Volume 27 Number 1 April 2003 ISSN 0350-5596

Informatica

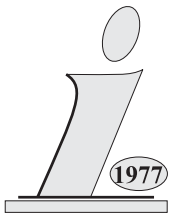
An International Journal of Computing and Informatics

Special Issue:

Bioinformatics Tools and Applications

Guest Editors:

Johann Eder, Omran Bukhres



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.

A Decentralized Approach to the Integration of Life Science Web Databases

Zina Ben Miled,
ECE Department, Indiana University Purdue University Indianapolis,
723 W. Michigan St, SL 160C, Indianapolis, IN, 46202
zmiled@iupui.edu

Nianhua Li, Mark Baumgartner
CSCI Department, Indiana University Purdue University Indianapolis,
723 W. Michigan St, SL 280, Indianapolis, IN, 46202
niali@iupui.edu, maabaumg@iupui.edu

Yang Liu,
ECE Department, Indiana University Purdue University Indianapolis,
723 W. Michigan St, SL 160C, Indianapolis, IN, 46202
liuy_yang@yahoo.com

Keywords: integration, biological databases, distributed architecture

Received: June 8th,2002

In the recent decades technological breakthroughs in science and engineering have led to an explosion in the amount of data available in several fields such as environmental, biological and chemical fields. One of the obstacles preventing this data from empowering new discoveries is the lack of adequate methods that can manage this data and turn it into knowledge. This paper presents a scalable solution to the management of life science databases. Life science web databases are often heterogeneous, geographically distributed and contain semi-structured data. The proposed system (BACIIS: Biological and Chemical Information Integration System) integrates these web databases on-demand. The architecture of BACIIS is decentralized. This design choice was made in order to overcome some of the limitations of remote web-based querying and to create a system that can adapt to an increasing number of users. This paper discusses the architecture of BACIIS and presents an analysis of its performance in response to queries submitted by multiple users.

1 Introduction

The highlight of the last decade in the life sciences was the production of massive amount of data. The objective of the next decade is to analyse this data and turn it into knowledge that can enable discoveries. In order for scientists to turn the available data into knowledge, they have to be able to formulate hypothesis and validate them. This process involves accessing multiple databases that are often only accessible through a web interface. Furthermore, while these databases contain large amount of valuable data, they do not easily interoperate. There are hundreds of life science databases that provide access to scientific data and literature. These databases use different nomenclatures, file formats, and data access interfaces. Furthermore, they may include redundant and conflicting data.

BACIIS *(Biological and Chemical Information Integration System)* [1] is an on-demand information integration system for life science web-databases. Figure 1 shows BACIIS integrating four widely used life science web databases. These databases are GenBank [2], SwissProt[3], OMIM[4] and PDB[5].

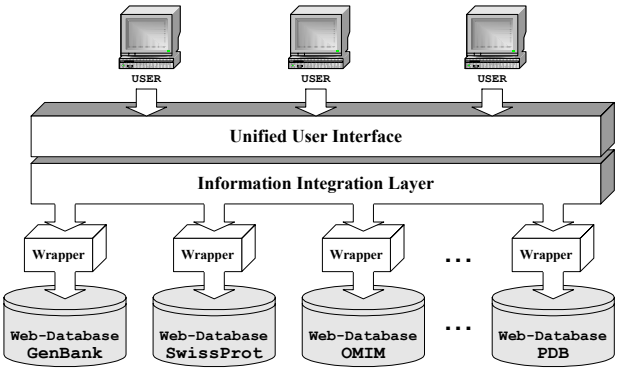


Figure 1: Information integration of life Science web database.

For example, PDB (Protein Data Bank) contains information on 3-D biological macromolecular structure and GenBank is a genetic sequence database, which consists of an annotated collection of all publicly available DNA sequences. These databases represent only a subset of the available life science databases that are in excess of 100[6]. The objective of BACIIS is to integrate a large number of these databases in order to provide

wide coverage. This goal can only be achieved through a robust and scalable architecture.

BACIIS supports the seamless integration of various life science web databases in order to facilitate knowledge discovery. It allows a user to issue a multi-database query without any knowledge of the actual content or data representation of the individual web databases being integrated by BACIIS. The querying process is completely transparent to the user. Thus, allowing him or her to concentrate on the biological aspects of the project being conducted rather than on the implementation details of the web databases that house the needed information.

The integration method used in BACIIS is based on the semantic understanding and representation of the life science domain knowledge. BACIIS provides users with the ability to submit queries across multiple geographically distributed heterogeneous life science web-databases. The integration is performed on-demand as opposed to in advance. That is, when a user issues a multidatabase query, the corresponding data is retrieved directly and on-line from the websites of the target life science web databases. An integration in advance approach relies on first downloading the various databases to a local server ahead of time and responding to the user query using the data from these local databases. Given that BACIIS performs integration on-demand, its decentralized architecture enhances its ability to perform multidatabase queries with reduced response times even when multiple users are accessing the system.

In this paper, the decentralized architecture of BACIIS is presented and its query response time is analysed. Although, BACIIS integrates life science databases, the proposed architecture can also be used as a model for systems that process transactions over the Internet. Section 2 of this paper describes the functionality and implementation of the decentralized architecture of BACIIS. Experiments that illustrate the scalability of BACIIS and its performance are presented in Section 3. Related work is summarized in Section 4. Conclusions are included in Section 5.

2 Decentralized Architecture

BACIIS was designed with several goals in mind including correctness, reduced query response time, and maximum coverage. In order to fulfil the first goal, an on-demand integration approach was selected because it provides the user with up-to-date query results. Furthermore, correctness dictates a semantic based integration. The biological databases use disparate scientific terms and representations. For example, the term *citation* is used by PDB and the term *references* is used by GenBank to refer to literature references. When constructing the data source schema for these two databases, the two terms are tagged by the same ontology term: *REFERENCE*. Thus, establishing their equivalency. In addition, once the result corresponding to these two terms is returned from each database, the records in each result have to be combined in order to keep only the unique records. This example illustrates one of the many cases where only a semantic integration approach can resolve the variability among the databases and provide a correct response to a multidatabase query.

Preserving the local autonomy of the individual life science web databases was dictated by the culture of the biological field. Most of the databases are organized around disciplinary interest or institutional convenience. This gave rise to data silos that integrate data vertically (within a domain) but not horizontally (across domains). The goal of BACIIS is to support this horizontal integration using a semantic-based approach.

The above mentioned constraints (i.e. correctness, reduced query response time, maximum coverage and preservation of the local autonomy of the databases) often lead to design trade-offs. For example, returning the most complete result data set for a query (i.e. maximum coverage through the integration of a large number of databases) will most likely require a longer response time than returning a selected reduced data set result. This paper focuses on the decentralized architecture of BACIIS and its ability to integrate on-demand, multiple web databases with fast query response time.

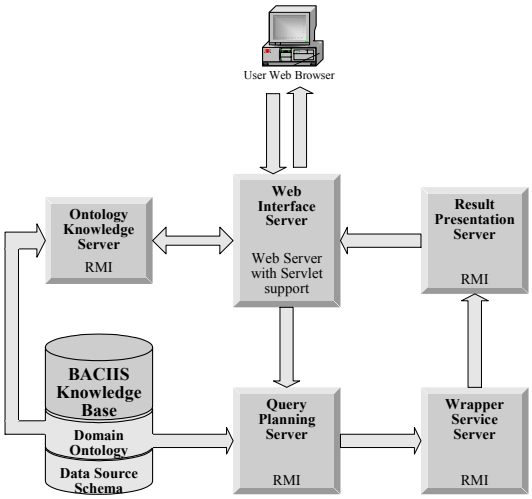


Figure 2: BACIIS decentralized architecture

BACIIS is composed of five servers (Figure 2): Web interface server, Query Planner Server, Ontology Knowledge Server, Wrapper Service Server, and Result Presentation Server. These servers collaborate in order to generate query plans, match queries with target data sources, perform data source interrogation, and integrate the results returned from data sources. The functionality and implementation of these servers are discussed in the following subsections.

The architecture of BACIIS (Figure 3) is based on a mediator-wrapper approach [7, 8, 9] augmented with a knowledge base. The mediator transforms data from its

format in the source database to the internal format used by the integration system. Some of the functions of the mediator are included in the Query Planning Server and the remaining functions are included in the Result Presentation Server.

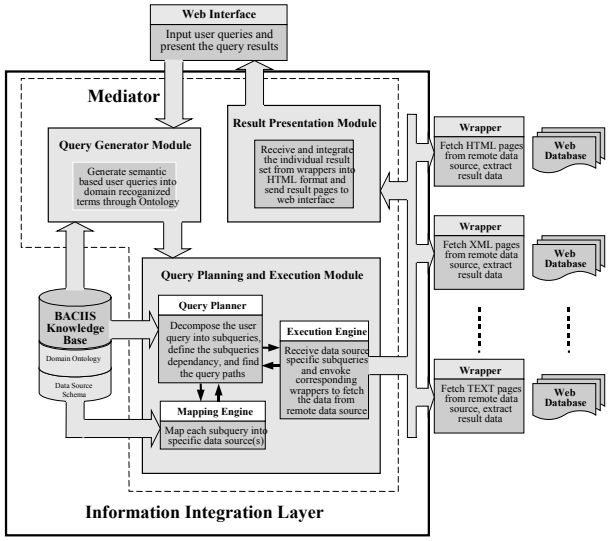


Figure 3: BACIIS system architecture

Each database participating in the integration is associated with a wrapper. The wrapper acts as an intelligent proxy user and extracts information from the corresponding remote data source. All the wrappers are part of the Wrapper Service Server.

The life science web databases include semi-structured data with an unknown or dynamically changing schema. This characteristic makes it impractical to collect the schema from each remote database and attempt to maintain a unified static global data schema using the traditional relational model. BACIIS uses a different strategy. Instead of building a global data schema, each remote database schema is mapped onto the domain ontology [10]. Both the ontology and the data source schema for the various databases are included in the BACIIS knowledge base (Figure 2). The ontology is independent from the data schema of the remote databases. It will only change when the biological domain evolves to include new discoveries and findings.

The decentralized architecture of BACIIS was implemented using Java Remote Method Invocation (RMI) [11]. Queries in BACIIS often result in a large volume of retrieved data. The complexity of the queries and the high volume of retrieved data, make a centralized architecture inadequate. Furthermore, a centralized architecture quickly becomes a bottleneck if multiple users are trying to submit queries. The decentralized architecture of BACIIS yields other benefits beyond a performance enhancement and a better quality of service, including increased modularity and better maintainability.

2.1 Web Interface Server

The web interface server is developed using JavaBeans and JSP. It accepts user queries and presents the query results. Queries in BACIIS are formulated by using the query by example approach. This is a very popular approach in the biological domain. The user composes a query by combining terms from the domain ontology. These domain ontology terms are retrieved from the ontology service server.

The user formulates a query in BACIIS through the interface shown in Figure 4. The first box in this interface shows the ontology classes and subclasses. When one of the classes is highlighted the corresponding properties are displayed in the properties window. This allows the user to select the desired output property from the available properties list. The user can select several classes and several properties. This process in the query composition allows the user to retrieve only the data that he or she may need. Filtering the output according to the preferences of the user will reduce the amount of data retrieved, which will in turn improve query response time.

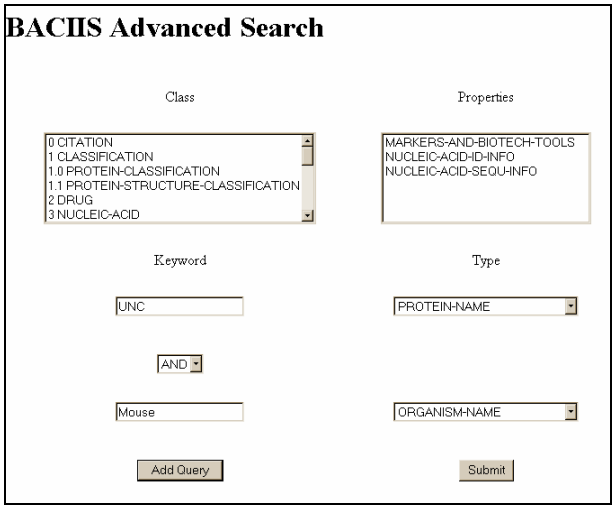


Figure 4: BACIIS Advanced Search Interface.

The next step of the query composition is to define the input part of the query. The input is defined as a Boolean expression over a set of input keywords. The user can have unlimited number of predicates in the input query. However, for each predicate, the user must also specify an input type that is selected from a drop down menu. For example, in Figure 4, the keyword *mouse* is associated with the type *ORGANISM-NAME*. The type specification is necessary when integrating heterogeneous life science databases because it promotes a more accurate query result. Most of the life science web databases identify specific terms, such as organism name, in each record of the database. Specifying the type organism name for the input keyword "*mouse*" will use the metadata knowledge available in the underlying life science database to prevent BACIIS from returning irrelevant results. For

example the keyword "*mouse*" may be used in a different context such as in "by a click of a mouse". Records containing reference to the keyword *mouse* in this context are irrelevant to the input query and should not be returned by BACIIS. A subset of the type list available in BACIIS is shown in Figure 5.

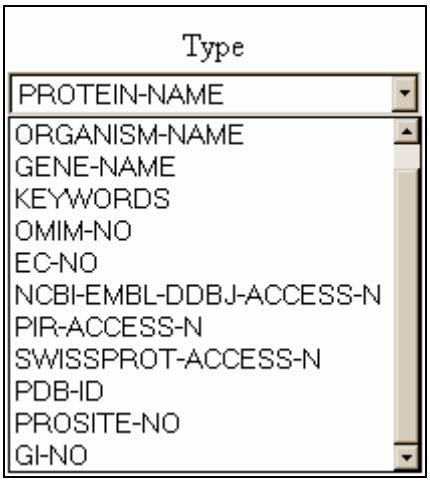


Figure 5: User Interface Type Selection

Once the query entry process is completed, the web interface server will package the query including information about the output properties, the input and the output types selected by the user, and forwarded it to the query planner server.

2.2 Query Planner Server

The role of the query planner server is to decompose a given query into sub-queries, define the dependencies between these sub-queries, find proper query execution paths, map the sub-queries to the appropriate data sources, and call the wrapper service server to invoke the corresponding data source wrappers.

Finding a proper query execution plan entails determining the shortest execution path for the query. The plan is composed of database specific sub-queries, where each sub-query can be answered using a single database. This task involves building a graph of possible paths, where the sub-queries are the nodes of the graph, and determining the shortest path.

Consider the following query: *For all the enzymes of the EC family 1.1.1.1 and the human ADH2 protein, retrieve their 3D structure information, their coding gene sequence and their related literature references.*

The above query is entered in the BACIIS interface as follows, where the Boolean operators that combine the clauses are underlined:

[Protein Name = *ADH2*] AND [Organism Name = *Human*] OR [EC Family Number = *1.1.1.1*].

The BACIIS interface uses the query-by-example approach which simplifies the entry of complex queries such as the above query. Also, as previously mentioned, the interface uses the ontology to guide the user in formulating the query.

Query decomposition transforms a complex query into sub-queries. This is accomplished by breaking the query string into sub-strings along the partitions of logical operators. Internally in BACIIS the query is interpreted using the left anchored operator precedence. The result of the query decomposition tree for the example query introduced in Section 1 is shown in Figure 6.

