in Figure 16. The rule manager gets rules triggered by the event, and then processes each rule according to different coupling modes. Recall that no rule with a before modifier can be raised by an internal or an external event, because it is impossible for an active system to control when an internal or an external event occurs. Similar to the processing of a top-level application transaction, once the processing arrives at the *end-proc* stage, asynchronous rules must join the triggering transaction. After the *pre-commit* state, all deferred rules are processed.

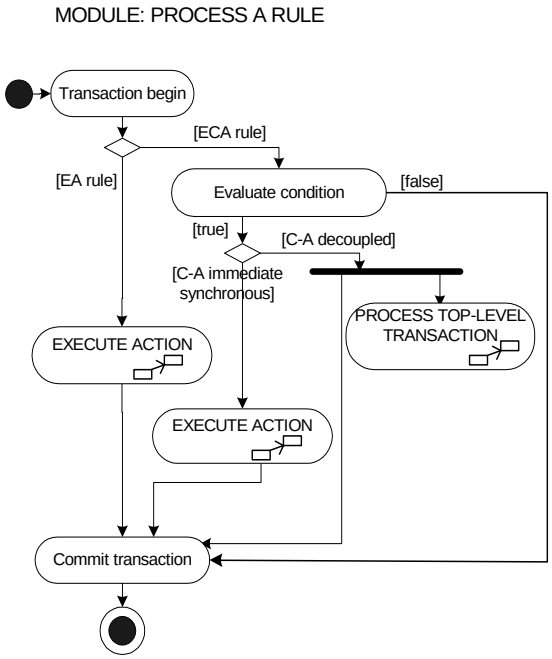


Figure 14: Process A Rule

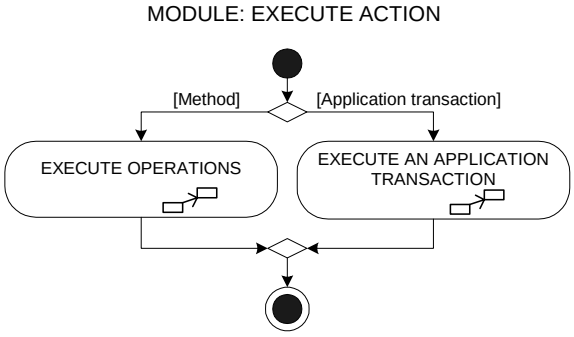


Figure 15: Execute Action

5.3 Execution Scenario of an Investment Application

To illustrate the rule processing algorithm, this section presents an execution scenario for selling stocks using the investment example presented in Section 3. A preliminary version of the scenario from Figures 17-19 appears in [6] without the notion of cycles and levels.

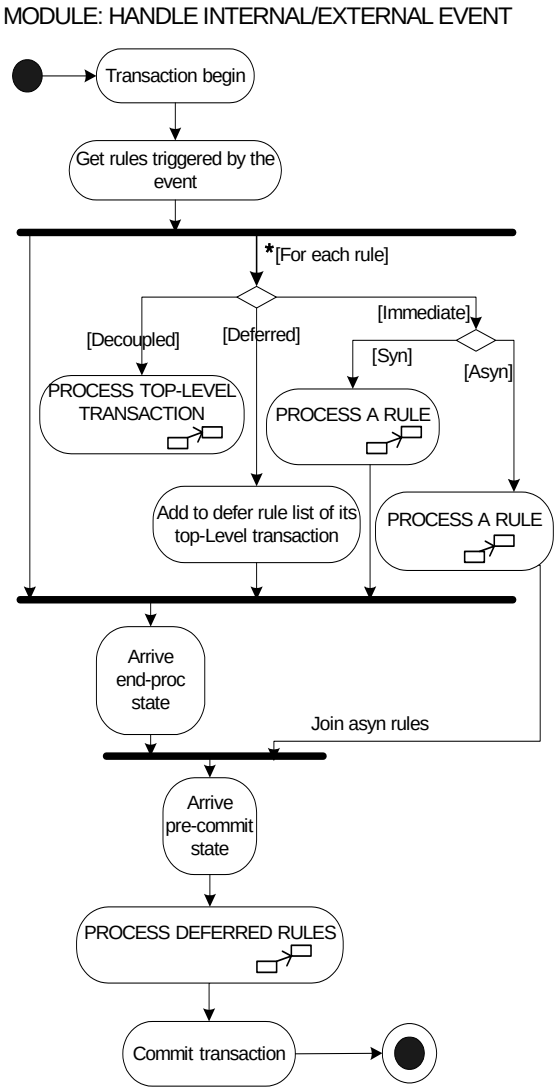


Figure 16: Handle internal/external event

As shown in Figure 17, the application transaction clientWantsToSellStock is the request from a user to perform the function of placing an order to sell a stock. The transaction creates an order to sell a stock at a desired price, and then prints a report. An event is generated after this application transaction, triggering the integration rule clientWantsToSellStockRule in Figure 18. The integration rule examines the desired prices for the stock to be sold and compares it with the current price, selling the stock if the condition is satisfied. The action part of the rule executes the sellStockOnNewPO application transaction in Figure 17, which raises two events. The first event (afterSellStock) is generated after the stock is sold but before the end of the transaction, allowing rules to be triggered in reaction to each sell operation. In particular, the scenario has an active rule stockBuyOnUpdateCash that allows a portfolio to exercise pending purchases when sufficient funds are available in the user's account. The afterSellStockOnNewPO event is signaled after the action of sellStockOnNewPO is complete to trigger a rule billingToAccountOnSell. This rule sends billing information to the user. Both of the stockBuyOnUpdateCash and billingToAccountOnSell rules are shown in Figure 18.

```
application transaction clientWantsToSellStock(String pnId,
String portfolioId, String stockId, int numOfShares,float
desiredPrice, String action, Stock actUpon, Portfolio
orderedBy)
tcl newInstance
{
set pn [newInstance PendingOrder $pnId $portfolioId
$stockId $numOfShares $desiredPrice $action $actUpon
$orderedBy $irulesId];
printPendingOrderInfo $pn $irulesId;
}

application transaction sellStockOnNewPO(String stockId,
float price, String portfolioId, int numOfShares,
StockBroker.PendingOrderComponent.PendingOrder pn)
tcl printSellInfo
{
set session [newInstance PortfolioSessionBean $irulesId];
$session sellStock $stockId $price $portfolioId
$numOfShares $irulesId;
$pn setStatus "executed" $irulesId;
printSellInfo $pn $irulesId;
}
```

Figure 17: Examples of Application Transactions for the Investment Scenario

Figure 19 illustrates the execution scenario that occurs as a result of the IRP algorithm. Figure 19 uses a notation that is based on UML activity diagrams. There are four transactions represented by four different swimlanes [45], one for each transaction context of the application transactions and rules. We use notation such as T1, e1, R11, as the abbreviated names of transactions, events, and rules, respectively.

When a user invokes the clientWantsToSellStock application transaction, the transaction manager creates a top-level transaction (T1) to process the application transaction. The top-level transaction T1 executes at ($Cycle^1_0$, $Level^1_0$) in coordinate system G^1. The clientWantsToSellStock application transaction generates an event named afterClientWantsToSellStock (e1). The event e1 triggers the rule clientWantsToSellStockRule (R11) presented in the second column of Figure 19. Because the E-C coupling mode of R11 is immediate synchronous, the condition of R11 is evaluated immediately. T1 suspends until R11 completes. The execution of immediate rule R11 is within the context of subtransaction T11. Because R11 is an immediate rule triggered by an event at ($Cycle^1_0$, $Level^1_0$) , the rule is executed at ($Cycle^1_0$, $Level^1_1$).

```
create rule clientWantsToSellStockRule
event     afterClientWantsToSellStock(pnId,
          portId, stockId, numOfShares,
```

```
            desPrice, pnaction, actUpon,
            orderedBy)
condition immediate
            when pnaction = "sell"
            define stockAndPendingOrder as
            select struct ( stk: s, newPo: pn )
            from s in stocks, pn in pendingOrders
            where pn.id=pnId and pn.actUpon=s
            and desPrice<=s.price
action      immediate
            from sp in stockAndPendingOrder
            do sellStockOnNewPO(stockId,
            sp.stk.price, portId, numOfShares,
            sp.newPo)

create rule stockBuyOnUpdateCash
event       afterSellStock(stockId, price, portfolioId,
            numOfShares)
condition asynchronous
            define portfolioOnUpdate as
            select p
            from p in portfolios
            where p.portfolioId=portfolioId and p.cash >
            p.buyThreshold
action      decoupled
            from p in portfolioOnUpdate
            do buyStockOnUpdateCash(p)

create rule billingToAccountOnSell
event       afterSellStockOnNewPO(stockId, price,
            portfolioId, numOfShares, pn)
action      deferred
            from p in portfolios
            where p.portfolioId= portfolioId
            do setAccountBillingOnSell(stockId, price,
            portfolioId, numOfShares, p.accountId)
```

Figure 18: Examples of Integration Rules for the Investment Scenario

The condition of R11 is evaluated in the second column. If the condition evaluation returns a non-null structure containing stocks and pending orders, then the action of R11 is performed using the structure as input bindings. Because of the immediate synchronous C-A coupling mode, the action is executed immediately. The action part of R11 is wrapped in a subtransaction named sellStockOnNewPO (T11a), which has four operations. The second operation sellStock is a method that generates a method event afterSellStock (e2). The event e2 triggers the rule stockBuyOnUpdateCash (R111) in the third column. Since the E-C coupling mode of R111 is immediate asynchronous, the condition of R111 is evaluated immediately. Moreover, the condition evaluation of R111 is concurrent with the execution of the triggering transaction T11a. Because R111 is an immediate rule triggered by an event at ($Cycle^1_0$, $Level^1_1$), R111 is executed at ($Cycle^1_0$, $Level^1_2$).

If the condition evaluation of R111 returns a non-null set of portfolios, the action of R111 will be performed upon the set. Due to the decoupled C-A coupling mode, the action of R111 becomes a new top-level transaction named buyStockOnUpdateCash (T2) since the action is an application transaction. Since the C-A coupling mode is decoupled, the action part of R111 will be executed in a different coordinate execution system G^2 at ($Cycle^2_0$, $Level^2_0$).

Once T2 is started in the fourth column, T111 resumes and commits. As shown in the second column, when the set status and printInfo operations of T11a finish executing, T11a is at the end of execution. At this time T11a waits until all the triggered asynchronous rules join. In this example, T111 joins T11.

As shown in the second column, the completion of sellStockOnNewPO generates an event (e3) that triggers an EA Rule named billingToAccountOnSell (R112). Because the E-A coupling mode of R112 is deferred, R112 is scheduled to the end of the top-level transaction (T1). Subtransaction T11 finishes execution and commits. Because R112 is a deferred rule triggered by an event at $Cycle^1_0$, R112 will be executed at ($Cycle^1_1$, $Level^1_0$).

In the first column, the commit of T11 releases the suspension of T1. Just before T1 commits, deferred rule R112 is processed. After R112 finishes executing, T1 commits.

5.4 Performance Analysis of the IRules Environment

The IRules system is a Java implementation that uses the BEA Weblogic Server [46] to provide EJB components. The Jini distributed computing environment is used as the backbone of the system, with IRules architectural components implemented as Jini Services. Java Message Service (JMS) [42] provides asynchronous event notification for communication between the event-signaling components and the event-handling components. JavaSpaces [43] is used for the storage of metadata. The blocking call mechanism of JavaSpaces is also used in the synchronization space to synchronize the execution of a transaction and its triggered immediate rules, releasing the suspension of the transaction upon the completion of the rule processing [11].

We have evaluated the performance of the IRules environment using the OBJECTIVE benchmark [47] as the basis for the evaluation. The OBJECTIVE Benchmark was originally designed to identify bottlenecks and to evaluate the functionality of an active object-oriented database. The OBJECTIVE benchmark was adjusted and extended as part of this research to apply the benchmark to a distributed component integration environment. The full details of the performance analysis and how the benchmark was adapted to the IRules environment is beyond the scope

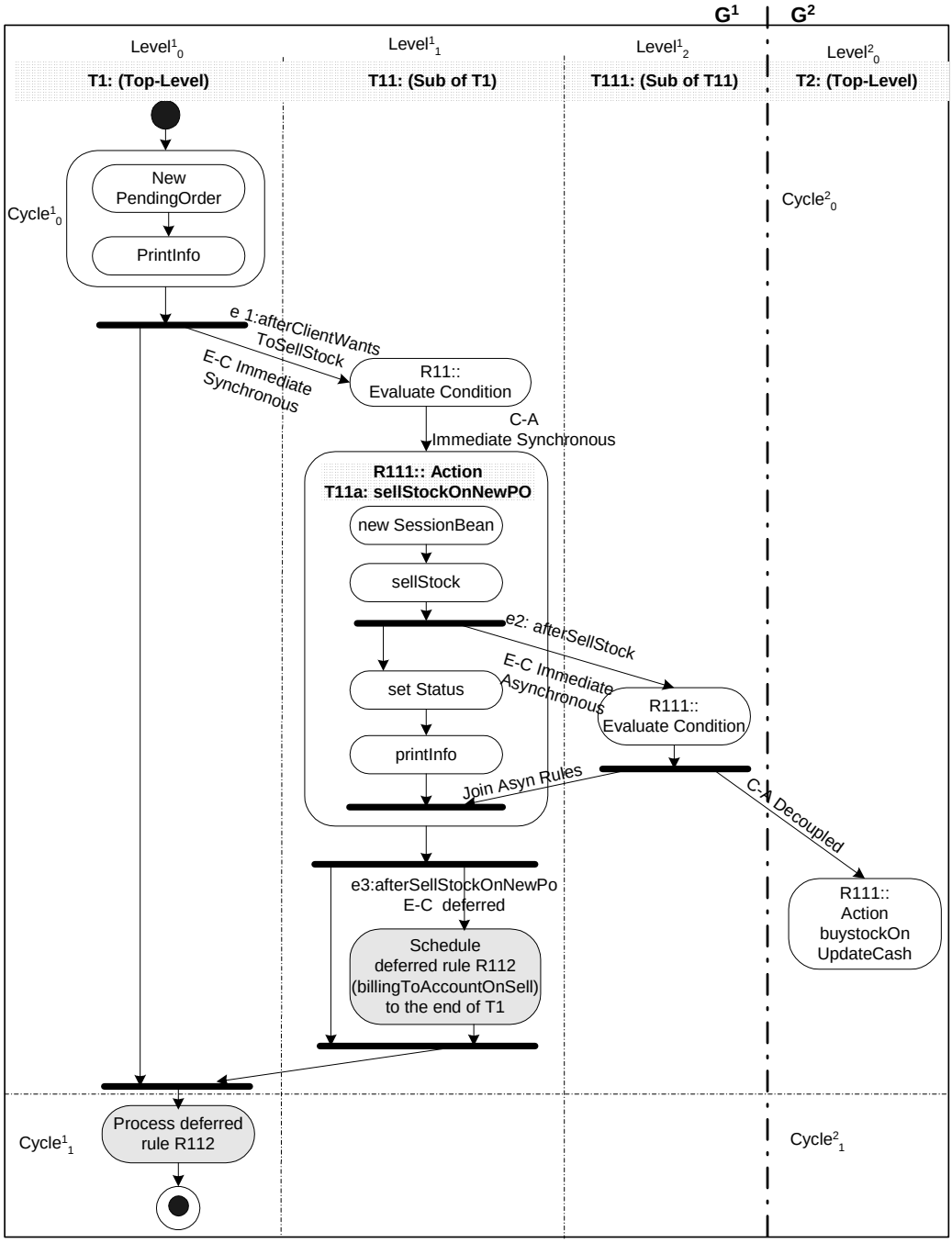


Figure 19: Execution Scenario of the Investment Application

of this paper and can be found in [44, 48]. The evaluation was conducted within the IRules environment and has not been applied to any industrial environment.

As a brief summary of the performance evaluation process, the system was implemented and evaluated using four Windows NT 4.0 computers. The metadata and object manager illustrated in Figure 6 were co-located on one physical machine, while in two different Java virtual machines. The rule manager, event handler, and the Weblogic EJB server, also illustrated in Figure 6, were each physically located on one machines. The evaluation was conducted using four parameters of configuration: 1) the number of events, 2) the number of rules, 3) the number of application transactions, and 4) the number of component instances. The primary focus of the evaluation was on four different aspects of the execution environment: 1) three phases of active behavior (event detection, rule retrieval, and rule execution), 2) performance of different coupling modes, 3) performance of the rule processor under a heavy event load, and 4) the time for event detection

for the different types of IRules events. The primary results of the evaluation indicate that:

1) The decoupled and immediate asynchronous modes provide the best performance since they allow concurrent execution. The deferred mode is the slowest due to the need to schedule rules for execution at the end of the top-level transaction.
2) Execution time is somewhat affected by a large number of rules and transactions due to the larger amount of metadata that must be searched during rule and transaction retrieval.
3) A heavy event load can cause the rule processor to be interrupted to queue events, thus slowing down the performance of the rule processor, but the performance eventually levels off to a consistent execution speed regardless of the event arrival load.
4) Access to EJB components is the primary point of slow performance, affecting the time for method execution as well as the time for method event generation. A future improvement of the IRules system should involve re-design of the wrapper structure to reduce the EJB component layers, thus reducing the time associated with method invocation.

6 Summary and Future Directions

This paper has presented the integration rule processing algorithm of the IRules environment, with supporting descriptions of the rule execution model and transaction model. The IRP algorithm illustrates an approach for active rule processing in the context of distributed component integration, where events are used to trigger rules that invoke application transactions and methods on components. The IRP algorithm is presented in a form that can be reused in other environments for a rule-based approach to integration logic, defining the manner in which immediate coupling modes can be used together with nested rule execution in a distributed environment. The IRules integration system allows application integrators to specify the integration logic in a declarative fashion, which does not require an integrator's low-level knowledge of programming and transaction management. Integrators can focus on mediating the interaction between components, rather than the technical details of event handling and transaction processing.

It has been a challenging effort to develop a distributed rule and transaction processing environment such as IRules, since it involves the combination of issues such as component autonomy, rule distribution, cascaded rule triggering, and distributed synchronization. The implementation of the execution environment presented in the paper has been completed. One future direction is to expand the environment to support multiple component models. The transaction model also needs further investigation to address failure in the execution process, especially when global transactions execute over different component models with heterogeneous transaction processing semantics. These future research directions will be explored in the context of Grid services for virtual organizations, where a Grid service provides a service-oriented view of a component [49, 50] and the Grid environment forms the foundation of the underlying architecture.

References

[1] Object Management Group: The Common Object Request Broker, Architecture and Specification. (1999) John Wiley Publishing.

[2] Enterprise Java Beans Specification (2000) Sun Microsystems, version 2.0.

[3] S. D. Urban, S. W. Dietrich, Y. Na, Y. Jin, and A. Sundermier (2001) The IRules Project: Using Active Rules for the Integration of Distributed Software Components, Proc. of the 9th IFIP 2.6 Working Conf. on Database Semantics: Semantic Issues in E-Commerce System, Hong Kong, April 2001, pp. 265-286.

[4] S. D. Urban, S. W. Dietrich, Y. Jin, S. Kambhampati, and Y. Na (2002) Distributed Software Component Integration: A Framework for a Rule-Based Approach, Handbook of Electronic Commerce in Business and Society, Watson, R., Lowery, P. and Cherrington, J. Ed.

[5] S. W. Dietrich, S. D. Urban, A. Sundermier, Y. Na, Y. Jin, and S. Kambhampati (2001) A Language and Framework for Supporting an Active Approach to Component-Based Software Integration, Informatica, Vol. 25, No. 4, pp. 443-454.

[6] Y. Jin, S. D. Urban, S. W. Dietrich, and A. Sundermier (2002) An Execution and Transaction Model for Active, Rule-Based Component Integration Middleware, Proceedings of the Engineering and Deployment of Cooperative System, Beijing, China, pp. 403-417.

[7] N. W. Paton, O. Diaz (1999) Active Database Systems, ACM Computing Surveys, Vol. 31, No. 1, pp. 3-27.

[8] J. Widom and S. Ceri (Eds.) (1996) *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Publisher.

[9] Unified Modeling Language (UML) Specification, version 2.0. http://www.uml.org/#UML2.0

[10] R. Patil (2003) A Framework Supporting an Active Approach to Component-Based Software Integration, M.S. Thesis, Arizona State University, Department of Computer Science and Engineering.

[11] S. Urban, S. Kambhampati, S. Dietrich, Y. Jin, and A. Sundermier (2004) An Event Processing System for Rule-Based Component Integration, Proceedings of the International Conference on Enterprise Information Systems, Porto, Portugal, pp. 312-319.

[12] M. Stonebraker, E. N. Hanson, and S. Potamianos (1998) The POSTGRES Rule Manager, IEEE Transactions on Software Engineering, Vol. 14, No. 7, pp. 897-907.

[13] J. Widom (1992) The Starburst Rule System: Language Design, Implementation and Application, IEEE Data Engineering Bulletin, December, pp. 15-18.

[14] U. Dayal, B. Blaustein, A. Buchmann, and S. Chakravarthy (1998) The HiPAC Project: Combining Active Databases and Timing Constraints, *ACM SIGMOD Record*, Vol. 17, No. 1, pp. 51-70.

[15] S. Gatziu, K. R. Dittrich (1992) SAMOS: An Active Objective-Oriented Database System, Data Engineering bulletin, pp. 23-26.

[16] S. W. Dietrich, S. D. Urban, J. V. Harrison and A. Karadimce (1992) A DOOD RANCH at ASU: Integrating Active, Deductive and Object-Oriented Databases, IEEE Data Engineering Bulletin: Special Issue on Active Database Systems, Vol. 15, No. 1-4, pp. 40-43.

[17] H. Branding, A. P. Buchmann, T. Kudrass, and J. Zimmermann (1993) Rules in an Open System: The REACH Rule System, Rules in Database Systems, pp. 111-126.

[18] P. Gulutzan and T. Pelzer (1999) SQL-99 Complete Really, Miller Freeman Publishing.

[19] P. Fraternali and L. Tanca (1995) A Structured Approach for the Definition of the Semantics of Active Databases, ACM Transactions on Database Systems, Vol. 20, No. 4.

[20] L. B. Warshaw (2001) Facilitating Hard Active Database Applications, Ph.D. Dissertation, The University of Texas at Austin, Department of Computer Science.

[21] S. Chakravarthy, and R. Le (1998) ECA Rule Support for Distributed Heterogeneous Environments, International Conference on Data Engineering, pp. 601.

[22] Object Management Group (OMG) Interface Definition Language (IDL), International Organization for Standardization (ISO) International Standard, number 14750. http://www.omg.org/gettingstarted/omg_idl.htm

[23] A. Koschel, and P. C. Lockemann (1998) Distributed Events in Active Database Systems - Letting the Genie out of the Bottle, Journal of Data and Knowledge Engineering, Vol. 25, pp. 11-28

[24] H. Fritschi, S. Gatziu, K. and R. Dittrich (1998) FRAMBOISE – an Approach to Framework-Based Active Database Management System Construction, Proceedings of the 7th ACM International Conference on Information and Knowledge management, pp. 364-370.

[25] M. Cilia, C. Bornhovd, and A. Buchmann (2001) Moving Active Functionality from Centralized to Open Distributed Heterogeneous Environments, Proceedings of 9th International Conference on Cooperative Information Systems (CoopIS'01), Trento, Italy, pp. 195-210.

[26] C. Liebig, M. Malva, A. Buchmann (2000) Integrating Notfications and Transactions: Concepts and X2TS Prototype, Proceedings of the 2nd International Workshop on Engineering Distributed Objects, University of California, Davis, USA, pp.194-214.

[27] W. K. Edwards (2000) Core Jini, Prentice-Hall PTR, Second Edition.

[28] C. Bussler, and S. Jablonski (1994) Implementing Agent Coordination For Workflow Management Systems Using Active Database Systems, The International Workshop On Active Database Systems, Houston TX, pp. 53-59.

[29] F. Casati, S.Ceri, B. Pernici, and G. Pozzi (1996) Deriving Active Rules for Workflow Enactment, DEXA, Switzerland, pp. 94-115.

[30] K. Karlapalem and P. C. K. Hung (1998) Security Enforcement in Activity Management Systems, Workflow Management Systems and Interoperability, Ed. Dogac, etc., Springer-Verlag publisher, pp. 165-193.

[31] G. Kappel, and W. Retschitzegger (1998) The TriGS Active Object-Oriented Database System - An Overview," SIGMOD Record, 27(3), pp.36-41.

[32] Workflow on Intelligent Distributed database Environment. http://dis.sema.es/projects/WIDE/

[33] B. Benatallah, M. Dumas, Q. Sheng, and H. Ngu (2002) Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, Proceedings of the 18th International. Conference on Data Engineering, San Jose, CA, pp. 297-308.

[34] R. Peri (2002) Compilation of the Integration Rule Language, M.C.S. Report, Arizona State University, Department of Computer Science and Engineering.

[35] K. Marimuthu (2003) An Object-Oriented Query Processor Based on an Extended Monoid Algebra, M.S. Thesis, Arizona State University, Department of Computer Science and Engineering.

[36] M. DeJong, C. Laird, "TCL+Java = A Match Made for Scripting," http://www.sunworld.com/sunworldonline/swol-11-jacl.html.

[37] J. Ousterhout (1994) TCL and the TK Toolkit, Addison-Wesley Publishing.

[38] T. Abdellatif (1999) An Architecture for Active Database Systems Supporting Static and Dynamic Analysis of Active Rules Through Evolving Database States, Ph.D. Dissertation, Arizona State University, Department of Computer Science and Engineering.

[39] J. Gray and A. Reuter (1994) Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers.

[40] S. Jajodia and L. Kerschberg (1997) Advanced Transaction Models and Architectures, Kluwer Academic Publishers.

[41] M. T. Ozsu and P. Valduriez (1999) Principles of Distributed Database Systems, Prentice Hall Publishing.
[42] JMS 2002. Java Messaging Service. Version 1.1. http://java.sun.com/products/jms/docs.html
[43] E. Freeman, S. Hupfer, K. Arnold (1999) JavaSpace: Principles, Patterns, and Practice, Addison-Wesley Publisher.
[44] Y. Jin (2004) An Architecture and Execution Model for Component Integration Rules, Ph.D. Dissertation, Arizona State University, Department of Computer Science and Engineering.
[45] G. Booch, J. Rumbaugh, I. Jacobson (1999) The Unified Modeling Language User Guide. Addison-Wesley Publisher.
[46] BEA Systems Weblogic Server (2003). http://www.bea.com
[47] U. Cetintemel (1995) OBJECTIVE: A Benchmark for Object-Oriented Active Database Systems, M.S. Thesis, Bikent University, Turkey.
[48] Y. Jin, S. D. Urban, D. W. Dietrich (2005) Extending the OBJECTIVE Benchmark for Evaluation of Active Rules in a Distributed Component Integration Environment, submitted for journal publication, 2005.
[49] S. D. Urban, V. Kumar, and S. W. Dietrich (2005) A Prototype for Integration of Web Services into the IRules Approach to Component Integration, Proceedings of the International Conference on Enterprise Information Systems, Miami, FL., pp.3-10.
[50] H. Ma, S. D. Urban, Y. Xiao, and S. W. Dietrich (2005) GridPML: A Process Modeling Language and History Capture Systems for Grid Service Composition, Proceedings of the International Conference on e-Business Engineering, Beijing, China.

A PC-based Decision Support System for Optimal Cutting of Logs in Veneers Production

Anton Čižman and Marko Urh
University of Maribor, Faculty of Organizational Sciences
Kranj, Kidričeva cesta 55a
E-mail: anton.cizman@fov.uni-mb.si, marko.urh@fov.uni-mb.si

Keywords: decision support systems, cutting, production, operations management, linear programming

Received: July 14, 2005

We report on a user-friendly decision support system (DSS) for solving specific cutting stock problem from a smaller wood-processing company. A prototype DSS is developed for use in veneers production and is designed to aid mangers in create or improving existing tailoring of logs using their experience and preference s. The results of testing a typical cutting-stock problem are shown to point out how such DSS, which utilizes linear and mixed-integer programming, can reduce inventory costs and improves the exploitation of input material.

Povzetek: V prispevku je prikazan odločevalni informacijski sistem (OIS) za reševanje problema razreza hlodov na podhlode v proizvodnji furnirja.

1 Introduction

Decision Support Systems (DSS) have emerged as the computer-based system to assist decision makers address semi structured problems by allowing them to access and use data and analytic models. Such systems have the following characteristics: they are interactive computer-based systems; they are aimed at semi-structured problems. They utilize models with internal and external databases, and they emphasize flexibility, effectiveness, and adaptability. These characteristics have guided much of the research in the DSS area, but the potential benefits of DSS in the business environment have not been fully realized [1, 2, 3].

Although the definition of the DSS concept has been elusive, the field has flourished with the development of computer technology. Keen [8] reviewed the decade of DSS development and concluded that there is a need for balance between each of the three DSS elements: decision, support and systems. To achieve the mission of DSS - helping people make better decisions - Keen stressed the need for an active supporting role for decisions that really matter. This paper focuses on the decision component of the DSS.

PC technologies are becoming accepted and incorporated into organizations and our personal lives. PC-based systems have the potential to improve both individual and organizational performance. As decision makers recognize the potential benefits, many companies are investing in information technology. PC-based systems have been generally hailed as a revolution that will change the nature of professional work and transform the way people work. It is expected that almost all knowledgeable workers are likely to have their own PC to perform both stand-alone tasks and network services.

Despite the proliferation of microprocessor-based systems, the potential benefits of these systems, as aids to managerial decision making, may not be fully realized due to poor design and low acceptance by users. It is recognized that individuals are sometimes unwilling to use these systems, even if the system may increase their productivity. While some of these systems may have an impact on individuals and organizations, the adoption and acceptance of these systems among decision-makers has been limited. This may be due to the inflexibility in the systems as well as their narrow design. Therefore, it is important to understand the environment of the decision makers and the type of support they need in order to make effective decisions, and to examine the models appropriate for addressing their problems [1, 5].

In this paper we describe a PC-based DSS which addresses the optimal cutting of logs in the manufacturing of veneers. The development of DSS is proposed on generalized prototyping approach [2, 3] that leads to more efficient and simpler use of such systems. The effectiveness of the DSS is shown by relatively simple example which utilizes the pattern-oriented LP-based methods for solving one-dimensional cutting stock problem. The results of testing show that the application of such managerial DSS enhances problem solving capability for achieving greater competitiveness of an organization.

In the first part of the paper, we present a description of the major operations in the production of veneers focusing on the process of efficient longitudinal tailoring of logs. The second part of the paper describes

design and development of a DSS for this environment. Finally, we illustrate how the DSS can be used in practice to improve existing intuitive tailoring of logs.

2 The application problem

The application problem that is considered in this case study was identified in a small-to-medium size Slovenian veneers factory. The factory belongs to a group of connected wood-processing enterprises that have successfully passed the process of economical restructuring in the past few years. The factory is producing steamed and sliced veneers from domestic and imported wood species. It has been operating since 1926 and is famous all over Europe particularly for its top-quality steamed beech veneers. On one side, this reputation is a result of a long tradition. On the other side, the management is continuously striving to improve the competitive position of the enterprise by gradual automation of production facilities as well as by introducing modern information systems and advanced information technology.

In this sense, a particular challenge for the manager of the company in recent years has been a better integration of production and business processes by means of various DSS. Such integration became a necessity due to a change in market conditions. In previous decades of successful operation for the largely unsaturated market, their production relied on the so-called "make-to-stock" strategy. With recent economy trends, such as: loss of the market share in the countries of the former Yugoslavia, privatization and restructuring of the Slovenian national economy, as well as further globalization of trade, the management has been forced to switch over to the strategy of "make to order". In such a situation, we would define the problem as follows.

While the partially intuitive planning, assisted with simple calculations, was enough to manage the previous strategy, the new production planning strategy requires more sophisticated information systems. Due to very tight resources in terms of available time, qualified people and investment money, the management currently cannot deal satisfactorily with these problems. In this situation, our research group decided to perform some exploratory work, with the aim to develop a suitable DSS for a specific Mathematical Programming (MP) application.

2.1 Analyzing the decision-making process

The essential operation to be discussed in this paper is longitudinal tailoring of logs. It is done by cutting the selected logs in the transversal direction to a pre-determined number of sub-logs. The tailoring plan is prepared in advance according to the actual purchasing orders, as it is explained later. The aim of this operation is to get proper initial length of sub-logs before slicing them to veneer sheets that have the same length as sub-logs. In terms of optimal wood exploitation, this operation has the greatest potential for achieving substantial cost savings. Therefore this operation was chosen to receive the main attention in the presented case study.

Primary analysis of the decision-making process applied to the management of the veneers production operations described in a paper [3] reveals a number of decision-making situations. One of the relevant problems in this regard, such as determination the optimal log-cutting strategy and the quantity of logs to be purchased, is related to optimal exploitation of input material (logs). This situation is shown in Fig. 1.

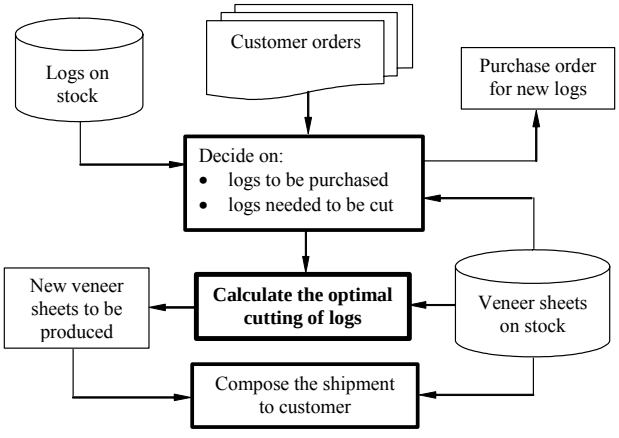


Fig. 1: The decision-making process in the production of veneers.

The discussion is focused to the indicated initial decision-making problem from Fig. 1, i.e. the optimal cutting of logs. The current decision-making model in the production of veneers is based on obtainable data samples about logs and related veneer sheets that have been produced in several periods (see Table 1).

Up to the present, the **longitudinal tailoring** was performed by an intuitive “rule of thumb”. Thereby, the

goal of the operations manager is to establish a "reasonable minimum" of the edge-waste remainders of logs which, unfortunately, does not take into account the end-waste remainders of logs. This simple approach to log cutting causes gradual accumulation of an undesired inventory of veneer sheets in the warehouse which significantly increases inventory costs. Thus the idea is to improve the existing decision-making process using scientific approach to minimize at the same time both, the edge waste as well as the end waste from cutting the logs.

Table 1: Typical data samples about the logs and the veneer sheets

	Log ID number	Length (cm)	Volume (m^3)		Log ID number	Length of sheets (cm)	Area (m^2)
Log 1	1234	563	1,19505	Packet 1	1234	220	13,024
Log 2	1535	657	2,047	Packet 2	1535	182	8,91072
Log 3	1623	446	0,741	Packet 3	1623	151	6,2816
Log 4	1854	721	1,904	Packet 4	1854	193	12,4678
Log 5	1910	512	0,701	Packet 5	1910	112	2,0608

2.2 The problem-solving paradigm

The problem of optimal cutting of logs in veneer production is called *one-dimensional cutting stock problem* (1d-CSP). This problem consists of determining the smallest number of logs of length L_p, $(p = 1, 2, ..., s)$ that are available in sufficiently large quantity and have to be cut in order to satisfy the required number b_i sub-logs of lengths l_i ($i = 1, 2, ..., m$). The lengths of sub-logs l_i are determined by the required length of ordered veneer sheets. On the other hand, the required number of sub-logs (b_i) can be calculated from the required area and thickness of ordered veneer sheets taking into account the diameter and quality of logs to be cut. A combination of required sub-logs lengths in the log of length L_p is called cutting pattern. For each log length L_p, the number n_p ($p = 1, 2,..., s$) of all different cutting-patterns can be determined with regard to the particular sub-log length l_i. The element a_{ijp} of matrix A that describes each cutting pattern, represents the number of sub-logs of length l_i obtained in cutting pattern j related to log length L_p. The number of logs of length L_p to be cut according to cutting pattern j is denoted by a decision variable x_{jp}.

In this problem situation, the objective is to minimize the number of logs (x_{jp}) of length L_p, that have to be cut according to the j-th pattern. Thus, our 1d-CSP represents a pure integer programming (IP) problem that can be modeled as follows:

$$\min \sum_{p=1}^{s} \sum_{j=1}^{n_p} L_p x_{jp} \qquad (1)$$

subject to

$$\sum_{p=1}^{s} \sum_{j=1}^{n_p} a_{ijp} x_{jp} \geq b_i, \quad i = 1,...,m \qquad (2)$$

where

$$x_{jp} \geq 0 \text{ and integer}, \quad j = 1,...,n_p; \quad p = 1,..., s$$

For cutting pattern to be valid:

$$\sum_{i=1}^{m} a_{ijp} l_i \leq L_p \qquad (3)$$

$$a_{ijp} \geq 0 \text{ and integer}, \quad j = 1,...,n_p; \quad p = 1,..., s$$

The IP problem described above can be solved using different LP-based methods [3, 4, 9]. In practice, most IP problems can be solved by the technique of Branch-and-Bound (BB).

In actual situation, the ordered number of different veneer sheet lengths and the number of different log lengths that are available for cutting at the same time is usually small; that means: ($m < 5$ and $s < 3$). Therefore, the number of all possible log-cutting patterns that can be enumerated is not large. Because of this property, a normally good solution for many real log cutting problems can be obtained by using the BB method of selected DSS development software.

In the case of larger number of different veneer sheets lengths (large-scale 1d-CSP) the use of the BB method is impractical, because the number of all possible cutting patterns (columns) can be very large. As a result, the IP will be too large to even formulate and difficult or impossible to solve. To tackle this problem, the "*delayed column generation approach*" can also be included into DSS [6, 7].