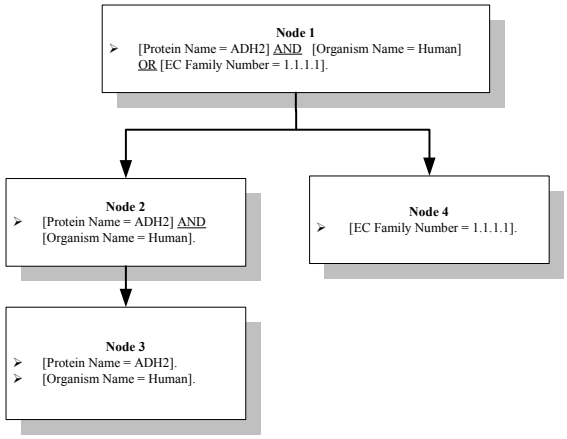


Figure 6: Query Decomposition Tree

In this figure node 1 is processed and all the paths originating with the databases that can accept the query in node 1 (i.e. [Protein Name = *ADH2*] AND [Organism Name = *Human*] OR [EC Family Number = *1.1.1.1*]) and terminating with the databases that can provide *Protein-3D-Structure, Coding gene sequence, or Related literature references* are determined. For most complex queries, it is very unlikely that a single life science database can accept all the query clauses. Therefore, this query needs to be decomposed further. Without an information integration system for life science databases, the user would need to decompose the query manually, process the various sub-queries and combine the results of these sub-queries. This fact is one of the motivations underlying the need for the integration of life science databases. The query of node 1 is decomposed along the OR operator into two nodes: node 2 (i.e. [Protein Name = *ADH2*] AND [Organism Name = *Human*]) and node 4 (i.e. [*EC Family Number = 1.1.1.1*]) . For these nodes also, the paths that start with databases that can accept the two sub-queries and terminate in a database that can generate *Protein-3D-Structure, Coding gene sequence, or Related literature references* are identified. The goal of query decomposition is to explore all the possible execution paths for the original query submitted by the user. Each of the nodes 2 and 4 will result in a set of paths. If the final execution plan includes these nodes, a given path from the first set will be combined with a path from the second set. Since the union (OR) of node 2 and

node 4 forms the original query, the union of the data set resulting from the first path and the data set of the second path is computed in order to collect the result data set for the original query. The union is performed by the information integration layer in BACIIS because of the limited query capabilities of the individual life science databases. Similarly, if two sub-queries are joined using the AND operator, then the information integration layer in BACIIS will compute the intersection of the result data sets.

The decomposition process continues until nodes with simple sub-queries (i.e. one input constraint with no operators) are obtained. For example, the query decomposition terminates in Figure 6 with nodes 3 and 4. Each of these nodes contains a set of simple sub-queries.
The first step in the query processing is the decomposition (Figure 6). Once the query decomposition tree is obtained, the second step consists of identifying all the databases that can accept the original query (node 1) or any of its sub-queries (nodes 2 through 4). For example, in order to process node 3, the databases that can accept either one of the following sub-queries must be determined as input databases:

- Protein Name = *ADH2*
- Organism Name = *Human*

Moreover, all databases that can provide the desired query output (i.e. *Protein-3D-structure, Coding gene sequences, or Related literature references* for the example query) will be identified as output databases. Among the four databases that are currently integrated by BACIIS, SwissProt and PDB can provide *Protein-3D-structure*, SwissProt and GenBank can provide *Related literature references*, and GenBank can provide *Coding gene sequences*.

The third step consists of combining the result of the previous step into paths that originate from the databases that can service any of the above queries and terminate in an output database that contains the query result (i.e. *Protein-3D-Structure, Coding gene sequence, or Related literature references* information for the example query). Figure 7 shows the complete plan for the above sample query. The plan contains five paths: A1, A2, B1, C1, and C2.

The forth step is to translate the plan in Figure 7 into an executable plan. The plan in Figure 7 contains 5 plan paths. Each of them contains several steps. In the mapping process, each of the plan steps is mapped to a specific data source that contains the requested data using the information contained in the knowledge base. Specifically, the metadata associated with each web database being integrated by BACIIS is represented using an XML schema file that is stored in the knowledge base. This information is used to map a given plan step to the specific URLs and extraction rules of the corresponding web database. This mapping process will be explained in more details in section 2.5. In the final executable plan, each plan step includes the URL address of the associated web database, the input and output properties, and the extraction rules. An extraction rule consists of the set of operations to be performed on the web database in order to retrieve a given output based on a given input.

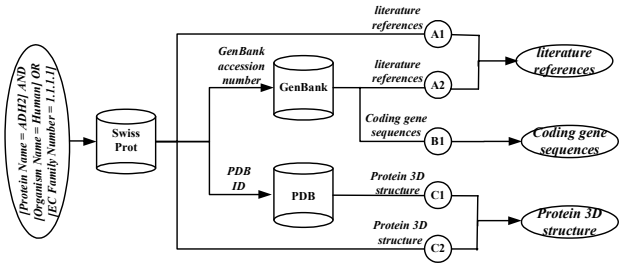


Figure 7: Query plan

2.3 Wrapper Service Server

The wrapper service server receives the executable plan from the query planner server. The wrapper service server will fork a thread for plan paths that it receives. Five threads will be generated for the above sample query plan. Each thread will access the web databases associated with a given path in a sequential order. For example, the thread for plan path C1 in Figure 7 will use the original query to retrieve corresponding SwissProt entries and extract the cross-references to PDB from these SwissProt entry pages, then it will fetch the cross-referenced PDB entries and extract the protein 3D structure information from the PDB database. In this process, URLs are used to fetch source database data entry pages, whereas the extraction rules are used to obtain the desired information from that page. Once the data is returned for all the plan paths, the wrapper service server makes a remote method invocation to the result presentation server.

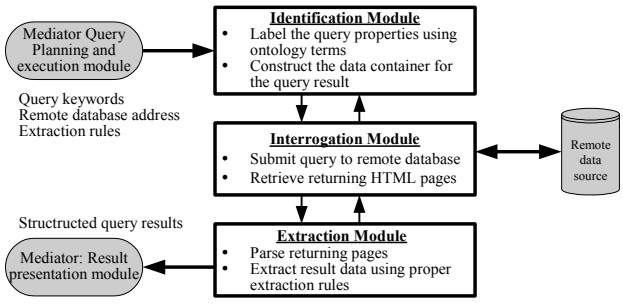


Figure 8: Wrapper Architecture

Figure 8 shows the architecture of the wrapper. Each wrapper consists of three modules: the identification module, the interrogation module and the extraction module. The identification module interfaces with the query planner server. The role of this module is to label the terms in the query using the ontology properties. It also constructs the container that will hold the returned

result. The interrogation module communicates with the identification module, the extraction module and the associated web database. The role of this module is to submit the query to the remote database and to retrieve the result pages. These pages are passed to the extraction module. In the extraction module the result pages are parsed in order to extract the information requested by the user. The structured query results also contain tags that identify the various fields using terms from the ontology. These structured results are passed to the result presentation server.

The complexity of retrieving the correct data for a given query may involve multiple accesses to more than one web database. The data returned from a given web database may consist of a page that includes the desired data or just a link to another data source page. In the second case, another access to a web database is required. Furthermore, often, the data can only be retrieved from one web database if a preliminary step of retrieving data from another database is performed. For example, given the *GI number* (GenBank ID) of a gene, in order to get the *3D structure information* of the protein encoded by this gene, three databases have to be queried sequentially: *GI number* –> GenBank database –> *Accession Number* –> SwissProt database –> *PDB ID* –> PDB database –> *3D structure information.*

2.4 Result Presentation Server

In the wrapper service server, once the data is returned by the wrappers, the client will make a remote method invocation to the result presentation server. The result presentation server will fork a thread for each remote method invocation. Once the results from all the sub-queries of a given query are integrated, the information is passed to the web interface server.

In order for the results retrieved by the wrappers to be usable, the data retrieved must be semantically integrated into a correct and cohesive format. Integrating data source results in the biological domain is particularly challenging because data from different data sources must be checked for equivalency. Furthermore, the data schema and the data format are different in the databases. For example, a reference information is expressed as a single record in SwissProt, while it is divided into different fields (i.e. AUTHORS, TITLE, JOURNAL, etc…) in GenBank.

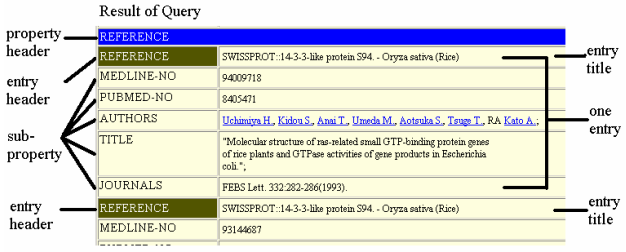


Figure 9: Example query result entry from SwissProt.

In the result presentation server, an information unit is called a result entry and it is extracted from one record in one of the web databases. A result entry starts with a title that contains the source database. Figure 9 shows the result entry associated with the reference information extracted from SwissProt for "14-3-3-like protein S94. - Oryza sativa (Rice)". In this figure the sub-properties of the selected property REFERENCE are shown under the field header.

Figures 9 and 10 are both result entries returned for a single query issued against BACIIS. The difference in these result entries illustrates another challenge that the result presentation server faces when integrating the result returned from different sub-queries. Figure 9 is a result entry returned from SwissProt. Figure 10 is a result entry retuned from GenBank. The SwissProt database stores the sub-property PUBMED-NO. The PUBMED-NO is an identification number given to a record in the PubMed database. GenBank does not store the PubMed identification number. Therefore, this number is included in the result entry of Figure 9 but not in the result entry of Figure 10. The result presentation server needs to be able to handle the differences in both representation and database information content. These differences are very common in life science web databases.

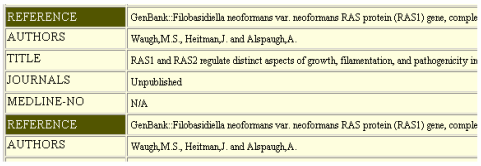


Figure 10: Example query result entry from GenBank.

2.5 Ontology Knowledge Server

The BACIIS knowledge base contains the domain ontology and the data source schema. Figure 11 shows the high level structure of the ontology used in BACIIS. The ontology is organized along three dimensions: Object, Property and Relation. In this figure, nucleic acid and protein are object classes, polymorphism and protein-mut-info are property classes, and encode and regulate are relations. An instance of a property class can only exist if there is at least one instance of the associated object class. The classes in the object dimension are arranged in a tree using the ordering "is-a-subset-of". The relations in the relation dimension provide an association between the classes in the object dimension and between the classes in the property dimension as well as between classes across the two dimensions.

The domain ontology in BACIIS is represented using PowerLoom [12]. The ontology knowledge server is used to manage this domain ontology and service ontology terms and relations inquiries from the BACIIS web interface server. When a user starts building a query, the web interface server will first retrieve all the ontology classes through the ontology knowledge server. These

classes populate the left window in the BACIIS interface shown in Figure 4. As the user selects certain classes, the web interface server will further query the ontology knowledge server to retrieve the sub-classes or the properties for the selected ontology classes. These properties will populate the right window in the BACIIS interface shown in Figure 4.

The data source schema is the other component of the knowledge base in BACIIS. It maps the schema of individual databases to the domain ontology. The mapping information contained in the data source schema is stored in a file using the XML format. This convenient and structured data format is well suited for storing data source information because of the hierarchical nature of the data. The concepts of sub-sets of information residing at multiple levels are well represented by XML. For better system performance, XML files are read into Java Document Object Model (DOM) structures only on system start up or after a knowledge base update.

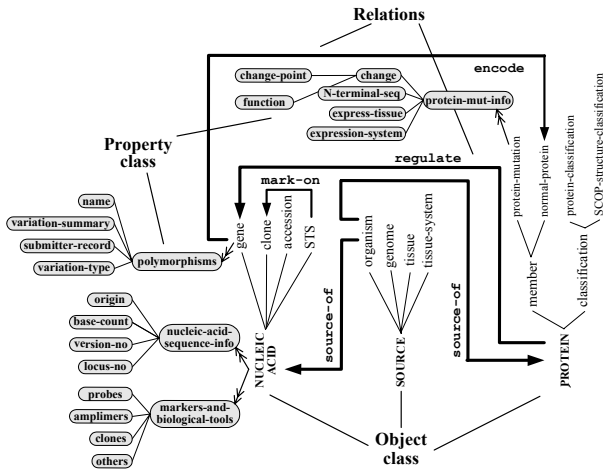


Figure 11: Structure of the Domain Ontology

A partial data source schema for GenBank is shown in Figure 12. This XML file contains three sections: the metadata, the query input types and the query output types. The metadata section includes information about the associated database. For example, one of the tags is DBNAME which represents the database name. This tag has the value GenBank in Figure 12. The query input type section includes all the types of input keywords that are accepted by the database. For example, GENE-NAME which is a tag in the query input type section of the GenBank data source schema, can be accepted by GenBank. The value of this tag is a URL (i.e. http://www.ncbi.nlm.nih.gov/query?db=2&form=1&term =*XXXX[TITLE]*). If the user types in an input keyword (e.g. *ras*) in the BACIIS user interface and specifies its type as a Gene-Name, then the keyword *ras* will be inserted into the URL as follows:
http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=2&form=1&term=ras[TITL]

This final URL will be used to fetch data entries related to gene *ras* from GenBank.

The last section of the data source schema contains the types of output that can be produced by the database. In the case of GenBank, NUCLEIC-ACID-SEQUENCE-INFO can be retrieved. All of the tags in the query input type section and the query output section of the data source schema are terms defined in the ontology. In addition to being used by the query planner to identify if a given web database can answer a given query, these tags are also used to mark the object, property and relation terms in the result pages returned from the wrappers. For example, the REFERENCE is an ontology term that is used to tag the records returned from SwissProt for citation and the records returned from GenBank for reference for the BACIIS result entries shown in figures 9 and 10.

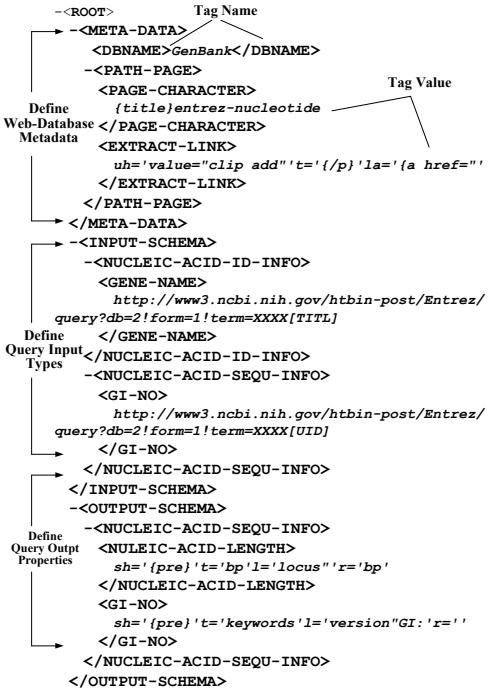


Figure 12: Example Data Source Schema

3 Performance Analysis

Initially, BACIIS was implemented using a centralized architecture. All system components were integrated into a single application built to run on a single machine. The complexity of the system components made it clear that a decentralized approach is necessary. In this section the performance gain of the decentralized architecture of BACIIS is compared to the centralized architecture. This performance is measured in terms of query response time.

3.1 Experiment Set-up

In order to evaluate the performance of the centralized and decentralized BACIIS architectures, three representative queries where identified. Because the response time of BACIIS varies with the complexity of

the query, the selected queries have three different complexity levels as shown in Table 1.

The first query is rated as simple because only one database (SwissProt) will be used to generate the query result. SwissProt is the only database among the four databases currently integrated in BACIIS that can provide general information for a protein. Furthermore, the query is stated in terms of an access number that is recognized by SwissProt. Finally, only one SwissProt record matches this query. Thus, the wrapper will only access SwissProt once and the returned result is small in size.

The second query is classified as average. The execution plan generated by BACIIS for this query consists of the following two steps:

1. Retrieve the entry list of all ADH2 proteins from SwissProt.
2. Use the cross-references in each SwissProt entry to retrieve related GenBank entries and extract the desired information.

Complexity	Query
simple	*Retrieve general information about the protein whose ACCESS NUMBER in the database SwissProt is P00326.*
average	*Retrieve general information of all genes encoding the ADH2 (alcohol dehydrogenase 2) proteins.*
complex	*For all the enzyme of the EC family 1.1.1.1 and the human ADH2 protein, retrieve their 3D structure information, their gene sequence and their related literature references.*

Table 1: Queries with different level of complexities used to evaluate the response time of BACIIS.

The second step is complicated by the biological terminology, which uses the same name for close homologous proteins in all species. For example, SwissProt contains 74 records for the ADH2 proteins from different species (yeast, tomato, human, etc …). Furthermore, each entry is linked to multiple other records in GenBank that are also retrieved. This query mainly stresses the wrapper service server and provides moderate complexity for the other servers.

The third query is the one previously mentioned in Section 2.2. It is repeated here for convenience. This query is complex. It seeks information about the Enzyme family EC1.1.1.1 also known as "alcohol dehydrogenase". The execution plan for this query consists of the following steps

1. Obtain all SwissProt entries of the EC family 1.1.1.1 proteins and human ADH2 protein and extract reference information and cross-references to GenBank and PDB.
2. Retrieve the gene sequences from GenBank and the protein 3D structure information from PDB.

This query plan is more complex than the previous two because it includes a join operation over data sets from multiple databases. Thus, it takes the wrapper service server and the result presentation server more time to process this query. By selecting three queries with varying complexity it possible to assess the performance of the BACIIS system under different scenarios.