2.3 Selecting the DSS development

2.3.1 Software

Criteria for the evaluation of DSS software have been dealt with by some authors (e.g. Buede, 1992). For the purpose of our selection, the process of setting up requirements for a proper software tool was considered from three different viewpoints: (1) broad viewpoint of the enterprise, (2) viewpoint of the OR expert (developer) and (3) pragmatic viewpoint of the user (i. e. manager). Results of preliminary selection

Table 2: List of commercial DSS software development tools selected as closer candidates

Name	Short description	Supplier
LINDO	linear programming (LP) solver	LINDO Systems
LINGO	linear, non-linear programming (NLP) and integer programming (IP) solver with specific mathematical language	
What's Best	spreadsheet (EXCEL) add-in for solving LP, NLP and IP problems	
ILOG	resource optimization suite (includes CPLEX)	ILOG, Inc.
MS EXCEL	standard solver, Frontline solvers for EXCEL for solving LP, IP and NLP problems	Microsoft Inc., Frontline Systems Inc.
CPLEX optimizer	general large-scale MP software and services for resource optimization	CPLEX division of ILOG, Inc.
SAS/OR module	software tools for MP, scheduling, decision analysis, project management, statistical analysis	SAS Institute, Inc.

based on these criteria, and at the same time taking into account local preferences, have shown that a couple of OR or general-purpose software candidates (see Table 2) are remarkable for further consideration.

If compared on the basis of requirements given previously, all commercial MP software systems given in Table 2 have favorable features as candidates to be selected for further development of a manager-friendly DSS in our specific circumstances. They appear to have an almost equal rank from the viewpoint of the OR expert. Therefore, the choice of the basic DSS development software may depend largely on the particular needs and circumstances of the enterprise where the new DSS will be used.

A general argument for SAS/OR software selection was the fact that the SAS System offers to the user enterprise a wider selection of software tools and technology, not only for MP [9, 10, 11, 12]. In the context of a production enterprise, such as that of veneers production, the software tools for forecasting (the SAS/ETS module), for statistical analysis (the SAS/STAT module) and quality control (the SAS/QC module) are of particular interest. In this way, the benefit of this selection for the company is to have a single software provider, the same source of technical support and a truly integrated information delivery system with similar user interfaces.

Another important argument for having selected the SAS System is the possibility of effective exploitation of modern technologies for effective development and implementation of end-user oriented DSS such as: Data Warehousing and On-Line Analytical Processing (OLAP), object-oriented programming concepts (the SAS/AF software) with powerful Screen Control Language (SCL), and interfaces that provide easily and timely access to data from SAS-format and outside databases (the SAS/ACCESS software).

The other two arguments that have led to the selection of the SAS/OR software are bounded to the circumstances that are important for the enterprises in Slovenia. In particular, the SAS System provides data-exchange interfaces for ERP systems used in a number of important Slovenian enterprises (specifically the SAP R/3). Finally, the local SAS office in the capital of Slovenia, Ljubljana, has a good reputation for their technical support of SAS products and training courses in the Slovenian language.

3 The system design, development and testing

A specific DSS was build to assist the operations manger during his tasks of log cutting in production of veneers, such that the system: (1) is based on the mental model of the user; (2) does not require any special MP knowledge for operating it; (3) can be used through simple point-and-click commands; (4) has simple data-input and data-output procedures; (5) can be operated from a standard personal computer running the Microsoft Windows operating system; (6) exploits as much as possible the features and functions of the selected SAS/OR software; (7) is flexible and easy to modify as well as to maintain. During the initial development steps of the reported DSS it was confirmed that the above requirements can be satisfied by the SAS software.

The DSS includes four components: the user-interface, the modeling base, the database and solution techniques. The integration of the four components of the DSS, i.e., the structure of our DSS, is presented in Fig. 2.

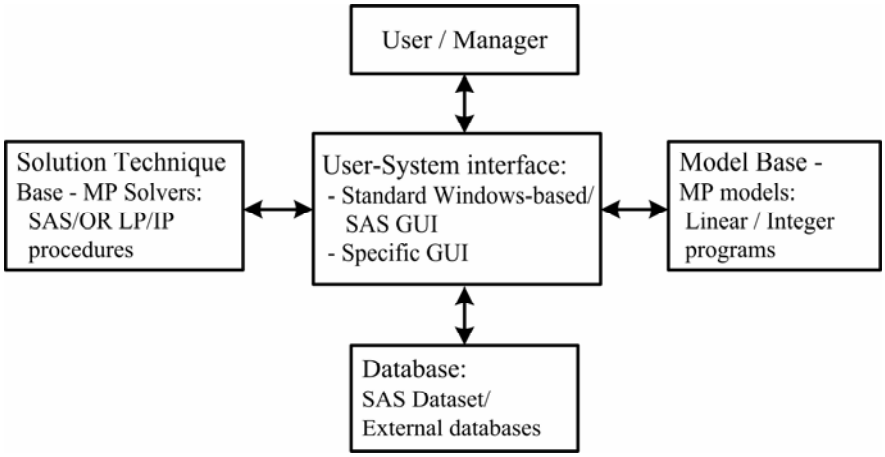


Fig. 2: The structure of a decision support system

However, the standard SAS/OR graphical user interface provides a specific programming language that was not considered appropriate for an operations manger. Therefore, development was focused around building a special window-based graphical user interface (GUI) allowing easier use of related solution technique. The graphical user interface was developed specifically for this application by means of the SAS/AF software. The application is started from standard SAS GUI screen by clicking to the application-specific "PROC" icon (see Fig.3).

Fig. 3: Basic data-definition screen

3.1 Test case

A typical test case is described here for illustrating an ordinary production task in log cutting that can be solved by the developed DSS. Let suppose that a customer orders three different quantities of veneers of equal thicknesses and of three different sheets lengths: l_1 = 182 cm and total area A_1 m^2; l_2 = 151 cm and total area A_2 m^2; l_3 = 112 cm and total area A_3 m^2. In this situation it is calculated that the 350 sub-logs (b_1) of length l_1, 250 sub-logs (b_2) of length l_2 and 400 sub-logs (b_3) of length l_3 are needed if the original logs of tree type 3 with length of 446 cm is used. All possible cutting patterns that can be created from selected log of length 446 cm are determined. Results of this step are given in Table 3.

The operations manager has to input data into the DSS, according to the Table 3. Input data relate to the computation objective (min or max), to the number of constraints and decision variables and to the file name (existing or new). Fig. 3 and the upper-left/central part of Fig. 4 illustrate how the test data can be fed into the new DSS by means of the two data-input GUI screens.

Table 3: Possible cutting patterns for a log of length L_1 = 446 cm, j = 1, 2, …, 12, p = 1

Cutting patterns (j)	182-cm sublogs	151-cm sublogs	112-cm sublogs	Edge waste (cm)
Pat. 1- x1	1	0	0	264
Pat. 2 - x2	2	0	0	82
Pat. 3 - x3	1	1	0	113
Pat. 4 – x4	1	1	1	1
Pat. 5 – x5	1	0	1	152
Pat. 6 – x6	1	0	2	40
Pat. 7 – x7	0	1	0	295
Pat. 8 – x8	0	2	0	144
Pat. 9 – x9	0	2	1	32
Pat. 10 – x10	0	0	1	334
Pat. 11 – x11	0	0	2	222
Pat. 12 – x12	0	0	3	110
Req.nb. of sublogs	350	250	400	

After performing data input, the presented test case brings the operations manager to the results of the SAS "PROC LP" optimization procedure. To get them, he has to click on the "SOLVE" button shown in Fig. 4. The results of the optimal solution appear in the left-bottom part of the same (main data input/output) screen. In this solution, the values of decision variables (X1, X2,…, X12) practically represents the number of 446 cm logs that have to be cut according to twelve (n_1 = 12) cutting patterns given in Tab. 1. In this case, the minimally required number of original logs to fulfill this customer order is **338**. Optional sensitivity analysis is available by clicking either to the "RANGE/PRICE" or to the "RANGE/RHS" button. Recalling Fig. 4, the user can combine input data from other databases by clicking the "ACCESS" button.

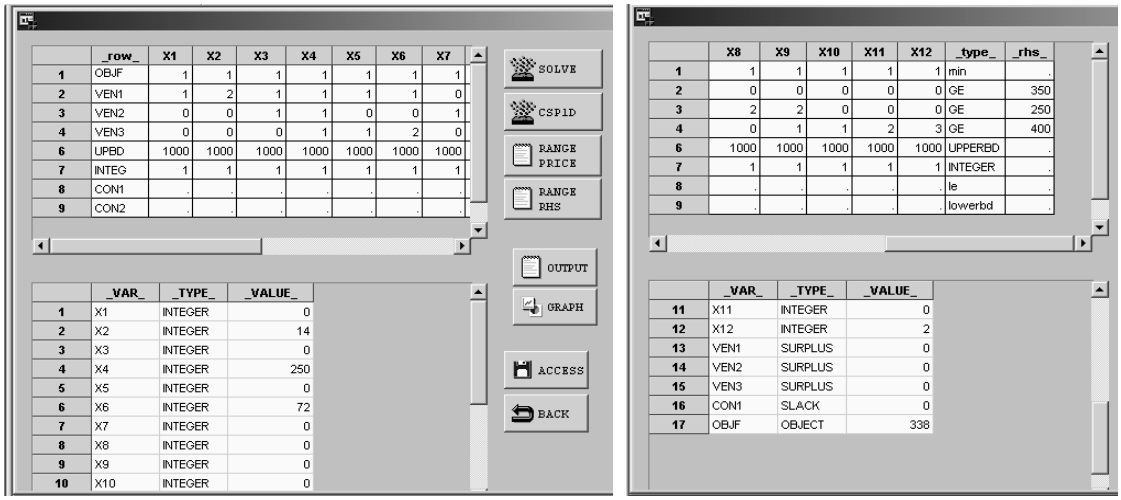


Fig. 4: Main data I/O screen: input test data and the results of optimal solution

4 Analysis and discussion

The approach to DSS development used in this paper represents an attempt to make the implementation and use of DSS in industry and other smaller businesses more efficient. It recognizes the fact that effective DSS are aimed at semi-structured problems, utilize models with internal and external databases, and emphasize flexibility, effectiveness and adaptability. Introduction of such systems in the business environment requires to solve a range of non-technical and people-related problems, e.g. limited resources, low level of IT and MP knowledge, open organizational issues, sometimes resistance to use quantitative models for decision making, etc. In this situation, the approach recommends to take full advantage of commercially available MP software and modern information-processing facilities, i.e. personal computers, mostly affordable to or available even in smaller firms, respectively.

Looking at the case study from the practical viewpoint, let us discuss the benefits of using the new DSS, in comparison to the intuitive decision-making used in the veneers manufacturing firm to date. For this purpose, we will look closer to the test case given in Table 3. First, let us look how the operations manager solves the problem by his ad-hoc approach. He may intuitively decide to cut one original 446 cm log directly

into the required lengths of sub-logs of 182, 151 and 112 cm, leaving an edge-waste of 1 cm. This requires to cut 400 original 446 cm logs and gives a total **edge waste of 4 m** only, a total **end waste of 317,5 m.** In this situation total length of original logs required for cutting is 1784 m.

One reason is that the manager is looking only on reducing edge waste and is not used to care about end waste. This was acceptable before, as the over-production of veneer sheets and the consequent inventory costs were not problematic because the additional veneers packets could be sold at a later date. The next reason is that the manager is not enough skilled now to create intuitively different cutting patterns and/or to choose the best one. Thus, the simplest way for him is to cut the logs as the above example shows.

On the other hand, the optimal solution that is generated by the DSS results in total 338 original logs required for cutting, where **total edge waste is 44,98 m** that is much greater then 4 m, but total **end waste is zero and** the consumption of original logs is reduced to 1507,48 m.

There are also some other possibilities in the same production process where this DSS can be used, such as:

- Help in ordering new logs.
- Log cutting optimization with consideration of logs on stock.
- Consideration of cutting-machine setup costs.
- Consideration of a greater number of different sub-log lengths using the delayed column generation method.
- Broader use of DSS for similar decision-making situations in other types of production.
- Optimization in the same organization, such as product-mix problems, blending problems and transportation problems.

The GUI is the essence of the new DSS which is built around the original SAS/OR software. Actually, it works with four data input/output screens having a relatively simple layout. Developing a DSS application directly from the original SAS/OR software requires from the user knowing the SAS high-level programming language and some basic computer knowledge.

On the other hand, this new GUI is an user-friendly interactive windowing point-and-click application that gives a user quick, easy access to desired information. For using the new GUI efficiently, the user has to understand: the goal and the basic assumptions, the principles of LP, the 1d-CSP formulation in LP terms, the structure of the screens, the meaning of command options and the contents of data-input/output windows. Once instructed about the procedure of using the DSS for solving the log-cutting problem, the user can get information that will help him at more appropriate decision making.

When comparing the use of this DSS with the procedure of using the original SAS GUI for solving the same 1d-CSP, it turns out that the main advantage of such a DSS is the fact that the user does not need to formulate the LP model in the specific SAS language. The other advantage is the possibility of using the other SAS optimization procedures [9] for solving other types of optimization problems. The DSS prototype that includes the PROC LP of the SAS/OR software is not very case-specific. This means that it is useful for solving other types of LP and MIP problems and can also be used for training the users about the new approach to operational decision-making.

5 Conclusion

Through a case study from the production of veneers, the paper presents a pragmatic approach to the development of a DSS, to be used by the operations managers. The basic idea of the proposed approach is to choose a good commercially available OR software and then upgrade it with a user-friendly graphical interface. The feasibility of the approach is illustrated by a simple prototype DSS for optimal cutting of logs in the manufacturing of veneers.

By choosing to test the DSS which utilizes the *pattern-oriented* LP-based Methods for solving the standard 1d-CSP, it was attempted to show that the power of OR can be more successfully presented to the industrial practitioners when the underlying theory is relatively straightforward. The results of testing a typical cutting-stock problem with actual production data indicate that such a user-oriented DSS can indeed improve the competitiveness of a (small) organization. It does this by supporting the operations manager to process customer orders more efficiently and, at the same time, by helping him (her) to keep the inventory of finished goods as low as possible. Therefore, it is considered that the challenge mentioned at the beginning of this paper can be met satisfactorily by using the prototyping approach even for solving of more complex OR problems.

By taking together the cumulative benefits from all these possible future DSS applications, it can be imagined that a significant financial/economic/ commercial benefit for the firm can be expected. Of course, only a detailed cost-benefit analysis, based on real production data, can give a proper basis for investing in this direction of further DSS development.

6 References

[1] Basnet, C., Foulds, L., Igbaria, M. (1996) Fleet *Manager:* a microcomputer-based decision support system for vehicle routing, *Decision Support Systems* 16, pp. 95-207.

[2] Buede, M.D. (1992) Superior Design Features of Decision Analytic Software, *Computers & Operations Research*, pp. 19 (1), 43 - 57.

[3] Chaudhry, S., Salhenberger, L. and Beheshstian, M. (1996) A small business inventory DSS: design,

development and implementation issues, *Computers & Operations Research 23 (1)*, pp. 63-72.

[4] Čižman, A., Černetič, J. (2004) Improving competitiveness in veneers production by a simple-to-use DSS, *European Journal of Operational Research 156*, pp. 241-260.

[5] Dyckhoff, H. (1990) A typology of cutting and packing problems, *European Journal of Operational Research 44*, pp. 145-159.

[6] Foulds L. and Thachenkary C., 2001. Empower to the people – how decision support systems and IT can aid the OR analyst, OR/MS Today, June 2001, p. 3. (Downloadable from website http://www.lionhrtpub.com/orms/orms-6-01/foulds.html).

[7] Gillmore, P.C. and Gomory, R.E. (1961) A linear programming approach to the cutting stock problem, *Operations research, 9*, pp. 849-859.

[8] Haessler R.W.and Sweeney P.E. (1991) Cutting stock problems and solution procedures, *European Journal Of Operational Research, 54*, pp. 141-150.

[9] Keen, P.G.W. (1988) Decision Support Systems: the Next Decade, *Decision Support systems 3*, pp. 253-265.

[10] Kearney, D.T. (1999) Advances in Mathematical Programming and Optimization in the SAS System, *Proceedings SAS European Users Group International (SEUGI) 99*, SAS Institute Inc.

[11] SAS (1999) SAS/OR User's Guide: Mathematical Programming, *Version 8, SAS Institute Inc.*, Cary, NC.

[12] SAS (1995) SAS/AF Software Frame Application Development Concepts, Version 6, First Edition, *SAS Institute Inc*., Cary, NC.

[13] SAS (1994) The SAS System for Information Delivery-Scientific and Technical Applications, *SAS Institute Inc.*

Dissipationless Waves for Information Transfer in Neurobiology – Some Implications

Danko D. Georgiev
Division of Pharmaceutical Sciences
Kanazawa University Graduate School of Natural Science and Technology
Kakuma–machi, Kanazawa, Ishikawa 920–1192, JAPAN
E-mail: danko@p.kanazawa-u.ac.jp

James F. Glazebrook
Department of Mathematics and Computer Science
Eastern Illinois University
600 Lincoln Avenue, Charleston IL 61920–3099, USA
E-mail: jfglazebrook@eiu.edu

Keywords: soliton, energase, microtubule, SNARE complex.

Received: June 17, 2005

We describe a biophysical framework for subneuronal processing of information via certain quantum mechanical processes and solitonic interactions as applicable to neuronal microtubules. In particular, we describe how certain energase actions and vibrationally assisted tunneling may influence the conformational dynamics of the neuronal cytoskeletal protein network. Some implications are also discussed in relationship to special neurophysiological processes as basic to the study of mind and memory.

Povzetek: Opisan je model neurobiološkega delovanja.

1 Introduction

Solitons are dissipationless waves whose theory and applications prevail in fields such as quantum physics, atmospherics, oceanography, cellular automata, and biophysical systems. Some well known examples appearing in the wealth of literature on the subject include the equations of Korteweg-de Vries, Boussinesq, Klein–Gordon, and the nonlinear Schrödinger (NLS) equation (Dodd et al. 1982, Calogero and Degasperis 1982, Davydov 1991). These robust, often bell–shaped waves can propagate in a pulsating manner while retaining their form and velocity in undergoing collisions; so in a sense they can be compared with interacting particles. On the other hand, their universality as a nonlinear scientific phenomenon suggests they are essential to understanding life and information within a unified framework, and therefore provide an essential contribution to the understanding of consciousness.

Soliton equations constitute part of a hierarchy of integrable, or 'solvable', systems admitting high degrees of symmetry (Ablowitz and Clarkson 1991, Calogero and Degasperis 1982, Miwa et al. 1982), but seen as solutions to nonlinear wave equations, solitons do not normally obey the superposition principle, so that when two solutions are combined, a complicated wave is formed. Eventually however, pairs of soliton waves are seen to actually pass through each other thus revealing an unusual phenomenon that has far–reaching applications. Of specific interest here are 'kink' and 'antikink' solutions which are common to a number of solvable systems where spatial derivatives are localized; typically, the resulting wave pulsates in a twisting fashion with certain asymptotic properties. Besides kink and antikink solutions, there may also be oscillatory solutions known as 'breathers' which will play an instrumental role in the discussion following.

For biomolecular/physical systems, the works of Davydov (1982, 1991) provide a foundation for applying the theory of solitons for dissipationless energy transfer in hydrogen bonded systems, DNA, membraneous flexing, muscular contraction and other phenomena (we refer also to the excellent article by Scott 1992 on this subject). Our interest here draws upon the role that soliton dynamics can play in neurobiology/neurophysiology in a particular situation; namely, we survey how such effects theoretically related to systems such as the sine–Gordon and the class of evolutionary equations considered in Davydov (1982, 1991), might influence the mechanisms of dendritic and axonal microtubules, subneuronal processing of information, and synaptogenesis in cerebral architecture.

2 Microtubules and C–terminal tubulin tails

Neuronal structures within the brain are known to be dynamically regulated by strings of self–assembling protein networks forming the cytoskeleton, a skeleton–like protein network that regulates cellular dynamics. The main constituents of the cytoskeleton consist of microtubules which are like hollow cylinders of 25 nm in diameter, of variable length (from micrometers to milimeters, depending on whether they are contained within dendrites or axons) and are composed of assemblies of α/β tubulin dimers. Microtubules interact with intermediary and actin filaments, MAPs (microtubule associated proteins), as well as different scaffold proteins, thus organizing the intracellular space and tuning the biochemical activity of microtubule anchored enzymes (mostly phosphatases and kinases). The assembly by α/β tubulin dimers is a process requiring nucleotide GTP (guanosine triphosphate) to bind to both α and β tubulins. The α–bound GTP never hydrolyzes, whereas the GTP–molecule which is tied to the β–tubulin, is hydrolyzed to nucleotide GDP (guanosine diphosphate) soon after the dimer is incorporated into the growing microtubule lattice. The released energy is then stored in the microtubule wall as an elastic strain, and the β–tubulin bound GDP cannot be further phosphorylated or exchanged for GTP because the successive α–tubulin in the protofilament occludes the preceding β–tubulin nucleotide binding pocket (Heald and Nogales, 2002).

Experimental data by Sackett (1995) revealed the form of microtubules not as smooth cylinders, since extending from each tubulin are tiny 'hairy' projections of 4–5 nm in length, referred to as *tubulin tails*. Since these projections are highly flexible, their PDB structure was revealed only recently by Jimenez et al. (1999) who determined the helicity of α (404–451) and β (394–445) tubulin C–terminal recombinant peptides with the use of NMR (nuclear magnetic resonance). They showed that the C–terminal domain of tubulins has a different length and structure in both α– and β–tubulin. In general, the C–terminal domain has a C–terminal helix H12 and a random coil C–terminal tubulin tail. In α–tubulin molecules aminoacid residues 418–432 form the C–terminal helix H12 and aminoacid residues 433–451 comprise the α–tubulin tail. The α–tubulin C–terminal tail aminoacid sequence is EEVGVDSVEGEGEEEGEEY. The α–tubulin tail is 19 aminoacids long and possesses 10 negatively charged residues. The situation in the β–tubulin C–terminal domain is more interesting. Jimenez et al. (1999) have computed a 9 aminoacid longer helix of the β–tubulin compared to previous PDB models (cf Nogales et al. 1998). This suggests an extension in the protein, supporting the possibility of a functional coil–to–helix transition at the C–terminal zone. The β–tubulin C–terminal helix H12 is formed by aminoacid residues 408-431, but it seems that the reversible transition between coil and helix of the last 9 aminoacid residues 423–431 from the C-terminal helix (with sequence QQYQDATAD) could either decrease or increase the length of the helix H12, at the same time increasing or decreasing the β–tubulin tail length. The β–tubulin tail aminoacid sequence (residues 432–445) is EQGEFEEEEGEDEA. It has 14 aminoacids and 9 negatively charged residues, but depending on the conformational status of the residues 423–431, the β–tubulin tail random coil can extend to 23 aminoacid residues bearing 11 negative charges. Following the C–terminal helices α–H12 and β–H12, the 19 and 14 C–terminal residues of the respective α– and β–tubulin tails are observed to be disordered by NMR. In particular, this is a dynamical disordering and is effectively the manifestation of the extreme sensitivity of the tubulin tails to environmental conditions, and local electric fields yielding a plethora of metastable conformations (Georgiev 2003a).

Located within dendrites and axonal projections, microtubules serve as tracks for the transportation of post–Golgi vesicles by microtubule bound motor proteins (such as kinesin and dynein). Microtubules however are not passive elements in the vesicle transport and it has been shown that the tubulin C–terminal tails modulate kinesin function. Experiments performed by Skiniotis et al. (2004) have shown that the β–tubulin tail interacts with the kinesin switch II domain, while the α–tubulin tail possibly interacts with the kinesin α 7–helix in such a way that after the kinesin bound ATP (adenosine triphosphate) is hydrolyzed, the kinesin perambulates along the microtubule surface. Native microtubules that possess tubulin tails cannot be decorated by ADP (adenosine diphosphate)–kinesin molecules because of the weak ADP–kinesin/tubulin tail binding, while subtilisin treated microtubules that lack tubulin tails bind stably ADP–kinesin, thus blocking the kinesin walk. The conclusion is that the tubulin tails catalyze the detachment of the kinesin–ADP complex from the microtubule surface allowing the kinesin dimer to take a 'step' along the microtubule protofilament.

Microtubules do not only regulate motor protein function but also attach with their C–terminal tubulin tails different MAPs and protein kinases and phosphatases, thus organizing the intraneuronal space. The proper attachment/detachment of these proteins could regulate their enzymatic activity. In case studies of schizophrenia, Arnold et al. (1991) have found altered expressions of MAP2 and MAP5 that result in abnormalities in the neuronal cytoarchitecture. Whereas in Alzheimer's disease, the primary alteration is the phosphorylation status of axonal MAP–tau and the activity of protein phosphatase 2A (PP2A) regulated via attachment/detachment to microtubules (Sontag et al. 1999).

We propose that the mechanism of the tubulin tail enzymatic action is generated by vibrationally assisted tunneling – a key concept which emerged and was experimentally verified over the last several years (Sutcliffe and Scrutton, 2000). A locally formed tubulin tail standing breather could promote or suppress conformational tunnel-

ing of a molecule attached to the tubulin tail. The effect of vibrations on mixed-tunneling could be either to promote or to suppress the tunneling process and this depends on the boundary conditions (Takada and Nakamura 1994, 1995). Formally, the mechanism of the tubulin tail breathing action could be manifestly a form of enzymatic energase process. Energases do not have source of energy, but rather induce conformational transitions in a molecule that has accumulated energy in an intermediate highly energetic conformational state (Purich, 2001). The accumulated energy is derived from hydrolyzed ATP or GTP in previous biochemical steps, so for that reason this energy is usually called 'primed energy' and the process of energy accumulation in metastable protein states is referred to as 'priming'.

The idea that microtubules might be agents of subneuronal processing of information was originally suggested by Hameroff and Watt (1982). Hameroff and colleagues (Hagan et al. 2000) conjectured that the energy for computation could be delivered from the tubulin bound GTP molecules. Since it had been already observed that in stable microtubules there is no possibility for tubulin bound nucleotide cycling, we propose that tubulin tail energase action releases the energy accumulated in metastable conformational states of kinesin, dynein, or phosphorylated MAPs. The metastable states of these proteins are produced via ATP hydrolysis through previous 'priming' steps. We mention that ideas involving GTP–hydrolysis, ferroelectric phase and (C–terminal) tubulin tails as possible agents of information transfer, have been suggested in Georgiev (2003a, 2003c, 2004), Georgiev et al. (2004), Satarić and Tuszyński (2003) and the appropriate references therein.

3 The water laser as a pumping mechanism

As the organizing framework for special neurobiological processes, the cytoskeleton is the major intracellular structure providing a protein surface to which water molecules cling thus facilitating the water ordering. We point out that the term 'water' used here is not quite the same as its mundane sense, but instead should be regarded as a protein–like saturated mixture. Ordered (vicinal) water molecules are microscopic dipoles that interact with each other via hydrogen bonds whose effect influences a relatively high viscosity, surface tension and dielectric constant. They form the water electric dipole (WEDP)–field occurring on either side of a brain cell. Within the interior of the cell, the water molecules generate a WEDP–field in the vicinity of the cytoskeleton, whereas in the exterior of the cell, the molecules form an intercellular flow completing the regions between neighbouring cells. Del Giudice et al. (1983) have proposed that electromagnetic waves arising from the WEDP–field within the body of the cytoskeleton, create signals compatible in size with the internal diameter of a given microtubule.

To proceed, we adopt in part the development of Jibu et al. (1994, 1996, 1997). Let $\mathbb{V}$ denote a perimembranous region or a spatial region in the vicinity of a cytoskeletal microtubule. The WEDP–field in $\mathbb{V}$ taken within a cylindrical neighbourhood, is represented by a 2–spinor field

$$\psi(\mathbf{x},t) = \begin{bmatrix} \psi^+(\mathbf{x},t) \\ \psi^-(\mathbf{x},t) \end{bmatrix} , \tag{3.1}$$

where $\psi^+(\mathbf{x},t)$ and $\psi^-(\mathbf{x},t)$ are spinor components. The electric dipole moment is given by

$$\mu = \psi(\mathbf{x},t)^* \, \frac{\hbar}{2} \sigma \, \psi(\mathbf{x},t) , \tag{3.2}$$

where $\sigma = [\sigma_1, \sigma_2, \sigma_3]$ is a 3–vector whose components consist of the Pauli spin matrices. The dipole moment μ exhibits the water molecule as similar to a quantum–mechanical spinning top. In other words, it is due to μ that the water molecules interact dynamically with the quantized electromagnetic field in $\mathbb{V}$. If m_p and e_p denote the proton mass and charge respectively, then the average moment of inertia of a water molecule is estimated as $I = 2m_p d^2$ with $d \approx 0.82$Å, whereas μ is estimated as $\mu = 2e_p P$, with $P \approx 0.2$Å .

Given $\psi(\mathbf{x},t) \neq 0$ only holds at each position $\mathbf{x} = \mathbf{x}_k$ of the k–th manifestation of localization, the WEDP–field with N localizations are describable in terms of N spin variables as given by

$$s^k(t) = \psi(\mathbf{x}_k,t)^* \, \sigma \, \psi(\mathbf{x}_k,t) , \qquad 1 \leq k \leq N . \tag{3.3}$$

The Hamiltonian of the WEDP–field for N water molecules with energy difference ϵ, is given by

$$H_{WM} = \epsilon \sum_{k=1}^{N} s_3^k(t) , \tag{3.4}$$

where for a given wave vector k_0, it is convenient to assume that a normal mode has an angular frequency ω_{k_0} resonating to the energy difference between two principal eigenstates for which $\epsilon = \hbar\omega_{k_0}$ ($\epsilon \approx 24.8$ meV), in accordance with the predictions of dominance over other possible energy exchanges (Del Giudice et al. 1988). The radiation field of $\mathbb{V}$ is given by a scalar electric field operator $E = E(\mathbf{x},t)$ whose associated Hamiltonian is

$$H_{EM} = \frac{1}{2} \int_{\mathbb{V}} E^2 \, d^3\mathbf{x} . \tag{3.5}$$

A main premise of Jibu and Yasue (1997) is that the dynamics of the WEDP–field and the quantized electromagnetic (EM) field is an energy interchange through creation and annihilation operators of photons. In order to see this, consider a decomposition of the electric field operator $E = E^+ + E^-$ into its positive and negative frequency

components. Then the Hamiltonian for the interaction between the WEDP–field and the EM–field is given by

$$H_I = -\mu \sum_{k=1}^{N} \{E^-(r^k,t)s_-^k + s_+^k E^+(r^k,t)\}\ , \quad (3.6)$$

where $s_\pm^k = s_1^k \pm \iota s_2^k$. The total Hamiltonian H_{QM} which governs the quantum mechanical dynamics of the electromagnetic field, the dipolar vibrational field of water molecules along with their interaction, is then expressed by

$$H_{QM} = H_{EM} + H_{WM} + H_I\ . \quad (3.7)$$

Since parts of the region $\mathbb{V}$ in the vicinity of a cell can be considered as a cavity for the electromagnetic wave, we introduce the normal mode expansion of E given by

$$E^\pm(\mathbf{x},t) = \sum_\lambda E_\lambda^\pm(t)\ \exp[\ \pm\iota(\lambda\cdot\mathbf{x} - \omega_\lambda t)\ ]\ . \quad (3.8)$$

From a motivational viewpoint, let us mention that the process of signaling response in synapses is influenced by certain classes of cellular adhesive molecules (CAMs) in which the actin cytoskeleton provides a suitable structural mechanism for assimilating the signaling inputs. The formation of functional synapses at an axonal growth cone involves identifying and initiating contacts with suitable companion cells (Brose 1999). Of special importance for synapse formation are two types of CAMs known as β-*neurexin* and *neuroligin* forming a heterologous adhesive interaction. Remarkably, β-neurexin-neuroligin interaction alone has the unique ability to act as a bidirectional trigger of synapse formation (Dean and Dresbach, 2006). β–neurexin is located in axons and interacts presynaptically with CASK, a multidomain scaffolding protein that organizes the presynaptic space and emits signals to the actin cytoskeleton via protein 4.1. β–neurexin also directly interacts with the synaptic vesicle protein synaptotagmin-1, thus controlling exocytosis and neuromediator release (see later). Synaptotagmin-1 per se might act as MAP molecule binding to β–tubulin tails stabilizing microtubules in high Ca^{2+} concentration presynaptically. Neuroligins are located in dendrites and transmit information to postsynaptic density protein (PSD-95), which is a multidomain scaffold protein that anchors different ion channels to the active zones of the postsynaptic membrane. Neuroligin-1 is a specific CAM for excitatory (glutamatergic) synapses, while neuroligin-2 is a specific CAM for inhibitory (GABAergic) synapses. PSD-95 is anchored to postsynaptic microtubules via another protein known as CRIPT. Neuroligins on binding with presynaptic β–neurexins, comprise an adhesive system facilitating learning processes manifest as a morphological reorganization of the synapse. Relevant here is that the radiation field of (3.8) could be considered as falling within this junction as shielded by ordered water molecules, and so assists the signaling mechanism between neighbouring neurons (Georgiev 2003b, 2003c).

Next, we introduce collective dynamical variables $S_\lambda^\pm$ for water molecules given by

$$S_\lambda^\pm(t) = \sum_{k=1}^{N} s_\pm^k(t)\ \exp[\ \pm\iota(\lambda\cdot\mathbf{x} - \omega_\lambda t)\ ]\ . \quad (3.9)$$

On setting $S \equiv \sum_k s_3^k$, we can express (3.7) in the form

$$H_{QM} = H_{EM} + \epsilon S - \mu \sum_\lambda \{E_\lambda^- S_\lambda^- + S_\lambda^+ E_\lambda^+\}\ . \quad (3.10)$$

Equation (3.10) resembles that of the Hamiltonian for a laser radiation process, and in this way suggests that the water molecules of $\mathbb{V}$ exhibit a laser–like coherent optical property, *provided the energy is sustained above a certain threshold*; this threshold will be represented by equation (3.15) below. The dynamically ordered region of water molecules and quantized EM–field, are considered within a coherence length of $50\ \mu m$. The explanation given by Jibu et al. (1997) is that by increasing the ordering of water on the microtubule surface, spontaneous symmetry breaking occurs (see below), thus creating Nambu–Goldstone (NG) bosons, the quanta of long–range correlation waves of the aligned electric dipoles referred to as *dipole wave quanta*, denoted DWQ.

The Hamiltonian H_{EM} can also be expressed in terms of canonical operators (observables) $P_\lambda(t)$ and $Q_\lambda(t)$ as defined by

$$\begin{aligned} P_\lambda(t) &= \sqrt{\frac{\hbar\omega_\lambda}{2}}\ \iota(E_\lambda^- - E_\lambda^+)\ , \\ Q_\lambda(t) &= \sqrt{\frac{\hbar}{2\omega_\lambda}}\ (E_\lambda^- + E_\lambda^+)\ , \end{aligned} \quad (3.11)$$

and which satisfy the well–known canonical commutation relations of the Heisenberg algebra. On making the necessary transformations and substituting into (3.10), we obtain

$$\begin{aligned} H_{QM} = \frac{1}{2}\sum_\lambda \{P_\lambda^*(t)P_\lambda(t) + \omega_\lambda^2 Q_\lambda^*(t)Q_\lambda(t)\} \\ + \epsilon\sum_{k=1}^{N} s_3^k(t) \\ - \sqrt{\frac{2}{\hbar}}\mu\ \sum_{k=1}^{N}\sum_\lambda \{\sqrt{\omega_\lambda}Q_\lambda(t)s_1^k \\ - \frac{1}{\sqrt{\omega_\lambda}}P_\lambda(t)s_2^k\}\ . \end{aligned} \quad (3.12)$$

Consider when a system possesses a certain symmetry but through which the vacuum state is altered (through this symmetry) and may be transformed into some other degenerate state, whereas the Lagrangian symmetry remains independent of the vacuum solution. In other words, the Hamiltonian may be invariant under the symmetry transformation but the vacuum (or lowest energy) state is not. In this way, spontaneous symmetry breaking (SSB) occurs

and results in massless quanta governed by Bose–Einstein (BE) statistics that are assigned to repair the broken symmetry. The NG bosons are understood to be the quanta of long range coherence induced by the vacuum state, which violated the original dynamical symmetry. Typically, what might otherwise be two massive fields emerge from SSB as one massive and one massless field, the latter in this case is a NG boson. In Jibue–Yasue (1997) this is explained when the corresponding Heisenberg equations of (3.12) are considered in order to study the dynamically ordered state of the WEDP–field in terms of a long–range alignment of associated spin variables. Under an $\mathrm{SO}(2)$–transformation of the canonical variables, the Hamiltonian H_{QM} is invariant, whereas a time independent solution is not invariant.