The web interface server and ontology knowledge server are involved in the query building process, but not in the query processing. Both of them are not included in the experiment for three reasons. First, they have limited effect on the overall system performance. Second, the variability in the interactivity of a user with the interface may make the results of the experiment unreliable. Third, the experiment includes a test that submits multiple queries in order to mimic the performance of the system when multiple users are using it. It is impractical to use the web interface and the ontology knowledge server for such a test in a controlled environment. Instead of using the web interface server, a simple query generator module is used to send test queries, receive results and measure the query response time.

For the centralized BACIIS architecture, the queries were executed on a Sun Ultra 5 workstation. In the case of the decentralized BACIIS architecture, the query planner server, the wrapper service server and the result presentation servers were each executed on a different Sun Ultra 5 workstation. All the Sun Ultra5 workstations used in the experiment have 384Mbyte of memory and are equipped with one 400Mhz UltraSPARC-IIi processor. During execution, these servers communicate and exchange data through Java RMI procedure calls.

In an information integration system, the workload is affected by the query arrival rate and the query complexity. The first factor was tested by simultaneously initiating multiple instances of the query generator module where each instance submits one test query. Thus, multiple copies of the same test query will be sent to BACIIS concurrently. In order to test the second factor, three queries (Table 1) with varying complexities were selected.

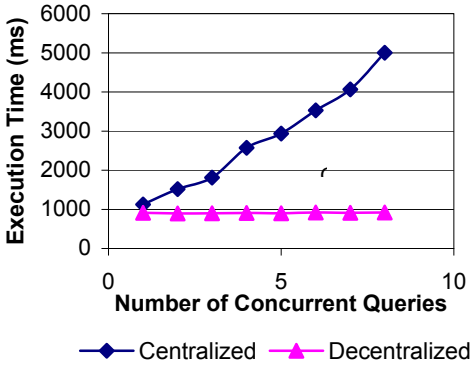
3.1 Results

Figures 3, 4 and 5 show the execution times of the queries under both the centralized and decentralized architectures. Each figure includes four graphs. They correspond to the execution time spent in the query planner server, the wrapper service server, the result presentation server and the overall execution time,

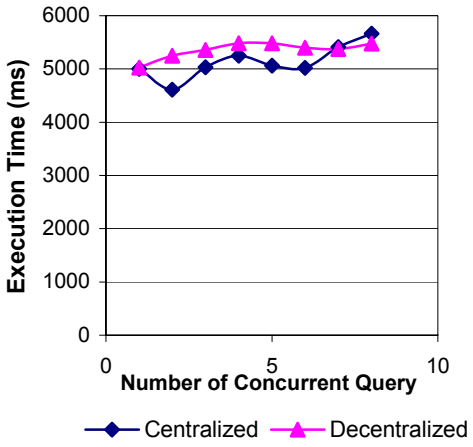
respectively. Figures 3, 4, and 5 show the execution times for the simple, average and complex queries, respectively. In each graph the x-axis represents an increasing number of queries. The first data point refers to the case when only one query is issued. The last data point shown on the graphs corresponds to the case when eight copies of the same query are submitted to BACIIS. The y-axis represents the execution time per query.

As expected, the query response time for both the centralized and decentralizes architectures degrades as more queries are issued (figures 3d, 4d and 5d).

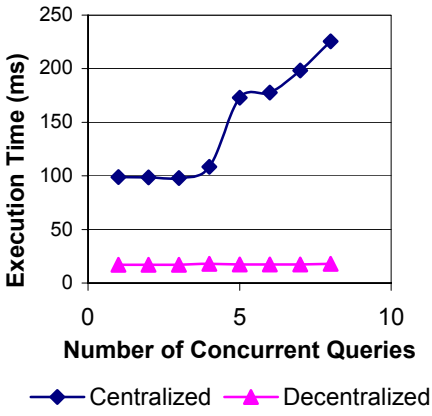
The query planner server is CPU and memory intensive, so its performance heavily depends on the resources available on the host machine. In the decentralized architecture, since each server executes on a different machine, there are less contention for resources. The difference in contention in resources between the centralized and decentralized architectures for the query planner server can be observed in figures 3a, 4a and 5a. In each of these graphs, the execution time for the query planner in the decentralized architecture nearly remains constant as the number of queries increases while that of the centralized architecture increases rapidly.



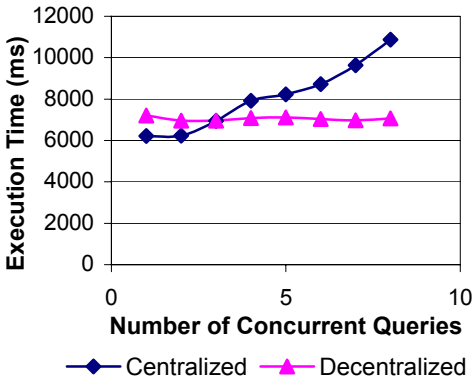
(a) Query planner server execution time



(b) wrapper service server execution time



(c) Result presentation server execution time

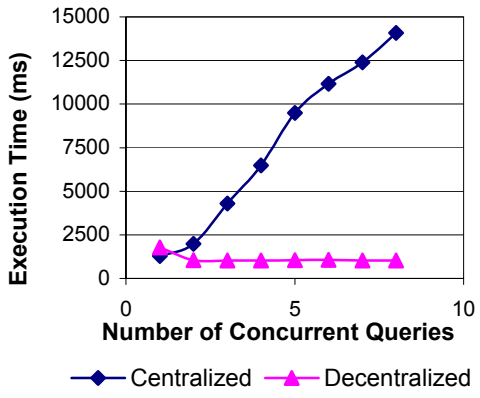


(d) Overall query response time

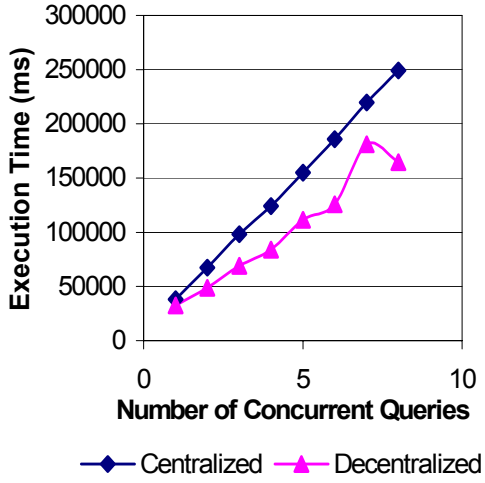
Figure 3: Execution times for simple query.

The result presentation server is also memory intensive. For the same reason mentioned above, its performance exhibits a similar pattern to that of the query planner server.

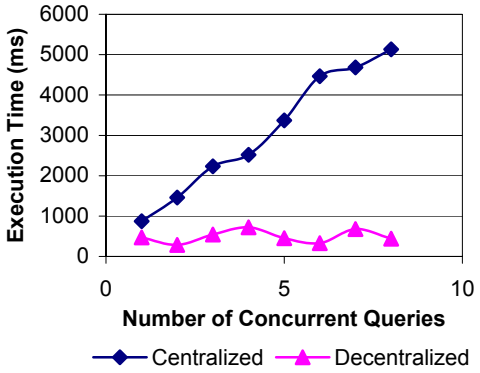
The performance of the wrapper server is mainly dictated by network traffic and response times from remote database queries such as PDB and SwissProt. Because multiple copies of the same query are issued concurrently, these queries will connect to the same URL and access the same records in the remote databases integrated by BACIIS at almost the same time. Thus, these requests will experience longer delays as the number of query increases.



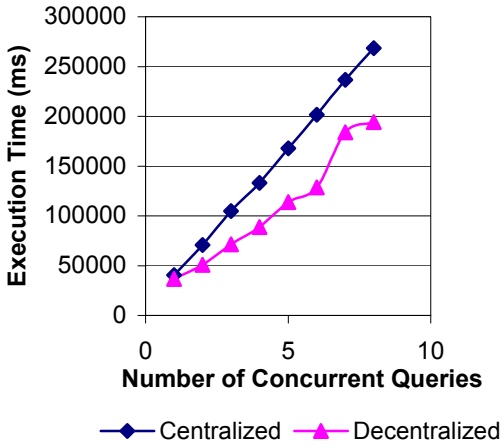
(a) Query planner server execution time



(b) Wrapper service server execution time



(c) Result presentation server execution time



(d) Overall query response time

Figure 4: Execution times for average query.

The overall execution time in the case of the simple query (Figure 3) is nearly constant for the decentralized architecture and it increases rapidly in the centralized architecture. However, in the case of the average and complex query the overall execution time increases for both architectures. This can be explained by investigating the percentage of the execution time spent in the wrapper service server. For the average and complex query, the wrapper service execution time dominates the overall execution, whereas for the simple query this is not the case. Moreover, as stated earlier, the execution time of the wrapper service server is mostly dominated by the access to the remote databases which incurs the same overhead in both the centralized and decentralized architectures.

As the usage of the biological databases increases, mirror web databases may be created. Furthermore, we are currently redesigning the wrapper service server so that it itself can have a distributed architecture. These two factors will improve the performance of the wrapper service server in the cases of average and complex queries.

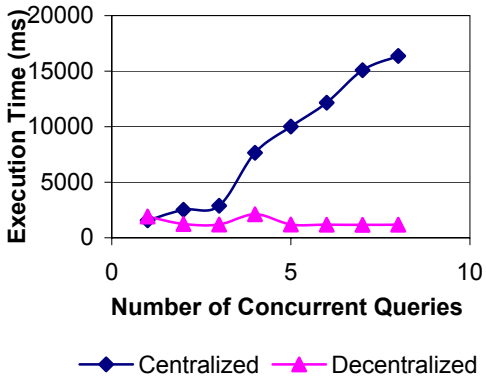
4 Related Work

There are other projects that aim at integrating heterogeneous databases such as TAMBIS [13] and TSIMMIS [14]. BACIIS also shares many of its design features with other distributed systems that perform the same functions of query entry and data retrieval. These systems include the Distributed Information Systems Control World (DISCWorld) [15]. In this section, BACIIS is compared to these systems.

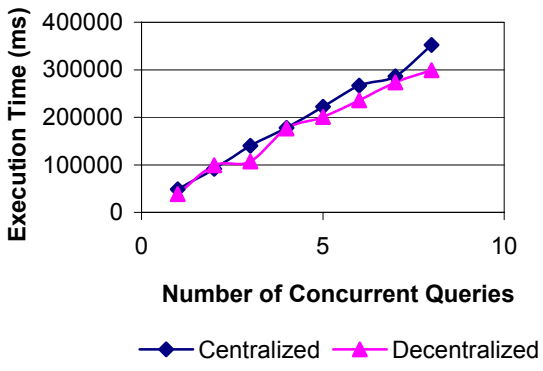
TAMBIS integrates a specific set of life science databases that consists of protein and nucleic acid data sources. BACIIS differs from TAMBIS in that it aims at integrating all possible life science data sources. In addition the architecture of TAMBIS is not distributed.

The goal of the TSIMMIS Project is to develop tools that facilitate the rapid integration of heterogeneous information sources that may include both structured and semi-structured data. TSIMMIS has components that translate queries and information (source wrappers), extract data from web sites, combine information from several sources (mediator), and allow browsing of data sources over the Web. TSIMMIS itself is not an information integration system, but a tool for building integration systems. TSIMMIS utilizes a distributed CORBA-based protocol for submitting queries to data sources and obtaining results [16]. This distributed protocol is asynchronous in nature and will return partial results to the client during data retrieval. The protocol requires server side support, which is impractical for most of the web databases that BACIIS integrates.

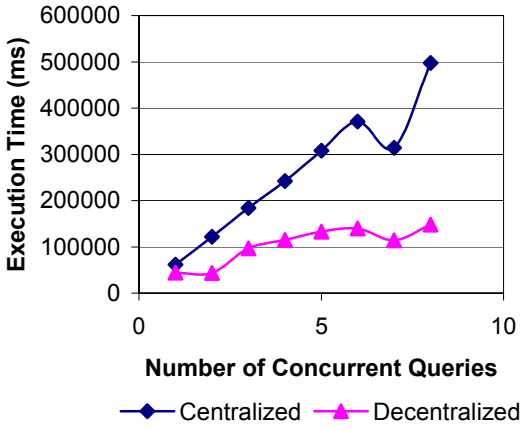
DISCWorld is a Java-based middleware for integrating distributed computing component applications across wide-area networks[17]. Like TISMMIS, DISCWorld also requires the integrated services to be well defined and described, thus it is not suitable for the integration of web databases. DISCWorld focuses on issues such as scalability, platform heterogeneity and the ability to operate over wide-area networks[18].



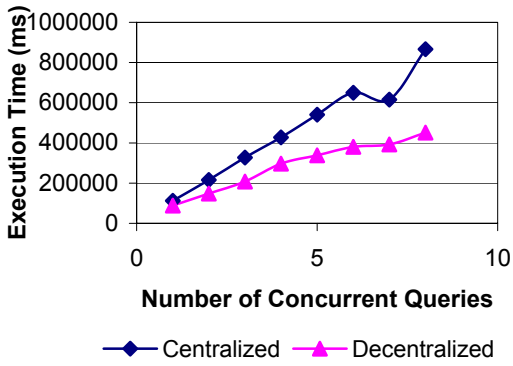
(a) Query planner server execution time



(b) Wrapper service server execution time



(c) Result presentation server execution time



(d) Overall query response time

Figure 5: Execution times for complex query.

6 Conclusion

This paper describes the decentralized architecture of BACIIS. This architecture consists of five servers that cooperate to answer multi-database queries over a set of geographically distributed life science databases. These servers can be executed on the same host machine or on different machines. This decentralized implementation is scalable and maximizes resource utilization. In addition, the decentralized implementation reduces the effort needed to add new services to BACIIS.

The decentralized architecture of BACIIS shows a performance gain in query response time for simple queries when compared to the performance of the centralized architecture. The difference in performance is less apparent when more complex queries are submitted to BACIIS. In this case, the query execution time is dominated by the time it takes to retrieve data from remote web databases (i.e. the wrapper service server). This time is controlled by the access time to the remote databases. This effect was exacerbated with the fact that in the experiment multiple copies of the same query were used. All of these copies access the same records in the

same web databases. The impact of the access time to remote web databases on the performance of BACIIS will be reduced further if mirror web sites for the web databases can also be used. In addition, this impact can be reduced if BACIIS can support the replication of services. That is, initiating multiple copies of each of the servers on different hosts when BACIIS receives multiple queries.

BACIIS is available at http://baciis.engr.iupui.edu.

Acknowledgment

This research is supported by the National Science Foundation under grants CAREER DBI-0133946 and DBI-0110854, by the Indiana 21st Century Research and Technology Fund, and by Eli Lilly and Company.

References

[1] Z. Ben Miled, O. Bukhres, Y. Wang, N. Li, M. Baumgartner, B. Sipes, Biological and Chemical Information Integration System, *Network Tools and Applications in Biology*, Genoa, Italy, May 2001.

[2] http://www.ncbi.nlm.nih.gov/Genbank/index.html

[3] http://www.expasy.ch/sprot/

[4] http://www.ncbi.nlm.nih.gov/entrez/

[5] http://www.rcsb.org/pdb/

[6] http://www.expasy.ch/alinks.html#Proteins

[7] Wiederhold, G., Mediators in the Architecture of Future Information Systems, *IEEE Computer*, pages 38-49, 1992.

[8] Knoblock, C.A., Minton, S. The ariadne approach to web-based information integration. *IEEE Intelligent Systems*, 13(5), September/October 1998.

[9] Levy, A.Y., The Information Manifold Approach to Data Integration, *IEEE Intelligent System*, 1998.

[10] Ben Miled, Z., Wang, Y., Li, N., Bukhres, O., Martin, J., Nayar, A. and Oppelt, R., BAO, A Biological and Chemical Ontology For Information Integration, *Online Journal Bioinformatics*, Vol. 1, pages 60-73, 2002.

[11] George Coulouris, Jean Dollimore, and Tim Kindberg., "Distributed Systems Concepts and Design", P194-200, Addison-Wesley Publishers Limited, Third Edition, 2001

[12] MacGregor, R.M., Chalupsky H., and Melz, E.R. Powerful knowledge representation and reasoning with deliver in Common-Lisp, C++, (and, eventually, Java). *PowerLoom Manual*, Nov.1997.