In order for the coherent emission of photons to have the proper biological impact, it is necessary to consider timescales of the order of 10–15 picoseconds which are compatible with that of protein action. In the presence of a disordered thermodynamic system, thermal fluctuations, noise and dissipation have to be take into consideration. However, the laser–like emission of coherent photons may still be realized under such circumstances once the protein molecules achieve dynamics sufficient to engage a pumping effect of the WEDP–field. This 'slow phenomenon' involving the water laser is preferred in this situation to the 'fast phenomena' of superradiance. Jibu and Yasue (1997) consider the relevant system of Heisenberg–Langevin equations governing the collective dynamics of the quantized EM–filed in $\mathbb{V}$. On assuming a certain coherent state representation, these are seen to reduce to the stochastic Langevin equation

$$\frac{dZ}{dt} = \alpha_1 Z - \alpha_2 \overline{Z} Z^2 + B \,, \tag{3.13}$$

where $Z = Z(t)$ is a Markov process in $\mathbb{C}$ of the corresponding EM–field operator, $B = B(t)$ is a (complex) Gaussian white noise of thermal fluctuations of quantized EM–field, and α_1, α_2 are particular constants depending on the volume V of the region, thermal fluctuations for the EM and WEDP–field, damping coefficients (denoted γ, γ_0) for the WEDP–field, and a parameter of pumping rate (denoted S_∞) resulting from the interaction of the WEDP–field with the dynamics of the microtubule protein molecules. In turn, these parameters are used to define a diffusion constant D, which along the probability density function $f = f(z, \bar{x}, t)$ of $Z(t)$, transform equation (3.13) to its corresponding Fokker–Planck equation

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial z}[(\alpha_1 z - \alpha_2 \overline{z} z^2) f] + D \, \frac{\partial^2 f}{\partial z \, \partial \overline{z}} \,. \tag{3.14}$$

Finally, and again referring to Jibu and Yasue (1997) for details, the required level of excitations of the quantized EM–field, namely the photon emission as induced by the electric dipoles of tubulin, is attained when the pumping rate S_∞ satisfies the estimate

$$S_\infty > \frac{\hbar^2 V \gamma_0 \gamma}{4\pi\epsilon f^2} \,. \tag{3.15}$$

Thus it is suggested that the energy for the coherent pulse emission by vicinal water in a proximity of 4-5 nm of the microtubule's outer surface could be gained from the tubulin electric dipole oscillations and/or from vibrations along the microtubule walls. The transmission of pulse mode coherent photons is determined by Maxwell's equation as derived from the total Hamiltonian H_{QM} . For $E = E(z,t)$ it is given by the quantum dynamical equation of motion (Jibu et al. 1994, 1997, Abdalla et al. 2001) :

$$\frac{\partial E^\pm}{\partial z} + \frac{1}{c}\frac{\partial E^\pm}{\partial t} = \mp\iota \frac{2\pi\epsilon\mu}{\hbar V} S^\pm \,. \tag{3.16}$$

In terms of a quantum average, denoted $\langle \, \rangle_q$, the expression for the electric field is

$$\theta^\pm(z,t) = \frac{2\mu}{\hbar} \int_{-\infty}^{t} \langle E^\pm(z,u) \rangle_q \, du \,. \tag{3.17}$$

This leads to a soliton equation of sine–Gordon type

$$\frac{\partial^2}{\partial t \partial \sigma} \theta^\pm = -2A \sin \theta^\pm \,, \tag{3.18}$$

expressed in Lorenztian coordinates, where $A = \frac{2\pi\epsilon\mu^2 N}{\hbar^2 V}$, in which $\frac{N}{V}$ is the number of water dipoles per unit of volume, and $\sigma = t + \frac{z}{c}$. The indices $\pm$ indicate the transverse directions of the electric field where it is assumed there is no propagation in the longitudinal direction. The soliton equation (3.18) is an equation characteristic of self–induced transparency as realized in nonlinear optics and here suggests how the cumulative effects of the WEDP–field might induce a transfer of energy via dissipationless waves. Time–differentiating (3.17), leads to

$$E = \frac{\hbar}{\mu}\sqrt{A\rho} \, \mathrm{sech} \, [\, \sqrt{A\rho} \, (t - \frac{z}{c}) \,] \,, \tag{3.19}$$

where $\rho = \frac{v_0}{c - v_0}$. The above equations were taken up by Abdalla et al. (2001) who studied the correspondence between information configurations induced by solitonic interactions and the DWQ at certain levels of excitation. As is part represented by the sine–Gordon equation (3.18), the cumulative effect of the WEDP–field then induces a source of resonant–propulsive energy.

Let us mention several alternative models which consider different dynamics, based on equations of 'solvable' type, which are relative to the lattice structure of microtubules. For instance, in Chou et al. (1994) energy releasing effects of GTP–hydrolysis could generate certain kinks and pulsations which propagate along the microtubule via elastic flexing of the dimers. In Satarić and Tuszyński (2003) a liquid crystal property of microtubules is considered relative to kink 'shifting' through GTP hydrolysis whose rate may increase given additional Ca^{2+} and where possible impediments to the kink motion, polymerization, and microtubular caps are taken into account. These models, however, investigate effects in dynamic microtubules that undergo assembly/disassembly while not addressing

the contrasting situation for stable microtubules (such as the neuronal types). Another model involving solitonic interactions, as considered by Mavromatos et al. (2002), entails possible quantum coherent states of the DWQ on the tubulin dimer walls where the DWQ are paired to electrons in the dimer hydrophobic pockets via Rabi field coupling.

A model suggested in Georgiev (2004) relates to how the water dipoles from the tubulin tail hydration shells that form a 4–5 nm layer on the outer microtubular surface, strongly interact with the local electromagnetic field thus affecting the conformational state of the tiny C–tubulin tails. The model is based on a long–range interaction of the water molecule dipoles and local EM field resulting in a coherent emission of photon pulses propagating via tunneling. The resulting solitons could be viewed as traveling conformational waves in the tubulin tails that do not dissipate under thermal fluctuations, but could be pumped by the water laser provided the threshold inequality (3.15) is satisfied. This model also considers solutions to the sine–Gordon equation as providing the necessary dynamics. To facilitate matters, consider a change of parameters from Lorentzian coordinates to laboratory coordinates, so that equation (3.18) is now expressed by :

$$u_{tt} - u_{xx} = \pm \sin u \ , \ \ u \equiv u(x,t) \ . \tag{3.20}$$

We have chosen for now a description based on the elastic ribbon model, and recall that a kink soliton involves a twist in a solution, $u = u(x,t)$ say, which moves from one solution $u = 0$ to an adjacent solution $u = 2\pi$. Vacuum states as constant solutions of zero energy, correspond to $u = 0(mod\ 2\pi)$. In this respect, the traveling solitons of Jibu–Yasue can be regarded as tunneling photons coupled with tubulin tail hydration shells. The assumption is that there is a prevailing coherence time of 10–15 picoseconds.

Such a kink (K) solution u_K of (3.20) as given by :

$$u_K = 4 \tan^{-1} \exp[\ \gamma_K (x - v_K t - x_K)\ ] \ , \tag{3.21}$$

where $0 \leq v_K < 1$ is the kink velocity, x_K the kink position at $t = 0$, and

$$\gamma_K^{-1} = (1 - v_K^2)^{\frac{1}{2}} \ , \tag{3.22}$$

the kink width. The kink energy is given by $E_K = 8\gamma_K$.

On setting $G = \gamma_K(x - v_K t - x_K)$, one also finds the derived equations :

$$\begin{aligned} u_x &= 2\gamma_K \ \mathrm{sech}\ G \ , \qquad \text{(magnetic field)} \\ u_t &= -2\gamma_K v_K \ \mathrm{sech}\ G \ , \qquad \text{(electric field)} \\ \sin \frac{1}{2} u &= \mathrm{sech}\ G \ , \end{aligned} \tag{3.23}$$

(see Dodd et al. 1982).

The antikink (AK) solutions correspond to reversing the velocity, $v \mapsto -v$, and taking the negative square root in (3.22). At this stage we mention the role of certain solutions, called *breathers* which are manifestly local oscillating waves resulting from how a kink and antikink can merge into a combined state. Breathers admit more structure compared to a usual traveling wave because of the former's internal oscillations, and in contrast to (topological) ribbon solitons, can evolve without energy activation. In practice they have been realized as linear phonon modes which are excitable within thermal fluctuations (Russell et al. 1997). It was suggested earlier that some class of propagating solitons may influence the conformational states of the tubulin tails. To this extent, in Georgiev (2004, 2003a) several possibilities involving sine–Gordon kink–antikink–breather soliton collisions were proposed, where for instance, a standing breather soliton could be coupled to the energase action of the tubulin tails through vibrationally assisted tunneling. Further, we are reminded how the β–tubulin tails may interact with kinesin switches and the role of the α–tubulin tail in activating the kinesin walk (Skiniotis 2004).

As outlined in Dodd et al. (1982), the scheme of Bäcklund transformations can be employed to derive 3–soliton from 2–soliton solutions. In relationship to the kink solution u_K in (3.21), we follow Dmitriev et al. (1998) to describe a 3–soliton solution u_{KB} representing the elastic collision (without exchange of energy or momentum) between a kink and a breather, as it is given by the sum

$$u_{KB} = u_K + w_B \ , \tag{3.24}$$

where the term w_B is explained as follows. Firstly, if ω denotes the frequency of the breather, $0 \leq \omega < 1$, we set $\eta = (1 - \omega^2)^{\frac{1}{2}}$. Then

$$\begin{aligned} w_B = 4 \tan^{-1} \Big\{ \Big(& 2\omega\eta(\sinh D - \cos C \sinh G) \\ & + 2\eta\gamma_K\gamma_B(v_K - v_B) \sin C \cosh G \Big) \cdot \\ & \Big(2\omega\eta(\cos C + \sinh D \sinh G) \\ & - 2\omega\gamma_K\gamma_B(1 - v_K v_B) \cosh D \cosh G \Big)^{-1} \Big\} \ , \end{aligned} \tag{3.25}$$

where we have set

$$C = -\omega\gamma_B(t - v_B(x - x_B)) + 2\pi m \ ,$$

m an integer,

$$D = \eta\gamma_B(x - x_B - v_B t) \ ,$$

$\gamma_B^{-1} = (1 - v_B^2)^{\frac{1}{2}}$ is the kink width in which v_B denotes the velocity of the breather $0 \leq |v_B| < 1$, and lastly, x_B denotes the position of the breather at time $t = 0$. In the continuum limit, the breather's wavelength λ and period T are related via

$$|v_B| = \frac{\lambda}{T} \ , \ \lambda = 2\pi\gamma_B |v_B| \frac{1}{\omega} \ , \tag{3.26}$$

whereas the amplitude A and energy E_B are given by $A = 4\tan^{-1}(\frac{\eta}{\omega})$, $E_B = 16\eta\gamma_B$.

Particularly interesting is the collision between a standing breather ($v_B = 0$) and a traveling kink. After the collision the kink and breather recover their velocity and shape. However, the interaction results in a phase shift of the standing breather that oscillates at a new position. Therefore we can consider the sine–Gordon soliton collisions as a kind of application of computational gates.

In the process of collision between a moving kink and a standing breather, the shift Δ_B of the breather is given by the formula

$$\Delta_B = \frac{2\tanh^{-1}\sqrt{(1-\omega^2)(1-v_K^2)}}{\sqrt{1-\omega^2}}\,, \qquad (3.27)$$

where v_K is the velocity of the kink. If the original position is denoted x_0, then post–collision, the new position will be $x = x_0 + \Delta_B$.

Thus as a result of a pushing/pulling kink or antikink collision with a standing breather, the latter through its phase shift is conjectured to cause a deflection of the tubulin tails so as to influence the kinesin walk across the microtubule surface. Making the necessary change in parameters, a kink–breather or an antikink–breather collision might actually implement the required 'pushing' effect (this question remains open) if indeed a breather does function as a catalytic agent registering transitions, influencing MAPs and as noted, the workings of the prevailing motor proteins (kinesin and dynein) through tunneling and the energase action. It is possible there are other combinations and permutations of kink–antikink–breather collisions in, say, the pendulum or discrete models (cf Miroshnichenko et al. 2000, even perhaps a configuration of moving breathers as in Russell et al. 1997), which could provide the relevant dynamics. At the same time we keep in mind the kink etc. counterparts in other integrable/solvable systems which might also serve as models of regulatory or computational gates that could influence cytoskeletal processes.

These last issues are discussed in Georgiev (2004) in relationship to some finer neurobiological processes. Concerning these, we comment on two important mechanisms corresponding to protein constituents such as synapsin-1 and synaptotagmin-1. Hirokawa et al. (1989) have proposed that phosphorylation of synapsin-1 by Ca^{2+} dependent kinases, on releasing synaptic vesicles from actin filaments, may accelerate vesicles to the presynaptic membrane. In Honda et al. (2002) it is shown that cytoskeletal protein tubulin binds directly to synaptotagmin-1 which promotes tubulin assembly. At the same time, synaptotagmin-1 functions by attaching synaptic vesicles to microtubules in high concentrations of Ca^{2+}. Presynaptic microtubules may attach directly to the synaptotagmin/SNARE complexes (SNARE abbreviates *soluble NSF attachment protein receptor*, where NSF abbreviates *N-ethyl-maleimide-sensitive fusion protein*) where β–tubulin tails may trigger synaptotagmin dimerization which is essential for accomplishing exocytosis. A further open possibility is that presynaptic microtubules remain crosslinked to docked synaptic vesicles by means of a complex presynaptic scaffold protein network referred to as *the cytomatrix of the active zone* (CAZ).

The SNARE complex, while functioning as a fusion mechanism, may be capable of receiving Ca^{2+} signals transmitted by synaptotagmin-1 Ca^{2+} binding, which may result in the fusion of synaptic vesicle with the presynaptic membranes. This opens up the possibility that a traveling antikink (for instance) on collision with a stationary breather, typically located at a penultimate tubulin tail, may push the breather to the microtubule end β–tail which is attached to the synaptotagmin Ca^{2+} sensor molecule located above the SNARE complex. If indeed the case, then such a model should be relevant to questions posed by Chapman (2002) concerning how synaptotagmin-1 may be realized as a catalyst of exocytosis. Answering these and other questions may well reflect upon the earlier ideas of Beck and Eccles (1992) who hypothesized long–range quantum correlation resulting from the exocytosis of synaptic vesicles when propagating into a bouton.

4 Solitons in α–helix protein molecules

The relevance of soliton dynamics to biophysics can be traced back in part to the studies of Fröhlich (1968, 1975) who considered one–dimensional electron systems occurring in biology. When these systems admit holes of some kind, it was conjectured that electron–hole pairing leads to the existence of intracellular solitonic dynamics inducing dissipationless energy transfer. Fröhlich postulated unusual protein dipole moments and wave frequencies as exhibited by cell membranes and certain enzymes. Such dielectric systems were considered as producing longitudinal electric oscillations across the matter. At suitable levels, energy can be channeled into a single mode and sufficiently ordered so as to sustain coherent electric waves, an ordering suggestive of long range quantum–coherence comparable to BE–condensation. In short, particles forsake their individual characteristics and unite into a condensate regulated by a single wave function, whereas particles outside of the condensate disperse erratically.

Further studies revealed molecules beneath the cell membrane as exhibiting dipolar vibrational activity where thin layers appear to act like biological superconductors in which the resulting wave propagation leads to Fröhlich waves possessing a frequency of order 10^{11} to $10^{12}\,\mathrm{sec}^{-1}$ (see e.g. Grundler and Keilmann 1983). The evidence suggests that protein dipoles in a common electromagnetic field exhibit resonating effects when energy is supplied. Such waves are seen to be induced by dipolar oscillations maintained by hydrogen bonds and non–localized electrons

within hydrophobic regions of protein molecules. The interaction between dipolar excitations and harmonic vibrations of certain biological lattice structures can be modeled on a Hamiltonian from which, as shown in e.g. Satarić et al. (1991), Davydov solitons can be derived relative to rates of chemical reactions. In a broader perspective, the ideas of Fröhlich are linked to electron superconductivity and are closer to utilizing this class of solitons.

Next, we recall the basic principle of how proteins act in converting chemical into mechanical energy, and when aided by lipids they generate the traffic of ions and molecules in and around cellular membranes. As we have mentioned, the protein chain can coil into a helix-like form which is manifestly the structure of hydrogen bonded peptide groups of the protein molecule. Protein molecules incorporated into the cytoskeleton create transduction energy and intracellular couplings all of which assist and determine energy release of hydrolysis of ATP molecules while at the same time portions of the helix constitute part of the cytoskeleton's protein composition. On the other hand, the excited states of a protein molecule are related to the resonant interaction between peptide groups within distinct chains.

According to Davydov (1982, 1991), a class of solitons evolve at the origin of each chain and so can be created within short intervals of α–helix proteins. The propagation of a soliton within an α–helix protein molecule could be either symmetric or asymmetric. Of these, the asymmetric soliton is the more stable and its radiation life–span does not depend on velocity and can increase sharply as the angle between the spiral axis and vibrational dipole moment decreases. This explains why the asymmetric solitons are favourable for transferring the energy of ATP hydrolysis without loss of energy along the α–helix protein chain over suitably large distances. Recall that Jiminez et al. (1999) have predicted a helicial structure to the C–terminal domains, and for certain C–terminal recombinant peptides, this helicity has been determined with evidence supporting a functional coil to the helix transition at the C–terminal zone. As also seen in Amos (2000), each tubulin monomer possesses twelve α–helices (labeled from H1 to H12), so in terms of short–range localization, it is plausible that the above asymmetric soliton propagation is applicable.

In order to see how the corresponding solutions arise, consider the Hamiltonian H_{PM} for collective excited states of the protein molecules as given by

$$\begin{aligned} H_{PM} = \sum_{n,\alpha} \Big\{ & (\mathcal{E} + D_{n\alpha}) B^*_{n\alpha} B_{n\alpha} \\ & + J_{n,\alpha;n+1,\alpha}(B^*_{n\alpha} B_{n+1\alpha} + B^*_{n+1\alpha} B_{n\alpha}) \\ & + J_{n\alpha;n,n+1}(B^*_{n\alpha} B_{n\alpha+1} + B^*_{n\alpha+1} B_{n\alpha}) \Big\} \\ & + H_{ph} \ , \end{aligned} \tag{4.1}$$

(Davydov 1982). In this expression the $B^*_{n\alpha}$ and $B_{n\alpha}$ are creation/annihilation operators for the excitation $\mathcal{E}$ of the peptide group $n\alpha$; the term $J_{n\alpha;m\beta}$ denotes the energy of the resonance inter–dipolar coupling between the peptide groups $n\alpha$ and $m\beta$; $D_{n\alpha}$ denotes the deformation energy of interaction with neighbouring groups arising from excitations of the group $n\alpha$, and H_{ph} is the displacement operator of the groups from their equilibrium position along hydrogen bonds. This is given by

$$H_{ph} = \frac{1}{2} \sum_{n\alpha} [\ \frac{1}{M} P^2_{n\alpha} + w(U_{n\alpha} - U_{n+1\alpha})^2\ ] \ , \tag{4.2}$$

where M denotes the effective mass displaced along with the peptide group, w is the elasticity coefficient of the chain along the hydrogen bonds, and $P_{n\alpha}$ is the momentum operator conjugated to the displacement operator $U_{n\alpha}$ of the peptide group.

Associated to the Hamiltonian H_{PM} is the wave function describing the collective vibrations of the system as given by :

$$|\Psi(t)\ \rangle = \sum_{n\alpha} a_{na}(t) e^{\sigma(t)} B^*_{n\alpha} |0\ \rangle \ , \tag{4.3}$$

where $|0\rangle$ denotes a function for which all of the groups are in the ground-state with vibrationless excitations away from their equilibria, and where

$$\sigma(t) = -\frac{\iota}{\hbar} \sum_{n\alpha} [\ \beta_{n\alpha}(t) P_{n\alpha} - \pi_{n\alpha}(t) U_{n\alpha}\ ] \ . \tag{4.4}$$

In this last expression, the functions $\beta_{n\alpha}(t)$ and $\pi_{n\alpha}(t)$ depend on the average values for the displacement of the groups $n\alpha$ and their momenta in the above state. The coefficient function $a_{n\alpha}(t)$ satisfy $\sum |a_{n\alpha}(t)|^2 = 1$, where the latter corresponds to the distributive probability over the groups $n\alpha$ in their collective excitation states. The complex–valued functions $a_{n\alpha}(t)$ and the real–valued functions $\beta_{n\alpha}(t), \pi_{n\alpha}(t)$ are obtained from minimizing the functional

$$\langle \Psi(t) | H | \Psi(t) \rangle \ , \tag{4.5}$$

and on applying a certain approximation, the following system of equations is deduced (Davydov 1982 §22.4). Firstly, since the functions $a_{n\alpha}(t), \beta_{n\alpha}(t)$ are continuous in n, they are replaced by $a_\alpha(\xi, t), \beta_\alpha(\xi, t)$ respectively. The system in question is then :

$$\begin{aligned} & \Big\{ \iota\hbar \frac{\partial}{\partial t} - [\ \mathcal{E}_0 + W - 2J\ ] - 2\chi\ \frac{\partial \beta_\alpha}{\partial \xi} \Big\} a_\alpha \\ & + J\ \frac{\partial^2 a_\alpha}{\partial \xi^2} - L(a_{\alpha+1} + a_{\alpha-1}) = 0 \ , \\ & [\ \frac{\partial^2}{\partial t^2} - v_\alpha \frac{\partial^2}{\partial \xi^2}\ ] \beta_\alpha = \frac{2\chi}{M}\ \frac{\partial}{\partial \xi}\ |a_\alpha|^2 \ . \end{aligned} \tag{4.6}$$

Here χ is formed from coupling parameters for internal excitations of the peptide groups and their displacements from the equilibrium positions; J denotes the resonant energy of inter–dipolar interactions between neighbouring groups in the same chain, and L the energy of the

same interaction between neighbouring groups from different chains ($J \approx 967$ μeV, $L \approx 1537$ μeV). Also, $v_\alpha^2 = w/M$, the term W is the average density for displacement of molecules from the equilibria position, and $\mathcal{E}_0$ is the excitation energy of the peptide group relative to the deformation potential. It is from this system that the symmetric and asymmetric solitons are derived (see Davydov 1982 §22.4 for explicit details).

It is worth pointing out that Davydov solitons can subserve the function of local effectors (e.g. responsible for local tubulin–kinesin interaction) but are not suitable for long–range dissipationless transfer of information. The BE condensation of tunneling photons in a macroscopic coherence region of $\approx 50\mu m$, however, is sufficiently long–ranged to mediate a global coupling between distant parts within the neuron. The tunneling photons have boson mass of 13.6 eV and their condensation is sustainable even at body temperature of 310 K (Jibu and Yasue, 1997). This is the main reason to didactically separate the possible quantum effects into local (Davydov solitons) and global (BE–condensation of tunneling photons) interactions.

5 DWQ and arrows of time

5.1 Dipole wave quanta and arrows of time

Following the model of Ricciardi and Umezawa (1967) (cf Stuart et al. 1979) that memory entails a phase transition from a chaotic vacuum state to one that is relatively ordered, Vitiello (1995, 2003) proposes the DWQ when in their lowest ground state as inducing a stability of memory with the distinctions of long–term as 'stable' and inherent to the vacuum state, whereas short–term (memory) corresponds to the excitations of the DWQ condensates which were described earlier. Order parameters correspond to the WDQ, 'symmetron' (Riccardi and Umezawa 1967, Vitiello 1995) and the WEDP field electric polarization, the 'corticon' (Stuart et al 1979, Vitiello 1995). These order parameters are considered as corresponding to the code strength specifying the vacuum, the value of which is determined by the density of condensed NG bosons. In turn, information storage is proposed to be represented by coding of the ground state via symmetron condensation.

We have described the vacuum state in a conventional QM–sense, but now we mention an alternative characterization following Vitiello (1995, 2003). Firstly, on denoting the DWQ by $A(k)$ for some k, the number $N(A)$ for all k of the $A(k)$–modes in the vacuum state $|0(N)\rangle$, is taken to define a coding of information relative to the order parameters. Taking a time reversal $\widetilde{A}(k)$ of the copies of the $A(k)$, the vacuum state is then characterized by setting $N(A) - N(\widetilde{A}) = 0$, for all k . The same applies to differing values of the code $N(A)$, that is, all ground states for which $N'(A) \neq N(A)$. It is proposed that the brain ground state is the entirety of memory states $|0(N)\rangle$, for all N, and further, memory is manifestly how the brain may accomodate a multitude of co-existent macroscopic quantum states.

Such a proposition commences with the premise that the brain forms an 'open system' and its environment, in the appropriate sense, forms the 'closure'. Given that the DWQ frequency depends on time t, modes $A(k,t), \widetilde{A}(k,t)$, are considered so that the coupled system of differences $A - \widetilde{A}$ is describable in terms of an oscillator frequency. In the continuum limit, the system of differences $A - \widetilde{A}$ also becomes closed. A finiteness of size for the corresponding domains implies then a transition through distinct vacuum states for a given t . In the presence of external stimuli, the reversal of time symmetry is broken and the purported dissipation results in multifold degenerate vacuua, in turn, resulting in a vast memory storage. Possibly the memory state $|0(N)\rangle$ as a finite temperature state corresponds to thermodynamic effects in brain activity, further suggesting that in view of increasing entropy, the thermodynamic arrows of time may have the same orientation as the psychological arrows as emergent in the dissipative process.

It is possible that there may be any number of experimental studies that could prove or disprove such a hypothesis. The question bears some similarity to the relationship between (brain) cortical versus thermodynamical phase transitions (Steyn–Ross et al. 2001): how cortical entropy varies under the effects of anesthetics from a state of consciousness (ordered phase transitions) to unconsciousness (disordered phase transitions). On the surface, such findings might suggest an 'emergent' process through objective time intervals, at least as far as clinical consciousness is concerned. The soliton–like mechanisms we have described are in essence derived from time evolution equations, and thus mathematically involve a time flow. However, the mechanisms by themselves do not explain away the questions of objective versus subjective time. The various models in which they feature, suggest how they may be tied to memory storage and retrieval, and to the irreversibility of consciousness. We can see the relevance of the mechanisms to objective time as manifest in the brain within certain cortical regions, but we cannot tie these (mechanisms) to the subjective feeling of time, as exemplified in the case of individuals suffering from time agnosia : the comprehension of time is altogether lost, although most normal mental processes may still function nevertheless.

6 Conclusion

We have discussed some mechanisms for solitonic interactions ambient to microtubular surfaces, suggesting possibilities for interaction between local EM–fields of electro–neural impulses and the cytoskeletal structure. The broader model suggests how these processes might actually recover an EM–field through the chain of events *EM–field* $\Longrightarrow$ *tubulin–tail solitons* $\Longrightarrow$ *exocytosis* $\Longrightarrow$ *EM–field*. This progression may be crucial towards understanding the neurobiological basis for mind and memory, as well

as for the possible implementation of quantum or semi–classical computational schemes which are to be assessed in a future work.

Acknowledgement

The authors wish to thank the referees for comments.

References

[1] E. Abdalla, B. Maroufi, B. C. Melgar and M. B. Sedra. Information transport by sine–Gordon solitons in microtubules, *Physica* A **301** (2001), 169–173.

[2] M. J. Ablowitz and P. A. Clarkson. *Solitons, Nonlinear Evolution Equations and Inverse Scattering*, Cambridge Univ. Press, Cambridge UK 1991.

[3] L. A. Amos. Focusing on microtubules, *Current Opinion in Structural Biology* **10** (2000), 236–241.

[4] S. E. Arnold, V. M. Lee, R. E. Gur and J. Q. Trojanowski. Abnormal expression of two microtubule–associated proteins (MAP2 and MAP5) in specific subfields of the hippocampal formation in schizophrenia, *Proc. Nat. Acad. Sci. USA* **88** (1991), 10850–10854.

[5] F. Beck and J. C. Eccles. Quantum aspects of brain activity and the role of consciousness, *Proc. Natl. Acad. Sci. USA* **89** (1992), 11357–11361.

[6] N. Brose. Synaptic cell adhesion proteins and synaptogenesis in the mammalian central nervous system, *Naturwissenschaften* **86** (1999), 516–524.

[7] F. Calogero and A. Degasperis. *Spectral Transform and Solitons: Tools to Solve and Investigate Nonlinear Evolution Equations*, North Holland, New York 1982.

[8] E. R. Chapman. Synaptotagmin. A $Ca2^{+}$ sensor that triggers exocytosis?, *Nat. Rev. Mol. Cell Biol.* **3** (2002), 1–11.

[9] K. C. Chou, C. T. Zhang and G. M. Maggiora. Solitary wave dynamics as a mechanism for explaining the internal mechanism during microtubule growth, *Biopolymers* **34** (1994), 143–153.

[10] A. S. Davydov. *Biology and Quantum Mechanics*, Pergamon Press, Oxford 1982.

[11] A. S. Davydov. *Solitons in Molecular Systems*, Kluwer, Dordrecht 1991.

[12] C. Dean and T. Dresbach. Neuroligins and neurexins: linking cell adhesion, synapse formation and cognitive function, *Trends in Neuroscience* **29** (2006), 21–29.

[13] E. Del Giudice, S. Doglia and M. Milani. Self–focusing and ponderomotive forces of coherent electric waves–a mechanism for cytoskeleton formation and dynamics, in *Coherent excitations in biological systems* (ed. H. Frölich and F. Kremer) Springer–Verlag, 1983.

[14] E. Del Giudice, S. Doglia, M. Milani and G. Vitiello. Electromagnetic field and spontaneous symmetry breaking in biological matter, *Nucl. Phys.* B **275** (1986), 185–199.

[15] E. Del Giudice, G. Preparata and G. Vitiello. Water as a free electric dipole laser, *Phys. Rev. Lett.* **61** (1988), 1085–1088.

[16] S. V. Dmitriev, T. Shigenari, A. A. Vasiliev and A. E. Miroshnichenko. Effect of discreteness on a sine–Gordon three soliton solution, *Phys. Lett.* A **246** (1998), 129–134.

[17] R. K. Dodd, J. C. Eilbeck, J. D. Gibbon and H. C. Morris. *Solitons and Nonlinear Wave Equations*, Academic Press, New York 1982.

[18] H. Fröhlich. The extraordinary dielectric properties of biological materials and the action of enzymes, Proc. Natl. Acad. Sci. USA **72** (1975), 4211–4215.

[19] H. Fröhlich. Long–range coherence and energy storage in biological systems, *Int. J. Quantum Chem.* **2** (1968), 641–649.

[20] D. Georgiev. Electric and magnetic fields inside neurons and their impact upon the cytoskeletal microtubules, (2003a) http://cogprints.org/3190/

[21] D. Georgiev. The β–neurexin–neuroligin–1 interneuronal intrasynaptic adhesion is essential for quantum brain dynamics, (2003b) http://arXiv.org/abs/quant-ph/0207093

[22] D. Georgiev. On the dynamic timescale of mind–brain interaction, (2003c) in proceedings of *Quantum Mind II : Consciousness, Quantum Physics and the Brain* 15–19 March 2003, The Convention Center, Tucson Arizona. http://cogprints.org/4463/

[23] D. Georgiev. Solitonic effects of the local electromagnetic field on neuronal microtubules–tubulin tail sine–Gordon solitons could control MAP attachment and microtubule motor protein function, (2004) http://cogprints.org/3894/

[24] D. Georgiev, S. Papaioanou, and J. F. Glazebrook. Neuronic system inside neurons: molecular biology and biophysics of neuronal microtubules, *Biomedical Rev.* **15** (2004), 67–75.

[25] W. Grundler and F. Keilmann. Sharp resonances in yeast growth proved nonthermal sensitivity to microwaves, *Phys. Rev. Letts.* **51** (1983), 1214–1216.

[26] S. Hagan, S. R. Hameroff and J. A. Tuszyński. Quantum computation in brain microtubules? Decoherence and biological feasibility, *Phys. Rev.* E **65**(2002), 061901, 1–11.

[27] S. R. Hameroff and R. C. Watt. Information processing in microtubules, *J. Theor. Biology* **98** (1982), 549–561.

[28] R. Heald and E. Nogales. Microtubule dynamics, *J. Cell Sci.* **115** (2002), 3–4.

[29] N. Hirokawa, K. Sobue, K. Kanda, A. Harada and H. Yorifuji. The cytoskeletal architecture of the presynaptic terminal and molecular structure of synapsin-1, *J. Cell Biol.* **108** (2002), 111–126.

[30] A. Honda, M. Yamada, H. Saisu, H. Takahashi, K. J. Mori and T. Abe. Direct Ca^{2+}–dependent interaction between tubulin and synaptotagmin-1. A possible mechanism for attaching synaptic vesicles to microtubules, *J. Biol. Chem.* **277** (2002), 20234–20242.

[31] M. Jibu, S. Hagan, S. R. Hameroff, K. H. Pribram, and K. Yasue. Quantum optical coherence in cytoskeletal microtubules : Implications for brain function, *Biosystems* **32** (1994), 195–209.

[32] M. Jibu and K. Yasue. What is mind? – Quantum theory of evanescent photons in brain as quantum theory of consciousness, *Informatica* **21** (1997), 471–490.

[33] M. Jibu, K. Yasue and K. H. Pribram. From conscious experience to memory storage and retrieval: The role of quantum brain dynamics and boson condensation of evanescent photons, *Int. J. Mod. Phys.* B **10** (1996), 1735–1754.

[34] M. Jibu, K. Yasue and S. Hagan. Evanescent (tunneling) photon and cellular vision, *Biosystems* **42** (1997), 65–73.

[35] M. A. Jiminez, J. A. Evangelio, C. Aranda, A. Lopez-Braet, D. Andreu, M. Rico, R. Lagos, J. M. Andreu and O. Monasterio. Helicity of α (404-451) and β (394-445) tubulin C–terminal recombinant peptides, *Protein Science* **8** (1999), 788–799.

[36] N. E. Mavromatos, A. Mershin and D. V. Nanopoulos. QED–cavity model of microtubules implies dissipationless energy transfer and biological quantum teleportation, *Int. J. Modern Physics* B **16** (2002), 3623—3642.

[37] A. E. Miroshnichenko, S. V. Dmitriev, A. A. Vasiliev and T. Shigenari. Inelastic three–soliton collisions in a weakly discrete sine–Gordon system, *Nonlinearity* **13** (2000), 837–848.

[38] T. Miwa, M. Jimbo and E. Date. *Solitons–Differential equations, symmetries and infinite dimensional algebras*, Cambridge Tracts in Math. **135**, Cambridge University Press, 2000.

[39] E. Nogales, S. G. Wolf and K. H. Downing. Structure of the α/β tubulin dimer by electron crystallography, *Nature* **391** (1998), 199–203.

[40] D. L. Purich. Enzyme catalysis: a new definition accounting for noncovalent substrate- and product-like states, *Trends Biochem. Sci.* **26** (2001), 417-421.

[41] L. M. Ricciardi and H. Umezawa. Brain and physics of many–body problems, *Kybernetik* **4** (1967), 44–48.

[42] F. M. Russell, Y. Zolotaryuk and J. C. Eilbeck. Moving breathers in a chain of magnetic pendulums, *Phys. Rev.* B **55** (1997), 6304–6308.

[43] D. L. Sackett. Structure and function in the tubulin dimer and the role of the acid carboxyl terminus, *Subcellular Biochemistry-Proteins: Structure, function and engineering* **24** (1995), 255–302.

[44] M. V. Satarić and J. A. Tuszyński. Relationship between the nonlinear ferroelectric and liquid crystal models for microtubules *Phys. Rev.* E **67** (2003), 011901, 1–11.

[45] M. V. Satarić, R. Zakula, Z. Ivic and J. Tuszyński. Influence of a solitonic mechanism on the process of chemical catalysis, *J. Molecular Electronics* **7** (1991), 39–46.

[46] A. C. Scott. Davydov's solitons, *Phys Rev. Lett.* **217** (1992), 1–67.

[47] G. Skiniotis, J. C. Cochran, J. Mueller, E. Mandelkow, S. P. Gilbert and A. Hoenger. Modulation of kinesin binding by the C–termini of tubulin, *The EMBO J.* **23** (2004), 989–999.

[48] E. Sontag, V. Nunbhakdi-Craig, G. Lee, R. Brandt, C. Kamibayashi, J. Kuret, C. L. White, M. C. Mumby and G. S. Bloom. Molecular Interactions among Protein Phosphatase 2A, Tau, and Microtubules: Implications for the regulation of tau phosphorylation and the development of tauopathies, *J. Biol. Chem.* **274** (1999), 25490-25498.

[49] M. L. Steyn–Ross, D. A. Steyn–Ross, J. W. Sleigh and L. C. Wilcocks. Toward a theory of the general–anesthetic–induced phase transition of the cerebral cortex. I. A thermodynamics analogy *Phys. Rev.* E **64** (2001), 011917, 1–16.

[50] C. I. J. M. Stuart, M. Takahashi and H. Umezawa. Mixed–system brain dynamics. Neural memory as a macroscopic ordered state, *Foundations of Physics* **9** (1979), 301–327.

[51] M. J. Sutcliffe and N. S. Scrutton. Enzyme catalysis: over-the-barrier or through-the-barrier? *Trends Biochem. Sci.* **25** (2000), 405-408.

[52] S. Takada and H. Nakamura. Wentzel-Kramer-Brillouin theory of multi-dimensional tunneling: General theory for energy splitting. J. Chem. Phys. **100** (1994), 98-113.

[53] S. Takada and H. Nakamura. Effects of vibrational excitation on multi-dimensional tunneling: General study and proton tunneling in tropolone, *J. Chem. Phys.* **102** (1995), 3977-3992.

[54] G. Vitiello. Quantum dissipation and information : a route to consciousness modeling, *Neuroquantology* **2** (2003), 266–279.

[55] G. Vitiello. Dissipation and memory capacity in the quantum brain model, *Int. J. Mod. Phys.* B **9** (1995), 973–989.

Actors as a Coordinating Model of Computation

N. Raja and R.K. Shyamasundar
School of Technology & Computer Science
Tata Institute of Fundamental Research
Mumbai 400 005, India
Email: {raja, shyam}@tifr.res.in

Keywords: Actors, agents, pi-calculus

Received: November 24, 2004

This paper relates two prominent models of concurrent computation, namely Actors *and the* π-calculus. *We build on a thesis that proclaims – Actors enact the role of a* coordinating *model of computation. We enrich the Actor model by defining a mechanism for achieving a higher level of abstraction. This helps in reasoning with collections of Actors termed* Actor Troupes. *We identify a notion of* interaction equivalence *between Actor Troupes; and provide a semantic foundation for the enriched Actor model, in terms of the* π*-calculus – which has emerged as the canonical process calculus for the semantic analysis of object-based concurrent systems. Furthermore, we show that the algebraic notion of barbed bisimilarity in the* π*-calculus, corresponds precisely to interaction equivalence of the corresponding Actor Troupes.*

Povzetek: Predstavljena sta dva računska modela - z akterji in π*-računi.*

1 Introduction

There has been an exponential increase in the number of new paradigms that are being proposed to model concurrent distributed computation. This number far exceeds the corresponding figure for sequential computation. (This is understandable as the basic sequential architecture, the von Neumann model, has essentially remained unchanged.) Despite this fact, concurrency is less well understood than sequential computation. It would be an understatement to say that there is an urgent need for a *coordinating* model of computation which can interconnect and unify the varied paradigms. The solution to the problem of finding such a model should commence with a search among the existing models, rather than in the immediate proposal of yet another novel paradigm. The factors guiding us in this search should be at least threefold – expressive power of the model; relative efficiency of execution of the model with respect to paradigms based on orthogonal features; and the demonstrated existence of a sound semantic basis for the model.

This paper is based on a thesis that proclaims, *Actors* enact the role of a coordinating model of computation. Existing evidence corroborating such a thesis is the following: Actors have been shown to be an expressive medium which can easily mimic other paradigms [2, 3]; and the execution efficiency of Actors has been shown to be as efficient [7] as the shared memory models (which are orthogonal to the message passing paradigm of Actors). The only factor that remains is the provision of a semantic foundation. This paper aims to further the above thesis by taking a step in the direction of providing a semantic basis to Actors.

Among the various process-calculus approaches to the algebraic analysis of concurrency, the π-calculus is conspicuous by its success. With a well-developed body of theoretical work supporting it, the π-calculus has attained a canonical status in the semantic frameworks of object-based concurrent systems, analogous to the λ-calculus in sequential programming. The semantic power of the π-calculus has been demonstrated in many ways – by encoding the λ-calculus [26]; by embedding various data-types [23]; by translating higher-order primitives [23]; and by using it as a semantic domain for various object-based concurrent languages [31]. All these facts provide a compelling basis to choose the π-calculus as a semantic foundation for Actors.

This paper relates two prominent models of concurrent computation – Actors and the π-calculus – in a precise way, and has the following significant contributions:

1. It argues that Actors can play the role of a coordinating model of computation, due to the simplicity and inherent flexibility of the Actor primitives.

2. It enriches the Actor model by defining the notion of an *Actor Troupe* – which is a mechanism to achieve a higher level of abstraction – by restricting the visibility of some Actors.

3. It provides a semantic foundation to the enriched Actor model by mapping it to the π-calculus – which has emerged as the canonical process calculus for the semantic analysis of object-based concurrent systems.

4. It identifies an equivalence relation, *interaction equivalence* on Actor Troupes, and shows that under the translation this corresponds to the algebraic notion of barbed bisimilarity in the π-calculus semantics.

The rest of this paper is organized as follows: Section 2 gives an introduction to the Actor model of computation; Section 3 introduces the basic π-calculus notions required for the purposes of this paper; Section 4 develops a higher level of abstraction called Actor Troupe and defines a notion of equivalence between them; Section 5 demonstrates the translation process from actor systems to the π-calculus; Section 6 shows that the embedding is semantics preserving; Section 7 reviews related work; and finally Section 8 examines avenues for further research.

2 Actor Model of Computation

Actors [15] form one of the earliest proposed models of concurrent distributed computation. They include very few primitive constructs, but serve as a framework for studying various issues in computation. An *actor system* consists of a finite set of three basic entities – *actors, messages,* and *behavior definitions*.

The formal abstract syntax is shown in Figure 1. (Note that in Figure 1 the following non-terminals: $< actorName >$,$< behaviorName >$, $< method >$, and $< var >$ are all identifiers for which there are no corresponding production rules. Furthermore, although $< acqList >$ and $< parameters >$ are defined by the same production rule, they are distinguished for the sake of clarity of exposition.)

Actors embody the spirit of objects. Every actor has a unique name, and a unique mailbox address which remains unaltered throughout the lifetime of the actor. In other words, there is an implicit injective function mapping actor names to mailbox addresses. (We shall make this function explicit in the translation we provide.) It is at this address that the actor receives messages. The body of an actor consists of a *state,* an *acquaintance list,* and a collection of *methods* with relevant *actions*. The *state* – encapsulated, persistent, and private – is made up of variables, which in turn contain references to other actors. The *acquaintance list* is a collection of names of actors which are known to the present actor at the time of its creation. It is important to note that, the name of an actor may not be known to all other actors in the system. Conversely, an actor may not be aware of the names of all the other actors. *Messages* can be sent only to those actors whose addresses are known. Apart from the addresses that make up the acquaintance list initially, an actor might receive the addresses of more actors through the contents of incoming messages. Furthermore, every actor is also aware of its own mailbox address. Thus every actor can send messages to itself. The address of an actor is contained in its own acquaintance list. In particular, we stipulate that it occurs as the last element of its acquaintance list. Thus the acquaintance list of an actor is never empty, and always contains at least a single element namely its own address. Each *method* has a set of parameters, which is received along with the method name to be serviced. Corresponding to each method, is a set of *actions* that the actor performs. We shall explain the actions after we deal with *behavior definitions*.

Behavior definitions are parametric actor definitions, parameterized over the state variables and acquaintance names. Behavior definitions by themselves are not actors – they provide templates for the creation of new actors. The parameters corresponding to the state and the acquaintances have to be specified and instantiated at the time of creating new actors.

Messages are the driving force of an actor system. This is due to the fact that computation in an actor system is carried out in response to messages received by the actors of the system. Every message has two distinct parts – *destination address* and *message contents*. The destination address is the mailbox address of an actor in the system to which the message is to be delivered. The message content comprises a *method name* and corresponding *parameters*. The actor at the destination is known to respond to this method name. The parameters comprise names of other actors. Message passing in actors is point to point and asynchronous. Message delivery is guaranteed but the despatch order need not be preserved even when we consider the arrival order of a sequence of messages addressed to the same actor. The guarantee of message delivery forms a type of fairness assumption [2].

Each instance of an actor can receive only one message. In response to a message received, which requires one of the methods of an actor to be processed, an actor may change its state, and may also perform a finite number of the following actions:

Send a message to another actor whose mail address is known;

Create a new actor using a behavior template, by providing all the parameters required for initialization;

Become an actor, which specifies the replacement behavior to come into effect, when the next message is processed.

It may be noted that the next incoming message at the same mail address is processed by another instance of the actor with the specified replacement behavior. The processing of the current message need not be completed before the replacement is specified. During start up time, an actor system consists of a collection of behavior definitions, together with a declaration of initial actors and messages. The actor system evolves in response to the messages sent to the system.

Example 2.1 (Actor R)**.** Actor R has an empty state, an acquaintance list comprising three actor names, and it processes the method name f. It takes requests for possibly transforming data by the method name f, and sends the result to the first actor on its acquaintance list by the method name g. It specifies an identical replacement behavior to replace itself. The template of actor R is given by the following behavior definition, which is parametrized by the two actor names on its acquaintance list:

$$
\begin{array}{rcl}
< System > & ::= & \{< actorName > \ \leftarrow \ < behaviorDef >\}^* \\
< behaviorDef > & ::= & \texttt{Bdef} < behaviorName > \texttt{with} < state > \texttt{and} < acqList > \\
 & & \quad < method > (< parameters >) \ \rightarrow \ \{< actions >\}^* \\
 & & \qquad \vdots \\
 & & \texttt{endBdef} \\
< state > & ::= & \{< var > \ \leftarrow \ < actorname >\}^* \\
< acqList > & ::= & \{< actorName >\}^* \\
< parameters > & ::= & \{< actorName >\}^* \\
< actions > & ::= & \quad \texttt{become} < behaviorName > \texttt{with} < state > \texttt{and} < acqList > \\
 & & | \ \texttt{create} < behaviorName > \texttt{with} < state > \texttt{and} < acqList > \\
 & & | \ \texttt{send} < method > (< parameters >) \texttt{to} < actorName > \\
 & & \quad \text{where} < actorName > \in < acqList >
\end{array}
$$

Figure 1: An Abstract Syntax for a System of Actors.

Bdef
R with $<>$ and a', x', r'
$f(d) \ \rightarrow$ send $g(d)$ to a'
become R with $<>$ and a', x', r'
endBdef
An instance of actor R can be created by the following message:
create R with $<>$ and a', x', r'

As mentioned before, the state is made up of variables which in turn are represented by actors. The state may contain integers or other data-types. However for the sake of simplicity we consider only empty state configurations in the examples in this paper.

Example 2.2 (Actor A)**.** Actor A has an empty state, an acquaintance list comprising two actor names, and it processes the method name g. It takes requests for possibly transforming data by the method name g, and sends the result to the first actor on its acquaintance list by the method name h. It specifies an identical replacement behavior to replace itself. The template of actor A is given by the following behavior definition, which is parametrized by the actor name on its acquaintance list:
Bdef
A with $<>$ and b', a'
$g(d) \ \rightarrow$ send $h(d)$ to b'
become A with $<>$ and b', a'
endBdef
An instance of actor A can be created by the following message:
create A with $<>$ and b', a'

Example 2.3 (Actor A_1)**.** Actor A_1 has an empty state, an acquaintance list comprising two actor names, and it processes the method name g. It accepts requests for possibly transforming data by the method name g, and sends the result to the first actor on its acquaintance list by the method name m. It specifies an identical replacement behavior to replace itself. The template of actor A_1 is given by the following behavior definition, which is parametrized by the two actor names on its acquaintance list:
Bdef
A_1 with $<>$ and x', a'
$g(d) \ \rightarrow$ send $m(d)$ to x'
become A' with $<>$ and x', a'
endBdef
An instance of actor A_1 can be created by the following message:
create A_1 with $<>$ and x', a'

Example 2.4 (Actor B)**.** Actor B has an empty state, an acquaintance list comprising two actor names, and it processes the method name h. It accepts data by the method name h, and sends the untransformed data to the first actor on its acquaintance list by the method name m. It specifies an identical replacement behavior to replace itself. The template of actor B is given by the following behavior definition, which is parametrized by the two actor names on its acquaintance list:
Bdef
B with $<>$ and x', b'
$h(d) \ \rightarrow$ send $m(d)$ to x'
become B with $<>$ and x', b'
endBdef
An instance of actor B can be created by the following message:
create B with $<>$ and x', b'

Example 2.5 (Actor X)**.** Actor X has an empty state, an acquaintance list comprising two actor names, and responds to the method name m. It accepts data by the method name m, does nothing with the data, and specifies an identical replacement behavior to replace itself. The template of actor X is given by the following behavior def-

inition, which is parametrized by the two actor names on its acquaintance list:

`Bdef`
X `with <> and` r', x'
$m(d)$ $\rightarrow$ `become` X `with <> and` r', x'
`endBdef`

An instance of actor X can be created by the following message:

`create` X `with <> and` r', x'

3 The Polyadic π-calculus

In this section, we include a brief review of the polyadic π-calculus (PPC) [24, 23, 25] and also introduce the specific syntax that we use for it in this paper.

Following Milner's idea [22], a number of calculi for concurrent computation have been proposed, where the communication mechanisms are similar. Communication consists of synchronously sending and receiving messages through a shared labeled channel. PPC [23, 25, 9] is a model of concurrent computation that supports process mobility by naming and passing channels. It consciously forbids the transmission of processes as messages. One of its goals is to demonstrate that in some sense it is sufficiently powerful to allow only channel names to be the content of communications. PPC has two kinds of entities – *names* (*channels*) and *processes* (*agents*).

Definition 3.1 (Names and Processes)**.** Names $(x, y, \ldots \in \mathcal{X})$ *are atomic entities while* Processes $(P, Q, \ldots \in \mathcal{P})$ *have the following structure:*

$$P \ ::= \ N \ \mid \ (P|Q) \ \mid \ !P \ \mid \ (\nu x)P$$

where, Normal Processes $(M, N, \cdots \in \mathcal{N})$ *are defined as:*

$$N \ ::= \ \pi.P \ \mid \ 0 \ \mid \ M + N$$

and, the Prefix (π)*, is given by:*

$$\pi ::= x(\tilde{y}) \ \mid \ \overline{x}[\tilde{y}]$$

where, $\tilde{y}$ refers to a finite sequence of names.

The term 0 represents an inactive process, which cannot perform any action. We shall omit the trailing ".0" from process terms. Basic actions in PPC constitute *sending* or *receiving* names on channels. The construct $x(y)$ (called an *input prefix*) represents an atomic action, where name x *binds* name y. The process term $x(y).P$ waits for a name to be transmitted along channel x, *substitutes* the *received* name for all free occurrences of y in P, and then triggers P. The construct $\overline{x}[y]$ (representing an atomic action) *outputs* the name y along x, but *does not bind* name y. The form $P|Q$ denotes that P and Q are *concurrently* active, independent, and can *communicate*. The form $M + N$ means that the process can indulge in precisely one of the alternatives, given by M and N, for communication. Operator "!" is called *replication*, and $!P$ denotes $P|!P$. Finally, $(\nu x)P$ *restricts* the use of name x to P. Apart from input prefix, "ν" is another mechanism for *binding* names within a process term in API. Operator "ν" may also be thought of as *creating* new channels. As both mechanisms – input prefix and ν – bind names, we define $BoundNames(P)$ as those names with a bound occurrence in P, and $FreeNames(P)$ as those with a not bound occurrence in P. The basic rule of computation in PPC is provided by the *parallel composition* of processes which communicate along the same channel.

The operational semantics of PPC is given in two stages. A structural congruence is first defined over processes, as shown below; and then a reduction relation is defined as shown in Figure 2. Note that the rules do not allow reduction under prefix, sum, or replication.

Definition 3.2 (Structural Congruence)**.** The relation '$\equiv$' is the smallest congruence relation over processes such that the following laws hold:

1. Processes are identified if they only differ by a change of bound names.
2. $(\mathcal{N}/\equiv, +, 0)$ is an abelian monoid.
3. $(\mathcal{P}/\equiv, |, 0)$ is an abelian monoid.
4. $!P \equiv P|!P$
5. $(\nu x)0 \equiv 0, (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
6. If $x \notin FreeNames(P)$ then $(\nu x)(P|Q) \equiv P|(\nu x)Q$

Furthermore, the synchronous π-calculus outlined above, can be suitably modified to yield the asynchronous π-calculus [10, 17]. The word 'asynchrony' in this calculus, means that message output is non-blocking. This is ensured by restricting the formation of a term $\overline{x}[\tilde{y}].P$ in the π-calculus to the case where P is a 'nil' process. However, it has been shown that under certain natural assumptions, the asynchronous version is strictly less expressive than the synchronous one [29].

PPC allows for the definition of a variety of equivalences between processes. Following Milner [27, 23], we define the notion of barbed bisimulation for PPC:

Definition 3.4 (Unguarded Process)**.** *A process Q occurs* unguarded *in P if it has some occurrence in P which is not under a prefix.*

Definition 3.5 (Observable Action)**.** *A process P can perform an* observable action *at x, written $P \downarrow_x$, if for some $x, \tilde{y}$, either the input prefix $x(\tilde{y}).Q$ or the output prefix $\overline{x}[\tilde{y}].Q$ occurs unguarded in P with x unrestricted.*

Let "$\rightarrow^*$" denote the transitive reflexive closure of "$\rightarrow$". We shall use $Q \rightarrow^* \downarrow_x$ to denote "$Q \rightarrow^* Q'$ for some Q', and $Q' \downarrow_x$".

Definition 3.6 (Barbed Bisimulation)**.** *A relation R_w over processes, is a* barbed simulation*, if $P \ R_w \ Q$ implies:*

Definition 3.3 (Reduction Relation)**.** The reduction relation $\rightarrow$ over processes is the smallest relation satisfying the following rules:

$$\text{Comm} \quad (\ldots + x(\tilde{y}).P) \mid (\ldots + \overline{x}[\tilde{z}].Q) \;\rightarrow\; P\{\tilde{y} \leftarrow \tilde{z}\} \mid Q$$

$$\text{Par} \quad \frac{P \rightarrow P'}{(P|Q) \rightarrow (P'|Q)}$$

$$\text{Struct} \quad \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

$$\text{Res} \quad \frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'}$$

Figure 2: Reduction Relation in PPC

1. *For each x, $P \downarrow_x$ implies $Q \rightarrow^* \downarrow_x$.*
2. *If $P \rightarrow P'$ then $Q \rightarrow^* Q'$ and $P'\ R_w\ Q'$;*

The relation R_w is a barbed bisimulation *if R and R^{-1} are* barbed simulations*. Processes P and Q are* barbed-bisimilar*, if $P\ R_w\ Q$ for some* barbed bisimulation R_w.

4 Actor Troupes: A Higher Level of Abstraction

An important requirement of a potential coordinating paradigm, which seeks to unify diverse models of concurrent computation, is the inherent support for various levels of abstraction. A desirable level of abstraction would be one which helps in dealing with bigger collections of Actors as if they were a single unit. Such a higher level of abstraction on actor systems can be defined by restricting and specifying the interface of actor systems (rather than individual actors) with the external world. We introduce the abstraction of *Actor Troupes* and also formally define a notion of *Interaction Equivalence* with respect to this abstraction.

Definition 4.1 (Actor Troupe)**.** *An* Actor Troupe *comprises actors, behavior definitions, and messages which satisfy the following conditions:*

1. *Certain actors within the troupe, declared* Receptionists*, are the only components whose existence is visible to the external world. Furthermore, only the mail addresses and the method names of Receptionists are visible outside.*
2. *Conversely, the components comprising the troupe are aware of the mail addresses and method names of a certain collection of* External actors *(fixed a priori) which are not members of the troupe. (Actor A is said to be aware of Actor B, if the acquaintance list of A contains the mail address of B).*

The notion of Actor Troupes helps in the modular development and composition of Actor programs, since it specifies the interface of a collection of actors with the external world. This can be observed by the fact that in any Actor Troupe only the Receptionists are capable of receiving messages from the external world. Furthermore, the *External Actors* are the only ones which can potentially receive messages from any member comprising the Actor Troupe. By adapting Milner's idea of experiments [22], we define a notion of *Interaction Equivalence* between actor troupes.

Definition 4.2 (Interaction Equivalence)**.** *Actor Troupes T_1 and T_2 are said to be interaction equivalent, when there is a nonempty relation R_a over Actor Troupes such that $T_1\ R_a\ T_2$ implies:*

1. *The* Receptionists *of T_1 and T_2 have the same locations, and respond to the same set of messages.*
2. *The* External Actors *that are known to T_1 and T_2 have the same locations, and respond to the same set of messages. (An external actor X is said to be known to Troupe T if the mail address of X is contained in the acquaintance list of any of the actors comprising Troupe T).*
3. *For every input message I sent to T_1 and T_2, where the message content of I does not contain the mail address of any component external to the troupe; the set of output messages O emanating from T_1 and T_2 are the same, and the message contents of O do not contain the mail address of any component internal to the troupe.*

4. *The Actor Troupes T_1' and T_2' which result from T_1 and T_2 respectively after they process message I, are in the relation $T_1'\ R_a\ T_2'$.*

The purpose of the definition is to ensure that if the above notion of equivalence is satisfied by the Actor Troupes T and T', then any occurrences of the Troupe T in an Actor program, may be replaced by the Troupe T' without any change in the meaning of the program. The above notion of equivalence does not permit the mail addresses of components to be carried in the messages because such communications violate and destroy the encapsulation of Actor Troupes.

Example 4.1 (Troupe T)**.** Comprises actors – R, A, and B, where R is the *Receptionist* – with a reference to an *External Actor* X. Actor R takes requests for transforming data by the message f, and sends the result to actor A by the message g. Actor A transforms the data further and sends the result to B by the message h. Actor B passes it on unchanged to the external actor X by the message m.

Example 4.2 (Troupe T_1)**.** Similar to Troupe T except for actor A which is replaced by actor A_1, which returns its results directly to the external actor.

5 Semantic Foundation for Actors

In this section, we present a semantic foundation for Actors in terms of the polyadic π-calculus (PPC). We shall concentrate on constructs that are unique to the Actor model. The translation uses, for each syntactic category, a mapping $[\![.]\!]$ from the Actor grammar to PPC process-terms. The translation also employs a set of auxiliary functions for "book-keeping" purposes, namely for maintaining correspondences between entities in the Actor formalism and PPC-names. Figure 3 is a formal description of the semantic function. The translation is explained in detail below.

Recall that in actor systems, behavior definitions are not actors. They are templates which enable creating actors. However, in the translation, we shall associate PPC processes with behavior definitions, and also with actors. The translation of *Behavior definitions* is given by:

$$[\![\texttt{BehaviorDef}]\!] \ = \ !\, l(\vec{s}, \vec{a}, i).([\![\texttt{Actor}]\!])$$

Behavior definitions are mapped to process terms containing the *Bang* operator, in order to model a resource which can create a new actor instance, every time it is requested. The PPC name l represents the location of the process term, and is statically determined by the function

$$\beta \ : \ BehaviourName \longrightarrow \text{PPC-}\textit{Name}$$

However, the association of actor addresses with PPC-names will be modeled by a dynamic mechanism. The tuples $\vec{s}$ and $\vec{a}$ are formal place holders for the *state* and *acquaintance* parameters which are supplied with each `create` and `become` request. The parameter i, again supplied by incoming requests, represents the address at which the newly created actor instance (or the replacement behavior) is to be located.

In the actor model the `create` primitive is endowed with an implicit capability of generating globally unique actor addresses on a purely local basis. This mechanism is modeled using the properties of the operator "ν" of PPC:

$$[\![\texttt{create}\ BehaviorName\ \texttt{with}\ state\ \texttt{and}\ acqList]\!]$$

$$= \ (\nu\, i)(\bar{l}[\vec{s}, \vec{a}, i])$$

The PPC-name l represents the location of the process term corresponding to the behavior definition which receives and services the `create` action. It is given as before by $\beta(BehaviourName) = l$. The function

$$\sigma \ : \ state \longrightarrow \text{PPC-}\textit{Names}$$

which maintains the correspondence between the *state* and PPC-names, gives $\sigma(state) = \vec{s}$. Similarly the correspondence between *acqList* and PPC-names is maintained by the function

$$\tau \ : \ acqList \longrightarrow \text{PPC-}\textit{Names}$$

which gives $\tau(acqList) = \vec{a}$.

The newly generated name i is guaranteed to be globally unique by the semantics of PPC [25, 23]. This can be explained as follows. Consider a PPC process term, $(\nu y)\ Q$, where the name y is bound by the restriction operator. The restriction mechanism combines two distinct roles in one operator. Firstly, it hides all interactions on the name y within Q, thus preventing external processes from interfering on communications along channel y. In effect, it declares a local name y, for use exclusively within Q. In this role it is similar to the '*let-in*' construct in programming languages, and the *hiding* mechanism of CCS. Secondly, the restriction operator also ensures that the name y is distinct from all external names too [25, 23]. This follows from the fact that PPC allows local names to be communicated to external processes. A term of the form $(\nu y)(\overline{x}[y].Q)$ can be viewed as simultaneously creating and transmitting a new name. The name y is at first local to Q and becomes active after the transmission. Thus the operator "ν" is a mechanism which creates globally unique channel names.

The "book-keeping" associated with the dynamic creation of addresses is managed in the semantic domain by the function

$$\alpha \ : \ \text{Actor} \longrightarrow \text{PPC-}\textit{Name}$$

whose definition enlarges after every creation of an actor instance.

The onus of providing a globally unique address for the newly created actor lies with the actor which issues the create command. This feature of the semantics guarantees the requirement of actor systems that at first the location of a

newly created actor is known only to its parent. Any other actor in the system becomes aware of the new arrival only on receiving the address of the newcomer. As we shall see later, this is a powerful mechanism to achieve modularity in actor systems by keeping certain actors hidden from the view of certain other actors.

The translation of `become` action is similar to that of `create` action:

$$[\![\texttt{become}\ BehaviorName\ \texttt{with}\ state\ \texttt{and}\ acqList]\!]$$
$$= \bar{l}[\vec{s}, \vec{a}, i]$$

but more simple. In the case of `become`, no new actor address needs to be generated because, the replacement is to be at the same location as its parent (namely '*i*'), even though both may exist concurrently. However, the parent cannot accept messages any longer. It is worthwhile to point out that we have modeled the `become` action exactly as envisaged by the pioneering work on Actors [15]. We place absolutely no restriction on the type of replacement behavior an actor instance might specify. For example, an actor instance created from a behavior definition A could specify its replacement to be created from the behavior definition Z.

The translation of the *actor instance* created by the *behavior definition* is:

$$[\![\texttt{Actor}]\!] = (\nu\ \vec{s})(\nu\ \vec{m})(\bar{i}[\vec{m}].\Sigma\vec{m}(\vec{p})[\![actions]\!])$$

The *actor instance* resides at location i, and its *state* $\vec{s}$ is encapsulated as shown by the restriction operator. The *actor instance* creates a tuple of channels $\vec{m}$ – which has as many elements as the number of messages the actor responds to. The newly created channel names are accessible at its location i, and are used for receiving parameters corresponding to each of the messages that are available. However, a single instance of an actor can service only one message – as indicated by the summation operator of PPC. The translation of the `send` action:

$$[\![\texttt{send}\ method\ (parameters)\ \texttt{to}\ Actor]\!]$$
$$= i(\vec{m}).\overline{m_j}[\vec{p}]$$

shows that in order to execute the method m_j of actor i, the parameters $\vec{p}$ have to be sent on the corresponding channel name. The function

$$\mu_i\ :\ methods \longrightarrow \text{PPC-}\textit{Name}$$

which maintains the correspondence between the *methods* and PPC-names of actor i, gives $\mu_i(method) = m_j$. Also $\vec{m}$ is a list of all the PPC-names which can be used to access the different methods provided by the actor. Quite naturally we have $m_j \in \vec{m}$. Similarly the correspondence between *parameters* and PPC-names is maintained by the function

$$\rho\ :\ parameters \longrightarrow \text{PPC-}\textit{Names}$$

which gives $\rho(parameters) = \vec{p}$. Notice that the `send` action is serviced by actor instances, while the `create` and `become` actions are serviced by the behavior definitions.

In the pure actor formalism that we have considered, the entities *state*, *acqList*, and *parameters* refer to sequences of *Actor Names*. The translation of these entities is provided by the functions σ, τ, and ρ respectively. As explained earlier, these functions map the *Actor Names* pointed to by these entities to the corresponding PPC-names. Note that it is possible to add *integers* and other *data-types* to the actor formalism and translate them to PPC-processes [23, 26]. However, for the sake of simplicity we shall not consider the encoding of *data-types* in this paper.

Actor Troupes correspond to *Systems* of actors which satisfy the additional constraints imposed by Definition 4.1. These constraints can be easily imposed and verified by monitoring the messages from the troupe to the external world, and vice versa. The translation of *Actor Troupes* is then very similar to the translation of the *System* of actors explained till now in this section.

Example 5.1 (Translation of Troupe T)**.** Consider the Troupe T of example 4.1. Suppose that the templates of the actors A, B, R, and X are located at a, b, r, and x respectively. The PPC translations of the templates are as follows:

$$A \equiv !a(b', a').(\nu g)(\overline{a'}[g].g(d).b'(h).\overline{h}[d].\overline{a}[b', a'])$$
$$B \equiv !b(x', b').(\nu h)(\overline{b'}[h].h(d).x'(m).\overline{m}[d].\overline{b}[x', b'])$$
$$R \equiv !r(a', x', r').$$
$$(\nu f)(\overline{r'}[f].f(d).a'(g).\overline{g}[d].\overline{r}[a', x', r'])$$
$$X \equiv !x(r', x').(\nu m)(\overline{x'}[m].m(d).\overline{x}[r', x'])$$

The translations of the `create` operations which initialize the troupe T are as follows:

$$[\![\texttt{create}\ A\ \texttt{with}\ <>\ \texttt{and}\ b', a']\!] = (\nu\ a')\overline{a}[b', a']$$
$$[\![\texttt{create}\ B\ \texttt{with}\ <>\ \texttt{and}\ x', b']\!] = (\nu\ b')\overline{b}[x', b']$$
$$[\![\texttt{create}\ R\ \texttt{with}\ <>\ \texttt{and}\ a', x', r']\!]$$
$$= (\nu\ r')\overline{r}[a', x', r']$$
$$[\![\texttt{create}\ X\ \texttt{with}\ <>\ \texttt{and}\ r', x']\!] = (\nu\ x')\overline{x}[r', x']$$

As all the above four `create` operations are meant to initialize the same troupe T in the example under consideration, they can be combined with the translation of the behaviour definitions using parallel composition. Further evolution of the troupe is given in later examples.

Example 5.2 (Translation of Troupe T_1)**.** Consider the Troupe T_1 of example 4.2. The Troupe T_1 has been built by modifying Troupe T of example 4.1. The template A is replaced by the template A_1 at the same location as A; the template B is discarded; and the templates of R and X

$$[\![\texttt{System}]\!] = [\![\texttt{BehaviourDef}_1]\!] \mid \ldots \mid [\![\texttt{BehaviourDef}_n]\!]$$
with the auxiliary functions ($\alpha,\ \beta,\ \mu,\ \rho,\ \sigma$ and τ)

$$[\![\texttt{BehaviourDef}]\!] = !\, l(\vec{s}, \vec{a}, i).[\![\texttt{Actor}]\!]$$
where $\beta(BehaviourName) = l$

$$[\![\texttt{Actor}]\!] = (\nu\ \vec{s})(\nu\ \vec{m})(\bar{i}[\vec{m}].\Sigma\vec{m}(\vec{p})[\![actions]\!])$$

$$[\![\texttt{create}\ BehaviourName\ \texttt{with}\ state\ \texttt{and}\ acqList]\!] = (\nu i)(\bar{l}[\vec{s}, \vec{a}, i])$$
where $\beta(BehaviourName) = l;\ \sigma(state) = \vec{s};\ \tau(acqList) = \vec{a}$

$$[\![\texttt{become}\ BehaviourName\ \texttt{with}\ state\ \texttt{and}\ acqList]\!] = \bar{l}[\vec{s}, \vec{a}, i]$$
where $\beta(BehaviourName) = l;\ \sigma(state) = \vec{s};\ \tau(acqList) = \vec{a}$

$$[\![\texttt{send}\ method\ (parameters)\ \texttt{to}\ Actor]\!] = i(\vec{m}).\overline{m_j}[\vec{p}],\ \text{where}$$
$$\alpha(Actor) = i;\ \mu_i(method) = m_j,\ m_j \in \vec{m};\ \rho(parameters) = \vec{p}$$

Figure 3: Mapping Actors to PPC.

remain the same. The PPC process term corresponding to the template A_1 is given by

$$A_1 \equiv !a(x', a').(\nu g)(\overline{a'}[g].g(d).x'(m).\overline{m}[d].\overline{a}[x', a'])$$

The translation of the creation of an instance of actor A_1 is given by:
$[\![\texttt{create}\ A_1\ \texttt{with}\ <>\ \texttt{and}\ x', a']\!] = (\nu a')\overline{a}[x', a']$

5.1 Preserving fairness on message delivery

As we mentioned before, the guarantee of message delivery in the Actor model forms a type of fairness assumption [2]. Message delivery is guaranteed but the despatch order need not be preserved. In the semantic domain, the corresponding property which provides the means to preserve fairness, is given by the fact that the reduction rules of π-calculus ensure that allowed reductions do take place within an unbounded but finite number of reduction steps.

The semantics ensures the fairness condition on message delivery. This can be seen from the fact that the only cases where the *bang* ("!") operators arise in the semantic mapping are from the translations of behaviour definitions. Such infinitely replicating terms are always guarded by input prefix operators. However, infinitely replicating terms that are guarded by matching output prefixes never arise.

This suffices to rule out the occurrence of situations like the following:

$$P = (a(x).Q \mid \overline{a}[w]) \mid (!b(x) \mid !\overline{b}[z])$$

Situations similar to the above will never arise from our semantic mappings. It is important to note that if the translation allowed processes with behaviors similar to those above, then fairness is not ensured, since there is no guarantee that in the above process the following allowed reduction:

$$a(x).Q \mid \overline{a}[w]$$

would ever take place.

6 Semantic Correspondence

In this section we demonstrate that our embedding is a semantic preserving mapping from actor troupes to PPC processes. In particular, the semantic function defined by our embedding maps interaction equivalence of actor troupes to barbed bisimilarity of PPC processes.

Consider actor troupes T and T_1 whose definitions are given in examples 4.1, and 5.1 respectively. Not surprisingly, it turns out that Troupe T is *interaction equivalent* to Troupe T_1 (A detailed evolution of Troupes T and T_1 is provided in examples 6.1 and 6.2). So an actor program containing Troupe T can be transformed into an actor program which has Troupe T_1 in place of T. In the semantic domain this would correspond to the replacement of one PPC process with another. Such an operation would make sense only if the PPC processes corresponding to T and T_1 are equivalent in some way. In fact, our semantic mapping has precisely the required property of equivalence preservation. The equivalence in the semantic domain corresponds to *barbed bisimilarity* [23] of PPC processes (defined in Section 3).

The following theorem establishes that the semantic function preserves interaction equivalence of Actor Troupes.

Lemma 6.1. *If T_1, T_2 denote arbitrary actor troupes which are interaction equivalent, and $[\![T_1]\!]$, $[\![T_2]\!]$ denote their corresponding semantic mappings in* PPC*, then:* $[\![T_1]\!] \downarrow_x$ implies $[\![T_2]\!] \rightarrow^* \downarrow_x$

Proof: By simple induction over reduction rules.

Lemma 6.2. *If T_1, T_2 denote arbitrary actor troupes which are interaction equivalent, and $[\![T_1]\!]$, $[\![T_2]\!]$ denote their corresponding semantic mappings in* PPC*, then:* $[\![T_1]\!] \rightarrow^* \downarrow_x$ implies $[\![T_2]\!] \rightarrow^* \downarrow_x$

Proof: By simple induction over reduction rules.

Theorem 6.3 (Semantic Correspondence)**.** *For any two Actor Troupes T_1, T_2 and their corresponding semantic mappings $[\![T_1]\!]$, $[\![T_2]\!]$ in* PPC*, we have the following: If T_1 R_a T_2 where R_a is an interaction equivalence, then $[\![T_1]\!]$ R_w $[\![T_2]\!]$ where R_w is a barbed bisimulation.*

Proof: From the definition of Actor Troupes and from the definition of the semantic mapping it is easy to see that there is a one-to-one correspondence between the set comprising 'Receptionists and method names' and a certain 'subset of PPC channel names'. The channel names on which actions are observable, belong to this subset. Actions on all other channel names which do not correspond either to the Receptionists or their methods, are unobservable since they are bound by the restriction operator. By Lemma 6.1, for all x, $[\![T_1]\!] \downarrow_x$ implies $[\![T_2]\!] \rightarrow^* \downarrow_x$. Furthermore if $[\![T_1]\!]$ reaches a state $[\![T_1]\!]'$ through an unobservable action, then $[\![T_2]\!]$ can also reach a state $[\![T_2]\!]'$ through a series of unobservable actions such that the observable actions of $[\![T_1]\!]'$ and $[\![T_2]\!]'$ coincide (by Lemma 6.2) (where $[\![T_1]\!]'$ and $[\![T_2]\!]'$ denote PPC processes). Thus the semantic mapping preserves interaction equivalence by transforming it to bisimilarity. □

The following examples provide a concrete and detailed illustration of the fact that the troupes T and T_1 (of examples 4.1 and 4.2) are equivalent.

Example 6.1 (Evolution of Troupe T)**.** Consider the Troupe T of examples 4.1 and 5.1. In the following we shall use A, B, R, and X as abbreviations for the PPC terms of the corresponding templates. In the beginning, the configuration is

$(\nu\ a, b, r)(R \mid A \mid B) \mid X$

When the `create` messages are introduced for the sake of initializing troupe T, and the external actor X, we get

$(\nu\ a, b, r, a', b', r')(R \mid A \mid B \mid \overline{r}[a', x', r']$
$\mid \overline{a}[b', a'] \mid \overline{b}[x', b']) \mid X \mid \overline{x}[r', x']$

This causes transitions to occur on r, a, b, and x to give rise to

$(\nu\ a, b, r, a', b', r')$
$(R \mid A \mid B \mid R' \mid A' \mid B') \mid X' \mid X$

where R, A, B and X are as before; while

$$R' \equiv (\nu\ f)(\overline{r'}[f].\ f(d).a'(g).\overline{g}[d'].\ \overline{r}[a', x', r'])$$

$$A' \equiv (\nu\ g)(\overline{a'}[g].g(d').b'(h).\overline{h}[d''].\ \overline{a}[b', a'])$$

$$B' \equiv (\nu\ h)(\overline{b'}[h].h(d'').x'(m).\overline{m}[d''].\ \overline{b}[x', b'])$$

$$X' \equiv (\nu\ m)(\overline{x'}[m].m(d'').\ \overline{x}[r', x'])$$

R', A', B' and X' correspond to particular instances of actors, with addresses r', a', b', and x' respectively. They are created from the templates R, A, B and X respectively.

The next step of the evolution is caused as the result of the message, `send` $f(d)$ `to` r', being sent to the troupe. The above `send` operation translates to the PPC expression $r'(f).\overline{f}[d]$ which is placed in parallel with the existing configuration.