[13] Paton, N.W., Stevens, R., Baker, P.G., Goble, C.A., Bechhofer, S., and Brass, A. Query Processing in the TAMBIS Bioinformatics Source Integration System, *Proc. 11th Int. Conf. on Scientific and Statistical Databases (SSDBM), IEEE Press*, pages 138-147, 1999

[14] J. Hammer, M. Breunig, H. Garcia-Molina, S. Nestorov, V. Vassalos, R. Yerneni. Template-based Wrappers in the TSIMMIS System. *In Proceedings of the Twenty-Sixth SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 12-15, 1997.

[15] K. A. Hawick, H. A. James, and J. A. Mathew., Remove Data Access in Distributed Object-Oriented Middleware, *Parallel and Distributed Computing Practices*, 2000

[16] Garcia-Molina, H. and Paepcke, A., Proposal for I**3 Client Server Protocol, *Technical Report*, September 1996

[17] Hawick, K.A., James, H.A., and Coddington, P.D. A Reconfigurable Component-based Problem Solving Environment. *Proc. Of Hawiaii International Conference on System Sciences (HICSS-34),* 2000.

[18] Hawick, K.A., James, H.A., Silis, A.J., Grove, D.A., Patten, C.J., Mathew, J.A., Coddington, P.D., Kerry, K.E., Hercus, J.F., and Vaughan, F.A. DISCWorkd: An Environment for Service-Based Metacomputing. *Future Generation Computer Systems* (15), pages 623-640, 1999.

An Algorithm for Computing the Optimal Cycle Time of a Printed Circuit Board Assembly Line

Dušan M. Kodek and Marjan Krisper
University of Ljubljana, Faculty of Computer and Information Science
Tržaška 25, 1000 Ljubljana, Slovenia
E-mail: duke@fri.uni-lj.si

Keywords: combinatorial optimization, integer programming, minimax approximation

Received: December 24, 2002

We consider the problem of optimal allocation of components to a printed circuit board (PCB) assembly line which has several nonidentical placement machines in series. The objective is to achieve the highest production throughput by minimizing the cycle time of the assembly line. This problem can be formulated as a minimax approximation integer programming model that belongs to the family of scheduling problems. The difficulty lies in the fact that this model is proven to be NP*-complete. All known algorithms that solve the* NP*-complete problems are exponential and work only if the number of variables is reasonably small. This particular problem, however, has properties that allow the development of a very efficient type of branch-and-bound based optimal algorithm that works for problems with a practically useful number of variables.*

1 Introduction

The problem of optimal allocation of components to placement machines in a printed circuit board (PCB) assembly line is *NP*-complete and is often considered too difficult to solve in practice. This opinion is supported by the experience with the general integer programming programs that are typically very slow and do not produce solutions in a reasonable time. It is therefore not surprising to see many attempts of replacing the optimal solution with a near-optimal one. The reasoning goes as follows: A near-optimal solution is often good enough and is usually obtained in a significantly shorter time than the optimal solution. Although this is true in many cases, it does not hold always. The difficulty with the near-optimal methods is that they, as a rule, do not give an estimate of closeness to the optimal solution. This means that a significantly better optimal solution, about which the user knows nothing, may exist. Given a choice, the user would probably always choose the optimal solution provided that it can be obtained in a reasonable time.

This paper challenges the opinion that the optimal solution is too difficult to compute. An algorithm that takes advantage of the special properties of the minimax approximation optimal allocation problem is developed. This optimal algorithm is much faster than the general integer programming approach mentioned above. The algorithm produces, in most practical cases, the optimal solution in a time that is similar to the time needed for near-optimal methods. Because of its exponential nature, it will of course fail in the cases when the number of variables is large. But it should be noted that the algorithm practically always produces one or more suboptimal solutions which can be used in such cases. These suboptimal solutions are comparable to those obtained by the near-optimal methods like local search, genetic algorithms, or knowledge based systems. Or in other words, the user can only gain if the optimal algorithm is used.

Let us start with an investigation of the PCB assembly line problem. The cycle time T of a PCB assembly line is defined as the maximum time allowed for each machine (or station) in the assembly line to complete its assembly tasks on the board. This time becomes important when the quantity of PCBs is large: A minor reduction in the cycle time can result in a significant cost and time savings. Moreover, a PCB line in the problem has several non-identical placement machines. As a board contains hundreds of surface mounted components in different shapes, sizes, and patterns, different placement machines in the line are installed to cope with different components. The line efficiency depends on the combination of the machine types. Due to the costly placement machines, the optimization of the assembly process can significantly increase the competitiveness of the production.

Many factors affect the efficiency of the PCB assembly, namely customer orders [1], component allocation [2], PCB grouping [3], component sequence [4], and feeder arrangement [5]. Different algorithms have been developed to optimize different factors in PCB assembly [6], [7]). The genetic algorithm technique is one of the heuristic methods that has been used recently to find a near-optimal solution [8].

Machine M_i	Placement times t_{ij} for component type j							Setup time s_i
	1	2	3	4	5	6	7	
1	0.3	0.7	0.7	0.5	∞	∞	∞	11.0
2	0.7	1.2	1.5	1.6	1.5	1.5	2.1	14.7
3	2.3	3.8	3.5	3.5	2.7	3.3	4.3	14.7
Number of type j components per board	324	37	12	5	7	5	4	

Table 1: An example of a PCB assembly line with 3 different placement machines and 7 different component types per board. The placemet times t_{ij} for different components and machines and the setup times s_i are in seconds.

2 Formulation of the problem

When a board is assembled on a production line the board's components are grouped and allocated to appropriate placement machines in order to achieve a high output of the line. The next machine can begin its tasks only after the previous machine has completed the placement of all components that were allocated to it. After the board passes through all the machines, the component placement process is completed. It is clear that the slowest task dictates the performance of the assembly line.

There are two important differences between the traditional assembly line problem and this PCB assembly line problem. First, unlike the traditional assembly line, the precedence of operations in the PCB assembly is not important and can be ignored. The second difference concerns the assembly times for the same component on different machines. Due to various types and configurations of the placement machines, different machines have different times for placement of the same kind of component. The components are usually of a surface mounted type, although this is not important here. An example from Table 1 is used to make the problem easier to understand. This example is the same as the one used in [8] and will allow the comparison of our optimal algorithm to the near-optimal one.

A PCB assembly line with three different placement machines M_1, M_2, M_3 and a board with seven types of components is used in the example. The placemet times t_{ij} for different components and machines are given in the Table 1. If a machine cannot handle a particular type of component, its placement time is assigned to be infinite (∞). The infinity is used here for simplicity of notation only — it is replaced by a large positive number for computation. In addition to the time that is needed to place a component there is also a setup time s_i for each of the machines M_i. The machine needs this time every time a new board arrives for its positioning and placement preparation. Finally, a total number of each type of a component per board c_j is also given.

Obviously, there are many possible ways of allocating the components j to the placement machines M_i. Each of them leads to its own cycle time T. The question is how to allocate the components in such a way that the assembly line has the best performance. The PCB assembly line cycle time T is formally defined as the maximum time needed by one of the machines M_i, $i = 1, 2, \cdots, m$, to complete the placement of the components allocated to it. Clearly, the time interval between two finished boards coming out of the assembly line is equal to T which means that the number of boards produced in a given time span is proportional to $1/T$. This number can be increased by allocating the components to the machines in such way that T is reduced. A mathematical model that describes this situation can now be given.

Suppose that there are m non-identical placement machines M_i in a PCB assembly line and that a board with n types of components is to be assembled on this line. It takes t_{ij} units of time to place the component of type j on a machine M_i. In addition, each machine M_i has a setup time s_i. There are exactly c_j components of type j per board. The component allocation problem can be formulated as

$$T_{opt} = \min_{x_{ij}} \max_{i=1,2,\ldots,m} \left(s_i + \sum_{j=1}^{n} t_{ij} x_{ij} \right) , \qquad (1)$$

subject to

$$\sum_{i=1}^{m} x_{ij} \;=\; c_j, \quad j = 1, 2, \ldots, n\,, \qquad (2)$$

$$x_{ij} \;\geq\; 0 \quad \text{and integer}\,. \qquad (3)$$

The solution of this problem is the optimal cycle time T_{opt} and the optimal allocation variables $x_{ij}^{(opt)}$. The variable x_{ij} gives the number of components of type j that are allocated to machine M_i. Constraints (2) ensure that all of the components will be allocated. The components are indivisible and (3) ensures that x_{ij} are positive integers. Note that t_{ij} and s_i are by definition positive[2].

3 Complexity of the problem

The problem (1)–(3) is a combination of assignment and flowshop scheduling problems [9]. It is *NP*-complete for

[2]The times t_{ij} and s_i can be arbitrary positive real numbers. It is easy to reformulate the problem and change t_{ij} and s_i into arbitrary positive integers. This does not change the complexity of the problem.

$n \geq 2$. Proving the *NP*-completness is no too difficult. First, it is trivial to show that the problem is in *P*. Second, it is possible to show that the well known PARTITION problem can be polynomially transformed into (1)–(3) [10]. Since PARTITION is *NP*-complete, so is our problem.

A typical approach to solving this problem is to treat it as a general mixed integer linear programming problem. The minimax problem (1)–(3) is reformulated as

$$\begin{aligned} &\min_{x_{ij}} T_{opt}\,, \\ &T_{opt} - s_i - \sum_{j=1}^{n} t_{ij}x_{ij} \geq 0,\ \ i = 1, 2, \ldots, m, \\ &\sum_{i=1}^{m} x_{ij} = c_j,\ \ j = 1, 2, \ldots, n, \\ &x_{ij} \geq 0 \quad \text{and integer.} \end{aligned} \tag{4}$$

All algorithms that are capable of solving this problem optimally work by starting with the noninteger problem where the variables x_{ij} can be any positive real number. Additional constraints are then gradually introduced into the problem and these constraints eventually force the variables x_{ij} to integer values. Many instances of suitably reformulated subproblems of the form (4) must be solved before the optimal solution is found.

An advantage of the formulation (4) is that general mixed integer programming programs can be used to solve it. Unfortunately, this advantage occurs at the expense of the computation time. The general programs use the simplex algorithm to solve the subproblems. The simplex algorithm is very general and slow since it does not use any of the special properties of the minimax problem. All these properties are lost if the original problem (1)–(3) is converted into the general problem.

The fact that the general problem (4) is so slow has led to the development of suboptimal heuristic algorithms that search for a near-optimal solution. These algorithms are faster and often good enough. The difficulty is that a significantly better optimal solution may exist which these algorithms do not find. It is the purpose of this paper to develop an optimal algorithm that does not use the generalized formulation (4). The algorithm takes advantage of the special properties of the minimax problem (1)–(3). It avoids using the simplex algorithm completely which leads to a much faster solution.

4 The lower bound theorem

The basic idea of our algorithm is to use a lower bound for T_{opt} as a tool that leads to the solution. This lower bound must be computed for each of the subproblems that appear within the branch-and-bound process. It must take into account the fact that some of the subproblem's variables x_{ij} are known integers. To derive it, let us assume that the subproblems's variables $x_{ij}, j = 1, 2, \ldots, k-1$, are known integers for all i. In addition, some, but not all, of the variables x_{ik} may also be known integers. Let I_k be the set of indices i that correspond to the known integers x_{ik}. The subproblem's variables can be formally described as

$$x_{ij} = \begin{cases} x_{ij}^I,\ j = 1, \ldots, k-1,\ i = 1, \ldots, m \\ x_{ij}^I,\ j = k,\ i \in I_k \\ x_{ij},\ j = k,\ i \notin I_k \\ x_{ij},\ j = k+1, \ldots, n,\ i = 1, \ldots, m, \end{cases} \tag{5}$$

where k can be any of the indices $1, 2, \ldots, n$. Notation x_{ij}^I is used to describe the variables that are already known integers. The remaining variables x_{ij} are not yet known. The number of indices in the set I_k lies in the range 0 to $m-2$. If there were $m-1$ known integers x_{ik}^I the constraint (2) gives the remaining variable which contradicts the assumption that not all of the variables x_{ik} are known. The index k changes to $k+1$ when all x_{ik} are known integers.

Definition (5) assumes that a certain rule is used to introduce the constraints which force the variables x_{ij} to integer values. This rule is simple: For every index k it is necessary to constrain x_{ik} to known integers x_{ik}^I for all $i, i = 1, 2, \ldots, m$, before k can change. The rule follows from the structure of constraints given by (2) and is needed to derive the lower bound theorem. There is no problem with this rule because the branch-and-bound method, on which our algorithm is based, allows complete freedom of choosing the variable x_{ij} that is to be constrained next. The indices k can be selected in any order. A simple ascending order $k = 1, 2, \ldots, n$, is used in (5). This also applies to the case when the problem is first reordered along the indices j in a way that gives the fastest rate of lower bound increase. Such a reordering is used in our algorithm.

To simplify the notation, let us first use the known integers x_{ij}^I and redefine s_i into s_i' as

$$s_i' = \begin{cases} s_i + \sum_{j=1}^{k} t_{ij}x_{ij}^I,\ \ i \in I_k \\ s_i + \sum_{j=1}^{k-1} t_{ij}x_{ij}^I,\ \ i \notin I_k\,. \end{cases} \tag{6}$$

Similarly, the known integers x_{ik}^I (if any) are used to redefine c_k into c_k' as

$$c_k' = c_k - \sum_{i \in I_k} x_{ik}^I\,. \tag{7}$$

The lower bound on T_{opt} over all possible not yet known variables x_{ij} is the most important part of our algorithm. It is developed along the lines used in a related integer polynomial minimax approximation problem that appears in a digital filter design [11], [12] and is given in the following theorem.

Theorem 1 *Let T_{opt} be the minimum cycle time corresponding to the optimal solution of the problem (1)–(3) in which some of the variables are known integers defined by*

(5). Then T_{opt} is bounded by

$$T_{opt} \geq \max_{j=k+1,\ldots,n} \left(\frac{c_j + \sum_{i=1}^{m} \frac{s'_i}{t_{ij}} + p_j + q_j}{\sum_{i=1}^{m} \frac{1}{t_{ij}}} \right) \tag{8}$$

where

$$\begin{aligned} p_j &= \sum_{\substack{r=k+1 \\ r\neq j}}^{n} c_r \min_{i=1,2,\ldots,m} \left(\frac{t_{ir}}{t_{ij}} \right), \\ q_j &= c'_k \min_{i\notin I_k} \left(\frac{t_{ik}}{t_{ij}} \right), \quad j = k+1,\ldots,n\,. \end{aligned} \tag{9}$$

Proof: Let h be a number that satisfies

$$h \geq \begin{cases} s_i + \sum_{j=1}^{k-1} t_{ij} x^I_{ij} + \sum_{j=k+1}^{n} t_{ij} x_{ij}, & i \in I_k \\ s_i + \sum_{j=1}^{n} t_{ij} x^I_{ij} + \sum_{j=k}^{n} t_{ij} x_{ij}, & i \notin I_k\,. \end{cases} \tag{10}$$

Note that h is a lower bound for T_{opt} if we can prove that (10) holds over all possible not yet known values x_{ij}. Using (6) eq. (10) is simplified

$$h \geq \begin{cases} s'_i + \sum_{j=k+1}^{n} t_{ij} x_{ij}, & i \in I_k \\ s'_i + \sum_{j=k}^{n} t_{ij} x_{ij}, & i \notin I_k\,. \end{cases} \tag{11}$$

It follows from (11) that variables x_{ij} can be expressed as

$$\begin{aligned} x_{ij} &\leq \frac{h}{t_{ij}} - \frac{s'_i}{t_{ij}} - \sum_{\substack{r=k+1 \\ r\neq j}}^{n} \frac{t_{ir}}{t_{ij}} x_{ir},\ i \in I_k, j = k+1,\ldots,n, \\ x_{ij} &\leq \frac{h}{t_{ij}} - \frac{s'_i}{t_{ij}} - \sum_{\substack{r=k \\ r\neq j}}^{n} \frac{t_{ir}}{t_{ij}} x_{ir},\ i \notin I_k,\ j = k,\ldots,n. \end{aligned} \tag{12}$$

Adding all x_{ij} by index i and using (2) and (7) gives

$$c_j \leq \sum_{i=1}^{m} \frac{h}{t_{ij}} - \sum_{i=1}^{m} \frac{s'_i}{t_{ij}} - \sum_{\substack{r=k+1 \\ r\neq j}}^{n} \sum_{i=1}^{m} \frac{t_{ir}}{t_{ij}} x_{ir} - \sum_{i\notin I_k} \frac{t_{ik}}{t_{ij}} x_{ik}, \quad j = k+1,\ldots,n\,, \tag{13}$$

and the lower bound for h can now be written as

$$h \geq \frac{c_j + \sum_{i=1}^{m} \frac{s'_i}{t_{ij}} + \sum_{\substack{r=k+1 \\ r\neq j}}^{n} \sum_{i=1}^{m} \frac{t_{ir}}{t_{ij}} x_{ir} + \sum_{i\notin I_k} \frac{t_{ik}}{t_{ij}} x_{ik}}{\sum_{i=1}^{m} \frac{1}{t_{ij}}}, \quad j = k+1,\ldots,n\,. \tag{14}$$