Thus we now have

$(\nu\ a, b, r, a', b', r')(R \mid A \mid B \mid R' \mid A' \mid B')$
$\mid X \mid X' \mid r'(f).\overline{f}[d]$
$\rightarrow (\nu\ a, b, r, a', b', r')(R \mid A \mid B$
$\mid a'(g).\overline{g}[d'].\overline{r}[a', x', r'] \mid A' \mid B') \mid X' \mid X$
$\rightarrow (\nu\ a, b, r, a', b', r')(R \mid A \mid B \mid \overline{r}[a', x', r']$
$\mid b'(h).\overline{h}[d''].\overline{a}[b', a'] \mid B') \mid X' \mid X$
$\rightarrow (\nu\ a, b, r, a', b', r')(R \mid A \mid B \mid R'$
$\mid \overline{a}[b', a'] \mid x'(m).\overline{m}[d''].\overline{b}[x', b']) \mid X' \mid X$
$\rightarrow (\nu\ a, b, r, a', b', r')(R \mid A \mid B$
$\mid R' \mid A' \mid x'(m).\overline{m}[d''].\overline{b}[x', b']) \mid X' \mid X$
$\rightarrow (\nu\ a, b, r, a', b', r')(R \mid A \mid B$
$\mid R' \mid A' \mid \overline{b}[x', b']) \mid \overline{x}[x'] \mid X$
$\rightarrow (\nu\ a, b, r, a', b', r')(R \mid A \mid B$
$\mid R' \mid A' \mid B') \mid X' \mid X$

Example 6.2 (Evolution of Troupe T_1)**.** Consider the Troupe T_1 of examples 4.2 and 5.2. The troupe T_1 has been built by modifying Troupe T, by replacing the template A by the template A_1 at the same location as A; the template B has been discarded; while the templates of R and X remain the same. The PPC process term corresponding to the template A_1 is given by

$A_1 \equiv (\nu\ g)(!a(x', a').\overline{a'}[g].$
$g(d').x'(m).\overline{m}[d''].\overline{a}[x', a'])$

After the appropriate create messages have been sent to the troupe, the configuration is as follows:

$(\nu\ a, r, a', r')(R \mid A_1 \mid R' \mid A_1') \mid X' \mid X$

where R, X, R', X' are as before; while

$A_1' \equiv (\nu\ g)(\overline{a'}[g].g(d').x'(m).\overline{m}[d''].\overline{a}[x', a'])$

We follow the changes in the troupe resulting from the arrival of the message, `send` $f(d)$ `to` r', which translates to $r'(f).\overline{f}[d]$.

Thus we now have

$(\nu\ a, r, a', r')(R \mid A_1 \mid R' \mid A_1')$
$\mid X' \mid X \mid r'(f).\overline{f}[d]$
$\rightarrow (\nu\ a, r, a', r')(R \mid A_1$
$\mid a'(g).\overline{g}[d'].\overline{r}[a', x', r'] \mid A_1') \mid X' \mid X$
$\rightarrow (\nu\ a, r, a', r')(R \mid A_1 \mid \overline{r}[a', x', r']$
$\mid x'(m).\overline{m}[d''].\overline{a}[x', a']) \mid X' \mid X$
$\rightarrow (\nu\ a, r, a', r')(R \mid A_1 \mid R'$
$\mid x'(m).\overline{m}[d''].\overline{a}[x', a']) \mid X' \mid X$
$\rightarrow (\nu\ a, r, a', r')(R \mid A_1 \mid R'$
$\mid \overline{a}[x', a']) \mid \overline{x}[x'] \mid X$
$\rightarrow (\nu\ a, r, a', r')(R \mid A_1 \mid R' \mid A_1') \mid X' \mid X$

Notice that the PPC processes corresponding to the Actor

Troupes T and T_1, are *barbed bisimilar*. Thus, the replacement is valid in the semantic domain as well.

7 Related Work and Comparisons

The Actor model is one of the earliest proposed paradigms of object-based concurrent computation. It has been treated in a rather informal fashion by most of the papers which originally proposed the model [15][14]. There have been a few attempts to give a formal semantics to Actors, in the decades that have passed since it was proposed [15]. Research work related to formulating a semantic basis for Actors may be broadly classified under three main subheadings, as explained in detail in the following three subsections.

7.1 Process-Algebraic Semantics for (Non-Actor) Object-Based Systems

Work that has used process algebras to give semantics to object-based systems can in turn be classified into two major groups depending on the type of process algebra they use – higher-order or first-order. Higher-order process algebras like CHOCS [37] allow only processes to be transmitted as messages. First-order process algebras like the π-calculus [25] allow only channels to be passed in communication.

Among papers that use first order process algebras are those by – Walker [40] who maps POOL to the π-calculus; Jones [19] who maps on object-based design notation called $\pi o \beta \lambda$ to the π-calculus; Pierce and Turner [31] who use the π-calculus for the design of concurrent object-based programming languages; Vaandrager [38] who maps POOL to the process algebra ACP [8]; and Honda and Tokoro [16] who map an object-based calculus to the process calculus reported in [17].

Researchers who use higher-order process algebras are: Nierstrasz [28] who uses the higher-order π-calculus to model his object-based calculus; Sangiorgi [34] proposes the higher-order π-calculus as a rival semantic domain to the π-calculus. But Papathomas [30] and Walker [41] show that the higher-order calculi provide no more conceptual advantages over the first-order process calculi, while modeling object-based systems.

7.2 Non Process-Algebraic Semantics for Actors

The work reported in [13], [12], and [39], model only the concurrent execution of Actors while completely ignoring the object-based features of Actors like persistent state, encapsulation, dynamic creation and reconfigurability. Hewitt and Baker [13] define the notion of an *activation order*, which is a partial order on events. An event x is said to *activate* an event y when y is related to some message created by x. They also define an *arrival order* on messages (for each actor), which is a linear order. The linear order arises from the condition that the mailing queue associated with an actor can receive only one message at a time. Concurrency is modeled by the history order which is defined as the transitive closure of the activation and arrival orders. Clinger [12] develops the above work by creating *event diagrams* from the activation order. An *event diagram* is a historical record of all the computations right from the initial stage. The event diagram together with the set of pending messages is said to form a *powerdomain* which is used to describe the concurrency of Actors. Vasconcelos and Tokoro [39] refine this work further by weakening the condition on activation orders which no longer requires each event to have at most one immediate predecessor. Thus they use the notion of *traces* to model concurrent executions where an event might be immediately be preceded by many events.

The papers [5, 36] deal with the object-based features of Actors as well as with the concurrency notions, Agha et al. [5] formulates an Actor language as an extension of a simple functional language, and employs transition systems to provide an operational semantics for Actors. It further demonstrates that in the presence of fairness assumptions, the three notions of equivalences (*testing, may,* and *must*), collapse into two classes. Talcott [36] characterizes actor languages by defining a notion of *abstract actor structure*, and provides a semantics for them using transition rules that use properties of concurrent rewriting systems. Janssens and Rozenberg [18] model a restricted version of actors using graph grammars, and introduce the notion of an abstract actor structure. Kahn and Saraswat [20] model actors as a special case of concurrent constraint programming.

7.3 Process-Algebraic Semantics for Actors

The only other paper in this category, apart from our work, is by Agha [2]. It is a preliminary attempt using CCS [22] to provide a semantics for Actors. However, only the concurrency features of Actors is modeled, while key aspects like encapsulation, dynamic creation, and unrestricted replacement behaviors are completely ignored. This is because of the limitations in the expressive power of CCS – which deals with synchronous agents having a static interconnection topology, and lacks dynamic creation of new channels and processes.

In contrast with all the related work on Actors, our approach gives a comprehensive process-algebraic interpretation of all the basic features of the Actor Model for the first time. Also noteworthy is the fact that we use π-calculus – which is synchronous – to simulate an asynchronous system like Actors. This is possible due to the power of the *Bang* (!) operator of the π-calculus, which allows unbounded replication of processes and thereby provides the capability to model asynchronous behavior. This particular feature of the π-calculus has been recognized by other researchers in a different setting [10, 17, 33].

8 Conclusion

We have related two prominent models of concurrent computation, namely Actors and the π-calculus, by presenting an elegant semantic mapping of all the fundamental constructs of actors in terms of the polyadic π-calculus. We have enriched the Actor model by defining a higher level of abstraction, and also have provided a notion of equivalence between them. There are several interesting avenues which present themselves for further exploration. The object-based nature of the primitive constructs of actors, should be extended to include other object-oriented notions as well. In particular the varieties of inheritance [21, 42, 11], like self-based inheritance, and delegation-based inheritance, and also the notion of subtyping should be studied in this setting. The actor model allows various levels of granularity and abstraction. This flexibility of the actor model should be explored further by encompassing the framework given by [6]. A sound basis for typed object-based computing [1], can be explored using extensions to the formalism of this paper. Such extensions should include the notion of types in the setting of Actors. The Actor model has been extended in a different way by [4], using constructs which seem to have strong connections with the paradigm of concurrent constraint programming. It would be meaningful to explore connections between ActorSpace [4] and the model proposed in this paper.

Acknowledgement

We wish to thank the anonymous referees for constructive comments which were of immense help in improving the content and presentation of this paper. Our thanks to Ms. Margaret D'Souza for typing and proofreading this paper.

References

[1] M. Abadi, and L. Cardelli (1995) A theory of primitive objects: Second-order systems, *Science of Computer Programming*, Vol. 25, pp. 81–116.

[2] G. Agha (1987) *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press.

[3] G. Agha (1989) Supporting Multiparadigm programming on Actor Architectures, *Proc. PARLE'89*, LNCS 366, Springer, pp. 1–19.

[4] G. Agha, and C. J. Callsen (1993) ActorSpace: An Open Distributed Programming Paradigm, *Proc. PPOPP'93*, ACM, pp. 23–32.

[5] G. Agha, I. A. Mason, S. Smith, and C. Talcott (1997) A Foundation for Actor Computation, *Journal of Functional Programming*, Vol. 7, pp. 1–72.

[6] J. M. Andreoli, H. Gallaire, and R. Pareschi (1995) Rule-Based Object Coordination, *Object-Based Models and Languages for Concurrent Systems*, LNCS 924, Springer, pp. 1–13.

[7] F. Baude, and G. Vidal-Naquet (1991) Actors as a Parallel Programming Model, *Proc. STACS'91*, LNCS 480, Springer, pp. 184–195.

[8] J. Bergstra, and J. Klop (1985) Algebra of communicating processes with abstraction, *Theoretical Computer Science*, Vol. 37, pp. 77–121.

[9] G. Berry, and G. Boudol (1992) The Chemical Abstract Machine, *Theoretical Computer Science*, Vol. 96, pp. 217–248.

[10] G. Boudol (1992) Asynchrony and the π-calculus, *Research Report*, Number 1702, INRIA Sophia-Antipolis.

[11] K. B. Bruce (1994) A Paradigmatic Object-Oriented Programming Language: Design, Static Typing and Semantics, *Journal of Functional Programming*, Vol. 4, pp. 127–206.

[12] W. Clinger (1981) Foundations of Actor Semantics, *AI-TR*, Number 633, MIT.

[13] C. Hewitt, and H. Baker (1977) Laws for Communicating Parallel Processes, *Proc. IFIP*, pp. 987–992.

[14] C. Hewitt (1977) Viewing Control Structures as Patterns of Passing Messages, *Artificial Intelligence*, Vol. 8, pp. 323–364.

[15] C. Hewitt, P. Bishop, and R. Sterger (1973) A Universal Modulator Actor Formalism for Artificial Intelligence, *Proc. IJCAI*, pp. 235–245.

[16] K. Honda, and M. Tokoro (1991) A Small Calculus for Concurrent Objects, *OOPS Messenger*, Vol. 2, pp. 50–54.

[17] K. Honda, and M. Tokoro (1991) An Object Calculus for Asynchronous Communication, *Proc. ECOOP'91*, LNCS 512, Springer, pp. 133–147.

[18] D. Janssens, and G. Rozenberg (1989) Actor grammars, *Math. Systems Theory*, Vol. 22 (2), pp. 75–107.

[19] C. B. Jones (1993) A pi-calculus Semantics for an Object-Based Design Notation, *Proc. CONCUR*, LNCS 715, Springer, pp. 158–172.

[20] K. M. Kahn, and V. A. Saraswat (1990) Actors as a Special Case of Concurrent Constraint Programming, *Proc. ECOOP/OOPSLA'90*, ACM Press, pp. 57–66.

[21] B. Meyer (1993) Systematic Concurrent Object-Oriented Programming, *Communications of the ACM*, Vol. 36, pp. 56–80.

[22] R. Milner (1989) *Communication and Concurrency*, Prentice Hall.

[23] R. Milner (1999) *Communicating and Mobile Systems: The Pi Calculus*, Cambridge University Press.

[24] R. Milner (1993) Elements of Interaction, *CACM*, Vol. 36 (1), pp. 78–89.

[25] R. Milner, J. Parrow, and D. Walker (1992) A calculus of mobile processes (Parts I and II), *Information and Computation*, Vol. 100, pp. 1–77.

[26] R. Milner (1992) Functions as processes, *Mathematical Structures in Computer Science*, Vol. 2, pp. 119–141.

[27] R. Milner, and D. Sangiorgi (1992) Barbed Bisimulation, *Proc. ICALP*, LNCS 623, Springer, pp. 685–695.

[28] O. Nierstrasz (1992) Towards an Object Calculus, *Proc. Object-Based Concurrent Computing*, LNCS 612, Springer, pp. 1–20.

[29] C. Palamidessi (1997) Comparing the expressive power of the Synchronous and the Asynchronous pi-calculus, *Proc. POPL*, ACM, pp. 256–265.

[30] M. Papathomas (1991) A Unifying Framework for Process Calculus Semantics of Concurrent Object-oriented Languages, LNCS 612, Springer, pp. 53–79.

[31] B. C. Pierce, and D. N. Turner (1994) Concurrent Objects in a Process Calculus, *Proc. TPPP*, LNCS 907, Springer, pp. 187–215.

[32] B. C. Pierce, and D. N. Turner (2000) Pict: A Programming Language Based on the Pi-calculus, *Proof, Language and Interaction*: Essays in Honour of Robin Milner, MIT Press, pp. 455–494.

[33] N. Raja, and R. K. Shyamasundar (1996) Actors as a Coordinating Model of Computation, *Proc. Perspectives of System Informatics*, LNCS 1181, Springer, pp. 196–202.

[34] D. Sangiorgi (1993) From pi-calculus to higher-order pi-calculus and back, *Proc. TAPSOFT'93*, LNCS 668, Springer, pp. 151–166.

[35] D. Sangiorgi, and D. Walker (2001) *The Pi-Calculus - A Theory of Mobile Processes*, Cambridge University Press.

[36] C. Talcott (1997) Interaction Semantics for Components of Distributed Systems, *Proc. IFIP Workshop on Formal Methods for Open Object-based Distributed Systems*, Chapman & Hall, pp. 154–169.

[37] B. Thomsen (1993) Plain CHOCS: A Second Generation Calculus for Higher Order Processes, *Acta Informatica*, Vol. 30, pp. 1–59.

[38] F. W. Vaandrager (1990) Process algebra semantics of POOL, *Applications of Process Algebra*, Cambridge University Press, pp. 172–236.

[39] V. Vasconcelos, and M. Tokoro (1992) Trace Semantics for Actor Systems, *Proc. Object-Oriented Concurrent Computing*, LNCS 612, Springer, pp. 141–162.

[40] D. Walker (1991) π-calculus Semantics of Object-Oriented Programming Languages, *Proc. TACS'91*, LNCS 526, Springer, pp. 532–547.

[41] D. Walker (1995) Objects in the π-calculus, *Information and Computation*, Vol. 116, pp. 253–271.

[42] A. Yonezawa (1990) *ABCL: An Object-Oriented Concurrent System*, MIT Press.

On Integrating Conversations into Web Services Composition

Zakaria Maamar
Zayed University, Dubai, U.A.E
zakaria.maamar@zu.ac.ae

Soraya Kouadri Mostéfaoui
Oxford Brookes University, Oxford, UK
E-mail: kouadris@acm.org

Keywords: Web service, conversation, composition, context.

Received: November 28, 2004

We present an approach for integrating conversations into the process of composing Web services. A Web service is an accessible application that other applications and humans can discover and trigger to satisfy various needs such as weather forecasts. While much of the work on Web services to date has focussed on low-level standards, it is becoming urgent to allow Web services engage in conversations, make decisions, and adjust their behavior according to the context of the situations in which they participate.

Povzetek: Predstavljena je integracija koverzacije v spletne storitve.

1 Introduction

In this paper, we highlight the role of *conversations* in the field of *Web services* and assess the value-added of integrating conversations into the composition of Web services. Composition primarily addresses the situation of a user's request that cannot be satisfied by any available Web service, whereas a *composite service* obtained by combining available component services (i.e., Web services or composite services) might be used [3]. While much of the work on Web services to date has focussed on low-level standards for publishing, discovering, and triggering Web services [2], the use of conversations promotes Web services to a higher level by giving them the opportunity to act as active components. A conversation is a consistent exchange of messages between participants who are involved in joint operations and thus have common interests.

In order to engage in conversations, Web services (also called services in the rest of the paper) have to be leveraged from passive components, which only respond to triggers, to active components, which make decisions and adjust their behavior according to the *context* in which they evolve. Context is the information that characterizes the interactions between humans, applications, and the surrounding environment [4]. Huhns backs the importance of leveraging Web services by using software agent architectures [9]. Indeed Web services, unlike software agents, are not designed to use and reconcile ontologies. Moreover, software agents are inherently communicative, whereas Web services are passive until invoked.

While some authors agree on the importance of leveraging Web services to the level of active components, others have already identified some similarities between Web services and software agents [6]. Services (i) advertise their capabilities after specification using for example WSDL, (ii) search for other services using for example UDDI, and (iii) invoke services without prior notice using for example SOAP. This kind of behavior bears many likenesses to software agents. For instance, a service accepts requests (sense) and returns responses (action) [6]. In addition, once a service is invoked, it performs tasks with or without further inputs (autonomy). However, it is the authors' belief that the aforementioned behavior of a service is mainly hard-coded and consequently, limits the service in its action selection. Enhancing Web services with conversation capabilities will first, enable an emergent behavior during composition and second, permit to services to be more flexible in managing the situations in which they participate.

Web services composition is a very active area of research and development. However, *very little* has been achieved to date regarding the seamless integration of conversations into composition approaches of Web services. In particular, several obstacles still hinder this integration such as: (i) Web services are dealt with as passive components rather than active components that can be embedded with context-awareness mechanisms, (ii) existing approaches for service composition (e.g., BPEL4WS) typically facilitate orchestration only, while neglecting contextual information on services, and (iii) lack of support techniques for modeling and specifying conversations between Web services. In this paper, the focus is *on the conversations happening among a group of Web services, which are called to constitute composite services*.

2 Conversations and Web services

Several authors note that current standards of Web services are used in systems featured by simple interactions [1, 7]. Simple because the interactions adopt a *request-response pattern* (e.g., announce/confirm). However there are multiple situations that need more than two turns of interaction (e.g., propose/counter-propose/accept $\oplus$ reject $\oplus$ counter-propose/...). The participants in these situations have to engage in conversations before they reach an agreement. Another initiative in the field of Web services is the Web Services Conversation Language (WSCL). This language describes the structure of documents that a Web service expects receiving and producing, as well as the order in which the exchange of documents is expected to take place. While conversations in [1] occur between end-users and providers of Web services, and WSCL focusses on specifying the operations that Web services support, our focus is on the conversations happening among Web services. These services might originate from different sources and have to engage in conversations in order to agree on what to exchange, how to exchange, when to exchange, and what to expect from an exchange during composition.

2.1 *Composition stages*

To assess the benefits of conversations to Web services composition, we decompose the composition into three stages: pre-composition, composition, and post-composition. In the following, only the pre-composition stage is discussed. Because similar services exist on the Internet, it is important to search for and identify the services that satisfy specific user-defined selection criteria (e.g., execution cost, reliability, availability [14]). Conversations in the pre-composition stage concern the following aspects:

1. Identification aspect: use search mechanisms (e.g., UDDI registries) to identify Web services. We assume that Web services are already specified and advertised.

2. Invitation aspect: invite Web services to participate in a composition. An invitation is either accepted or refused. The rationale of inviting services instead of directly triggering them is given in [12]. In addition, conversations occurring during service invitation are described towards the end of this paper.

3. Compatibility aspect: check if the Web services can exchange meaningful information because of data heterogeneity issues that may raise. Details on the semantic compatibility of Web services are found in [13].

2.2 *Conversation's components*

Pre-composition, composition, and post-composition stages are each concerned with some of the conversational aspects that take place during Web services composition. Because of the variety of these aspects, we decided on (i) associating each aspect with a *conversation session* and (ii) implementing a session with a *course of conversational actions*.

To handle the multiple conversation sessions, we use *Conversation Schemas* (CSs) as a technique for describing these sessions and their respective course of conversational actions. We define a conversation schema as a specification of the exchange of messages that is expected to happen between participants. This exchange depends on several factors including application domain, active context, and current chronology. For example, a conversation schema describes both the side-effects of a conversation that is misunderstood and the corrective actions that permit fixing these side-effects.

First of all, the initiator of a conversation downloads a conversation schema from the library of conversation schemas (Fig. 2) according to the following elements: current composition stage, progress in this stage with regard to the current aspect, and intention behind establishing the conversation. These elements are needed since conversations are context sensitive [5, 15]. Next, the initiator instantiates the conversation object (i.e., gives a value) and checks the activation condition before it transfers a message to a receiver. Upon reception of the conversation object that the message conveys, the receiver takes one of the following actions:

1. Accepts the conversation object without any change and starts acting based on the conversation object's content.

2. Changes the conversation object and submits the modified conversation object to the initiator for either further change, approval, or rejection.

3. Rejects the conversation object and either submits a new conversation object to the initiator or ignores the initiator (this one has a time-out constraint).

Modeling conversations is a complex process as several requirements need to be satisfied. Some of these requirements are identified in [10]: (i) conversation models should be task-oriented, (ii) conversation models should be associated with a semantics, (iii) conversation models must provide communication abstractions, and (iv) conversation models should be reusable and extendable. In this paper, we specify conversation schemas with state charts [8]. In addition to satisfying some of the aforementioned requirements such as (i) and (iv), encoding the flow of conversations using state charts has several benefits. First, state charts have a formal semantics, which is essential for reasoning on the content of conversations. Next, state charts are becoming a standard process-modeling language as they are being integrated into UML. This process modeling helps in managing admissible turns and decision makings during conversations. Finally, state charts offer most of the control-flow constructs that can be found in real conversations such as branching and looping. Fig. 1 depicts the

mapping of the concepts of a conversation schema onto the concepts of a state chart.

- Name of states is labelled with **S/R** (sender/receiver).
- Name of transitions is labelled with activation condition and conversation object.
- Actions of states implement the information that conversation objects convey.
- A complete state chart illustrates the conversation schema of a conversation session.

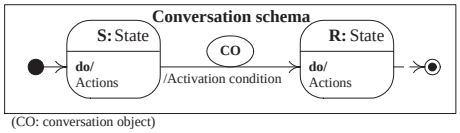


Figure 1: Conversation schema as a state chart

3 Context-aware conversations for Web services composition

To assess the way a composition of Web services progresses so relevant conversation sessions are entered and relevant conversation schemas are downloaded from the library of conversation schemas (Fig. 2), the use of awareness mechanisms is required. These mechanisms ensure that the status of each Web service is known and instantaneously reflected in a structure, which we denote by $\mathcal{W}$-context (context of $\mathcal{W}$eb service). using the information that $\mathcal{W}$-context caters, it is possible to know if a Web service is part of a composition, under execution, or invited to participate in a composition.

In one of our previous works [12], we strengthened the advantages of the combination (software agent, Web service, context). For example, agents track Web services in order to update their respective $\mathcal{W}$-contexts. The tracking is about the composition in which a service takes part, the current state that the service takes in the composition, and the type of conversations that the service has initiated during the composition. Fig. 2 presents the context-aware conversation approach for Web services composition that we developed. The features of this approach are below.

1. Three types of software agents are set: conversation-manager-agent, composite-service-agent, and service-agent.

   A conversation-manager-agent runs on top of the library of conversation schemas. It updates the library if a new specification of a conversation schema is developed (e.g, by a designer). Plus, the conversation-manager-agent responds to the requests of downloading conversation schemas that composite-service-agents and service-agents submit.

   A composite-service-agent triggers and monitors the deployment of the specification of a composite service (to keep Fig. 2 clear, the specification store is not represented[1]). This monitoring is reflected in a context, which we refer to as $\mathcal{C}$-context (context of $\mathcal{C}$omposite-service). In addition, the composite-service-agent interacts with service-agents when it comes to inviting services for composition or informing services of the changes in the specification of the composite services.

   A service-agent is responsible for getting a Web service ready for composition, monitoring its execution through its state chart, and updating its $\mathcal{W}$-context.

2. Besides the state charts of the conversation schemas (Fig. 1), additional state charts are associated with Web services (Fig. 2). The states that a Web service takes are immediately reflected in its $\mathcal{W}$-context. Interesting to note that the transitions in the state charts of services are *context-based* and *conversation-oriented* (Fig. 2-2). However, these conversations are less complex than the conversations that involve Web services (Fig. 2-1). Therefore, each state of a Web service is associated with a $\mathcal{T}$-context (context of Web-service s$\mathcal{T}$ate).

3. A library of conversation schemas that stores the specifications of conversation schemas (Fig. 1). The library is a resource from which composite-service-agents and service-agents download conversation schemas after interactions with the conversation-manager-agent.

In Fig. 2, the conversations occur in two separate locations. In the first location, the conversations concern the component Web services that participate in a composite service (Fig. 2-1). These conversations are specified using the conversation schemas of Fig. 1. In the second location, the conversations concern the states of the Web services (Fig. 2-2). It should be noted that the state chart of a conversation schema supports the interactions between the state charts of services. The distinction between states of services and states of conversations gives a much better flexibility in managing the aspects that each type of state chart is concerned with. State charts of services focus on the changes that apply to services such as availability, commitment, and execution, whereas state charts of conversations focus on the changes that apply to conversations such as formulation, reception, and response.

Because of both types of state (those associated with services and those associated with conversations), we annotate each conversation state with a context, which we refer to as $\mathcal{S}$-context (context of conversation $\mathcal{S}$tate). The rationale of $\mathcal{S}$-contexts of the states of conversations is similar to the rationale of $\mathcal{T}$-contexts of the states of services. Moreover,

[1]The specification of a composite service is based on state chart diagrams, where a state is labelled with a service chart diagram and a transition is labelled with events, conditions, and variable assignment operations [11].

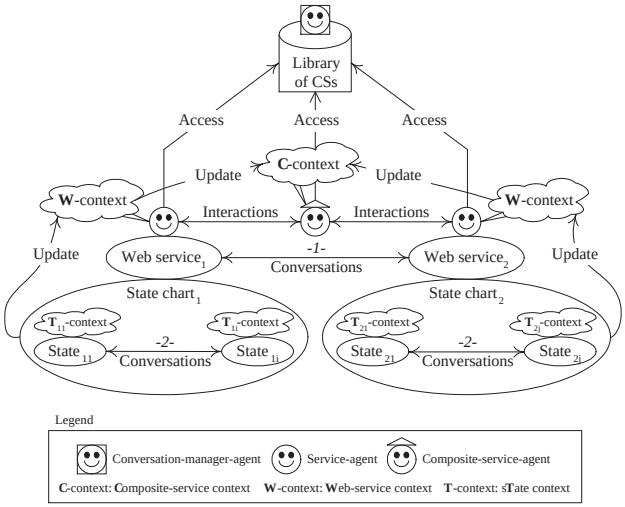


Figure 2: Context-aware conversation approach for Web services composition

to track the progress of a conversation a context, which we denote by $\mathcal{V}$-context (context of a con$\mathcal{V}$ersation), is used. This is similar to the $\mathcal{W}$-context of a Web service.

4 Development example of a conversation schema

We decomposed the composition of Web services into three stages: pre-composition, composition, and post-composition. Each stage consists of different aspects, which characterize the conversations that occur. In this section, we illustrate how a conversation schema is developed. This development consists of devising two state charts, one for the conversations and one for the services that participate in these conversations. Before we explain the development process, the following comments are made on both types of state charts:

- State labels are annotated with S/R (sender/receiver). If there is no annotation, the state identifies the communication network.
- Transitions implement the links between states. Two types of transition exist. The transitions that connect states of the same chart are represented with regular lines. The transitions that connect states of separate charts are represented with dashed lines.

To keep the paper self-contained, only the conversation schema featuring the invitation aspect is illustrated (Fig. 3). While a composite service monitors the component Web services that are currently under execution, the composite service submits an invitation to the next component Web service to join the composition.

States of services. There are four states, which are seen from two perspectives.

- Sender perspective: two states are associated with the sender service namely monitoring and assessment. Here, the sender corresponds to the composite service. One of the actions in the monitoring state consists of downloading the conversation schema for Web services invitation. We recall that this schema is stored in the library of conversation schemas (Fig. 2).
- Receiver perspective: two states are associated with the receiver service namely assessment and deployment. Here, the receiver corresponds to the next component Web service. One of the actions in the assessment state consists of checking the current commitments that the Web service has towards other composite services. Based on these commitments, the component service either accepts or rejects the invitation to participate in a composite service.

States of conversations. There are six states, which are seen from three perspectives. Ellipses glued to transitions correspond to conversation objects.

- Sender perspective: two states are associated with the sender service namely preparation and reception.
- Receiver perspective: two states are associated with the receiver service namely preparation and reception.
- Network perspective: two states are associated with the communication network namely transmission from the sender to the receiver, and transmission from the receiver to the sender.

5 Implementation

As a first step of validating the proposed approach of Fig. 2, we have developed a *conversation and Web services composition-manager*, using Borland JBuilder Enterprise

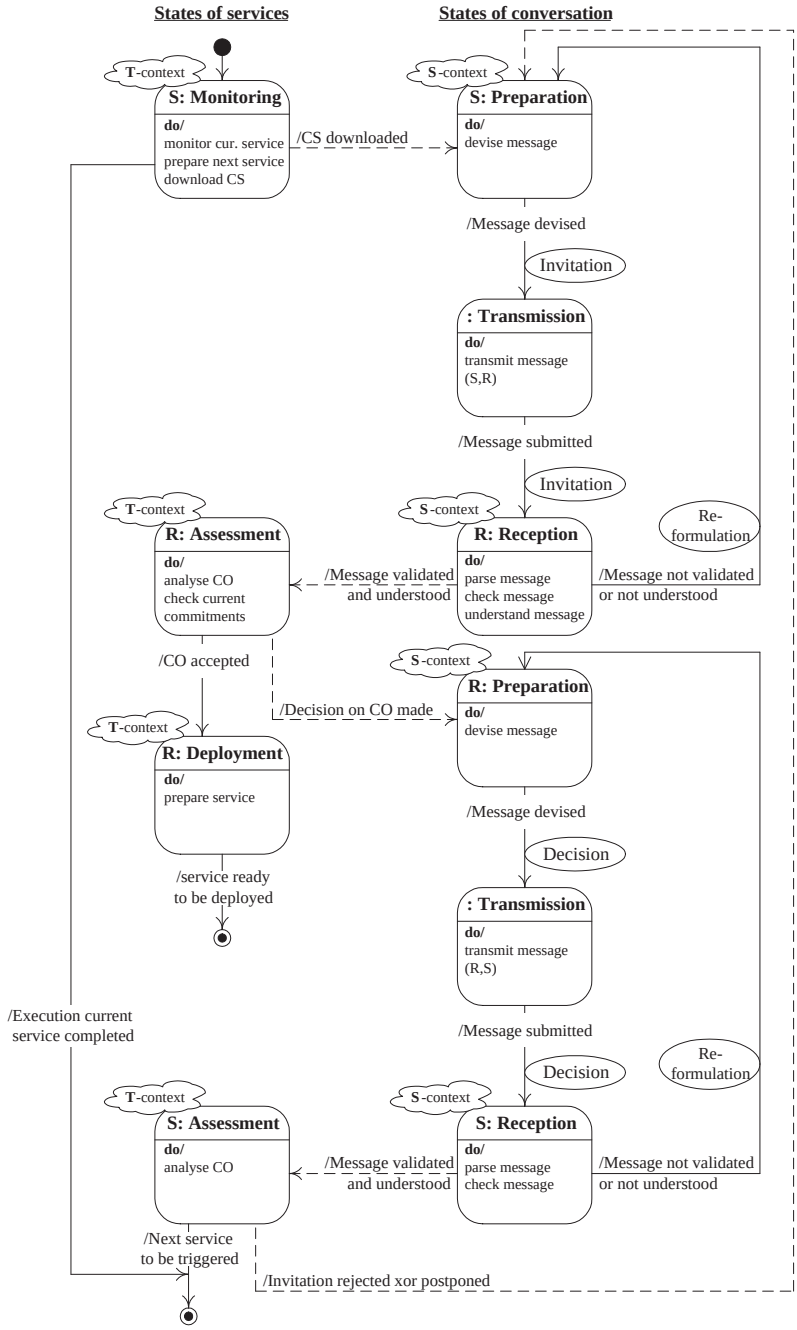


Figure 3: State chart of a conversation schema - invitation aspect

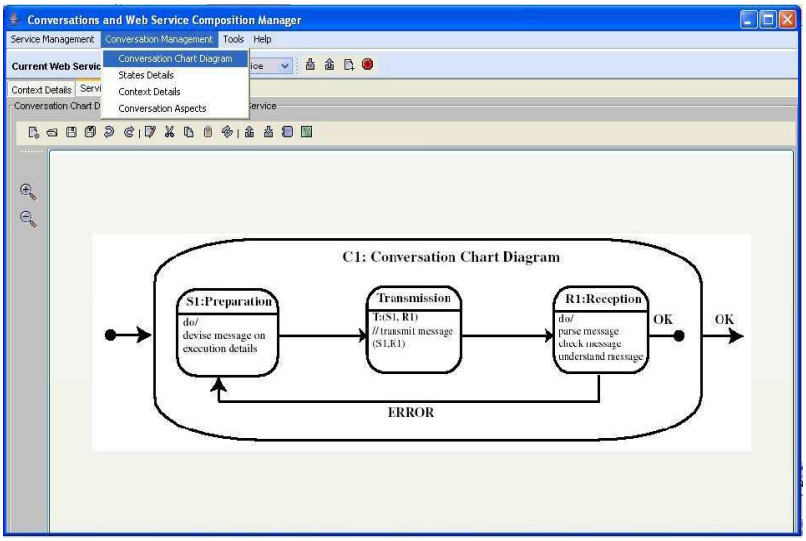


Figure 4: Graphical editor for conversation schemas and service chart diagrams

Edition version 9[2]. The prototype integrates a set of tools, which allow for instance Web services' providers and users to create, compose, and execute services based on the different contexts. WSDL is used for Web services specification, and UDDI is used for Web services announcement and discovery. Details of contexts and conversation sessions are structured as XML files. A dedicated XML editor was developed in order to create, validate, test, and monitor the different XML files. The validation of these files is based on two XML schemas (conversations.xsd and context.xds). In addition, the conversation manager offers an editor for describing the state charts of conversation sessions and composite services (Fig. 4). The graphical editor provides means for directly manipulating conversations and service chart diagrams, states, and transitions (add, remove, modify, etc.) graphically using drag and drop operations.

6 Conclusion

In this paper, we overviewed our approach for composing Web services using software agents, conversations, and context. Several types of conversation schemas are discussed such as those for inviting Web services to participate in composition. Conversation schemas have been associated with software agents and contextual information. Because needs and interests of users always change, it is important to ensure that the composition of Web services efficiently handles these changes. What is needed is to allow Web services to decide whether to join a composition, what states to take with regard to the outcomes of conversations, and what actions to perform within these states.

[2]http://www.borland.com/jbuilder/enterprise/index.html.

References

[1] L. Ardissono, A. Goy, and G. Petrone. Enabling Conversations with Web Services. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2003)*, Melbourne, Australia, 2003.

[2] B. Benatallah, Q. Z. Sheng, and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1), January-February 2003.

[3] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. A Foundational Vision for e-Services. In *Proceedings of The Workshop on Web Services, e-Business, and the Semantic Web (WES'2003) held in conjunction with The 15th Conference On Advanced Information Systems Engineering (CAiSE'2003)*, Klagenfurt/Velden, Austria, 2003.

[4] P. Brézillon. Focusing on Context in Human-Centered Computing. *IEEE Intelligent Systems*, 18(3), May/June 2003.

[5] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In *Proceedings of The Twelfth International World Wide Web Conference (WWW'2003)*, Budapest, Hungary, 2003.

[6] B. Burg. Agents in the World of Active Web Services. In *Proceedings of Second Kyoto Meeting on Digital Cities*, Kyoto, Japan, 2001.

[7] J. Dale, D. Levine, F. G. McCabe, G. Arnold, M. Lyel, and H. Kuno. Advanced Web Services. Technical

Report FLA-NARTM02-08, Fujitsu Laboratories of America, Inc., Sunnyvale CA, USA, 2002.

[8] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4), October 1996.

[9] M. Huhns. Agents as Web Services. *IEEE Internet Computing*, 6(4), July-August 2002.

[10] F. Lin and D. H. Norrie. Schema-based Conversation Modeling for Agent-oriented Manufacturing Systems. *Computers in Industry*, 46(3), October 2001.

[11] Z. Maamar, B. Benatallah, and W. Mansoor. Service Chart Diagrams - Description & Application. In *Proceedings of The Twelfth International World Wide Web Conference (WWW'2003)*, Budapest, Hungary, 2003.

[12] Z. Maamar, S. Kouadri Mostéfaoui, and H. Yahyaoui. A Web Services Composition Approach based on Software Agents and Context. In *Proceedings of The 19th Annual ACM Symposium on Applied Computing (SAC'2004)*, Nicosia, Cyprus, 2004.

[13] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing Web Services on the Semantic Web. *The VLDB Journal, Special Issue on the Semantic Web, Springer Verlag*, 12(4), 2003.

[14] D. A. Menascé. QoS Issues in Web Services. *IEEE Internet Computing*, 6(6), November/December 2002.

[15] J. P. Morgenthal. Web Service Conversations. *Business Integration Journal*, August 2003.

An Overview of Slicing Techniques for Object-Oriented Programs

Durga Prasad Mohapatra, Rajib Mall and Rajeev Kumar[1]
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, WB 721 302, India
E-mail: {durga, rajib, rkumar}@cse.iitkgp.ernet.in

Keywords: program slicing, program dependence graph, debugging, object-oriented programs, concurrent object-oriented program, multi-threading, distributed programming.

Received: April 15, 2005

This paper surveys the existing slicing techniques for object-oriented programs. Many commercial object-oriented programs are concurrent in nature. Concurrency is typically implemented in the form of multi-threading or message passing using sockets or both. We therefore review the available techniques in slicing of concurrent object-oriented programs. Another trend that is clearly visible in object-oriented programming is client-server programming in a distributed environment. We briefly review the existing techniques for slicing of distributed object-oriented programs

Povzetek: Opisana je tehnika analize objektnih programov.

1 Introduction

Program slicing is a program analysis technique. The main applications of program slicing includes various software engineering activities such as program understanding, debugging, testing, program maintenance, complexity measurement etc. It can also be used to extract the statements of a program that are relevant to a given computation. A *program slice* consists of the parts or components of a program that (potentially) affect the values computed at some point of interest, referred to as a *slicing criterion*. Typically, a slicing criterion consists of a pair $< s, V >$, where s is the statement number and V is a variable. The components of a program which have a direct or indirect effect on the values computed at a slicing criterion $< s, V >$ are called the *program slice with respect to the slicing criterion* $< s, V >$.

The concept of a program slice was introduced by Weiser [92]. Various slightly different notions of program slices have been proposed. There has also been a proliferation of the number of methods to compute slices. The main reason for this proliferation of slicing techniques is that different applications require different properties of slices. Weiser [91] defined a program slice S as a *reduced, executable program* obtained from a program P by removing statements, such that S replicates part of the behavior of P. The program slicing technique originally introduced by Weiser [91, 92, 93] is now called *static* backward slicing. It is static in the sense that the slice is independent of the input values to the program. It is *backward* because the control flow of the program is considered in reverse while constructing the slice. Another common definition of a slice is a subset of the statements and control predicates of the program which directly or indirectly affect the values computed at the slicing criterion, but which do not necessarily constitute an executable program.

Object-oriented programming style is becoming the norm. These programs may be very large as well as concurrent. In some applications the programs run in a distributed manner on several nodes connected through a network. The code size of these object-oriented programs often exceeds million of lines. Making such programs dependable and trust worthy is a major challenge. Program slicing is advocated as a technique to automatically analyze a program. The results of the analysis can be used to help in debugging, test case design, test coverage analysis and others [94, 82, 96, 19].

Slicing object-oriented programs presents new challenges which are not encountered in traditional program slicing. To slice an object-oriented program, features such as classes, dynamic binding, encapsulation, inheritance, message passing and polymorphism need to be considered carefully. Although the concepts of inheritance and polymorphism are strengths of object-oriented programming languages, they pose special challenges in program slicing. Due to *inheritance* and *dynamic binding* in object-oriented programs, the process of tracing dependencies becomes more complex than that in a procedural program. Larson and Harrold were the first to consider these aspects in their work [60]. To address these object-oriented features, they enhanced the system dependence graphs (SDG) [48] to represent object-oriented software. After the SDG is constructed, the two phase algorithm of Horwitz et al. [48] is used with minor modifications for computing static slices. Larson and Harrold have reported only a static slicing technique for object-oriented programs [60], and did not address dynamic slicing aspects. The dynamic slicing aspects have been reported by Zhao [100], Song et al. [84], Xu et

al. [94] and Wang et al. [90].

Most of the commercial object-oriented programs are concurrent in nature and run in different machines connected through a network. It is usually accepted that understanding and debugging *concurrent* and *distributed* object-oriented programs are much harder compared to the sequential programs. Slicing techniques promise to come in handy at this point. However, most of the research work in the program slicing area have focused attention on sequential programs. Research reports addressing slicing of *concurrent* and *distributed* object-oriented programs are scarce in literature [102, 17, 54, 55, 75].

Several comprehensive surveys are available for program slicing in general [86, 12, 63, 45, 95, 20]. But, surveys on slicing of object-oriented programs have not been reported to the literature to the best of our knowledge. In this paper, we present a brief survey of the existing slicing techniques for object-oriented programs. Also, we have reviewed the available literatures on the available techniques for slicing concurrent object-oriented programs. Subsequently, we have discussed the current trend in the area of slicing of distributed object-oriented programs. In this survey, we discuss the contribution of each work and compare the major difference between them.

In the following, we review some basic slicing concepts that would be useful to understand the rest of the paper.

1.1 Categories of Program Slicing

Several categories of program slicing as well as methods to compute them are found in literature. The main reason for the existence of so many categories of slicing is the fact that different applications require different types of slices. Slices can be *backward* or *forward* [48, 98], *static* or *dynamic* [3, 52, 27], *intra-procedural* or *inter-procedural* [48].

Static Slicing and Dynamic Slicing: *Static slicing* technique uses static analysis to derive slices. That is, the source code of the program is analyzed and the slices are computed for *all possible input values*. Therefore static slices are conservative and contain more statements than necessary. For object-oriented programs the situation is still worse as the computed static slice will contain all most all of the statements present in the program. This is due to the various relationships such as inheritance, polymorphism, dynamic binding etc. existing among classes. Therefore, static slices are of little use in the context of object-oriented programs.

Korel and Laski [52] introduced the concept of dynamic program slicing. Dynamic slicing makes use of the information about a *particular execution* of a program. *A dynamic slice with respect to a slicing criterion* $< s, V >$*, for a particular execution, contains those statements that actually affect the slicing criterion in the particular execution*. Therefore, dynamic slices are usually smaller than static slices and are more useful in interactive applications

```
1 main()
2 {
3   int i, sum;
4    cin>> i;
5    sum = 0;
6    while(i <= 10)
7    {
8      sum=sum+i;
9       ++ i;
10   }
11   cout<< sum;
12   cout<< i;
13 }
```

Figure 1: An example program

such as program debugging and testing. *Dynamic slicing* is more suitable for object-oriented programs than *static slicing* as the computed dynamic slice will contain only those statements that *actually* affect the slicing criterion. In other words, we can say that dynamic slicing techniques compute *precise slices*. A comprehensive survey on the existing dynamic program slicing algorithms is reported in Korel and Rilling [53] and Xu et al. [95].

Consider the C++ example program given in Fig. 1. The static slice with respect to the slicing criterion $< 11, sum >$ is the set of statements {4, 5, 6, 8, 9}. Consider a particular execution of the program with the input value $i = 15$. The dynamic slice with respect to the slicing criterion $< 11, sum >$ for the particular execution of the program is {5}.

Backward Slicing and Forward Slicing: As already discussed, a backward slice contains all parts of the program that might directly or indirectly affect the slicing criterion [92]. Thus a static backward slice provides the answer to the question: *"which statements affect the slicing criterion?"*.

A forward slice with respect to a slicing criterion $< s, V >$ contains all parts of the program that might be affected by the variables in V used or defined at the program point s [84, 98]. A forward slice provides the answer to the question: *"which statements will be affected by the slicing criterion?"*.

Intra-procedural Slicing and Inter-procedural Slicing: Intra-procedural slicing computes slices within a single procedure. Calls to other procedures are either not handled at all or handled conservatively. If the program consists of more than one procedure, inter-procedural slicing can be used to derive slices that span multiple procedures [48].

For object-oriented programs, intra-procedural slicing is meaning less as practical object-oriented programs contain more than one method. So, for object-oriented programs, inter-procedural slicing is more useful.

Other Slicing Categories: Many examples of slicing are combinations of the categories above. For example, Weiser's original work [91] describes backward, static, intra-procedural slicing; although he also later gave an algorithm for backward, static, inter-procedural slicing [93]. The work of Kamkar [49] produces backward, dynamic, inter-procedural slicing. It is also possible to combine the features of static slicing with the features of dynamic slicing. This new form of slicing is called hybrid slicing [40]. Hybrid slicing is an approach for refining static slices using dynamic information.

There are variants of slicing in between the two extremes of static and dynamic, where some but not all properties of the initial state are known. These are known as *conditioned slices* [13, 31, 42, 25] or *constrained slices* [29]. Traditional slicing methods are all based on statement deletion. In a recently reported form of slicing called *amorphous slicing* [11, 43, 46], slices are not necessarily produced by deleting statements and may not necessarily even be made from components of the original program being sliced. The slice is computed based on the *semantics* of the program. Recently, another form of slicing called *modular monadic slicing* has been developed where slices are computed based on the modular monadic semantics of the program analyzed [99]. This method computes slices directly on abstract syntax of the program without constructing intermediate representations such as dependence graphs.

1.2 Applications of Program Slicing

This section describes the use of program slicing techniques in various applications. In trying to use the basic slicing concepts in diverse domains, several variations of the notions of program slicing as described in Section 1 are developed. The program slicing technique was originally developed to realize automated static code decomposition tools. The primary objective of those tools was to aid program debugging [92, 64]. From this modest beginning, the use of program slicing techniques has now ramified into a powerful set of tools for use in such diverse applications as program understanding, program verification, automated computation of several software engineering metrics, software maintenance and testing, functional cohesion, dead code elimination, reverse engineering, parallelization of sequential programs, software portability, reusable component generation, compiler optimization, program integration, showing differences between programs, software quality assurance, software fault-injection etc. [10, 96, 65, 86, 32, 49, 44, 39, 104, 21, 30, 42, 19]. Slicing methods also play an important role in software fault-injection, see [89] for using slicing methods in software fault-injection. A comprehensive study on the applications of program slicing is made by Binkley and Gallagher [12], Lucia [63] and Qi et al. [82].

1.3 Paper Organization

The remainder of this paper is organized as follows. In Section 2, we discuss the inter-procedural slicing technique, that would be useful in understanding slicing of object-oriented programs. In Section 3, methods for slicing object-oriented programs are discussed. In Section 4, we review the slicing techniques for concurrent object-oriented programs. In Section 5, techniques for slicing of distributed object-oriented programs are discussed. Section 6 concludes the paper.

2 Inter-Procedural Slicing

Horwitz et al. [48] developed the *system dependence graph* (SDG) as an intermediate program representation and proposed a two-phase graph reachability algorithm on the SDG to compute inter-procedural slice. A system dependence graph is a collection of procedure dependence graphs, one for each procedure. A *procedure dependence graph* represents a procedure as a graph in which vertices are statements or predicate expressions and the edges represent the dependence relationships. There are two types of dependence edges: data dependence edge and control dependence edge. *Data dependence* edges represent flow of data between statements or expressions, and *control dependence* edges represent control conditions on which the execution of a statement or expression depends. Each procedure dependence graph contains an *entry vertex* that represents entry into the procedure. To model parameter passing, an SDG associates each procedure entry vertex with *formal-in* and *formal-out* vertices. An SDG contains a formal-in vertex for each formal parameter of the procedure and a formal-out vertex for each formal parameter that may be modified by the procedure. An SDG associates each call site in a procedure with a *call* vertex and a set of *actual-in* and *actual-out* vertices. An SDG contains an actual-in vertex for each actual parameter at the call site and an actual-out vertex for each actual parameter that may be modified by the called procedure. At procedure entry and call sites, global variables are treated as parameters. Thus, there are actual-in, actual-out, formal-in and formal-out vertices for these global variables. SDGs connect procedure dependence graphs at call sites. A *call* edge connects a procedure call vertex to the entry vertex of the called procedure's dependence graph. Parameter-in and parameter-out edges represent parameter passing. *parameter-in* edges connect actual-in and formal-in vertices, and *parameter-out* edges connect formal-out and actual-out vertices. Horwitz et al. compute inter-procedural slices by solving a graph reachability problem on the SDG. To obtain precise slices, the computation of a slice must preserve the calling context of called procedures, and ensure that only paths corresponding to legal $call/return$ sequences are considered. To facilitate the computation of inter-procedural slicing that considers the calling context, an SDG represents the flow of dependencies across call sites. A *transitive flow*

of dependence occurs between the actual-in vertex and an actual-out vertex if the value associated with the actual-in vertex affects the value associated with the actual-out vertex. The transitive flow of dependence may be caused by data dependencies, control dependencies or both. A *summary edge* models the transitive flow of dependence across a procedure call. Fig. 2 represents a simple example program containing two procedures i.e. *add* and *inc*. The system dependence graph of Fig. 2 is shown in Fig. 3. In the SDG of Fig. 3, circles represent program statements and ellipses represent parameter vertices.

After constructing the SDG, Horwitz et al. [48] applied a two-pass algorithm on the SDG to compute the static slices. The first pass of the inter-procedural slicing algorithm traverses backward along all edges except parameter-out edges, and marks those vertices reached. The second pass traverses backward from all vertices marked during the first pass along all edges except call and parameter-in edges, and marks reached vertices. The slice is union of the vertices marked during pass one and pass two.

3 Slicing of Object-Oriented Programs

In this section, we first discuss some work on static slicing of object-oriented programs. Then, we discuss how these basic slicing techniques have subsequently been extended by researchers to handle dynamic slicing of object-oriented programs.

3.1 Static Slicing of Object-Oriented Programs

Static slicing of object-oriented programs has drawn considerable research interest [56, 60, 88, 87, 62, 16, 59, 57, 58, 47, 66, 41]. While slicing object-oriented programs, how to represent the programs is an important problem. Larson and Harrold [60] extended the SDG of Horwitz et al. [48] to represent object-oriented programs. They have constructed Class Dependence Graphs (ClDG) for each class in an object-oriented program. A ClDG captures the control and data dependence relationships that can be determined about a class without knowledge of calling environments. Each method in a ClDG is represented by a procedure dependence graph [60]. Each method has a *method entry* vertex that represents the entry into the method. A ClDG also contains a *class entry* vertex that is connected to the method entry vertex for each method in the class by a *class member* edge. Class entry vertices and class member edges let us quickly access method information when a class is combined with another class or system. The ClDG construction expands each method entry by adding formal-in and formal-out vertices similarly as procedure dependence graphs.

Fig. 4 contains an example program written in C++ which creates the class *Elevator* and *AlarmElevator* depending on the command line arguments. Fig. 5 shows the ClDG for the *Elevator* class. A rectangle represents the class entry vertex and circles represent the statements. The ellipses represent the parameter vertices. For example in Fig. 5, the vertex 1 is the class entry vertex and 2, 6, 7, 9, 11, 13, 15 and 21 are method entry vertices. Bold dashed edges represent class member edges that connect class entry vertex to method entry vertex. For example (1, 2), (1, 6), (1, 7), (1, 9) and (1, 11) are class member edges. Each method entry vertex is the root of a subgraph that is itself a partial SDG containing control dependence edges, data dependence edges, call and parameter edges, and summary edges.

Since methods in a class can interact with each other or with other methods, a ClDG represents the effects of method calls by a *call* vertex. At each call vertex, there are actual-in and actual-out vertices to match the formal-in and formal-out vertices present at the entry to the called method. For example, in Fig. 5, vertices 18 and 20 represent calls to *add*().

To represent *derived class*, Larson and Harrold constructed a ClDG for the derived class by constructing a representation for each method defined by the derived class, and reusing the representations of all methods that are inherited from the base classes [60].

A *polymorphic method call* occurs when a method call is made and the destination of the call is unknown at compile time. The ClDG should represent the polymorphic method call. For this purpose, the ClDG uses a *polymorphic choice vertex* to represent the dynamic choice among the possible destinations. A call vertex corresponding to a polymorphic call has a call edge incident to a polymorphic choice vertex. A polymorphic choice vertex has call edges incident to subgraphs that represent calls to each possible destination. The polymorphic choice vertex represents the dynamic selection of a destination. In fig. 6 $P1$ is a polymorphic choice vertex that represents a dynamic choice between calls to *Elevator::go*() and *AlarmElevator::go*(). The two unlabeled vertices associated with *P1* represent the dummy polymorphic choice vertices.

At last, the SDG for a complete program is constructed by connecting calls in the partial system dependence graph to methods in the ClDG for each class. This process involves connecting call vertices to method entry vertices, actual-in vertices to formal-in vertices, and formal-out vertices to actual-out vertices. The summary edges for methods in a previously analyzed class are added between the actual-in and actual-out vertices at call sites. This construction of the SDG for an object-oriented system maximizes reuse of previously constructed portions of the representation.

Fig. 4 contains an example of an application program that instantiates an object. The SDG of Fig. 4 is given in Fig. 6. The variable *e_ptr* could point to an object of type *Elevator* or *AlarmElevator*. This graph was constructed by building a partial SDG for the *main* function, including the previously computed representation for the *Elevator* and

```
main( )
 int  s, i;
{
  s = 0;                      void  add(int a, int b)        void inc(int z)
  i = 1;                            {                              {
  while (i  < 10) do                  a = a + b;                     add(z,1);
     {                                return;                        return;
         add(s, i);                  }                              }
          inc(i);
      }
  write(s);
 }
```

Figure 2: An example program

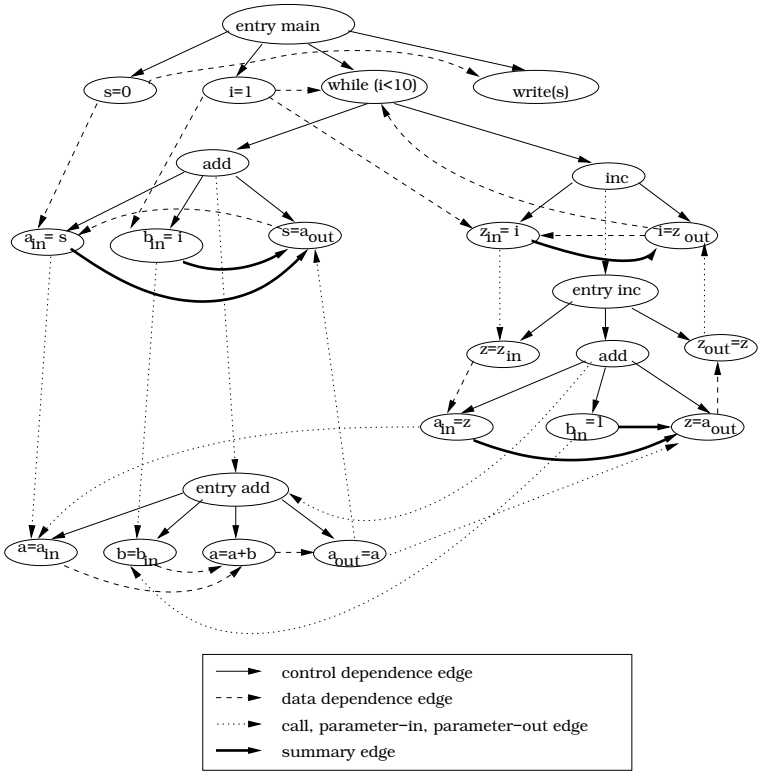


Figure 3: The system dependence graph of the example program of Fig. 2

```
1:  class Elevator{
        public:
2:      Elevator(int 1_top_floor)    /* initialization for Elevator */
3:        { current_floor = 1;
4:          current_direction = UP;
5:          top_floor = 1_top_floor; }     /* end of Elevator */
6:      virtual  ~Elevator() { }
7:      void up()
8:        { current_direction = UP; }
9:      void down()
10:       {  current_direction = DOWN;  }
11:     int which_floor()
12:       { return current_floor;  }
13:     Direction direction()
14:       { return current_direction;  }
15:     virtual void go(int floor)   /* declaration for method go() */
16:       { if  (current_direction = UP )
17:             { while (current_floor != floor)
                      &&  (current_floor <= top_floor)
18:               add(current_floor, 1); }
                  else
19:               { while (current_floor != floor)
                        && (current_floor  > 0 )
20:               add(current_floor, −1);  } /* end if */
            };
          private:
21:       add(int &a, const int &b) /* This method computes value of current_floor */
22:           {  a = a+b; } ;
          protected:
            int current_floor;
            Direction current_direction;
            int top_floor;
          };
23:     class AlarmElevator: public Elevator {  /* AlarmElevator is derived from  Elevato
          public:
24:       AlarmElevator(int top_floor);
25:           Elevator(top_floor)
26:               {alarm_on = 0;   }
27:       void set_alarm()
28:               {alarm_on = 1;  }
29:       void reset_alarm()
30:               {alarm_on = 0;  }
31:       void go(int floor)
32:           { if (! alarm_on)
33:             Elevator :: go(floor);
                };
          protected:
            int alarm_on;
          };
34: main(int argc, char  **argv)  {
          Elevator *e_ptr;
35:     if (argv[1])
36:           e_ptr = new Elevator(10);
          else
37:           e_ptr = new AlarmElevator(10);
38:     e_ptr −> go(3);      /* polymorphic method call */
39:     cout << "\n currently on floor:"
                << e_ptr −> which_floor();
        } /* end of main */
```

Figure 4: An example program

AlarmElevator classes, and connecting each graph using call, parameter-in, and parameter-out edges. In Fig. 6, the left hand side keys represent keys for formal parameter vertices and right hand side keys represent keys for actual parameter vertices.

After constructing the SDG for a complete object-oriented program, they have used the two-pass graph reachability algorithm [48] for computing slices. Fig. 6 shows the SDG of the example program given in Fig. 4 and the static slice with respect to the call to *which_floor()* at vertex 39, which includes all statements that may affect *current_floor*. The shaded vertices in the SDG represent the statements included in the slice. The static slice is shown in Fig. 7 in more detail. Since Larson and Harrold [60] have

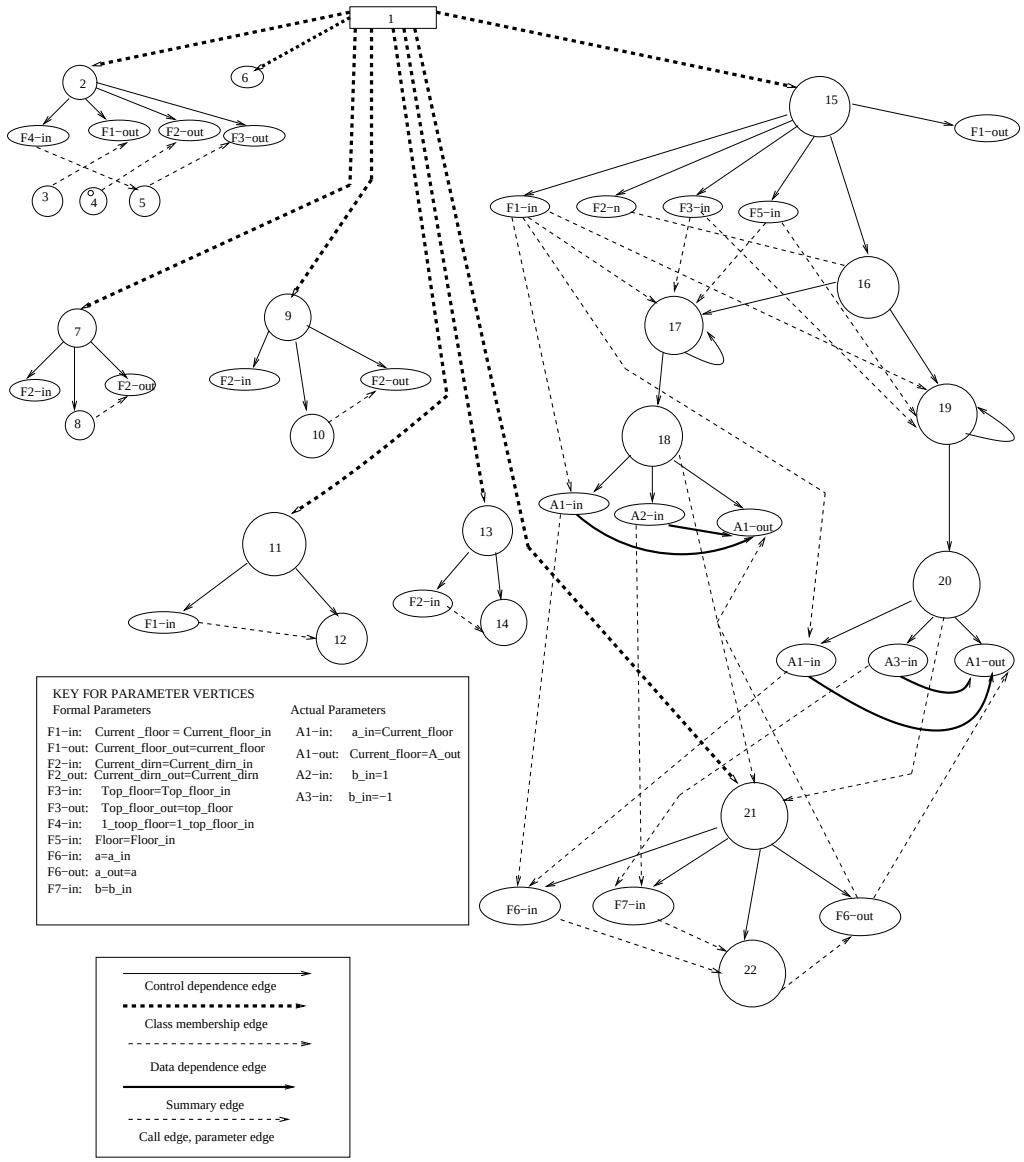


Figure 5: The ClDG for class *Elevator*

computed the static slice, so all most all of the statements in the example program are included in the slice.

One limitation of this approach is that the data dependencies obtained using the approach for creating the individual procedure dependence graphs are imprecise: by treating data members declared in a class as if they were global to the methods of that class, the approach fails to consider the fact that in different method invocations, the data members used by the methods might belong to different objects. A second limitation of the approach is that it does not handle cases in which an object is used as a parameter or as a data member of another object.

Tonella et al. [88] have addressed the first limitation by extending a methods signature to include data members of the class as formal parameters so that an object can pass its data members into the method as actual parameters. Their approach, however, is unnecessarily expensive because each method call site has actual parameter vertices for all data members of the object, even if only a few of them are referenced by the method. They addressed the second limitation by representing an object as a single vertex when the object is used as a parameter. This representation, however, might cause the slicer to produce imprecise slices because the slice may include all the data members of the object even if a few of them affects the slicing criterion.

Liang et al. [62] developed a more efficient intermediate representation to overcome the above limitations. To obtain more precision when an object is used as a parameter (*parameter object*), their modified SDG explicitly represents the data members of the object. They have represented the parameter object as a *tree*. The root of the tree represents the object itself, the children of the root represent the data members of the object and the edges of the tree represent the data dependencies between the object and it's

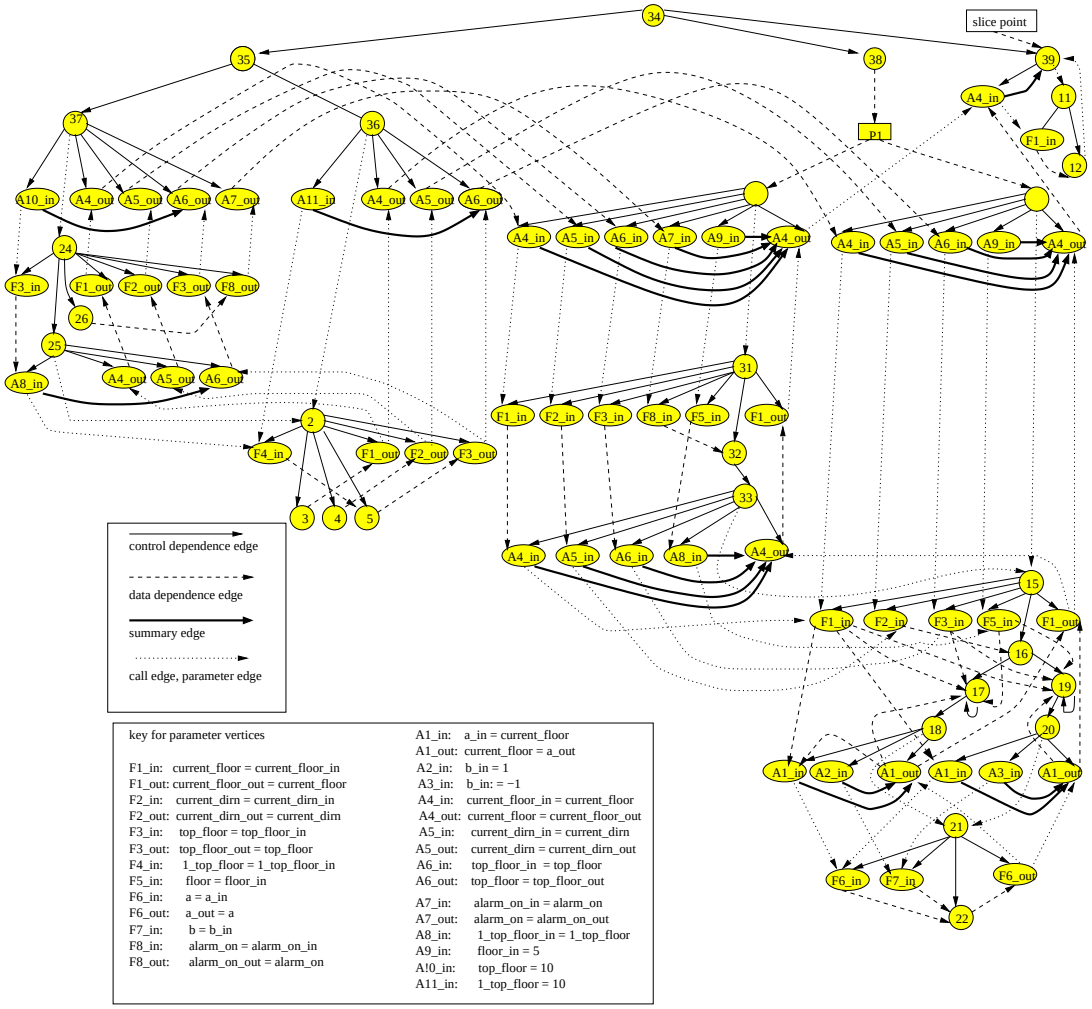


Figure 6: The system dependence graph of Fig. 4

data members. Under this representation, if a data member of the object is another object, then this data member can be further expanded into a subtree.

In this representation, a *polymorphic object* is represented as a tree in which the root of the tree represents the polymorphic object itself and the children of the root represent objects of the possible types. When the polymorphic object is used as a parameter, the children are further expanded into trees. When the polymorphic object receives a message, the children are further expanded into call sites. Note that, in this case, the technique of Liang et al. [62] differs from that of Larson and Harrold [60]. Liang et al. have used one call site for each possible object type, in their representation. But in the representation of Larson and Harrold, different call sites are used only for different implementations of a virtual method.

To represent inheritance, Liang et al. [62] have maintained one copy of the representation for a method within a class hierarchy. Then, this representation can be shared by different classes in the hierarchy. The class entry vertex in the SDG groups the methods belonging to one class together using *class member edges* [60]. But in some cases, a method might require a new representation when the program dependence graph for a new class to the hierarchy, is constructed. Liang et al. [62] suggested that, a method will require a new representation, if

- the method is declared in the new class, or
- the method is declared in a lower level class in the hierarchy and calls a newly redefined virtual method directly or indirectly.

Liang et al. [62] have also introduced a new concept called *object slicing*, which enables the user to inspect the effects of a particular object on the slicing criterion. Object slicing provides better support for debugging and program understanding for large scale programs. Sometimes the user may like to focus attention on one object at a time. To do this, they have designed a method to identify the statements in the methods of a particular object that might affect the slicing criterion.

The shortcomings of their method are that:

1. When slicing the object, we must obtain the complete

```
1:  class Elevator{
        public:
2:      Elevator(int 1_top_floor)          /* initialization for Elevator */
3:        { current_floor = 1;
4:          current_direction = UP;
5:          top_floor = 1_top_floor; }              /* end of Elevator */
6:        virtual  ~Elevator() { }
7:        void up()
8:           { current_direction = UP; }
9:        void down()
10:          {  current_direction = DOWN;  }
11:     int which_floor()
12:       { return current_floor;  }
13:       Direction direction()
14:          { return current_direction;  }
15:     virtual void go(int floor)          /* declaration for method go() */
16:       { if  (current_direction = UP )
17:           { while (current_floor != floor)
                  &&  (current_floor <= top_floor)
18:           add(current_floor, 1); }
              else
19:           { while (current_floor != floor)
                  && (current_floor  > 0 )
20:           add(current_floor, −1);  }                    /* end if */
            };
          private:
21:       add(int &a, const int &b)          /* This  method computes value of current_floor */
22:         {  a = a+b; } ;
          protected:
            int current_floor;
            Direction current_direction;
             int top_floor;
           };
23:     class AlarmElevator: public Elevator    /* AlarmElevator is derived from  Elevator */
          public:
24:       AlarmElevator(int top_floor);
25:         Elevator(top_floor)
26:           {alarm_on = 0;   }
27:         void set_alarm()
28:               {alarm_on = 1;  }
29:         void reset_alarm()
30:               {alarm_on = 0  }
31:       void go(int floor)
32:         { if (! alarm_on)
33:           Elevator :: go(floor)
                };
            protected:
                int alarm_on;
            };
34: main(int argc, char  **argv)  {
          Elevator *e_ptr;
35:     if (argv[1])
36:         e_ptr = new Elevator(10);
          else
37:         e_ptr = new AlarmElevator(10);
38:     e_ptr −> go(3);          /* polymorphic method call */
39:     cout << "\n currently on floor:"
                << e_ptr −> which_floor();
          }/* end of main */
```

Figure 7: The static slice of Fig. 4 on slicing criterion (39, current_floor)

slice first for the program. This might be too expensive.

2. When an object's method invokes other methods or is invoked by other methods, we must traverse backward through several methods.

Hammer et al. [41] proposed a new slicing algorithm for Java, which includes all dependencies between fields of nested objects but is more precise than previous algorithms [60, 88, 62]. Instead of limiting the tree level, Hammer et al. [41] unfold the tree completely. As this is not possible for recursive data structures, they have presented a *condition for safe termination of unfolding*. The condition is based on points-to information. This method keeps all trees finite but guarantees that no dependencies are lost. Points-to information is also used to constrain run-time targets of method calls. As a by-product, a call graph is extracted. But, even the best points-to analysis will not resolve all object polymorphism, and the object trees must represent all possible run-time types of an object. Unlike [62], Hammer et al. [41] do not represent polymorphic objects as a set of trees, but as one *merged tree*. To disambiguate fields with the same name but defined in different classes, they have used the fully qualified field name. Thus merging does not reduce the precision of the final SDG. It just reduces the size of the SDG. The short coming of this

approach is that it is more expensive than [60, 62].

Krishnaswamy [56] proposed a different approach to slicing object-oriented programs. He used another dependence-based representation called the *object-oriented program dependency graph* (OPDG) to represent the object-oriented programs. The OPDG of an object-oriented program represents control flow, data dependencies and control dependencies. The OPDG representation of an object-oriented program is constructed in three layers, namely: *Class Hierarchy Subgraph* (CHS), *Control Dependence Subgraph* (CDS), and *Data Dependence Subgraph* (DDS). The CHS represents inheritance relationship between classes, and the composition of methods into a class. A CHS contains a single *class header node* and a *method header node* for each method that is defined in the class. Inheritance relationships are represented by edges connecting class headers. Every method header is connected to the class header by a *membership edge*. Subclass representations do not repeat representations of methods that are already defined in the super classes. *Inheritance edges* of a CHS connect the class header node of a derived class to the class header nodes of it's super classes. *Inherited membership edges* connect the class header node of the derived class to the method header nodes of the methods that it inherits. A CDS represents the static control dependence relationships that exists within and among the different methods of a class. The DDS represents the data dependence relationship among the statements and predicates of the program. The OPDG of an object-oriented program is the union of three subgraphs: CHS, CDS and DDS. Slices can be computed using OPDG as a graph-reachability problem. He also computed the polymorphic slices of object-oriented programs based on the OPDG.

The OPDG of an object-oriented program is constructed as the classes are compiled and hence it captures the complete class representations. The main advantage of OPDG representation over other representations is that the representation has to be generated only once during the entire life of the class. It does not need to be changed as long as the class definition remains unchanged. Fig. 8 represents the CHS of the example program of Fig. 4.

Kung et al. [59, 57, 58] presented a representation for object-oriented software. Their model consists of an *object relation diagram* and a *block branch diagram*. The object relation diagram of an object-oriented program provides static structural information on the relationships existing between objects. It models the relationship that exists between classes such as inheritance, aggregation and association. The block branch diagram of an object-oriented program contains the control flow graph of each of the class methods, and presents a static implementation view of the program. Harrold and Rothermel [47] presented the concept of *Call Graph*. A call graph provides a static view of the relationship between object classes. A call graph is an inter-procedural program representation in which nodes represent individual methods and edges represent call sites. However, a call graph does not represent important object-oriented concepts such as inheritance, polymorphism and dynamic binding.

Chen et al. [14] proposed an intermediate representation called *Object-Oriented Dependency Graph* (ODG) to represent object-oriented programs. The ODG is a multi-diagraph which is extended from a directed graph by augmenting multiple edge types, vertex properties, and property relations. With this extension, the ODG can avoid some dependencies due to object encapsulation. Based on the ODG, Chen et al. [14] presented an algorithm for slicing of object-oriented programs.

Chen et al. [15] defined two types of program slices, state and behavior slices by considering the dependencies of object-oriented features. A state slice for an object is a set of messages and control statements that might affect the state of the object. A behavior slice for an object is a set of attributes and methods defined in related classes that might affect the behavior of the object.