All the terms in (14) are positive. This means that h is a lower bound over all variables if the lowest possible values of the terms containing variables x_{ir} and x_{ik} are used. The variables x_{ir} are subject to

$$\sum_{i=1}^{m} x_{ir} = c_r\,, \quad r = k+1,\ldots,n\,. \tag{15}$$

It is quite easy to see that the sum containing x_{ir} is bounded by

$$\sum_{\substack{r=k+1 \\ r\neq j}}^{n} \sum_{i=1}^{m} \frac{t_{ir}}{t_{ij}} x_{ir} \geq \sum_{\substack{r=k+1 \\ r\neq j}}^{n} c_r \min_{i=1,2,\ldots,m} \left(\frac{t_{ir}}{t_{ij}} \right) = p_j, \quad j = k+1,\ldots,n, \tag{16}$$

since it is obvious that a minimum is obtained if x_{ir} is given the value c_r for index i that corresponds to the lowest of the factors t_{ir}/t_{ij} while all other x_{ir} are set to zero. Similarly, the variables x_{ik} are subject to

$$\sum_{i\notin I_k} x_{ik} = c'_k\,, \tag{17}$$

and the sum containing x_{ik} is bounded by

$$\sum_{i\notin I_k} \frac{t_{ik}}{t_{ij}} x_{ik} \geq c'_k \min_{i\notin I_k} \left(\frac{t_{ik}}{t_{ij}} \right) = q_j\,, \quad j = k+1,\ldots,n\,. \tag{18}$$

Equations (16) and (18) are used in the definitions (9) and this completes the proof. □

Note that the Theorem 2 does not include the lower bound for the case $k = n$. The following trivial lower bound, which holds for all k, can be used in this case

$$T_{opt} \geq \max_{i\notin I_k} \left(s'_i + t_{ik} x_{ik} \right), \quad k = 1,\ldots,n. \tag{19}$$

Note also that index $j = k$ was not used in the derivation of the Theorem 1. The equivalent of (13) for $j = k$ is

$$c'_k \leq \sum_{i\notin I_k} \frac{h}{t_{ik}} - \sum_{i\notin I_k} \frac{s'_i}{t_{ik}} - \sum_{r=k+1}^{n} \sum_{i\notin I_k} \frac{t_{ir}}{t_{ik}} x_{ir}\,. \tag{20}$$

When I_k is not empty all x_{ir} in the sum over $i \notin I_k$ can be zero and still satisfy (15). The lowest possible sum containing x_{ir} is obviously zero in this case. This gives an additional lower bound

$$T_{opt} \geq \frac{c'_k + \sum_{i\notin I_k} \frac{s'_i}{t_{ik}}}{\sum_{i\notin I_k} \frac{1}{t_{ik}}}\,. \tag{21}$$

This lower bound is almost always much lower than the one given by (8). It can included in the algorithm to bring a small decrease in computing time which is on the order of 1%.

By choosing $k = 0$ one can use (8)–(9) to compute the lower bound over all possible values of variables x_{ij}. Applying this to the example from the Table 1 gives $T_{opt} \geq 96.084$. But there is more — the theorem plays a central role in our algorithm because it eliminates the need to use the simplex algorithm for solving the subproblems within the branch-and-bound process.

5 Application of the lower bound theorem

The usefulness of the Theorem 1 is based on the following observation: The problem of finding the all-integer solution that gives the lowest cycle time T_{opt} can be replaced by the problem of finding the all-integer solution that has the lowest lower bound for T_{opt}. Both approaches obviously lead to the same solution since T_{opt} equals its lower bound when all variables x_{ij} are integers.

This observation, however, is not enough. A new constraint must be introduced on one of the variables $x_{ik}, i \notin I_k$, at each branch-and-bound iteration. This constraint cannot be made on the basis of the Theorem 1 alone and requires additional elaboration.

To see how the lower bound depends on x_{ik} let us define the parameters $T_L(j,k)$ as

$$T_L(j,k) = \frac{c_j + \sum_{i=1}^{m} \frac{s'_i}{t_{ij}} + p_j + \sum_{i \notin I_k} \frac{t_{ik}}{t_{ij}} x_{ik}}{\sum_{i=1}^{m} \frac{1}{t_{ij}}}, \qquad (22)$$

where $j = k+1, \ldots, n$, and $k = 1, \ldots, n-1$. The $T_L(j,k)$ are simply (14) rewritten in a slightly different way. The Theorem 1 lower bound (8) in which the variables x_{ik} are left is now equal to

$$T_{opt} \geq \max_{j=k+1,\ldots,n} T_L(j,k)\,. \qquad (23)$$

This lower bound does not include the case $k = n$. This is easily corrected if (19) is included. To simplify notation we first define parameters $T_I(i,k)$ as

$$T_I(i,k) = s'_i + t_{ik} x_{ik}, \quad k = 1, \ldots, n, \qquad (24)$$

and define the new lower bound $T_{opt} \geq T_{LB}(k)$

$$T_{LB}(k) = \max\left(\max_{i \notin I_k} T_I(i,k), \max_{j=k+1,\ldots,n} T_L(j,k) \right). \qquad (25)$$

The $T_{LB}(k)$ are defined for $k = 1, \ldots, n$ (where $T_L(j,n) = 0$). They include $T_I(i,k)$ for all k even if it is strictly needed only for $k = n$. There is a good reason for that because the T_I lower bound sometimes exceeds the T_L lower bound. This can occur when the values of t_{ij} differ by several orders of magnitude as is the case in the example from Table 1 where a large positive t_{ij} is used instead of ∞. Although the algorithm works if T_I is used for $k = n$ only, experiments show that it is usually faster if it is used for all k.

The lower bound $T_{LB}(k)$ (25) is the basis of our algorithm. It is a linear function of the variables $x_{ik}, i \notin I_k$, and, as mentioned before, a new constraint must be introduced on one of them at each branch-and- bound iteration.

Let $i_c, i_c \notin I_k$, be the index of the variable $x_{i_c k}$ that is selected for constraining. Selection of the index i_c is simple — any of the indices $i, i \notin I_k$, can be used as i_c. It is more difficult to find the value $x^*_{i_c k}$ that will be used in the branch-and-bound iteration to constrain the selected variable to integers $x^I_{i_c k}$ which are the nearest lower and upper neighbours of $x^*_{i_c k}$. The $x^*_{i_c k}$ must be a number that gives the lowest possible lower bound $T_{LB}(k)$ over all possible values of the not yet known variables $x_{ik}, i \notin I_k$, and $x_{ij}, i = 1, \ldots, m, j = k+1, \ldots, n$. Or in other words, the $x^*_{i_c k}$ must be at the global minimum of $T_{LB}(k)$.

It is important to understand why $x^*_{i_c k}$ must be at the global minimum of $T_{LB}(k)$. It is because our algorithm uses the property that $T_{LB}(k)$ is a linear function of the variables x_{ik} and is therefore also convex. The convex property is crucial for the success of our algorithm since it ensures that every local optimum is also global. The algorithm uses this property by stopping the search along a variable in the branch-and-bound process when $T_{LB}(k)$ exceeds the current best solution T_u. This, however, can be used only if $x^*_{i_c k}$ is such that $T_{LB}(k)$ does not decrease when an arbitrary integer is added to $x^*_{i_c k}$. The $x^*_{i_c k}$ at the global minimum certainly satisfies this condition.

A great advantage of using the lower bound comes from the fact that the lower bound $T_{LB}(k)$ in (25) depends only on the variables $x_{ik}, i \notin I_k$, and is independent of the remaining variables $x_{ij}, i = 1, \ldots, m, j = k+1, \ldots, n$. This means that the number of variables is significantly reduced in comparison with the general approach (4). Solution of the minimax problem

$$\min_{\substack{x_{ik} \\ i \notin I_k}} \max\left(\max_{i \notin I_k} T_I(i,k), \max_{j=k+1,\ldots,n} T_L(j,k) \right), \qquad (26)$$

$$\sum_{i \notin I_k} x_{ik} = c'_k\,, \qquad x_{ik} \geq 0\,, \qquad (27)$$

gives the nonnegative numbers x^*_{ik} that give the global minimum of $T_{LB}(k)$ for a given k.

A complication arises when k changes to $k+1$ because the solution of (26)–(27) for $k+1$ depends not only on x^*_{ik+1} but also on x^*_{ik} (through s'_i). The problem is that x^*_{ik} are not at the global minimum of $T_{LB}(k+1)$. It is possible that the minimum of (26) for $k+1$ decreases if different x^*_{ik} are used. An error can occur if this is ignored because the algorithm stops the search along a variable if the minimum is $> T_u$ when in fact a lower value for $T_{LB}(k+1)$ exists. It is obvious that this error cannot occur if the minimum $T_{LB}(k+1) \leq T_{LB}(k)$.

The following corrective procedure is used in the algorithm when the minimum $T_{LB}(k+1) >$ minimum $T_{LB}(k)$. It consists of adding +1 and/or -1 to the $x^*_{i_c k}$ that was used

as the last constraint. Using the new $x^*_{i_c k}$ we simply recompute $T_{LB}(k)$ and solve again (26)–(27) for $k+1$. If $\max(T_{LB}(k), T_{LB}(k+1))$ decreases we continue in that direction until it stops decreasing or until (27) is violated ($T_{LB}(k)$ increases when the original $x^*_{i_c k}$ changes). The corrected $x^*_{i_c k}$ is a solution of

$$\min_{x_{i_c k},\, x_{ik+1}} \max\left(T_{LB}(k), T_{LB}(k+1)\right) . \qquad (28)$$

It is used to replace the original and this eliminates the possibility of error. Note that it is not necessary to correct the remaining variables x^I_{ik} even if they were not derived from the global minimum of $T_{LB}(k+1)$. This is because the branch-and-bound process ensures that all values of x^I_{ik} will be tried as long as their $T_{LB}(k)$ is lower than T_u. Additional details about the implementation of (28) are given in step 6 of the algorithm in section 7.

The minimax problem (26)–(27) must be solved many times within the branch-and-bound process and it is extremely important to have an efficient method that gives its solution. Most of the computing time in our algorithm is spent on solving this problem. The method that is used to solve it is worth a detailed description.

6 Solving the discrete linear minimax problem

The number of variables x_{ik} in (26)–(27) is equal to the number of indices $i, i \notin I_k$. Let $m', 1 \le m' \le m$, be this number and let $R(i), i = 1, \ldots, m'$, be the indices not in I_k. Equation (26) contains m' terms T_I and $n-k$ terms T_L. The total number of terms n' is equal to

$$n' = n + m' - k\,, \qquad m' \le n' \le n + m' . \qquad (29)$$

It helps to rewrite (26) using a new index v

$$\min_{x_{R(i)k}} \max\left(\max_{v=1,\ldots,m'} T_I(R(v),k), \max_{v=m'+1,\ldots,n'} T_L(v',k) \right), \qquad (30)$$

where $v' = v + k - m'$. Because of the sum constraint in (27) there are only $m'-1$ independent variables; the m'-th variable can be expressed as

$$x_{R(m')k} = c'_k - \sum_{i=1}^{m'-1} x_{R(i)k}. \qquad (31)$$

The minimax problem (26)–(27) can now be reformulated into a more general form

$$\min_{x_{R(i)k}} \max_{v=1,\ldots,n'} \left(f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x_{R(i)k} \right), \qquad (32)$$

$$\sum_{i=1}^{m'-1} x_{R(i)k} \le c'_k\,, \quad x_{R(i)k} \ge 0\,. \qquad (33)$$

Definitions of terms f_v and Φ_{vi} are somewhat tedious though they follow directly from (22) and (24)

$$f_v = \begin{cases} s'_{R(v)}\,, & v = 1, \ldots, m'-1 \\ s'_{R(v)} + t_{R(v)k} c'_k\,, & v = m' \\ \dfrac{c_{v'} + \sum_{r=1}^{m} \frac{s'_r}{t_{rv'}} + p_{v'} + \frac{t_{R(m')k}}{t_{R(m')v'}} c'_k}{\sum_{r=1}^{m} \frac{1}{t_{rv'}}}\,, & v > m' \end{cases} \qquad (34)$$

$$\Phi_{vi} = \begin{cases} t_{R(i)k} \text{ if } i = v,\ 0 \text{ if } i \ne v, & v = 1,\ldots,m'-1 \\ -t_{R(m')k}\,, & i = 1,\ldots,m'-1,\ v = m' \\ \dfrac{\frac{t_{R(i)k}}{t_{R(i)v'}} - \frac{t_{R(m')k}}{t_{R(m')v'}}}{\sum_{r=1}^{m} \frac{1}{t_{rv'}}}, & i = 1,\ldots,m'-1,\ v > m' \end{cases} \qquad (35)$$

for $v = 1, \ldots, n'$ and $i = 1, \ldots, m'-1$.

The process of solving (32)–(33) is simplified greatly by the theorem that gives the necessary and sufficient conditions for the variables $x^*_{R(i)k}, i = 1, \ldots, m'-1$, that minimize (32). The general version of the theorem is given in [15]. It is repeated here in the form that applies to our problem.

Theorem 2 *The variables $x^*_{R(i)k}, i = 1, \ldots, m'-1$, are the optimal solution of the minimax problem (32)–(33) if and only if the following holds*

$$\min_{z_i} \max_{v \in V_{max}(x^*)} \sum_{i=1}^{m'-1} \Phi_{vi}(z_i - x^*_{R(i)k}) = 0\,, \qquad (36)$$

over all numbers $z_i, i = 1, \ldots, m'-1$, that satisfy

$$\sum_{i=1}^{m'-1} z_i \le c'_k\,, \quad z_i \ge 0\,. \qquad (37)$$

The set $V_{max}(x^*)$ contains those of the indices $v, v = 1, \ldots, n'$, at which the maximum is obtained. That is

$$\max_{v=1,\ldots,n'} \left(f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x^*_{R(i)k} \right) = f_v + \sum_{i=1}^{m'-1} \Phi_{vi} x^*_{R(i)k}\,, \quad v \in V_{max}(x^*)\,. \qquad (38)$$

Only the indices $v, v \in V_{max}(x^*)$, that give the extremal values of the function (38) are used in the Theorem 2. The theorem says that $x^*_{R(i)k}$ is the optimal solution if there are no numbers z_i for which (36) is lower than zero. To show how this can be used to solve (32)–(33) let us assume that we have a set of numbers $x^*_{R(i)k}$ and would like to check if they are optimal. Depending on $V_{max}(x^*)$ and Φ_{vi} there are two mutually exclusive cases:

1. The set $V_{max}(x^*)$ contains at least two indices v_1 and v_2 for which the following holds

$$\Phi_{v_1 i}\Phi_{v_2 i} \leq 0\,, \quad i = 1, \ldots, m' - 1\,. \tag{39}$$

It is easy to see that the numbers z_i that give (36) lower than zero cannot exist. This is because of the opposite signs of $\Phi_{v_1 i}$ and $\Phi_{v_2 i}$ for all i. Any set of numbers z_i that is different from $x^*_{R(i)k}$ makes (36) greater than zero for at least $v = v_1$ or $v = v_2$. Thus, according to the Theorem 2, $x^*_{R(i)k}$ are optimal.

2. The set $V_{max}(x^*)$ does not contain two indices v_1 and v_2 for which (39) holds (this is always true if $V_{max}(x^*)$ contains only one index v). This means that there exists a set of indices I_p, containing at least one index i, for which

$$\Phi_{v_1 i}\Phi_{v_2 i} > 0,\ i \in I_p,\ v_1, v_2 \in V_{max}(x^*), \tag{40}$$

holds for any pair of indices v from $V_{max}(x^*)$. Or in other words, for each $i \in I_p$ the Φ_{vi} are nonzero and have the same signs for all $v \in V_{max}(x^*)$. Let us assume that there are numbers $z_i, i \in I_p$, that satisfy (37) and give

$$\sum_{i \in I_p} \Phi_{vi} z_i < \sum_{i \in I_p} \Phi_{vi} x^*_{R(i)k},\ v \in V_{max}(x^*). \tag{41}$$

These numbers, together with $z_i = x_{R(i)k}$ for $i \notin I_p$, obviously make (36) lower than zero. The numbers $x^*_{R(i)k}$ are therefore not optimal if such z_i exist. They exist almost always — the only exception occurs if the following holds

$$\sum_{i \in I_p} \Phi_{vi} x^*_{R(i)k} = \min_{z_i} \sum_{i \in I_p} \Phi_{vi} z_i\,, \tag{42}$$

for some $v, v \in V_{max}(x^*)$. It is clear that (41) cannot be satisfied in this case because the $x^*_{R(i)k}$ sum is already the lowest possible. The lowest possible sum in (42) is easy to compute by using $z_i = 0$ for $\Phi_{vi} > 0$ and $z_i = c'_k$ for the most negative of $\Phi_{vi} < 0$. This means that it is also easy to check if $x^*_{R(i)k}$ are optimal.