Although, these approaches [56, 60, 88, 87, 62, 16, 59, 57, 58, 47, 66, 41] represent many features of object-oriented programs, still there are some drawbacks with these approaches. First, these techniques are not fit to represent larger programs, because all the procedure dependence graphs of subprograms are connected in the SDG, and for a large program the SDG will be too large to manage and understand. Second, the existing techniques only slice statements in methods of a class. A class consists of a set of methods and data members. Statement slicing is not enough to analyze and understand classes. Finally, to improve the efficiency, most of the PDGs of methods should be reused.

To overcome these drawbacks, Chen and Xu [18] have proposed a new approach to represent dependence for object-oriented Java software that is quite different from the existing SDG representations [60, 88, 62, 56], which connect all PDGs of methods. This new program dependence graph is a set of PDGs with tags that are not connected. The PDG of a class consists of a set of PDGs of its methods. Each PDG is an independent graph, and does not connect to any other PDGs. The *tags* have the form (x, y) (where x and y are variables) and are used to distinguish the different definitions and dependencies in a statement. They have used the following sets in their approach: *def(s)*, *ref(s)*, *Def(s, x)*, *Dep_D(s, x)*, and *Dep_R(s)*. They have defined these sets as follows:

- Def(s) denotes the variables whose values are defined (modified) at s. The *in* formal parameters are defined at entry node of the subprogram.
- Ref(s) denotes the variables whose values are refereed, but not modified at s.
- Def(s, x) denotes the variables used when defining variable x at s.
- Dep_D(s, x) = $\{(x, s^1, y)$, such that $y \in Def(s, x) and y \in Def(s^1)$ and there exists a path from s^1 to s on which y is not redefined$\}$.

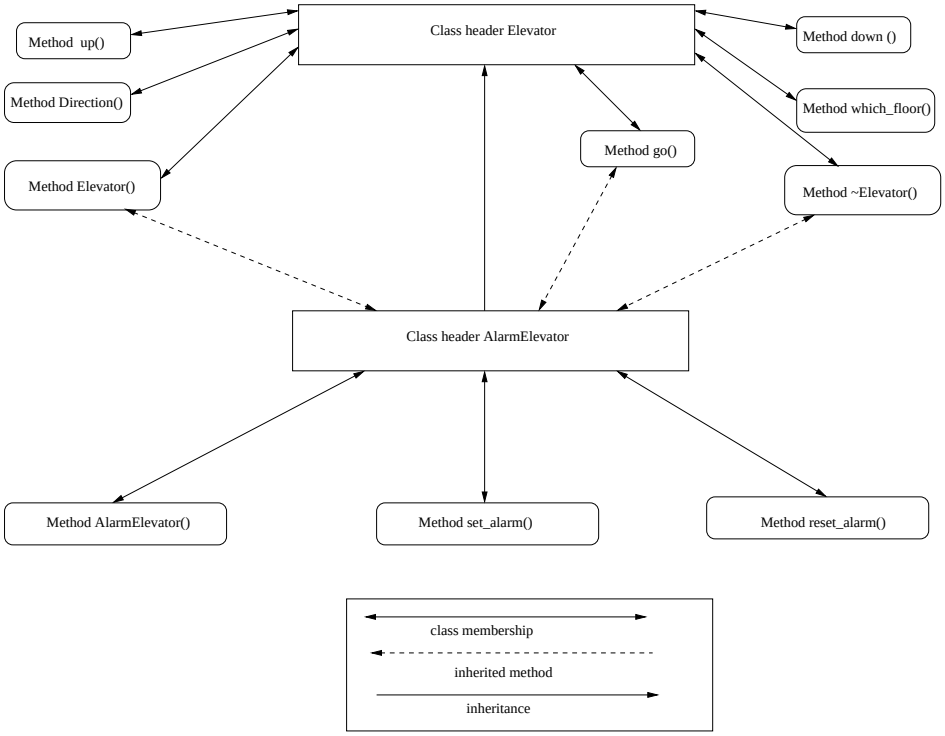


Figure 8: The CHS for the example program of Fig. 4

– Dep_R(s) = $\{(x, s^1, x)$, such that if s is a control statement, x is a conditional variable used at s, and $x \in Def(s^1)$ and there exists a path from s^1 to s on which x is not redefined$\}$.

Chen and Xu [18] defined the program dependence graph (PDG) of a method M as a directed graph with tags. According to this approach, PDG is triplet $< S^1, E^1, T >$, where node set $S^1 = S$ (S is the node set of M's CFG), edge set $E^1 = E_1 \cup E_2$, whereE_1 is the set of direct control dependence edges and $E_2 = \{< s_1, s_2 > suchthat(x, s_2, y) \in Dep\_D(s_1, x) and (x, s_2, x) \in Dep\_R(s_1)\}$ is the set of direct data dependence edges. T is a tag set. The tag on an edge $< s_1, s_2 >$ can be obtained in the following way:

– If $< s_1, s_2 > \in E_1$, then its tag is (*, *);

– If $< x, s_2, y > \in Dep\_D(s_1)$, then its tag is (y, x);

– If $< x, s_2, x > \in Dep\_R(s_1)$, then its tag is (x, x);

For example, consider the sample program in Fig. 4. The PDG of the method *Elevator*, according to the approach of Chen and Xu [18], is shown in Fig. 9. Similarly the PDG of other methods can be drawn. According to the approach of Chen and Xu [18], in Fig. 9, the tag for the edge (2, 3) is (current_floor, 1), the tag for edge (2, 4) is (current_direction, UP) and the tag for the edge (2, 5) is (top_floor, 1). There are three classes, *Elevator*, *AlarmElevator* and *main*, in the sample program. The PDG of class *Elevator* is shown in Fig. 10. It may be observed that the PDG of each method in Fig. 10,is independent. Using the PDG of a method, Chen and Xu [18] solved intra-method slicing as a graph-reachability problem with tags. The approach of Chen and Xu [18] differs from the previous approaches [48] in that it checks not only the edges but also the tags on these edges. Based on this new model, they have introduced the concepts of *partial slicing*, *object slicing* and *class slicing*.

Partial slicing: partial slicing can make the user pay attention to the interesting parts of the program, and slice incomplete programs or components from a third party without source codes. Informally, given a slicing criterion $< s, v >$, the partial slicing only slices the interested parts of the program such as a class, few methods of a class or an object. To slice parts of a program, they have constructed the PDGs of interested subunits. For other methods, only the interfaces, i.e., the dependencies among parameters are needed. It is enough to know the interface (how to use the method) of incomplete programs or components from a third party. When we construct the PDG of a program, all PDGs of methods have been constructed. Based on these PDGs, we can use the partial slicing algorithm. In the slicing algorithm, each method is sliced independently. If the method is not considered, we just do not slice it.

Object slicing: Object slicing was first introduced by Liang et al. [62]. It is mainly used for tasks, such as debugging and program understanding. Object slicing identifies statements in methods of an object that might affect the slicing criterion. To slice an object, the slicing criterion is changed to $< s, v, Object >$. Informally, given a slicing criterion $< s, v, Object >$, object slicing identifies the

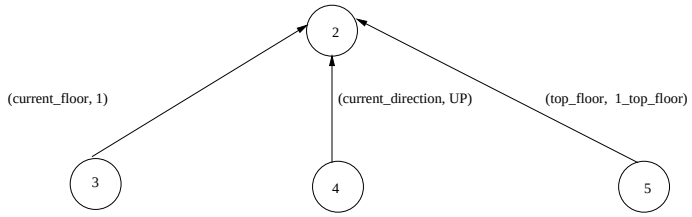


Figure 9: The PDG of method *Elevator* of the example program of Fig. 4

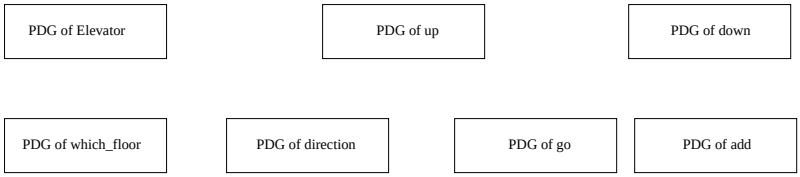


Figure 10: The PDG of class *Elevator* of the example program of Fig. 4

statements in the methods of the object that might affect slicing criterion $< s, v >$.

Class slicing: Class slicing identifies data members and statements in methods of the class that might affect the slicing criterion. Class slicing slices not only the methods, but also all the data members. To slice a class, the slicing criterion is changed to $< s, v, Class >$. Informally, given a slicing criterion $< s, v, Class >$, the result of class slicing of a class is a class that includes partial data members and statements in the methods of class, and these data members and statements might influence the variable defined at s. To slice a class, one method is to union all the object slices of the class and record the data members used. When the number of objects is large, this method will be too expensive. Another way is that, when constructing the PDG, we do not distinguish data members for different objects instantiated from the same class. But using such PDG will lose much information that is useful for other slicing. The best way is to traverse backward from s when slicing and mark the statements and data members used in the class based on this new PDG.

The advantages of this approach [18] are that:

- It distinguishes data members for different objects and represents the effects of polymorphism and dynamic binding.
- Using this representation, the PDGs can be constructed concurrently as each PDG is independent. So, this representation is quite fit for representing larger programs.
- Object slicing enables users to inspect statements in a slice, object by object. Class slicing enables users to inspect not only the statements in methods but also data members in classes.
- According to this slicing algorithm, when the slicing criterion changes, most PDGs need not be traversed, because the previous results that are saved on disks, can be reused.

The shortcoming of this approach is that when we only slice once or few times, the cost might be too much, because all the methods are analyzed first before slicing and the results are stored in libraries on disk.

Steindl [85] has developed a fully operational program slicing tool, *Oberon Slicing Tool*, for the programming language *Oberon-2*. It generates state-of-the-art algorithms and applies them to a strongly-typed object-oriented programming language. It extends them to support intermodular slicing of object-oriented programs. Control and data flow analysis considers inheritance, dynamic binding and polymorphism, as well as side-effects of functions, short circuit evaluation of Boolean expressions and aliases due to reference parameters and pointers. The algorithm for alias analysis is fast but effective by taking into account information about the type of variables and the place of their declaration. The result of static program analysis is visualized with active text elements: hypertext links connect the call sites with the possible call destinations, parameter information elements indicate the direction of data flow at calls. Since static program analysis must make conservative assumptions about actual program executions, the sets of possible aliases and call destinations due to dynamic binding are more general than necessary. Steindl has visualized these sets and allowed the programmer to restrict them via user interaction. These restrictions are then used to compute more precise control and data flow information. In this way, the programmer can limit the effects of aliases and dynamic binding and bring in his knowledge about the program into the analysis.

The disadvantages of this technique are:

- The layout of the the original source code is lost.
- The front-end of the compiler skips all comments, so they are lost and cannot be displayed.
- The front-end of the compiler performs some simple optimizations such as constant folding, transformation

of IF statements with constant conditions, replacement of integer multiplication by a power of two by arithmetic shifts, etc. These optimizations cannot be undone and the results are presented to the user. This may give insights, but may also confuse.

– The reconstruction of the source code is difficult, the module implementing the reconstruction and the user interface is very big, approximately 3000 lines.

3.2 Dynamic Slicing of Object-oriented Programs

Korel and Laski [52] introduced a new form of slicing. This new form of slicing is dependent on input data and is generated during execution-time analysis as opposed to Weiser's static slicing [92] and is therefore called *dynamic slicing*. Similar to the major objective of static slicing, dynamic slicing was specifically designed as an aid to debugging, and can be used to help in the search for offending statements which caused the program error [63].

Considerable research results on dynamic slicing of procedural programs are available [4, 52, 3, 49, 2, 78, 79, 37, 27, 98]. But dynamic slicing of object-oriented programs have scarcely been reported in the literature [100, 84, 94, 90].

Agrawal and Horgan [4] were the first to present algorithms for finding dynamic program slices using program dependence graphs. They proposed a dynamic slicing method by marking nodes on a static program dependence graph. The computed slice is not always precise, because some dependencies might not hold in dynamic execution. They also proposed a precise method based on the dynamic dependence graph (DDG) [4]. Zhao [100] extended the DDG of Agrawal and Horgan [4], known as *dynamic object-oriented dependence graph* (DODG) to represent various dynamic dependencies between statement instances for a particular execution of an object-oriented program. The DODG is an arc-classified diagraph (V, A), where V is the multi-set of flow-graph vertices, and A is the set of arcs representing dynamic control dependencies and data dependencies between vertices. Zhao's construction of DODG is based on dynamic analysis of control flow and data flow of the program, and similar to those for constructing dynamic dependence graphs for procedural programs [2]. Zhao constructed the DODG by creating a new node for each occurrence of a statement in the execution trace, and creating all the dependence edges associated with the occurrence at run-time. The execution trace of the example program in Fig. 4 on input argument *argv[1] = 3*, is given in Fig. 11. Fig. 12 shows the DODG of the example program in Fig. 4 with respect to the execution trace in Fig. 11.

Zhao [100] has considered the specific features of object-oriented programs such as method calls, inheritance, polymorphism and dynamic binding etc. in his algorithm. Zhao has regarded a call statement in an object-oriented program as one of the following statements:

– a statement that calls a free standing procedure,

– a statement that has function application,

– a statement that creates an object,

– a statement that invokes a method, or

– a statement that returns a value to its caller.

Using similar techniques proposed by Agrawal and Horgan [4], Zhao has solved the problem of representing a call statement in the DODG.

Zhao has adopted the following concepts for dynamic slicing of object-oriented programs:

– A *slicing criterion* for an object-oriented program is of the form *(s, v, t, i)*, where s is a statement in the program, v is a variable used at s, and t is an execution trace of the program with input i.

– A *dynamic slice* of an object-oriented program on a given slicing criterion *(s, v, t, i)* consists of all statements in the program that actually affected the value of a variable v at statement s.

Based on the DODG, Zhao has used a two-phase algorithm to compute dynamic slices of object-oriented programs. Computation of dynamic slices using the DODG is carried out as a graph-reachability problem. The two phases of the algorithm are:

1. Computing a dynamic slice over the DODG of the object-oriented program.
   (This can be done by using a usual depth-first or breadth-first graph traversal algorithm to traverse the DODG of the program by taking the vertex corresponding to the statement of interest as the start point of traversal.)

2. Mapping the slice over the DODG to the source code to obtain a dynamic slice of the program.
   (This can be done by simply defining a mapping function.)

It may be noted that the dynamic slice computed by Zhao [100] is not executable. This is in contrast to that presented in [52] which defines a dynamic slice as an executable subprogram. For program debugging and testing, a non-executable dynamic slice can also supply enough information as an executable dynamic slice, but can be computed more easily.

Fig. 13 shows the dynamic slice of the example program in Fig. 4 with respect to the slicing criterion *(39, current_floor, t, argv[1] = 3)*, where t is the execution trace given in Fig. 11. The statements within the boxes are included in the slice. It can be marked that the size of the resulting dynamic slice is reduced significantly compared to its corresponding static slice. The disadvantage of Zhao's approach is that the number of nodes in a DODG is equal

```
34(0)    main(argc, char  **argv)
35(0)    if   (argv[1])
37(0)    e_ptr = new Elevator(10);
2(0)     Elevator(int 1_top_floor);
3(0)     current_floor = 1;
4(0)     current_direction  = UP;
5(0)     top_floor = 1_top_floor;
38(0)    e_ptr          -.-.-.-.-.-> go(3);
15(0)       virtual void go(int floor);
16(0)    if (current _direction  = UP)
17(0)    while ( (current_floor  !=  floor) && (current_floor) <= top_floor));
18(0)     add(current_floor, 1),       ;
21(0)     add(int &a, const  int  &b);
22(0)      a = a + b;
17(1)    while ( (current_floor  !=  floor) && (current_floor) <= top_floor));
18(1)     add(current_floor, 1);
21(1)     add(int &a, const  int  &b);
22(1)      a = a + b;
17(2)    while ( (current_floor  !=  floor) && (current_floor) <= top_floor));
39(0)    cout  <<  "\n currently on floor:" << e_ptr          -.-.-.-.-.-> which_floor()  <<"
11(0)    int which_floor ( );
12(0)    return current_floor;
```

Figure 11: An execution trace of the example program in Fig. 4 on input argv[1] = 3.

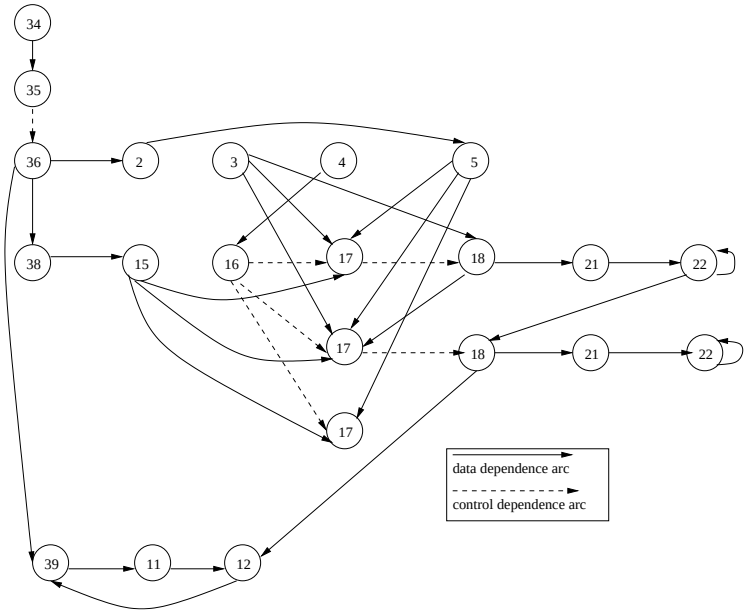


Figure 12: The DODG of the program of Fig. 4 on input argv[1]=3

to the number of executed statements, which may be unbounded for programs having many loops. Further, Zhao has used trace files to store the execution history which is expensive. The space complexity and the time complexity of this dynamic slicing algorithm are of $O(S)$ and $O(S^2)$, respectively, where S is the length of execution of the program.

Song et al.[84] proposed a method to compute forward dynamic slice of object-oriented programs using dynamic object relationship diagram (DORD). In this method, they computed the dynamic slices for each statement immediately after the statement is executed. When the last statement is executed, the dynamic slices of all executed statements have been obtained. However, only some special statements in the loops need to compute dynamic slices. So the dynamic slices computed by this technique is unnecessarily expensive.

Xu et al. [94] extended their earlier method [18] to dynamically slice object-oriented programs. Their method uses object program dependence graph (OPDG) and other static information to reduce the information to be traced during execution. Their method computes dynamic slices

```
1:  class Elevator{
        public:
2:      Elevator(int 1_top_floor)            /* initialization for Elevator */
3:        { current_floor = 1;
4:          current_direction = UP;
5:          top_floor = 1_top_floor; }                  /* end of Elevator */
6:      virtual  ~Elevator() { }
7:      void up()
8:         { current_direction = UP; }
9:      void down()
10:        {  current_direction = DOWN;  }
11:   int which_floor()
12:     { return current_floor;  }
13:     Direction direction()
14:       { return current_direction;  }
15:   virtual void go(int floor)           /* declaration for method go() */
16:     { if  (current_direction = UP )
17:         { while (current_floor != floor)
                 &&  (current_floor <= top_floor)
18:           add(current_floor, 1); }
              else
19:           { while (current_floor != floor)
                   && (current_floor  > 0 )
20:             add(current_floor, −1);  } /* end if */
          };

        private:
21:     add(int &a, const int &b)          /* This  method computes value of current_floor */
22:       {  a = a+b; } ;
          protected:
            int current_floor;
            Direction current_direction;
             int top_floor;
          };
23:     class AlarmElevator: public Elevator    /* AlarmElevator is derived from  Elevator */
          public:
24:         AlarmElevator(int top_floor);
25:           Elevator(top_floor)
26:             {alarm_on = 0;   }
27:         void set_alarm()
28:             {alarm_on = 1;  }
29:         void reset_alarm()
30:             {alarm_on = 0  }
31:         void go(int floor)
32:           { if (! alarm_on)
33:             Elevator :: go(floor)
                };
           protected:
              int alarm_on;
           };
34: main(int argc, char  **argv)  {
          Elevator *e_ptr;
35:     if (argv[1])
36:       e_ptr = new Elevator(10);
          else
37:       e_ptr = new AlarmElevator(10);
38:     e_ptr −> go(3);          /* polymorphic method call */
39:     cout << "\n currently on floor:"
              << e_ptr −> which_floor();
        }/* end of main */
```

Figure 13: The dynamic slice of the example program in Fig. 4 on slicing criterion (39, current_floor, t, argv[1] = 3).

by combining static dependence information and dynamic execution of the program. By analyzing the control flow graph of the given program, fewer breakpoints are inserted to trace the execution of the program. It is an approach combining forward analysis with backward one. In the forward process, it marks nodes on the OPDG and computes intermediate dynamic slices (which are used to record dynamic execution information) at the necessary points during the program execution. In the backward process, it traverses the OPDG marked to obtain the final dynamic slice. Based on this model, they have proposed algorithms to dynamically slice methods, objects and classes.

Wang et al. [90] presented a new dynamic slicing algorithm for Java programs which operates on compact byte code traces. According to their algorithm, first, the byte code stream corresponding to an execution of a Java program is compactly represented. Then, they perform a backward traversal of the compressed program trace to compute data/control dependencies on-the-fly. The slice is updated as these dependencies are encountered during the traversal.

The compactness of the trace representation is owing to several factors. First, byte codes which do not correspond to memory read/write (i.e., data transfer to and from the heap) or control transfer are not stored in the trace. These byte codes can be ignored for computing control and data dependencies. Secondly, the sequence of addresses used by

each memory reference, control transfer byte code is stored separately. Since these sequences typically have high repetition of pattern, they exploit such repetition to save space. They have extended the dynamic slicing algorithm to explain certain classes of omission errors.

The important advantage of their technique is that it is more space efficient than that of Zhao [100] since they use the results from data compression to compactly represent byte code traces of Java programs. The major space savings come from the optimized representation of data (instruction) addresses used by memory reference (branch) byte codes as operands. Also, their algorithm can directly traverse the compact traces without restoring to costly decompression. The disadvantage of this approach is that it uses trace files, which are expensive to handle.

Mohapatra et al. [70, 72] proposed a new algorithm for dynamic slicing of object-oriented programs. They have used *extended system dependence graph* (ESDG) as the intermediate representation. They have statically constructed the ESDG only once before the execution of the program starts. Their algorithm is based on marking and unmarking the edges of the ESDG as and when the dependencies arise and cease during run-time. So, they have named their algorithm *edge marking dynamic slicing* (EMDS) algorithm for object-oriented programs. The EMDS algorithm marks an edge of the ESDG when the corresponding dependency arises and unmarks an edge when the dependency ceases to exist. Mohapatra et al. [70, 74] also proposed another algorithm called *node marking dynamic slicing* (NMDS) algorithm for object-oriented programs. The NMDS algorithm also uses ESDG as the intermediate representation. The NMDS algorithm is based on marking and unmarking the executed nodes of the ESDG appropriately during runtime. The space complexity of both the algorithms (EMDS and NMDS) is $O(n^2)$, where n is the number of statements in the program. The time complexity of both the algorithms (EMDS and NMDS) is $O(n^2S)$, where S is the length of the execution trace. Each vertex of ESDG is annotated with its most recent dynamic slice during execution of program. Thus, slices can be extracted in constant time i.e., in $O(1)$ time.

The advantage of both the algorithms [72, 74] compared to the related ones [100, 84, 94, 90] is that they do not not require any new nodes to be created and added to the intermediate representation at run-time nor do they require to maintain any execution trace in trace files. This saves the expensive node creation and file I/O steps. Another important advantage of their algorithms is that when a request for a slice is made, it is already available. Once a slicing command is given the algorithms produce results almost instantus through a mere table-lookup and avoid on-demand slicing computation. They have shown that the EMDS and NMDS algorithms are more space and time efficient than the related algorithms [100, 84, 94, 90]. They have also shown that the NMDS algorithm is faster than the EMDS algorithm. Table 1 shows the comparison of various dynamic slicing algorithms for object-oriented programs.

Ohata et al. [81] observed that static slicing cannot compute precise slices and dynamic slicing requires too much computation time and memory space. So, they adopted an intermediate slicing method between static and dynamic slicing called *Dependence-Catch* (DC) slicing to object-oriented programs. DC slicing method uses dynamic data dependence analysis and static control dependence analysis. Dependence-Catch slicing computes more precise slices than static slicing and needs less computation time and memory space.

4 Slicing of Concurrent Object-Oriented Programs

Concurrent object-oriented programs are becoming more popular. Many of the real life object-oriented programs are concurrent which run on different machines connected to a network. It is usually accepted that understanding and debugging of concurrent object-oriented programs are much harder compared to those of sequential programs. The nondeterministic nature of concurrent programs, lack of global states, unsynchronized interactions among objects, multiple threads of control and a dynamically varying number of objects are some reasons for this difficulty [6, 9, 8]. An increasing amount of resources are being spent in debugging, testing and maintaining these products. Slicing techniques promise to come in handy at this point. However research attempts in the program slicing area have focused attention largely on sequential programs. But research reports dealing with slicing of concurrent object-oriented programs are scarce in literature [104, 101, 102, 17, 103, 105, 82, 97, 73].

4.1 Static Slicing of Concurrent Object-Oriented Programs

Static slicing of *concurrent procedural programs* has drawn the attention of many researchers [36, 35, 38, 7]. Also, static slicing of *concurrent object-oriented programs* has been addressed by some researchers [17, 103, 102, 105, 82, 97]. Excellent surveys on static slicing of concurrent object-oriented programs can be found in [20].

Zhao et al. [104] presented a dependence based representation called the *system dependence net* (SDN) which extends the previous dependence based representations [60] to represent various dependence relationships in concurrent object-oriented programs. An SDN of a concurrent object-oriented program consists of a collection of dependence graphs each representing a main procedure, a free standing procedure, or a method in a class of the program. It also consists of some additional arcs to represent direct dependencies between a call and the called procedure/method and transitive inter-procedural data dependencies. To represent interprocess communications between different methods in a class of a concurrent object-oriented program, they have introduced a new type of program dependence arc named as *external communication depen-*

Table 1: Comparison of algorithms for dynamic slicing of object-oriented programs

Approach	Category of slice	Break points inserted	Trace files used
Zhao	backward	no	yes
Song et al.	forward	no	yes
Xu et al.	backward	yes	yes
Wang et al.	backward	no	yes
Mohapatra et al.	backward	no	no

dence arc into the SDN. An SDN can be used to represent either object-oriented features or concurrency issues in a concurrent object-oriented program.

Based on the SDN, Zhao et al. [104] have used the two-phase algorithm [48] to compute static slices of concurrent object-oriented programs such as CC++. In CC++, synchronization between different threads is realized by using a single assignment variable. Threads that share access to a single assignment variable can use that variable as a synchronization element. Their system dependence net (SDN) is an extension of the SDG of Larson and Harrold [60] and therefore can be used to represent many object-oriented features in a CC++ program. To handle concurrency issues in CC++, they used an approach proposed by Cheng [22] which was originally used for representing concurrent procedural programs with a single procedure each. However, their approach, when applied to concurrent Java programs suffers from some problems due to the fact that the concurrency models of CC++ and Java are essentially different. While Java supports monitors and some low level thread synchronization primitives, CC++ uses a single assignment variable mechanism to realize thread synchronization. This difference leads to different sets of concurrency constructs in both the languages, and therefore requires different techniques to handle concurrency issues in computing slices.

Zhao [102] has also presented a dependence-based representation called the *multi-threaded dependence graph* (MDG) to represent concurrent Java programs. The MDG is composed of a collection of *thread dependence graphs* (TDG) each representing a single thread in the program, and some special kinds of dependence arcs to represent thread interactions between different threads. The TDG is used to represent a single thread in a concurrent Java program and is similar to the SDG [60]. The TDG of a thread is an arc-classified diagraph that consists of a number of *method dependence graphs* each representing a method, and some special kinds of dependence arcs to represent direct dependencies between a call and the called method and transitive inter-procedural data dependencies in the thread. The method dependence graph is similar to the procedure dependence graph proposed by Horwitz [48]. To represent synchronization among threads and communication among shared objects in different threads, Zhao has used two special types of dependence arcs in the MDG. He has used *synchronization dependence arcs* to represent dependence relationships between different threads due to inter-thread synchronization and *communication dependence arcs* to represent dependence relationships between different threads due to inter-thread communication.

Zhao [102] has constructed the MDG for a complete concurrent Java program by combining the TDGs for all threads in the program at synchronization and communication points by adding synchronization and communication dependence arcs between these points. Based on the MDG, Zhao [102] has presented a two-phase algorithm for computing static slices of concurrent Java programs.

Zhao et al. [105] developed another dependence-based representation called *concurrent program dependence graph* (CPDG) to represent program dependencies in a concurrent Java program. The CPDG is a diagraph which consists of a collection of dependence graphs each representing a single method in the class. Also, it includes a few additional vertices and arcs to model parameter passing between different methods in a class, and inter-thread synchronization and communication between different threads. Zhao et al. [105] used the two phase algorithm [48] to compute static slices of concurrent Java programs.

Cheng [23] introduced an intermediate representation called *program dependence net* (PDN) for parallel and distributed programs. Cheng [23] has also discussed various possible applications of PDN including slicing concurrent programs. Cheng has defined a dynamic slicing criterion of a concurrent program as a quadruplet (*s, V, H, I*), where s is a statement in the program, V is a set of variables used at s, and H is a history of an execution of the program with input I. According to Cheng, the dynamic slice *DS(s, V, H, I)* of a concurrent program on a given slicing criterion (*s, V, H, I*) consists of all statements in the program that actually affected the beginning or end of execution of s and/or affected the values of variables in V at s in the execution with I that produced H.

All these approaches [103, 102, 105, 23] slice concurrent programs by solving a node reachability problem in the graph. A shortcoming of these algorithms is that the resulting slice is not precise since they consider that dependencies between concurrently executed statements are transitive. But, in practice, the dependencies between concurrently executed statements are not transitive due to the presence of synchronization dependence and communica-

tion dependence [17].

To get a more precise slice than that of Zhao [102] and Cheng [23], Krinke [54] introduced a slicing algorithm without synchronization. Krinke has introduced a new type of dependence called *interference dependence*, among threads. In Krinke's algorithm, the interference dependence is not transitive. So, the resulting slice is more precise. However, synchronization is widely used in concurrent programs and in some environments unavoidable. Thus Krinke's algorithm can be used only in some restricted applications.

Krinke [55] has also developed another technique for context sensitive slicing of concurrent programs. In this technique, Krinke has extended the control flow graph (CFG) and program dependence graph (PDG) [48] to represent concurrent programs with interference. This technique does not require serialization or inlining of called procedures. Nanda and Ramesh [80] have extended Krinke's technique [54] to compute static slices of concurrent programs with synchronization. In their approach, they have considered *loop-carried data dependence* while computing the slice. They have proposed some optimizations to slice more efficiently. They have claimed that it could get near linear behavior for many practical concurrent programs.

Qi and Xu [82] have developed a *task synchronization reachability graph* (TSRG) for analyzing concurrent Ada programs. Based on the TSRG, they determine the synchronization dependencies in a concurrent Ada program, and construct a new type of program dependence graph, *TSRG-based program dependence graph* (RPDG). They have discussed various applications of RPDG including program understanding, debugging, testing and software maintenance etc. A limitation of this approach is that, it does not consider the communication dependencies in a concurrent program. But, communication dependencies do exist in many practical situations and is normally unavoidable in a concurrent object-oriented program. This makes Qi and Xu's approach [82] difficult to use in many practical situations.

Chen and Xu [17] have developed *concurrent control flow graphs* (CCFG) and *concurrent program dependence graphs* (CPDG) to represent concurrent Java programs. Based on the CPDG, they proposed a static slicing algorithm for concurrent Java programs [17]. In their algorithm, they have considered the fact that the inter-thread data dependencies are not transitive. So, the resulting slice is more precise than that of Zhao [102] and Cheng [23].

All the reported approaches [104, 101, 102, 23, 54, 55, 17, 82] focus on static slicing. They have not considered the *dynamic slicing* aspects.

4.2 Dynamic Slicing of Concurrent Object-Oriented Programs

Reports on dynamic slicing of concurrent object-oriented programs are scarcely available in the literature [71, 73, 76, 77].

Mohapatra et al. [71] extended the dynamic slicing algorithm of Zhao [100] to compute dynamic slices of concurrent object-oriented programs. They have used *dynamic multi-threaded dependence graph* (DMDG) as the intermediate representation. The DMDG is an arc-classified diagraph *(V, A)*, where V is the multi-set of flow graph vertices, and A is the set of arcs representing dynamic control dependencies, data dependencies, synchronization dependencies and communication dependencies between the vertices. Based on the DMDG, they have used a two-phase algorithm to compute dynamic slices of concurrent object-oriented programs. The space complexity and the time complexity of this algorithm are of $O(S)$ and $O(S^2)$, respectively, where S is the length of the execution trace. The disadvantage of this approach is that they have used a trace file to store the execution history, which is expensive.

Mohapatra et al. [70, 73, 77] have also proposed another algorithm for dynamic slicing of concurrent Java programs without using trace files. They have used *concurrent control flow graph* (CCFG) and *concurrent system dependence graph* (CSDG) as the intermediate representations. According to their approach, first the CCFG is constructed statically. Then, the CSDG is constructed by using the CCFG. A concurrent system dependence graph (CSDG) G_C of a concurrent object-oriented program *P* is a directed graph (N_C, E_C) where each node $n \in N_C$ represents a statement in P. For *x*, $y \in N_C$, $(x,y) \in E_C$ *iff* one of the following holds:

1. *y* is *control dependent* on *x*. Such an edge is called a *control dependence edge*.
2. *y* is *data dependent* on *x*. Such an edge is called a *data dependence edge*.
3. *y* is *synchronization dependent* on *x*. Such an edge is called a *synchronization dependence edge*.
4. *y* is *communication dependent* on *x*. Such an edge is called a *communication dependence edge*.

Based on the CSDG, they have proposed a marking based dynamic slicing (MBDS) algorithm for concurrent Java programs. The MBDS algorithm is based on marking and unmarking the edges of the CSDG as and when the dependencies arise and cease during run-time. MBDS algorithm permanently marks the control dependence edges as control dependencies do not change during program execution. The algorithm considers all the data dependence edges, synchronization dependence edges and communication dependence edges for marking and unmarking during run-time. During execution of the program P, MBDS algorithm marks an edge of the CSDG when its associated dependence exists, and unmarks when its associated dependence ceases to exist. After each statement *u* is executed, MBDS algorithm unmarks all incoming *marked* dependence edges excluding the control dependence edges, associated with the object *obj*, corresponding to the *previous* execution of the statement *u*. Then, the algorithm

marks the dependence edges corresponding to the *present* execution of the statement *u*.

MBDS algorithm operates in three main stages:

Stage 1: Statically constructing the intermediate program representation graph,

Stage 2: Managing the CSDG at run-time, and

Stage 3: Computing the dynamic slice.

In the first stage of MBDS algorithm, the CCFG is constructed from a static analysis of the source code. Also at this stage, using the CCFG the static CSDG is constructed. The stage 2 of the algorithm is responsible for maintaining the CSDG during run-time. The maintenance of the CSDG at run-time involves marking and unmarking the different dependencies such as data dependencies, synchronization dependencies and communication dependencies as they arise and cease. The stage 3 is responsible for computing the dynamic slices for a given slicing criterion using the uptodate CSDG. However, the third step is simply a look up as the dynamic slice computed during run-time is already available. So, when a request for a slice is made, it is immediately obtained. The space complexity of the MBDS algorithm is $O(n^2)$, where n is the number of statements in the program. The time complexity of the MBDS algorithm is $O(n^2S)$, S being the length of the execution trace. Each node of the CSDG is annotated with its most recent dynamic slice during execution of the program. Thus, the dynamic slices can be looked up in constant time i.e., in $O(1)$ time.

The important features of the MBDS algorithm are listed below.

- It computes *correct* dynamic slices with respect to any slicing criterion.
- It can handle *inter-thread synchronization* by using primitives such as *wait*() and *notify*().
- It can handle *inter-thread communication* through *shared objects*.
- No trace files are used. All information are maintained and updated dynamically for all threads and are discarded at run-time of a program on termination of a thread.
- It does not create any additional nodes during run-time. This saves the expensive node creation steps.
- When a request for a slice is made, it is already available.
- No serialization of the events of the concurrent program is required.
- As MBDS algorithm marks an edge of the CSDG only when the dependence exists, so the *transitive problem* [17] does not arise at all. So, MBDS algorithm often results in slices that are more precise.
- It can be easily extended to compute dynamic slices of distributed object-oriented programs as each component program of the whole distributed program can be considered as a single concurrent program.

Mohapatra et al. [70, 77] have developed a slicing a tool called *Dynamic Slicer for Concurrent Object-Oriented Programs* (DSCOP) to implement the MBDS algorithm. DSCOP can compute the dynamic slice of a concurrent object-oriented program with respect to any given slicing criterion. DSCOP can handle only a subset of the Java syntax. However, the tool supports inter-thread synchronization and inter-thread communication using shared memory. The lexical analyzer, parser and semantic analyzer components of DSCOP have been implemented using *ANTLR* (Another Tool for Language Recognition) [1, 67]. During semantic analysis, the input program code is appropriately instrumented so as to facilitate computation of dynamic slices and to update other associated run-time data structures after execution of each statement, as described in the MBDS algorithm. The Compile and Execute block of DSCOP compiles and links the instrumented source code using the Java compiler.

5 Slicing of Distributed Object-Oriented Programs

As software applications grow larger and become more complex, program maintenance activities such as adding new functionalities, porting to new platforms, and correcting the reported bugs consume enormous effort. This is especially true for distributed object-oriented programs. In order to cope with this scenario, programmers need effective computer-supported techniques for decomposition and dependence analysis of programs. Program slicing is one technique for such decomposition and dependence analysis.

Many real life object-oriented programs are distributed in nature and run on different machines connected to a network. The emergence of message passing standards, such as MPI, and the commercial success of high speed networks have contributed to making message passing programming common place. Message passing programming has become an attractive option for tackling the vexing issues of portability, performance, and cost effectiveness. As distributed computing gains momentum, development and maintenance tools for these distributed systems seem to gain utmost importance.

Development of real life distributed object-oriented programs presents formidable challenge to the programmer. Distributed object-oriented programs introduce several problems which do not exist in sequential programs. The non-reproducible behaviors, non-deterministic selection of communication events, lack of global states and unsynchronized interactions among threads are some of the problems which arise in case of distributed object-oriented

programs [83]. An increasing amount of effort is being spent in debugging, testing and maintaining these products. Slicing techniques promise to come in handy at this point. Through the computation of a slice for a message passing program, one can significantly reduce the amount of code that a maintenance engineer has to analyze to achieve some maintenance tasks. However, research attempts in program slicing area have focused attention largely on sequential programs. Slicing of distributed procedural programs [26, 51, 28, 22, 50, 24, 61] has also drawn the attention of many researchers. But, research reports on slicing of distributed object-oriented programs are scarcely available in the literature [34, 75].

Goel et al. [34] proposed compression schemes for representing execution profiles of shared memory parallel programs. Their representation captures control flow, data flow and synchronization in the execution of a shared memory multi-threaded program running on a multiprocessor architecture. According to their approach the control and data flow of each processor is maintained individually as whole program paths (WOP). The total order of the synchronization operations executed by all processors and the annotation of each processor's WOP with synchronization counts help to capture the inter-processor communications which are protected via synchronization primitives such as *lock*, *unlock* and *barriers*. They have illustrated the applications of compact execution traces in program debugging, program comprehension, code optimization, memory layout etc. They have used trace files to store the execution history. This leads to slow I/O operations. They have considered that the communication across different threads occurs only via synchronization primitives. Communication via shared variable accesses is not explicitly represented in their method. We have considered communications among threads through shared variables as well as message passing.

Garg et al. [33] introduced the notion of a slice of a *distributed computation*. They have defined the slice of a distributed computation with respect to a global predicate, as a computation which captures *those and only those* consistent *cuts* of the original computation which satisfy the global predicate. A *computation slice* differs from a *dynamic slice* in that it is defined for a property rather than a set of variables of a program. Unlike a program slice, which always exists, a computation slice may not always exist. They have proved that the slice of a distributed computation with respect to a predicate exists iff the set of consistent cuts that satisfy the predicate, forms a sub lattice of the lattice of consistent cuts. Mittal and Garg [68, 69] presented an efficient algorithm to *graft* two slices, that is, given two slices, either compute the smallest slice that contains all consistent cuts that are common to both slices or compute the smallest slice that contains all consistent cuts that belong to at least one of the slices.

Mohapatra et al. [75] were the first to propose an algorithm for dynamic slicing of distributed object-oriented programs. They have introduced the notion of *distributed program dependence graph* (DPDG) as the intermediate program representation. In distributed object-oriented programs, communication dependency may exist among sub programs running on different machines. A *rcvmsg*() call executed on one machine, might have a pairing *sndmsg*() on some other remote machine. To represent this aspect, they have introduced a logical (dummy) node in the DPDG. They have named this logical node as a *C-node*. They have defined a C-node in the following way:

Let G_{D_1} and G_{D_2} be the DPDGs of two sub programs P_1 and P_2 respectively. Let x be a node in G_{D_1} representing a statement invoking a *sndmsg*() method. Let y be a node in G_{D_2} representing the statement invoking the corresponding *rcvmsg*() method. A *C-Node* represents a logical connection of the node y of DPDG G_{D_1} with the node x of the remote DPDG G_{D_2}. Node x represents the pairing of *sndmsg*() with a *rcvmsg*() call at node y. Node y is *Communication dependent* on node x.

The *C-nodes* maintain the logical connectivity among DPDGs representing different sub programs. A *C-node* does not represent any specific statement in the source code of a sub program. Rather, it encapsulates the triplet: $<$ *send_PID*,*send_node_number*, *dynamic_slice_at_send_node* $>$ representing the pairing of the components in a distributed program. Here, *send_PID* represents the *id* of the process sending the message, *send_node_number* represents the particular label number of the statement sending the message and *dynamic_slice_at_send_node* represents the dynamic slice at the sending node. *C-nodes* capture communication dependencies among the processes of different sub programs. It may be noted that the number of *C-nodes* in the DPDGs of a distributed C++ program, equals the number of *rcvmsg*() calls present in the program. In the DPDG, for a *rcvmsg*() node *x*, the corresponding *C-node* is represented as *C(x)*.

They have defined a distributed program dependence graph (DPDG) in the following way:

Let $P = (P_1, \ldots, P_n)$ be a distributed C++ program, and P_i be a sub program of P. P is represented using a set of DPDGs $(G_{D_1}, \ldots G_{D_n})$. The distributed program dependence graph (DPDG) G_{D_i} of the component-program P_i is a directed graph (N_{D_i}, E_{D_i}) where each node *n* (excepting the dummy nodes) represents a statement in P_i. For *x, y* $\in$ N_{D_i}, (y,x) $\in$ E_{D_i} *iff* any one of the following holds:

1. *y* is *control dependent* on *x*. Such an edge is called a *control dependence edge*.
2. *y* is *data dependent* on *x*. Such an edge is called a *data dependence edge*.
3. *y* is *fork dependent* on *x*. Such an edge is called a *fork dependence edge*.
4. *y* is *communication dependent* on *x*. Such an edge is called a *communication dependence edge*.

For all the nodes x, representing *rcvmsg*() calls, in the sub program P_i, a dummy node *C(x)* is created, and a corresponding dummy communication edge (x, C(x)) is added.

The set of DPDGs for each sub program of the distributed program is constructed statically only once before the execution of the distributed program starts. Based on the DPDG, Mohapatra et al. [75] have proposed an algorithm for dynamic slicing of distributed object-oriented programs. They have named their algorithm *parallel dynamic slicing* (PDS) algorithm as the algorithm can run parallely on several machines connected through a network. The PDS algorithm is based on marking and unmarking the edges of the DPDG as and when the dependencies arise and cease at run-time. To achieve fast response time, the PDS algorithm can run parallely on several machines connected through a network. For this purpose, we use local slicers at each remote machine. Our slicing algorithm in effect operates as the coordinated activities of local slicers running at the remote machines. Each local slicer contributes to the dynamic slice by determining its local portion of the global slice in a fully distributed fashion.

The PDS algorithm addresses the concurrency issues of object-oriented programs while computing the dynamic slices. It also handles the communication dependency arising due to objects shared among processes on same machine and due to message passing among processes on different machines. The space complexity of the PDS algorithm is $O(N^2)$, N being the total number of statements of the distributed program. The time complexity of the PDS algorithm is $O(N^2S)$, where S is the total length of execution of the distributed program.

The advantage of PDS algorithm is that it does not require any trace file to store the execution history. Another important advantage of their algorithm is that when a slicing command is given, the dynamic slice is extracted immediately by looking up the appropriate data structure, as it is already available during run-time.

Mohapatra et al. [70] also have developed another algorithm for distributed dynamic slicing of Java programs. They have named their algorithm distributed dynamic slicing (DDS) algorithm for Java programs. To achieve fast response time, DDS algorithm can run in a fully distributed manner on several machines connected through a network, rather than running it on a centralized machine. They have used local slicers at each node in a network. A local slicer is responsible for slicing the part of the program executions occurring on the local machine.

DDS algorithm uses a modified program dependence graph (PDG) [48] as the intermediate representation. This intermediate representation is called as distributed program dependence graph (DPDG). First, the DPDG is constructed statically before run-time. DDS algorithm marks and unmarks the edges of the DPDG appropriately as and when dependencies arise and cease during run-time. Such an approach is more time and space efficient and also completely does away with the necessity to maintain a trace file. This eliminates the slow file I/O operations that occur while accessing a trace file. Another advantage of DDS algorithm is that when a request for a slice for any slicing criterion is made, the required slice is already available. This appreciably reduces the response time of slicing commands.

Mohapatra et al. [70] have developed a slicing tool to implement the DDS algorithm. The tool can compute the dynamic slice of a distributed Java program with respect to a given slicing criterion. The tool handles only a subset of Java language constructs. They have named their tool *Dynamic Slicer for Distributed Java programs* (DSDJ). To construct the intermediate graphs they have used the compiler tool *ANTLR* [1, 67]. A distributed Java program is given as the input to the ANTLR program. The ANTLR program automatically generates the DPDGs for the component programs. The lexical analyzer, parser and semantic analyzer components of DSDJ are combined and the joint component is termed as program analysis component [5]. The lexical analyzer, parser and semantic analyzer components of DSDJ have been implemented using *ANTLR* [1, 67]. During semantic analysis, the Java source code is analyzed token by token to gather the various program dependencies. The tokens are first used to construct the DCFG (Distributed Control Flow Graph). Next, using the DCFG the corresponding DPDG (Distributed Program Dependence Graph) is constructed. The source program is then automatically instrumented, by adding calls to the slicer module after every statement in the source program. After the execution of each statement, the *update_slice*() method is invoked, which marks and unmarks the edges of the DPDG appropriately and updates the dynamic slice. For storing the dynamic slice of each statement they have used a two dimensional integer array. When the dynamic slice of a particular statement is requested, the *compute_slice*() method is invoked, and it provides the dynamic slice for the given slicing criterion.

6 Conclusions

We have reviewed the recent works in the area of object-oriented program slicing including static slicing of object-oriented programs, dynamic slicing of object-oriented programs, static slicing of concurrent object-oriented programs and dynamic slicing of concurrent object-oriented programs. We have presented a brief review on slicing of distributed object-oriented programs. We have also discussed some available tools for slicing of object-oriented programs. Starting with the basic sequential program constructs researchers are now trying to address various issues of slicing distributed object-oriented programs. Since modern software products are often large and consist of millions of lines of code, processing a single data structure becomes very slow and therefore development of parallel algorithms for slicing has assumed importance.

References

[1] Antlr. http://www.antlr.org/.

[2] H. Agrawal, R. A. DeMillo, and E. H. Spafford. Dynamic slicing in the presence of unconstrained pointers. In *Proceedings of the ACM Fourth Symposium on Testing, Analysis and Verification (TAV4)*, pages 60 – 73, 1991.

[3] H. Agrawal, R. A. DeMillo, and E. H. Spafford. Debugging with dynamic slicing and backtracking. *Software Practice and Experience*, 23(6):589 – 616, 1993.

[4] H. Agrawal and J. Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN'90 Conference on Programmimg Lanuages Design and Implementation, SIGPLAN Notices, Analysis and Verification*, volume 25, pages 246 – 256, White Plains, NewYork, 1990.

[5] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[6] G. R. Andrews. *Concurrent Programming: Principles and Practice*. Addison-Wesley, 1991.

[7] G. R. Andrews and F. B. Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys*, 15:3 – 43, 1983.

[8] M. Awad and J. Ziegler. A practical approach to the design of concurrency in object-oriented systems. *Software Practice and Experience*, 27:1013 – 1034, 1997.

[9] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice Hall, 1990.

[10] D. Binkley. The application of program slicing to regression testing. *Information and Software Technology, Special Issue on Program Slicing*, 40(11-12):583 – 594, 1998.

[11] D. Binkley. Computing amorphous program slices using dependence graphs and a data flow model. In *Proceedings of the ACM Symposium on Applied Computing, ACM Press*, 1999.

[12] D. Binkley and K. B. Gallagher. *Program Slicing, Advances in Computers*, volume 43. Academic Press, San Diego, CA, 1996.

[13] G. Canfora, A. Cimitile, and A. D. Lucia. Conditioned program slicing. *Information and Software Technology*, 40:595 – 607, 1998.

[14] J. Chen, F. Wang, and Y. Chen. An object-oriented dependency graph. Technology of Object-Oriented Languages and Systems Tools, Beijing, China, 1997.

[15] J. Chen, F. Wang, and Y. Chen. Slicing object-oriented programs. In *4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC-97 / ICSC-97)*, Hong Kong, 1997.

[16] J. T. Chen, F. J. Wang, and Y. L. Chen. Slicing object-oriented programs. In *Proceedings of the APSEC'97*, pages 395 – 404, Hongkong, China, December 1997.

[17] Z. Chen and B. Xu. Slicing concurrent Java programs. *ACM SIGPLAN Notices*, 36:41 – 47, 2001.

[18] Z. Chen and B. Xu. Slicing object-oriented Java programs. *ACM SIGPLAN Notices*, 36:33 – 40, 2001.

[19] Z. Chen, B. Xu, and H. Yang. Test coverage analysis based on program slicing. In *Proceedings of IRI*, pages 559 – 565, 2003.

[20] Z. Chen, B. Xu, and J. Zhao. An overview of methods for dependence analysis of concurrent programs. *ACM SIGPLAN Notices*, 37(8):45 – 52, 2002.

[21] Z. Chen, Y. Zhou, B. Xu, J. Zhao, and H. Yang. A novel approach for measuring class cohesion based on dependence analysis. In *Proceedings of International Conference on Software Maintenance, IEEE Press*, pages 377 – 384, 2002.

[22] J. Cheng. Slicing concurrent programs - a graph theoretical approach. In *Automated and Algorithmic Debugging, AADEBUG'93, LNCS, Springer-Verlag*, pages 223 – 240, 1993.

[23] J. Cheng. Dependence analysis of parallel and distributed programs and its applications. In *International Conference on Advances in Parallel and Distributed Computing*, pages 370 – 377, 1997.

[24] J. D. Choi, B. Miller, and R. Netzer. Techniques for debugging parallel programs with flowback analysis. *ACM Transactions on Programming Languages and Systems*, 13:491 – 530, 1991.

[25] S. Danicic, M. Daoudi, C. Fox, M. Harman, R. M. Hierons, J. R. Howroyd, L. Ourabya, and M. Ward. ConSUS: a light-weight program conditioner. *Journal of Systems and Software*, 2005.

[26] S. Danicic, Mark Harman, and Yoga Sivagurunathan. A parallel algorithm for static program slicing. *Information Processing Letters*, 56:307 – 313, 1995.

[27] D. M. Dhamdhere, K. Gururaja, and P. G. Ganu. A compact execution history for dynamic slicing. *Information Processing Letters*, 85:145 – 152, 2003.

[28] E. Duesterwald, R. Gupta, and M. L. Soffa. Distributed slicing and partial re-execution for distributed programs. In *Fifth Workshop on Languages and Compilers for Parallel Computing, New Haven Connecticut, LNCS Springer-Verlag*, pages 329 – 337, August 1992.

[29] J. Field, G. Ramalingam, and F. Tip. Parametric program slicing. In *Conference Record of the Twenty-Second ACM Symposium on Principles of Programming Languages*, pages 379 – 392, San Francisco, CA, USA, 1995.

[30] I. Forgacs and A. Bertolino. Feasible test path selection by principal slicing. In *Proceedings of 6th Euoropean Software Engineering Conference*, September 1997.

[31] C. Fox, S. Danicic, M. Harman, and R. M. Hierons. Consit: a fully automated conditioned program slicer. *Software Practice and Experince*, 34:15 – 46, 2004.

[32] K. Gallagher and J. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, SE-17(8):751 – 761, 1991.

[33] V. K. Garg and N. Mittal. On slicing a distributed computation. In *Proceedings of 21st IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 322 – 329, 2001.

[34] A. Goel, A. RoyChoudhury, and T. Mitra. Compactly representing parallel program executions. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 191 – 202, 2003.

[35] D. Goswami and R. Mall. Fast slicing of concurrent programs. In *Sixth International Conference on High Performance Computing (HiPC), LNCS Springer-Verlag*, pages 38 – 42, December 1999.

[36] D. Goswami and R. Mall. Dynamic slicing of concurrent programs. In *Seventh International Conference on High Performance Computing (HiPC), LNCS Springer-Verlag*, pages 17 – 26, December 2000.

[37] D. Goswami and R. Mall. An efficient method for computing dynamic program slices. *Information Processing Letters*, 81:111 – 117, 2002.

[38] D. Goswami, R. Mall, and P. Chatterjee. Static slicing in unix process environment. *Software Pracice and Experience*, 30:17 – 36, 2000.

[39] R. Gupta, M. J. Harrold, and M. L. Soffa. Program slicing-based regression testing techniques. *Journal of Software Testing, Verification and Reliability*, 6, 1996.

[40] R. Gupta and M. L. Soffa. Hybrid slicing: An approach for refining static slices using dynamic information. In *Proceedings of ACM SIGSOFT*, pages 29 – 40, 1995.

[41] C. Hammer and G. Snelting. An improved slicer for Java. In *Proceedings of PASTE*, pages 107 – 112, 2004.

[42] M. Harman. Conditioned slicing supports partition testing. *Journal of Software Testing, Verification and Reliability*, 12:23 – 28, 2002.

[43] M. Harman, D. Binkley, and S. Danicic. Amorphous program slicing. *Journal of Systems and Software*, 68:45 – 64, 2003.

[44] M. Harman and S. Danicic. Using program slicing to simplify testing. *Journal of Software Testing, Verification and Reliability*, 5, 1995.

[45] M. Harman and R. M. Hierons. An overview of program slicing. *Software Focus*, 2:85 – 92, 2001.

[46] M. Harman, L. Hu, M. Mumro, X. Zhang, D. Binkley, and S. Danicic. Syntax-directed amorphous slicing. *Automated Software Engineering*, 11:27 – 61, 2004.

[47] M. J. Harrold and G. Rothermel. Performing data flow testing on classes. In *Second ACM SIGSOFT Symposium on the Foundation of Software Engineering*, pages 154 – 163, December 1994.

[48] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26 – 61, 1990.

[49] M. Kamkar. *Inter Procedural Dynamic Slicing with Applications to Debugging and Testing*. PhD thesis, Linkoping University, Sweden, 1993.

[50] M. Kamkar and P. Krajina. Dynamic slicing of distributed programs. In *International Conference on Software Maintenance, IEEE CS Press*, pages 222 – 229, October 1995.

[51] B. Korel and R. Ferguson. Dynamic slicing of distributed programs. *Applied Mathematics and Computer Science*, 2:199 – 215, 1992.

[52] B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155 – 163, 1988.

[53] B. Korel and J. Rilling. Dynamic program slicing methods. *Information and Software Technology*, 40:647 – 659, 1998.

[54] J. Krinke. Static slicing of threaded programs. *ACM SIGPLAN Notices*, 33:35 – 42, April 1998.

[55] J. Krinke. Context-sensitive slicing of concurrent programs. In *Proceedings of ACM SIGSOFT Software Engineering Notes*, pages 178 – 187, 2003.

[56] A. Krishnaswamy. Program slicing: An application of program dependency graphs. Technical report, Department of Computer Science, Clemson University, August 1994.

[57] D. Kung, J. Gao, P. Hisa, and Y. Toyoshima. Change impact identification in object-oriented software maintenance. In *Proceedings of International Conference on Software Maintenance*, pages 202 – 211, September 1994.

[58] D. Kung, J. Gao, P. Hisa, and Y. Toyoshima. Firewall regression testing and software maintenance of object-oriented systems. *Journal of Object-Oriented Programming*, 1994.

[59] D. Kung, J. Gao, P. Hisa, Y. Toyoshima, and C. Chen. Design recovery for software testing of object-oriented programs. In *Working Conference on Reverse Engineering*, pages 202 – 211, May 1993.

[60] L. D. Larson and M. J. Harrold. Slicing object-oriented software. In *Proceedings of the 18th International Conference on Software Engineering*, German, March 1996.

[61] Hon. F. Li, Juergen Rilling, and Dhrubajyoti Goswami. Granularity-driven dynamic predicate slicing algorithms for message passing systems. *Automated Software Engineering*, 11:63 – 89, 2004.

[62] D. Liang and L. Larson. Slicing objects using system dependence graphs. In *Proceedings of International Conference on Software Maintenance*, pages 358 – 367, November 1998.

[63] A. D. Lucia. Program slicing: Methods and applications. In *Proceedings of IEEE International Workshop on Source Code Analysis and Manipulation*, pages 142 – 149, 2001.

[64] J. R. Lyle and M. D. Weiser. Automatic program bug location by program slicing. In *Proceedings of the second International Conference on Computers and Applications, Peking, China*, pages 877 – 882, 1987.

[65] R. Mall. *Fundamentals of Software Engineering*. Prentice Hall, India, 2nd Edition, 2003.

[66] B. A. Malloy, J. D. McGregor, and A. Krishnaswamy. An extensible program representation for object oriented software. In *Proceedings of ISFST*, pages 105 – 112, 2004.

[67] A. J. S. Mills. Antlr. The University of Birmingham, 2002.

[68] N. Mittal and V. K. Garg. Computation slicing: Techniques and theory. Technical Report, TR-PDS-2001-02, The Parallel and Distributed Systems Laboratory, Department of Electrical and Computer Engineering, The University of Texas at Austin, 2001.

[69] N. Mittal and V. K. Garg. Computation slicing: Techniques and theory. In *Proceedings of Symposium on Distributed Computing*, 2001.

[70] Durga Prasad Mohapatra. *Dynamic slicing of object-oriented programs*. PhD thesis, Indian Institute of Technology, Kharagpur, India, 2005.

[71] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. Dynamic slicing of concurrent object-oriented programs. In *Proceedings of International Conference on Information Technology: Progresses and Challenges (ITPC)*, pages 283 – 290, Kathamandu, May 2003.

[72] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. An edge marking dynamic slicing technique for object-oriented programs. In *Proceedings of 28th IEEE Annual International Computer Software and Applications Conference, IEEE CS Press*, pages 60 – 65, September 2004.

[73] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. An efficient techinque for dynamic slicing of concurrent Java programs. In *Proceedings of Acian Applied Conference on Computing (AACC-2004), Kathmandu, LNCS Springer-Verlag*, volume 3285, pages 255 – 262, October 2004.

[74] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. A node marking dynamic slicing technique for object-oriented programs. In *Proceedings of Workshop on Software Development and Architecture (SoDA)*, pages 1 – 15, Bangalore, January 2004.

[75] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. A novel approach for dynamic slicing of distributed object-oriented programs. In *Proceedings of International Conference on Distributed Computing and Internet Technology (ICDCIT), Bhubaneswar, LNCS Springer-Verlag*, volume 3347, pages 304 – 309, December 2004.

[76] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. A novel method for computing dynamic slices of concurrent C++ programs. In *Proceedings of International Conference on Advanced Computing and Communications*, pages 744 – 750, Ahmedabad, December 2004.

[77] Durga Prasad Mohapatra, Rajib Mall, and Rajeev Kumar. Computing dynamic slices of concurrent object-oriented programs. *Information and Software Technology*, 2005.

[78] G. B. Mund, R. Mall, and S. Sarkar. An efficient dynamic program slicing technique. *Information and Software Technology*, 44:123 – 132, 2002.

[79] G. B. Mund, R. Mall, and S. Sarkar. Computation of intraprocedural dynamic program slices. *Information and Software Technology*, 45:499 – 512, April 2003.

[80] M. G. Nanda and S. Ramesh. Slicing concurrent programs. In *ACM International Symposium on Software Testing and Analysis*, August 2000.

[81] F. Ohata, K. Hirose, M. Fuji, and K. Inoue. A slicing method for object-oriented programs using dynamic light weight information. In *Eighth Asia-Pacific Software Engineering Conference (APSEC-01)*, China, 2001.

[82] X. Qi and B. Xu. Dependence analysis of concurrent programs based on rechability graph and it's applications. In *Proceedings of International Conference on Computational Science*, pages 405 – 408, 2004.

[83] M. Singhal and N. G. Sivaratri. *Advanced Concepts in Operating Systems - Distributed, Database, and Multiprocessor Operating Systems*. TATA McGRAW HILL, 2002.

[84] Y. Song and D. Huynh. *Forward Dynamic Object-Oriented Program Slicing, Application Specific Systems and Software Engineering and Technology (ASSET'99)*. IEEE CS Press, 1999.

[85] C. Steindl. *Program slicing for object-oriented programming languages*. PhD thesis, Johannes Kepler University Linz, 1999.

[86] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121 – 189, 1995.

[87] F. Tip, J. D. Choi, J. Field, and G. Ramalingam. Slicing class hierarchies in C++. In *Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 179 – 197, 1996.

[88] P. Tonella, G. Antoniol, R. Fiutem, and E. Merlo. Flow insensitive C++ pointers and polymorphism analysis and its application to slicing. In *Proceedings of 19th International Conference on Software Engineering*, pages 433 – 443, May 1997.

[89] M. Jeffrey Voas and Gary McGraw. *Software fault-injection: inoculating programs against errors*. Wiley and Sons, 1998.

[90] T. Wang and A. RoyChoudhury. Using compressed bytecode traces for slicing Java programs. In *Proceedings of IEEE International Confrence on Software Engineering*, pages 512 – 521, 2004.

[91] M. Weiser. *Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method*. PhD thesis, University of Michigan, Ann Arbor, MI, 1979.

[92] M. Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446 – 452, 1982.

[93] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352 – 357, 1984.

[94] B. Xu and Z. Chen. Dynamic slicing object-oriented programs for debugging. In *SCAM*, pages 115 – 122, 2002.

[95] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen. A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes*, 30(2):1 – 36, 2005.

[96] L. Xu, B. Xu, Z. Chen, J. Jiang, H. Chen, and H. Yang. Regression testing for web applications based on slicing. In *Proceedings of 28th IEEE Annual International Computer Software and Applications Conference, IEEE CS Press*, pages 652 – 656, 2003.

[97] J. Zeng, C. Soviani, and S. A. Edwards. Generating fast code from concurrent program dependence graph. In *Proceedings of ACM LCTES*, pages 175 – 181, 2004.

[98] X. Zhang, R. Gupta, and Y. Zhang. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams. In *International Conference on Software Engineering*, 2004.

[99] Y. Zhang, B. Xu, L. Shi, B. Li, and H. Yang. Modular monadic program slicing. In *Proceedings of 28th IEEE Annual International Computer Software and Applications Conference, IEEE CS Press*, pages 66 – 71, September 2004.

[100] J. Zhao. Dynamic slicing of object-oriented programs. Technical report, Information Processing Society of Japan, May 1998.

[101] J. Zhao. Multithreaded dependence graphs for concurrent Java programs. In *Proceedings of the 1999 International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'99)*, 1999.

[102] J. Zhao. Slicing concurrent Java programs. In *Proceedings of the 7th IEEE International Workshop on ProgramComprehension*, May 1999.

[103] J. Zhao, J. Cheng, and K. Ushijima. Static slicing of concurrent object-oriented programs. In *20th IEEE Annual International Computer Software and Applications Conference*, pages 312 – 320, August 1996.

[104] J. Zhao, J. Cheng, and K. Ushijima. A dependence-based representation for concurrent object-oriented software maintenance. In *Proceedings of 2nd Euromicro Conference on Software Maintenance and Reengineering*, pages 60 – 66, March 1998.

[105] J. Zhao and B. Li. Dependence based representation for concurrent Java programs and it's application to slicing. In *Proceedings of ISFST*, pages 105 – 112, 2004.

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan–Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research institution in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 700 staff, has 500 researchers, about 250 of whom are postgraduates, over 200 of whom have doctorates (Ph.D.), and around 150 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of S**love**nia (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

In the last year on the site of the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

At the present time, part of the Institute is being reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project is being developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park will take the form of a shareholding company and will host an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of Economic Relations and Development, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel.:+386 1 4773 900, Fax.:+386 1 219 385
Tlx.:31 296 JOSTIN SI
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Contact person for the Park: Iztok Lesjak, M.Sc.
Public relations: Natalija Polenec

INFORMATICA

AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS

INVITATION, COOPERATION

Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks. If the paper is accepted, the editor will also send copies to the Contact Person. The Executive Board will inform the author that the paper has been accepted, in which case it will be published within one year of receipt of e-mails with the text in Informatica LaTeX format and figures in `.eps` format. The original figures can also be sent on separate sheets. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

QUESTIONNAIRE

☐ Send Informatica free of charge

☐ Yes, we subscribe

Please, complete the order form and send it to Dr. Drago Torkar, Informatica, Institut Jožef Stefan, Jamova 39, 1111 Ljubljana, Slovenia.

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (more than ten years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: .....

Title and Profession (optional): .....

.....

Home Address and Telephone (optional): .....

.....

Office Address and Telephone (optional): .....

.....

E-mail Address (optional): .....

Signature and Date: .....

Informatica WWW:

http://www.informatica.si/

Referees:

Witold Abramowicz, David Abramson, Adel Adi, Kenneth Aizawa, Suad Alagić, Mohamad Alam, Dia Ali, Alan Aliu, Richard Amoroso, John Anderson, Hans-Jurgen Appelrath, Iván Araujo, Vladimir Bajič, Michel Barbeau, Grzegorz Bartoszewicz, Catriel Beeri, Daniel Beech, Fevzi Belli, Simon Beloglavec, Sondes Bennasri, Francesco Bergadano, Istvan Berkeley, Azer Bestavros, Andraž Bežek, Balaji Bharadwaj, Ralph Bisland, Jacek Blazewicz, Laszlo Boeszoermenyi, Damjan Bojadžijev, Jeff Bone, Ivan Bratko, Pavel Brazdil, Bostjan Brumen, Jerzy Brzezinski, Marian Bubak, Davide Bugali, Troy Bull, Sabin Corneliu Buraga, Leslie Burkholder, Frada Burstein, Wojciech Buszkowski, Rajkumar Bvyya, Giacomo Cabri, Netiva Caftori, Particia Carando, Robert Cattral, Jason Ceddia, Ryszard Choras, Wojciech Cellary, Wojciech Chybowski, Andrzej Ciepielewski, Vic Ciesielski, Mel Ó Cinnéide, David Cliff, Maria Cobb, Jean-Pierre Corriveau, Travis Craig, Noel Craske, Matthew Crocker, Tadeusz Czachorski, Milan Češka, Honghua Dai, Bart de Decker, Deborah Dent, Andrej Dobnikar, Sait Dogru, Peter Dolog, Georg Dorfner, Ludoslaw Drelichowski, Matija Drobnič, Maciej Drozdowski, Marek Druzdzel, Marjan Družovec, Jozo Dujmović, Pavol Ďuriš, Amnon Eden, Johann Eder, Hesham El-Rewini, Darrell Ferguson, Warren Fergusson, David Flater, Pierre Flener, Wojciech Fliegner, Vladimir A. Fomichov, Terrence Forgarty, Hans Fraaije, Stan Franklin, Violetta Galant, Hugo de Garis, Eugeniusz Gatnar, Grant Gayed, James Geller, Michael Georgiopolus, Michael Gertz, Jan Goliński, Janusz Gorski, Georg Gottlob, David Green, Herbert Groiss, Jozsef Gyorkos, Marten Haglind, Abdelwahab Hamou-Lhadj, Inman Harvey, Jaak Henno, Marjan Hericko, Henry Hexmoor, Elke Hochmueller, Jack Hodges, John-Paul Hosom, Doug Howe, Rod Howell, Tomáš Hruška, Don Huch, Simone Fischer-Huebner, Zbigniew Huzar, Alexey Ippa, Hannu Jaakkola, Sushil Jajodia, Ryszard Jakubowski, Piotr Jedrzejowicz, A. Milton Jenkins, Eric Johnson, Polina Jordanova, Djani Juričič, Marko Juvancic, Sabhash Kak, Li-Shan Kang, Ivan Kapustøk, Orlando Karam, Roland Kaschek, Jacek Kierzenka, Jan Kniat, Stavros Kokkotos, Fabio Kon, Kevin Korb, Gilad Koren, Andrej Krajnc, Henryk Krawczyk, Ben Kroese, Zbyszko Krolikowski, Benjamin Kuipers, Matjaž Kukar, Aarre Laakso, Sofiane Labidi, Les Labuschagne, Ivan Lah, Phil Laplante, Bud Lawson, Herbert Leitold, Ulrike Leopold-Wildburger, Timothy C. Lethbridge, Joseph Y-T. Leung, Barry Levine, Xuefeng Li, Alexander Linkevich, Raymond Lister, Doug Locke, Peter Lockeman, Vincenzo Loia, Matija Lokar, Jason Lowder, Kim Teng Lua, Ann Macintosh, Bernardo Magnini, Andrzej Małachowski, Peter Marcer, Andrzej Marciniak, Witold Marciszewski, Vladimir Marik, Jacek Martinek, Tomasz Maruszewski, Florian Matthes, Daniel Memmi, Timothy Menzies, Dieter Merkl, Zbigniew Michalewicz, Armin R. Mikler, Gautam Mitra, Roland Mittermeir, Madhav Moganti, Reinhard Moller, Tadeusz Morzy, Daniel Mossé, John Mueller, Jari Multisilta, Hari Narayanan, Jerzy Nawrocki, Rance Necaise, Elzbieta Niedzielska, Marian Niedq'zwiedziński, Jaroslav Nieplocha, Oscar Nierstrasz, Roumen Nikolov, Mark Nissen, Jerzy Nogieć, Stefano Nolfi, Franc Novak, Antoni Nowakowski, Adam Nowicki, Tadeusz Nowicki, Daniel Olejar, Hubert Österle, Wojciech Olejniczak, Jerzy Olszewski, Cherry Owen, Mieczyslaw Owoc, Tadeusz Pankowski, Jens Penberg, William C. Perkins, Warren Persons, Mitja Peruš, Fred Petry, Stephen Pike, Niki Pissinou, Aleksander Pivk, Ullin Place, Peter Planinšec, Gabika Polčicová, Gustav Pomberger, James Pomykalski, Tomas E. Potok, Dimithu Prasanna, Gary Preckshot, Dejan Rakovič, Cveta Razdevšek Pučko, Ke Qiu, Michael Quinn, Gerald Quirchmayer, Vojislav D. Radonjic, Luc de Raedt, Ewaryst Rafajlowicz, Sita Ramakrishnan, Kai Rannenberg, Wolf Rauch, Peter Rechenberg, Felix Redmill, James Edward Ries, David Robertson, Marko Robnik, Colette Rolland, Wilhelm Rossak, Ingrid Russel, A.S.M. Sajeev, Kimmo Salmenjoki, Pierangela Samarati, Bo Sanden, P. G. Sarang, Vivek Sarin, Iztok Savnik, Ichiro Satoh, Walter Schempp, Wolfgang Schreiner, Guenter Schmidt, Heinz Schmidt, Dennis Sewer, Zhongzhi Shi, Mária Smolárová, Carine Souveyet, William Spears, Hartmut Stadtler, Stanislaw Stanek, Olivero Stock, Janusz Stokłosa, Przemysław Stpiczyński, Andrej Stritar, Maciej Stroinski, Leon Strous, Ron Sun, Tomasz Szmuc, Zdzislaw Szyjewski, Jure Šilc, Metod Škarja, Jiří Šlechta, Chew Lim Tan, Zahir Tari, Jurij Tasič, Gheorge Tecuci, Piotr Teczynski, Stephanie Teufel, Ken Tindell, A Min Tjoa, Drago Torkar, Vladimir Tosic, Wieslaw Traczyk, Denis Trček, Roman Trobec, Marek Tudruj, Andrej Ule, Amjad Umar, Andrzej Urbanski, Marko Uršič, Tadeusz Usowicz, Romana Vajde Horvat, Elisabeth Valentine, Kanonkluk Vanapipat, Alexander P. Vazhenin, Jan Verschuren, Zygmunt Vetulani, Olivier de Vel, Didier Vojtisek, Valentino Vranić, Jozef Vyskoc, Eugene Wallingford, Matthew Warren, John Weckert, Michael Weiss, Tatjana Welzer, Lee White, Gerhard Widmer, Stefan Wrobel, Stanislaw Wrycza, Tatyana Yakhno, Janusz Zalewski, Damir Zazula, Yanchun Zhang, Ales Zivkovic, Zonling Zhou, Robert Zorc, Anton P. Železnikar

Informatica

An International Journal of Computing and Informatics

Archive of abstracts may be accessed at America: http://ocean.ocean.cs.siu.edu/informatica/index.html, Europe: http://www.informatica.si/, Asia: http://www3.it.deakin.edu.au/ hdai/Informatica/.

Subscription Information Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 1000 Ljubljana, Slovenia.
The subscription rate for 2006 (Volume 30) is
– 60 EUR (80 USD) for institutions,
– 30 EUR (40 USD) for individuals, and
– 15 EUR (20 USD) for students
Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

Typesetting: Borut Žnidar.
Printed by Dikplast Kregar Ivan s.p., Kotna ulica 5, 3000 Celje.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. Drago Torkar, Jožef Stefan Institute: Tel (+386) 1 4773 900, Fax (+386) 1 219 385, or send checks or VISA card number or use the bank account number 900–27620–5159/4 Nova Ljubljanska Banka d.d. Slovenia (LB 50101-678-51841 for domestic subscribers only).

Informatica is published in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society; Cognitive Science Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences / Engineering Academy of Slovenia (Igor Grabec)
ACM Slovenia (Dunja Mladenič)

Informatica is surveyed by: AI and Robotic Abstracts, AI References, ACM Computing Surveys, ACM Digital Library, Applied Science & Techn. Index, COMPENDEX*PLUS, Computer ASAP, Computer Literature Index, Cur. Cont. & Comp. & Math. Sear., Current Mathematical Publications, Cybernetica Newsletter, DBLP Computer Science Bibliography, Engineering Index, INSPEC, Linguistics and Language Behaviour Abstracts, Mathematical Reviews, MathSci, Sociological Abstracts, Uncover, Zentralblatt für Mathematik

The issuing of the Informatica journal is financially supported by the Ministry of Higher Education, Science and Technology, Trg OF 13, 1000 Ljubljana, Slovenia.

Volume 30 Number 2 June 2006 **ISSN 0350-5596**

Informatica

An International Journal of Computing and Informatics

Informatica **30** (2006) Number 2, pp. 141–279