Using (39)–(42) it becomes straightforward to solve (32)–(33). A starting solution for $x^*_{R(i)k}$ is selected and checked as described above. If it is found optimal, we have a solution. If not, one of the variables $x^*_{R(i_1)k}, i_1 \in I_p$, is tried; if it can change towards zero (if $\Phi_{vi_1} > 0$) or towards c'_k (if $\Phi_{vi_1} < 0$) without violating (33), it leads to an improved solution. It is ignored otherwise and a new variable is tried. The set I_p always contains at least one index i that leads to an improved solution.

The new value of $x^*_{R(i_1)k}$ is computed by trying all $v_1, v_1 \notin V_{max}(x^*)$, and solving

$$f'_{v_1} + \Phi_{v_1 i_1} x^*_{R(i_1)k} = f'_v + \Phi_{v i_1} x^*_{R(i_1)k},\ v \in V_{max}(x^*), \tag{43}$$

where f'_v are defined as

$$f'_v = f_v + \sum_{\substack{i=1 \\ i \neq i_1}}^{m'-1} \Phi_{vi} x^*_{R(i)k},\ v = 1, \ldots, n'. \tag{44}$$

Each of the equations (43) gives a possible new value for $x^*_{R(i_1)k}$. The one that is the least different from the current value must be used because the set $V_{max}(x^*)$ changes at that value. The new $x^*_{R(i_1)k}$ must of course also satisfy (33). Replacing $x^*_{R(i_1)k}$ with the new value gives a new solution $x^*_{R(i)k}, i = 1, \ldots, m' - 1$, for which the whole process is repeated until the optimal solution is found.

Selecting a good starting solution is important because it reduces the number of iterations. Our algorithm uses a solution that is found by choosing $x^*_{R(i)k} = c'_k$ (the remaining $x^*_{R(i)k}$ are zero) for $i = 1, \ldots, m'$, and computing the lower bound $T_{LB}(k)$ for each of them. The choice that gives the lowest $T_{LB}(k)$ is the starting solution. This starting solution is often optimal; when it is not, it usually takes only one or two iterations to find the optimum. Note that the search for the optimal $x^*_{R(i)k}$ is not necessary if the starting $T_{LB}(k)$ is lower than the lower bound (21). In such cases the algorithm simply uses the starting solution.

Having the optimal variables $x^*_{R(i)k}, i = 1, \ldots, m' - 1$, it remains to select the one that will be used as the new constraint. This is done by computing the products

$$t_{R(i)k} x^*_{R(i)k}, \quad i = 1, \ldots, m'\,, \tag{45}$$

where (31) is used to compute the remaining variable $x^*_{R(m')k}$. The index $R(i)$ that gives the highest product is selected as i_c. The reasons for this choice is obvious: The highest of products (45) is most likely to give the largest increase of the lower bound $T_{LB}(k)$.

7 The algorithm

The algorithm is based on the well known branch-and-bound method which is described in detail in many textbooks (see, for example, [13] or [14]). We assume that the reader is familiar with this method and continue with the description of the algorithm.

An important part of the branch-and-bound method is the branch-and-bound tree. Each node in the tree represents a subproblem that has some of the variables constrained to integers. Information that is stored at each node must contain the following: The node's lower bound $T_{LB}(k)$, index k, the size of set I_k (it is equal to $m - m'$), the indices i in I_k, integer variables $x^I_{ij}, j = 1, \ldots, k$, and the noninteger variable $x^*_{i_c k}$ that will be used as the next constraint (together with the index i_c). The efficient organization of the tree is important. It does not, however, influence the results of the algorithm and will not be discussed here. The algorithm is described in the following steps:

1. Set $k = 0$ and use (8)–(9) to compute

$$T_L(j,0) = \frac{c_j + \sum_{i=1}^{m} \frac{s'_i}{t_{ij}} + p_j + q_j}{\sum_{i=1}^{m} \frac{1}{t_{ij}}}, \qquad (46)$$

for $j = 1, 2, \ldots, m$. Sort the lower bounds $T_L(j,0)$ in the ascending order. The problem parameters t_{ij} and c_j are reordered accordingly. It is assumed from here on that $j = 1$ corresponds to the lowest $T_L(j,0)$, $j = 2$ to the next higher $T_L(j,0)$, and so on. The reasons for this reformulation of the problem are simple: We wish to eliminate the indices j that give the lowest contribution to the total lower bound $T_{LB}(k)$ and at the same time keep the indices that give the highest contribution to the total lower bound. Several other strategies for selecting the indices j were tested; none performed better over a large class of problems.

2. Set the current best solution T_u to ∞ (a large positive number). The corresponding variables $x_{ij}^{(u)}$ can be set to anything — they will be replaced by one of the solutions quickly. The index u indicates that T_u is an upper bound on T_{opt}. The alternative is to use some heuristic construction and compute a near-optimal starting solution T_u. We found that this is not really necessary because the algorithm quickly produces good near-optimal solutions.

3. Create the root node. This is done by making $k = 1$, $m' = m$ (this makes the set I_k empty), and solving the problem (32)–(33) as described by (36)–(45). The resulting information is stored in the branch-and-bound tree. Initialize the branching counter N to zero.

4. Choose the branching node by searching through the nodes of the branch-and-bound tree. Go to step 8 if no nodes with $T_{LB}(k) < T_u$ are found or if the tree is empty. Add 1 to the branching counter N and choose the branching node according to the following rule: If N is odd, choose the node with the lowest $T_{LB}(k)$, otherwise choose only among the nodes that contain the largest number of integer variables x_{ij}^I and select the one that has the lowest $T_{LB}(k)$. This branching strategy is a combination of the *lowest lower bound* and *depth first* strategies and is used to get many of the near-optimal solutions as fast as possible. This is especially important for large problems with several hundred variables x_{ij}.

5. Two subproblems are created from the branching node by fixing the node's variable $x_{i_c k}^*$ to integers

$$x_{i_c k}^I = \lfloor x_{i_c k}^* \rfloor, \qquad (47)$$

$$x_{i_c k}^I = \lfloor x_{i_c k}^* \rfloor + 1, \qquad (48)$$

where $\lfloor x_{i_c k}^* \rfloor$ denotes the nearest lower integer to $x_{i_c k}^*$. The integers $x_{i_c k}^I$ must of course conform to (27). If $x_{i_c k}^I$ in (48) does not, discard this subproblem (subproblem (47) is never discarded because $x_{i_c k}^*$ satisfies (33)). The number of noninteger variables x_{ik} is reduced by 1

$$m' \leftarrow m' - 1. \qquad (49)$$

If $m' \geq 2$ go to step 6. Otherwise there is only one noninteger variable x_{ik} left. Its integer value is already determined because (27) gives

$$x_{i_c k}^I + x_{ik}^I = c'_k, \qquad (50)$$

and x_{ik}^I is easily computed. All variables x_{ik} are known integers x_{ik}^I, $i = 1, 2, \ldots, m$. Because of this the index k is incremented as described by the definition (5)

$$k \leftarrow k + 1, \qquad (51)$$

The new set I_k is made empty ($m' = m$). If $k \leq n$, go to step 6. Otherwise we have a case where all of the subproblem's variables x_{ij} are integer. This is a *complete integer solution* and the cycle time T is simply computed as

$$T = \max_{i=1,2,\ldots,m} \left(s_i + \sum_{j=1}^{n} t_{ij} x_{ij}^I \right). \qquad (52)$$

If $T < T_u$, we have a new best solution; the current T_u is set to T and the current best solution $x_{ij}^{(u)}$ is replaced by x_{ij}^I. The branch-and-bound tree is searched and all nodes with $T_{LB}(k) \geq T_u$ are removed from the tree. Go to step 7.

6. Each of the non-discarded subproblems from step 5 is solved. The already known integers are taken into account by computing s'_i and c'_k using (6) and (7). Equations (34) and (35) are used next to compute f_v and Φ_{vi} and the problem (32)–(33) is solved as described by (36)–(45). The results are $T_{LB}(k)$ and $x_{i_c k}^*$. If $T_{LB}(k) \geq T_u$ ignore this subproblem since it obviously cannot lead to a solution that is better than the current best T_u. Otherwise if $m' = 2$ and $k < n$ do the corrective procedure (28) and replace $x_{i_c k}^*$ and $T_{LB}(k)$ with the new values. The newly computed $T_{LB}(k)$ will in most cases be greater than that of the branching node. This growth is not monotone and it is possible that the new $T_{LB}(k)$ is lower. Since the lower bound cannot decrease we use the branching node's $T_{LB}(k)$ as the subproblem's $T_{LB}(k)$ in such cases. The subproblem information containing $x_{i_c k}^*$ and $T_{LB}(k)$ is stored as a new node in the branch-and-bound tree.

7. The subproblem in the branching node from step 4 is modified (the root node is an exception — it is simply removed from the branch-and-bound tree and we go to step 4). The branching subproblem is modified by

Machine M_i	Allocation x_{ij} of components							Assembly time on machine M_i
	1	2	3	4	5	6	7	
1	274	0	2	5	0	0	0	97.1
2	50	37	2	0	0	0	0	97.1
3	0	0	8	0	7	5	4	95.3
Number of type j components per board	324	37	12	5	7	5	4	

Table 2: One of the 10 equivalent optimal solutions of the cycle time problem given in example Table 1. The solution was obtained with the algorithm described in this paper.

changing the integer variable x^I_{lk} that was created last. The modification is equal to

$$x^I_{lk} \leftarrow \begin{cases} x^I_{lk} - 1 \text{ if } x^I_{lk} \text{ was created by (47)} \\ x^I_{lk} + 1 \text{ if } x^I_{lk} \text{ was created by (48).} \end{cases} \quad (53)$$

This of course means that each node in the branch-and-bound tree must also contain information about the integer variable that was created last and about the way it was created (either by (47) or (48)). The branching node is removed from the tree if the new $x^I_{lk} < 0$ or if $x^I_{lk} > c'_k$ and we go to step 4. Otherwise the modified subproblem is solved exactly as in step 6. Note that k and m' remain unchanged and that this subproblem can never be a *complete integer solution*. If $T_{LB}(k) < T_u$ the modified subproblem is stored back into the tree, otherwise it is removed from the tree. Go to step 4.

8. The current best solution is the optimal solution. The optimal cycle time T_{opt} is equal to T_u and the optimal variables $x^{(opt)}_{ij}$ are equal to $x^{(u)}_{ij}$. Stop.

8 Experimental results and conclusions

The algorithm was implemented in a program and tested on many different cases. It is typical for the problem (1)–(3) that there are often many equivalent optimal solutions. One of the 10 optimal solutions of the example given in the Table 1 is presented in the Table 2. It took less than 0.01 seconds of computer time (on a 2.4 GHz Pentium 4) to find all 10 optimal solutions.

The computing time depends not only on the number of variables x_{ij} but also on the problem parameters t_{ij} and especially s_i. The lower values of s_i obviously make the search space smaller and reduce the computation time. Experiments have shown that for the problem parameters similar to those in the Table 1 all optimal solutions are typically found within a minute of computing time if the number of variables x_{ij} is 60 or fewer. For example, the 3-machine case from the Table 1 in which the number of different component types per board is increased to 20, takes less than a second to find all optimal solutions. This time increases to almost 2 hours if an additional machine is added (giving a problem with $4 \times 20 = 80$ variables x_{ij}). It should be noted, however, that for this example a suboptimal solution that is within 0.1% of the optimum was found after less than 0.1 second. This behaviour is typical for the branch-and-bound based algorithms where a great amount of time is often needed to prove the optimality of a solution that was found early in the process.

The algorithm was also tested on problems with a much larger number of variables x_{ij}. Cases with up to 10 machines and up to 100 different component types per board (giving up to 1000 variables x_{ij}) were tried. Because of the exponential nature of the algorithm the optimal solution is not found and/or proved optimal in a reasonable computing time for problems this large. But the algorithm is useful even in such cases — the branching strategy ensures that many good near-optimal solution are obtained. In addition, the algorithm gives a global lower bound on the optimal solution which allows the user to determine how close to the best possible solution a near-optimal solution is. The global lower bound on the optimal solution is the lowest of the $T_{LB}(k)$ in the branch-and-bound tree and is obtained in step 4 of the algorithm. It can be used to decide if a near-optimal solution is sufficiently close to the optimum and also if it is worth trying the longer computing time.

Acknowledgment

The authors would like to thank Prof. B. Vilfan for providing the formal proof of *NP*-completeness for the problem (1)–(3).

References

[1] P. Ji, Y.S. Wong, H.T. Loh, L.C. Lee, "SMT production scheduling: A generalized transportation approach," *International Journal of Production Research*, vol.32 (10), pp.2323–2333, 1994.

[2] J.C Ammons, M. Carlyle, L. Cranmer, G. Depuy, K. Ellis, L.F. Mcginnis, C.A. Tovey, H. Xu, "Component allocation to balance workload in printed circuit

card assembly system," *IIE Transactions*, vol.29 (4), pp.265–275, 1997.

[3] A. Schtub, O.Z. Maimon, "Role of similarity measures in PCB grouping procedure," *International Journal of Production Research*, vol.30 (5), pp.973–983, 1992.

[4] J. Sohn, S. Park, "Efficient operation of a surface mounting machine with a multihead turret," *International Journal of Production Research*, vol.34 (4), pp.1131–1143, 1996.

[5] Z. Ji, M.C. Leu, H. Wong, "Application of linear assignment model for planning of robotic printed circuit board assembly," *ASME Manufacturing Processes and Material Challenges in Microelectronics Packaging*, vol.ADM-v131/EEP-v1, pp.35–41, 1991.

[6] M. Sadiq, T.L. Landers, G. Taylor, "A heuristic algorithm for minimizing total production time for a sequence of jobs on a surface mount placement machine," *International Journal of Production Research*, vol.31 (6), pp.1327–1341, 1993.

[7] Y.D. Kim, H.G. Lim, M.W. Park, "Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process," *European Journal of Operational Research*, vol.91 (1), pp.124–143, 1996.

[8] P. Ji, M.T. Sze, W.B. Lee, "A genetic algorithm of determining cycle time for printed circuit board assembly lines," *European Journal of Operational Research*, vol.128 (3), pp.175–184, 2001.

[9] P. Brucker, "*Scheduling algorithms*," Second Ed., Springer, pp.274–307, 1998.

[10] B. Vilfan, "NP-completeness of a certain scheduling problem," (in Slovenian) *Internal report*, University of Ljubljana, Faculty of Computer and Information Science, June 2002.

[11] D.M. Kodek, "A theoretical limit for finite wordlength FIR digital filters," *Proc. of the 1998 CISS Conference*, vol. II, pp.836–841, Princeton, March 20-22, 1998.

[12] D.M. Kodek, "An approximation error lower bound for integer polynomial minimax approximation," *Electrotechnical Review*, vol.69 (5), pp.266–272, 2002.

[13] C.H. Papadimitrou and K. Steiglitz, "*Combinatorial optimization*," Prentice-Hall, pp.433–453, 1982.

[14] E. Horowitz, S. Sahni, "*Fundamentals of computer algorithms*," Computer Science Press, pp.370–421, 1978.

[15] V.F. Demyanov, V.N Malozemov, "*Introduction to minimax*," Dover, pp.113–115, 1990.

DoMosaic - Analysis of the Mosaic-like Domain Arrangements in Proteins

David T. Gerrard[1] and Erich Bornberg-Bauer[2]
School of Biological Sciences, University of Manchester, UK
2.205 Stopford Building, Oxford Road, M13 9PT, Manchester, UK; ebb@bioinf.man.ac.uk

Keywords: sequence analysis, domain evolution, data visualisation

Received: June 15, 2002

Sequence analysis is widely used to infer function from one protein sequence to another. One of the remaining hurdles in pushing further the limits of efficient usage is the modular and mobile architecture of proteins. Although many resources provide precompiled signatures, they are not yet used in conjunction with pairwise comparisons to investigate the modular architecture and phylogeny.
We present a program, doMosaic, which combines existing domain definitions with an adapted sequence alignment algorithm. It is based on the Smith-Waterman scheme, can be combined with trees derived from the domains and provides a user-friendly graphical interface. The method enables fast and efficient analysis of domain duplication events within one or in-between two sequences. Therefore, it enables refined functional annotation.

1 Introduction

Sequence data are a major resource for inferring molecular function by comparative analysis. Scoring schemes quantify the similarity between two genes or proteins, and algorithms compute the best alignment. This well established procedure is based on a reasonable model of molecular evolution. The first algorithms were introduced to molecular biology roughly 25 years ago and followed the recursive programming scheme which worked basically in $\mathcal{O}(n \times m)$ in time plus some overhead for the backtracking, typically of complexity m. More sophisticated and popular algorithms use statistics of tuple-frequencies in combination with a recursive principle or alone and trade off speed against selectivity [10]. An interesting visualisation tool is Dotter [23]. It was designed to display high scoring local alignments of proteins based on the dot plot idea [17].

To ease searching and browsing through the huge datasets many tools were developed to group proteins into functionally related "families" by clustering based on sequence similarity. This has been accomplished using full length or local comparison of regions, with or without demanding transitivity, splitting etc. [6, 11, 12, 15, 16, 19] The main hurdle in this context appears to be the irregular, domain-wise architecture of proteins. While proteins evolve, certain regions within a protein are functionally more important than others and thus remain better conserved while interspersed with regions that evolve more rapidly and have many gaps and insertions. These conserved substrings are generally known as domains if defined by structural units or motifs when characterised by sequence patterns which re-occur across different proteins. Often, the same feature of a sequence will be recognised as both a domain and a motif, but this is not always the case. However, since the methodology presented in the following works at the sequence level for both (structural) domains and (sequence) motifs, we use the word domain to denote both entity types. Searching for domains alone reduces the complexity of functional inference but depends on the reliability and availability of precompiled resources. Such searchable resources are provided for motifs using weighted matrices [5, 8, 14] or signatures of patterns [3, 4].

Domains may evolve at different speeds in different organisms and proteins, they can excise and incise into other proteins and sometimes other organisms (lateral gene/domain transfer). Because of various reordering processes domains are also highly mobile within their hosting protein: they can duplicate ($ABC \rightarrow ABBC$), swap ($ABCD \rightarrow ACBD$), undergo circular permutations ($ABC \rightarrow ABCABC \rightarrow BCA$) and so on. Consequently, the functional definition and inference by sequence similarity can become a very involved task when hierarchical clustering and canonical classifications are rendered useless.

Several groups have begun to take pragmatic approaches by using an existing tree, e.g. a species tree, as a "scaffold". By annotating the nodes and leaves of this scaffold with domain information it becomes possible to illustrate the most likely domain evolution. Thus, functional relationships which may remain hidden by simple sequence comparison or domain detection may be revealed.
TreeWiz uses a (species) tree as the underlying scaffold, displays sequences next to the leaves and allows the user to browse trees of several tens of thousands leaves interactively on a standard PC [22]. NIFAS, a tool building on Pfam, helps to analyse the trees and mobility of certain do-

[1]Present address: School of Biosciences, University of Birmingham, B15 2TT, UK; email: dtg124@bham.ac.uk.
[2]Correspondence: Phone/Fax: +44-161-275-7396/5082
Preprint version as of September 26, 2005. Final version to appear in *Informatica, special issue on Bioinformatics*(17),2002.

mains [25]. It generates a tree based on the phylogeny of one chosen domain and displays the resulting tree. The domain arrangement for every protein is displayed at the corresponding leaf. Thus, proteins hosting the domain under scrutiny in multiple copies will appear at all respective leaves.

We are currently developing algorithms with user-friendly graphical front ends to investigate the nature of domain architecture. Here we present a tool, based on variations of existing algorithms, which allows a quick and easy representation of major domain rearrangements between homo- and heteromeric paralogous and orthologous proteins. [1]

2 Methods and Application

In an ideal world, one would want to produce the most parsimonious tree in terms of duplications, insertions, losses, fusions, fissions etc. However, as there are far too many variables associated to these move sets (contrary to normal sequence alignments), these events can not be properly quantified. Therefore, one has to resort to a combination of approaches which, in our case, is the sequence alignment and the associated trees which can be produced from the pairwise similarities.

Algorithms: To make use of both the precompiled and fairly reliable motif resources and the pairwise comparison of full length sequences, doMosaic first brakes down the protein into the regions which are already annotated. Currently, the program requires SWISS-PROT entry format to scan in the feature tables correctly. Next, for each pair of domains which can be formed between both sequences, a full Smith-Waterman algorithm with affine gap-penalties is run at the sequence level. This procedure can be performed again for each single sequence against itself. Results are then displayed on a grid with size of filled squares proportional to the alignment score.

The raw scores generated by the Smith Waterman algorithm increase with the lengths of the sequences being aligned. The degree of length dependency varies with the alignment parameters (gap penalties, matrices) and with the residue frequencies in the amino acids. Empirical data suggest that the relationship lies somewhere between linear and logarithmic (see [1] for references). The inferences of doMosaic depend on the relative similarities of related domains which are normally of similar length. Therefore, the main point here is not so much do distinguish between true and false positives (or negatives) but rather to find a transformation that is fast and removes a significant portion of the length bias in the majority of cases.
The maximum local alignment score S from each comparison is divided by $ln(mn)$. So that doMosaic could filter out low scoring comparisons, significance cut off values were derived for each BLOSUM matrix from 300 comparisons of random sequences between lengths 50 and 500. Amino acid frequencies were taken from McCaldon and Argos (1998) . The value was chosen such that 99% of random comparisons fell below this cut off score (once transformed as above). The cut-offs were 5.0, 4.0 and 6.0 for the BLOSUM45, BLOSUM62 and BLOSUM80 matrices respectively. The cut off value is then subtracted from the length normalised score and 1.0 is added. If the value is still below 1.0, it is set to 1.0 so that in the next step, it will become 0. The natural log of the value is taken. This reduces to zero all comparisons that scored less than or equal to the cut-off value. The value is divided by 5 to give a value in the majority of biological cases between 0 and 1. Some long and very similar domains (e.g. vWA domains) will still score over 1.0. It is then multiplied by 100 for use by doMosaic in sizing the individual tiles between 0 and 100%
The normalisation could certainly be improved by taking into account the distribution of scores between real domain alignments or p-value statistics. However, in all examples the heuristics as described appear to work well enough to discriminate related domains from random similarities.

All pairwise domain alignments can be analysed using a neighbour-joining tree. This will group domains which have arisen from an internal duplication event closer together than domains of one kind which have been in linear order over a longer evolutionary period after, e.g., a more ancient duplication event. Domains which have been incised will also appear far off. Care must be taken if paralogous sequences are compared as will be seen in the following examples.

Applications: Figure 1 shows the domain-wise comparison of two paralogous human cadherin proteins. CADH_HUMAN is obtained from liver and intestines and has five consecutive cadherin domains ($A1, A2, ..., A5$) out of seven consecutive domains annotated in total ($A1, ..., A7$). CAD4_HUMAN is retinal, has seven consecutive cadherin domains ($B1, B2, ..., B7$) out of eight annotated domains in total ($B1, ..., B8$). $B1$, the first domain of CAD4_HUMAN has highest similarity to $A1$ and $A3$, the first and third domain of CADH_HUMAN and $B2$, the second domain of CAD4_HUMAN has highest similarity to the second and fourth domain of CADH_HUMAN. Figure 2 shows a tree derived from all pairwise domain-comparisons between these two proteins and a graphical illustration of an evolutionary model of domain duplication and insertion. Cadherins are commonly thought to have arisen by subsequent duplication events [13] but it is not clear if such an event has occurred before or after a paralogous duplication or a speciation event (leading to two orthologous sequences), if all have been duplicated one after another etc. The tree is difficult to decipher and some more ancient events as well as the recent domain additions can only be guessed. The necessity to add other paralogs and orthologs,

[1] Two genes are paralogs if they have arisen from a common ancestor and occur in the same organism. Orthologs are also descended from a common ancestor but appear in different organisms.

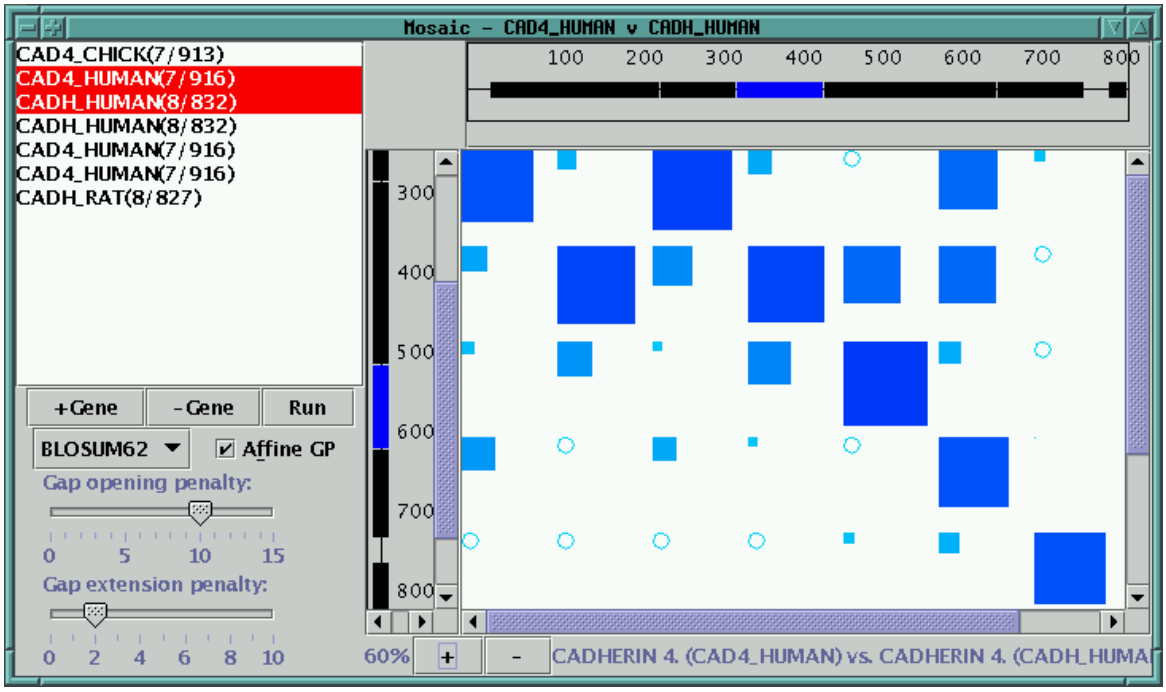


Figure 1: Comparison of CADH_HUMAN (North side) and CAD4_HUMAN (West side) using doMosaic. Each square is placed in a cell (all cells are of equal size) and denotes a domain for which a similarity value above the threshold has been found. The bigger and darker a square is, the more similar two domains are. Circles denote empty squares (similarity below threshold). The relative length and order of domains can be read from the bars on top and left from the panel. Upon mousing over a cell, information about the corresponding proteins is shown in a little window which pops up. Also, the corresponding domains in the north and west bars are high-lighted such that the relative orientation in each of the proteins becomes obvious even when the grid is dimensioned such that only a small part of all the cells are shown. Since the size of the cells has been computed from similarities within a limited range, cells can never "bump" into each other.

possible intermediate sequences, the inability to distinguish between domain loss and adding and so forth would make the tree even more involved. However, even without resorting to such a tree, doMosaic immediately gives a clear answer for these two paralogous proteins: because of conserved order, proximity and similar levels of homology, domains $B1$ and $B2$ from CAD4_HUMAN have most likely been added together to a precursor with the "classical" five-cadherin architecture in a fairly recent duplication event. Taking into account a few more CADH entities from other vertebrates, suggests that this duplication event has most likely occured only relatively recently, i.e., shortly before mammal speciation (data not shown).

Example 2 in figure 3 shows the self comparison of TRIO_HUMAN, a hetero-multimeric protein. Again, the co-duplication of the spectrin repeats (domains $1, 2, 4$ and 5 where 2 and 4 are only interspersed by a low-complexity region of a poly-gln) is apparent and so is the probable co-duplication of domains 6 and 7. Although the spectrin family is well studied [20] the strong conservation which will most likely have arisen from one single multiplication event has not been reported as yet.

GUI: doMosaic appears essentially as a GUI which allows changing parameters, loading and eliminating proteins from a list etc. It is possible to adjust screen size, gap penalties and choose the mutation matrix. Mousing over domains shows score, name of domains and highlights the associated domains in the string representations in the West and North margins of the main display window. Both size and colour intensity of cells in the grid indicate similarity score.

Performance: depends obviously on the length and number of domains but for typical applications a run does not take more than a few, mostly less than 2 real time seconds on a 700MHz Intel pentium III. Memory requirements are negligible.

Implementation status: To obtain platform independency, all parts have been programmed in JAVA and the version, as depicted on Figures 1 and 3 can be obtained from the authors (DTG) or our web-site (www.bioinf.man.ac.uk). A revised version with the tree generation routine fully integrated is under construction.

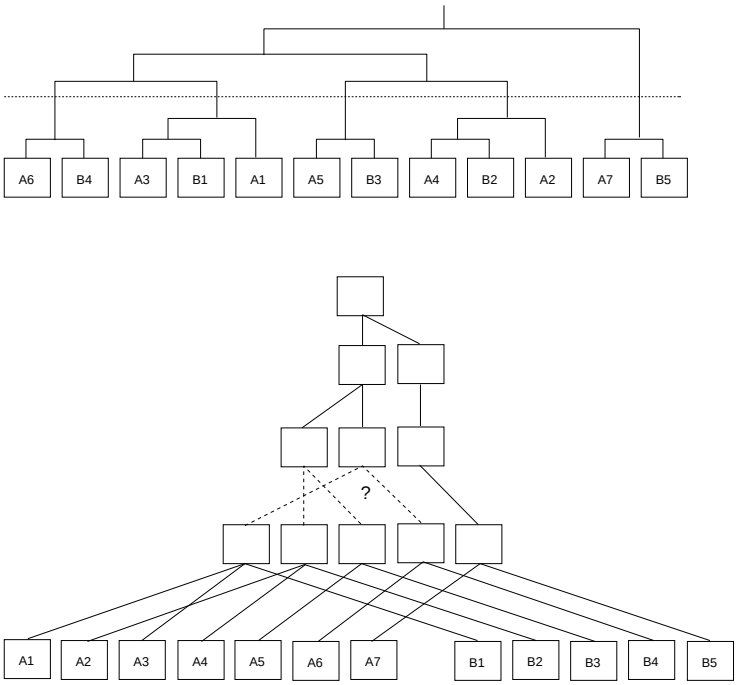


Figure 2: Top: The neighbour joining tree of all domains which were compared by doMosaic during the comparison between the two cadherins (CADH_HUMAN: A and CAD4_HUMAN: B) and from the self-comparison of each cadherin in Figure 1. All events above the dotted line (earlier events, closer to the root) probably have happened within one, ancestral protein, before the full-gene duplication into the two proteins. Bottom: The most probable flow of domains as obtained from the tree. At the question mark the situation is unclear as the order of the domains seems to be not conserved. Probably there have been more events which could only be reconstructed with the help of even more family members and/or more sophisticated tree programs.

3 Discussion and Conclusion

Since new experimental techniques are introduced, biological data rapidly accumulate and diversify and so do tools and methods to analyse these data. Many of these focus on specialised areas and help to gain qualitatively new insights, which in turn stimulate experimentalists to generate more data and new challenges for the bioinformatician. Analysing domain evolution and mobility is such a specialised area which has recently become an important issue since the increasing amount of sequence data requires more specialised tools to push the limits of functional inference further. Although the mobility of domains has been known of for a long time [9], it was only recently that it became apparent how complex a problem this imposes on their study [2, 7, 27]. [2]

Recent attempts to generate tools which provide further insight into the relationships between sequences have mostly focused on the sequence level. Dotter [23] is similar in spirit to doMosaic and displays the full $n \times m$ alignment of two proteins such that traces of all significant suboptimal alignments become visible. The domain arrangements for both sequence can be displayed next to the matrix similar to the two bars in doMosaic (see Figure 1). However, similarities between domains are not displayed. Therefore, evolutionary events such as duplications do not become directly obvious. NIFAS [25], displays the phylogenetic tree and domain arrangements of more than two sequences but does not allow the direct comparison of two sequences. Several other tools focus on the phylogenetic relationship, in particular of paralogy and orthology of full sequences [21, 24, 26].

However, these tools do no enable the direct and quantitative analysis of domain duplication events. This is what doMosaic does: domain rearrangements in homo- and heteromultimeric proteins can be compared and assigned to their origin. doMosaic is of course not a primary analysis tool, but it is particularly well suited for a quick analysis of domain based rearrangement events.

We plan to further develop the program such that it integrates with TreeWiz [22] and includes features from NIFAS.

[2] To illustrate the problem of domain mobility consider a graph G=(V,G). Let domains be vertices V_i, where each kind of a domain, e.g., all identified EGF-like domains or the p-kinase domains are represented by one vertex, (V_1) and (V_2) respectively. Let the set of all sequences which link the vertices V_i, V_j for a pair of two different domains be denoted by edge E_{ij}, irrespective of the order in which i and j occur in any of the sequences and whether other domains may also occur in that string. Then the graph has a small-world structure with scale free character and almost all domains appear in a single, giant, connected component. This holds for a variety of motif databases and organisms, [7, 27], for structural domains [2] and even for simplified model systems [7].

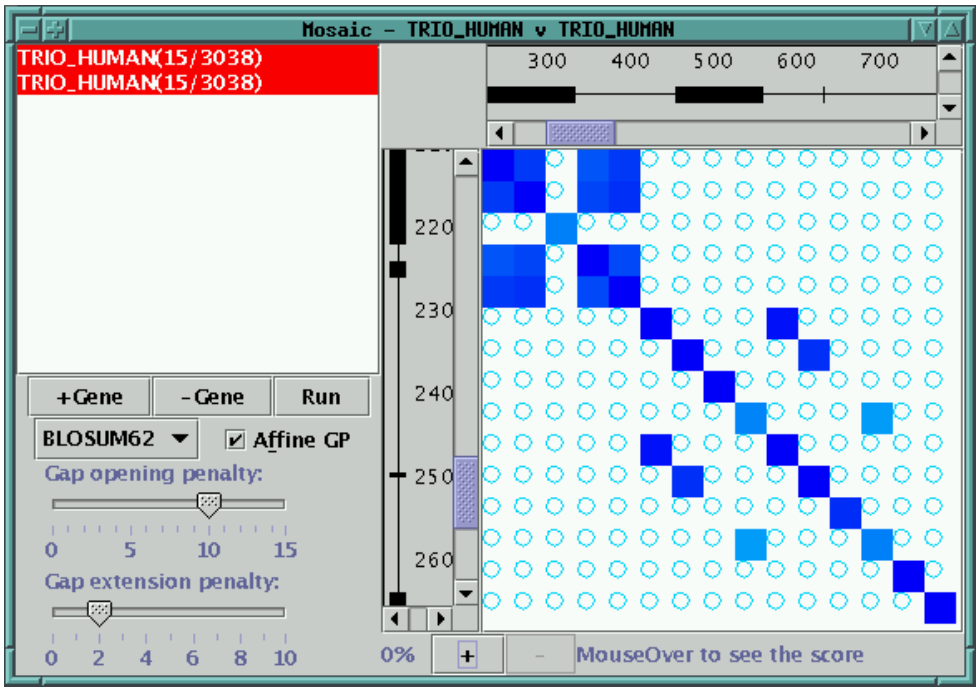


Figure 3: Self comparison of TRIO_HUMAN using doMosaic. The plot is symmetric because when comparing two domains transitivity holds.

Acknowledgements: We thank Julian Selley for technical assistance. EBB gratefully acknowledges support through an MRC international recruitment grant. We thank the referees for careful reading and useful comments which helped to improve the manuscript.

References

[1] S. F. Altshul, R. Bundschuh, R.Olsen, and T. Hwa. The estimation of statistical parameters for local alignment score distributions. *Nucleic Acids Res.*, 29:351 – 361, 2001.

[2] G. Apic, J. Gough, and S. A. Teichmann. Domain combinations in archaeal, eubacterial and eukaryotic proteomes. *J. Mol. Biol.*, 310:311–324, 2001.

[3] T. Attwood, M. Croning, D. Flower, A. Lewis, J. Mabey, P. Scordis, J. Selley, and W. Wright. PRINTS-S: the database formerly known as PRINTS. *Nucleic Acids Res.*, 28:225–227, 2000.

[4] A. Bairoch and B. Boeckmann. The SWISS-PROT protein sequence data bank, recent developments. *Nucleic Acids Res.*, 21:3105 – 3109, 1993.

[5] A. Bateman, E. Birney, R. Durbin, S. R. Eddy, K. L. Howe, and E. L. Sonnhammer. The Pfam protein family database. *Nucleic Acids Res.*, 28:263–266, 2000.

[6] E. Bolten, A. Schliep, S. Schneckener, D. Schomburg, and R. Schrader. Clustering protein sequences – structure prediction by transitive homology. *Bioinformatics*, 17:935 – 941, 2001.

[7] E. Bornberg-Bauer. Randomness, structural uniqueness, modularity and neutral evolution in sequence space of model proteins. *Z. Phys. Chem.*, 216:139 – 154, 2002.

[8] F. Corpet, F. Servant, J. Gouzy, and D. Kahn. ProDom and ProDom-CG: tools for protein domain analysis and whole genome comparisons. *Nucleic Acids Res.*, 28:267 – 269, 2000.

[9] R. F. Doolittle and T. L. Blundell. Sequence and topology: Unity and diversity all over again. *Curr. Opn. Struct. Biol.*, 3:377 – 378, 1993.

[10] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[11] A. J. Enright and C. A. Ouzounis. Gene RAGE: a robust algorithm for sequence clustering and domain detection. *Bioinformatics*, 16:451 – 457, 2000.

[12] J. Freudenberg, R. Zimmer, D. Hanisch, and T. Lengauer. A hypergraph-based method for unification of existing protein structure- and sequence-families. *In Silico Biology*, 2001. http://www.bioinfo.de/isb/2001/02/0031

[13] B. Geiger and O. Ayalon. Cadherins. *Ann. Rev. Cell and Dev. Biol.*, 8:307 – 332, 1992.

[14] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci., USA*, 89:10915–10919, 1992.

[15] A. Krause, P. Nicodeme, E. Bornberg-Bauer, M. Rehmsmeier, and M. Vingron. WWW-access to the SYSTERS protein sequence cluster set. *Bioinformatics*, 15:262 – 263, 1999.

[16] M. Linial, N. Linial, N. Tishby, and G. Yona. Global self-organisation of all known protein sequences reveals inherent biological signatures. *J. Mol. Biol.*, 268:539 – 556, 1997.

[17] J. V. Maizel and R. P. Lenk. Enhanced graphical matrix analysis of nucleic acid and protein sequences. *Proc. Natl. Acad. Sci., USA*, 78:7665 – 7669, 1981.

[18] P. McCaldon and P. Argos. Oligopeptide biases in protein sequences and their use in predicting protein coding regions in nucleotide sequences. *Proteins: Structure, Function and Genetics*, 4:99 – 122, 1988.

[19] J. Park and S. A. Teichmann. DIVCLUS: an automatic method in the GEANFAMMER package that finds homologous domains in single- and multi-domain proteins. *Bioinformatics*, 14:144 – 150, 1998.

[20] J. Pascual, J. Castresana, and M. Sarraste. Evolution of the spectrin repeat. *BioEssays*, 19:811 – 817, 1997.

[21] M. Remm, C. E. Storm, and E. L. Sonnhammer. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J. Mol. Biol.*, 314:1041–1052, 2001.

[22] U. Rost and E. Bornberg-Bauer. TreeWiz: interactive exploration of huge trees. *Bioinformatics*, 18:109 – 114, 2002.

[23] E. L. Sonnhammer and R. Durbin. A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis. *Gene*, 167:1–10, 1996.

[24] E. L. Sonnhammer and J. C. Wootton. Integrated graphical analysis of protein sequence features predicted from sequence composition. *Proteins*, 45:262 – 273, 2001.

[25] C. Storm and E. L. Sonnhammer. NIFAS: visual analysis of domain evolution in proteins. *Bioinformatics*, 17:343–348, 2001.

[26] C. E. Storm and E. L. Sonnhammer. Automated ortholog inference from phylogenetic trees and calculation of orthology reliability. *Bioinformatics*, 18:92 – 99, 2002.

[27] S. Wuchty. Scale-free behaviour in protein domain networks. *Mol. Biol. Evol.*, 18:1694 – 1702, 2001.

Introduction:

Bioinformatics Tools and Applications

Dear Readers,

Advances in computing have traditionally been driven by demands in rapidly evolving scientific areas. Examples of research areas that recently have been enjoying a rapid growth are life sciences. This rapid growth has in turn led to a high demand for computation tools that support the management, interpretation and analysis of the data generated by life science research. The field of Bioinformatics aims at addressing this demand.

The revolution in the life sciences has led to the emergence of new and challenging applications. These complex applications are driving the need for new algorithms and tools to facilitate the access, analysis and interpretation of life science data. The focus of this special issue of the journal is on algorithms, systems, techniques and tools that facilitate the way life science data is collected, interpreted and retrieved.

In order to expose the readers of Informatica to the recent trends in Bioinformatics, this special issue of the journal presents some of the emerging complex life science applications. The papers included in this issue cover various topics such as the interoperability of distributed biological databases, protein functional analysis and gene clustering. These topics will continue to be important in facilitating new discovery and are expected to be the subject of many future research contributions.

The special issue starts with an article that focuses on the interoperability of geographically distributed and heterogeneous science databases. The paper offers a summary of some of the challenges facing the support of such interoperability and proposes a scalable approach that addresses this issue. In addition, the authors analyze the query execution of multidatabase queries and identify the performance limitations of these queries. Inferring the function of a protein using sequence data is an active area of research. The process is generally based on sequence similarity algorithms that establish the similarities between known sequences and unknown sequences. There are several previous software tools that address this topic.

The second paper in this issue describes a system that greatly improves on these systems by using different processing techniques for different types of regions of the proteins. Certain regions of the proteins are functionally more important than others and therefore tend to be better conserved. The proposed system uses information about these highly conserved regions to facilitate the functional analysis of proteins.

The third paper in this issue concentrates on an important area of Bioinformatics: gene clustering. Increased attention to gene clustering was due to the recent availability of high throughput microarray technology. This technology allows the measurement of gene expression data for thousands of genes and generates a large amount of expression data. Analyzing and interpreting this data can be difficult. To assist scientists in this process, the authors of the third paper of this issue propose an integrated approach to gene clustering. One of the innovative aspects of the proposed approach is that it is highly automated and generates high quality clustering result based on a dynamic validation technique.

The editors would like to thank the authors for their strong contributions to this special issue and the reviewers for their diligent review of the papers. We hope that the readers of Informatica will enjoy this issue and will find it valuable in their future research.

Editors of the Special Issue,

Johann Eder — eder@isys.uni-klu.ac.at

Omran Bukhres — bukhres@cs.iupui.edu

Mining and Validating Gene Expression Patterns: an Integrated Approach and Applications

Shin-Mu Tseng and Ching-Pin Kao
Institute of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.
Email: tsengsm@mail.ncku.edu.tw

Keywords: gene expression, microarray, data mining, clustering, validation techniques

Received: July 5, 2002

The microarray technique has been widely used in recent years since it can capture the expressions of thousands of genes in a single experiment. To meet the challenge of high volume and complexity of microarray data, various data mining methods and applications have been proposed for analysing gene expressions. Although numerous clustering methods have been studied, they can not provide automation, high quality and high efficiency simultaneously for the biologists during the analysis process. In this research, we propose an integrated approach that can analyse large volume of gene expression data automatically and efficiently. Our approach integrates efficient clustering algorithms with a novel validation technique such that the quality of the discovered gene expression patterns can be evaluated on the fly. Through practical implementation and applications on real gene expression data, our approach was shown to outperform other methods in terms of efficiency, clustering quality and automation.

1 Introduction

With the innovation of microarray technology [5, 16], the biological researchers can examine the expressions of thousands of genes simultaneously in a single experiment. This advances greatly the progress in exploring the real functions of various genes. In recent years, large amounts of gene expression datum have been generated by the biologists. Thus, there is a great need to develop effective analytical methods to analyze and to exploit the information contained in gene expression data. Since genes with related functions tend to have similar expression patterns, possible roles for genes with unknown functions can be suggested based on the known functions of some other genes that are placed in the same cluster. Therefore, it is an important research issue to analyze and interpret the gene expression data obtained via microarray experiments [4, 15]. The gene expression patterns obtained by analysing microarray data can then be used for a variety of inference tasks, like measurement of a gene's involvement in a particular cellular event or process [1, 17, 19], predict regulatory elements [3], etc.

Clustering of gene expression is one of the most important processes in analysing gene expression data. Clustering methods aim at detecting groups of genes that have similar expression patterns. Basically, a clustering algorithm partitions entities into groups based on the given features of the entities, so that the clusters are homogeneous and well separated. For gene expression analysis, the main algorithmic problem involved is to cluster multi-condition gene expression patterns. More specifically, the aim is to identify sets of genes that behave similarly across the conditions. Furthermore, the clustering results can be utilized to help understand functions of genes. For example, the function of a gene may be predicted based on the known functions of genes within the same cluster.

A variety of clustering methods have been proposed for mining gene expression data [2, 4, 6-11]. For example, the average-link hierarchical clustering algorithm was widely used to identify groups of co-regulated yeast genes. Ben-Dor *et al.* [2] reported success of applying CAST algorithm on gene expression analysis. Although a number of clustering methods have been studied in the rich literature, they incur problems in the following aspects: 1) Automation, 2) Quality, and 3) Efficiency. In the aspect of automation, most clustering algorithms request the users to set up some parameters for conducting the clustering task. For example, *k*-means [9] requires the user to input the number of clusters *k* to be generated. However, in real applications, it is difficult for a biologist to determine the right parameters manually for the clustering tasks. Hence, an automated clustering method is required. In the aspect of quality, an accurate and efficient validation method is lacked for evaluating the quality of the clustering results. Consequently, it is difficult to provide users with the information regarding how good the clustering result is. Finally, in the aspect of efficiency, the existing clustering algorithms may not perform well when the optimal or near-optimal clustering

result is required from the global view.

In this paper, we propose an integrated approach for mining multi-condition gene expression and validating the clustering results. This approach integrates the density-based clustering method with the validation techniques to provide automation and accuracy for the clustering. Furthermore, an iterative computing process is adopted to reduce the computation in clustering such as to meet the requirement of efficiency. The approach is implemented and applied on real gene expression data, and it is shown to deliver higher efficiency, clustering quality and automation than other methods.

The rest of the paper is organized as follows: In Section 2, some related literatures are introduced; Our approach is described in section 3; Applications of the proposed method on analysing gene expression data is demonstrated in Section 4; the conclusion and future work is made in Section 5.

2 Related Work

In recent years, the biologists can produce large amounts of gene expression datum rapidly through the microarray experiments, which can be divided into two categories. The first types of microarray experiments are to monitor the expressions of a set of genes under a series of varied conditions; the second type of microarray experiments aim at observing the expressions of genes under a same environment but from different cells. The data generated from first type of experiments can be used to detect the trends and regularities of a gene under a series of conditions, while the data from the second type of experiments can provide information about the classifications of genes. In this research, we focus on the first type of gene expression data.

To analyse gene expression data effectively, a number of clustering methods were proposed [2, 4, 6-11, 21]. They can be classified into several different types: partitioning-based methods (like *k*-means [9]), hierarchical methods (like Hierarchical Agglomerative Clustering), density-based methods (like CAST [2]), model-based methods, etc. *k*-means partitions the dataset into *k* groups primarily based on the distance between data items, where *k* is a parameter specified by the user. Hierarchical clustering methods have been applied extensively and shown to be valuable on analyzing gene expression patterns. For example, hierarchical clustering can be used to separate normal and tumor tissues and to differentiate tumor types based on gene expression patterns in each tissue. Self-Organizing Maps were used by Tamayo et al. [7] for advanced gene expression analysis. CAST (Cluster Affinity Search Technique) takes as input a parameter named *affinity threshold t*, where $0 < t < 1$, and tries to guarantee that the average similarity in each generated cluster is higher than the threshold t. The main advantage of CAST is that it can detect the outliers more effectively and it executes efficiently. In [8], a detail survey was made on the main characteristics and applications of various clustering algorithms, which were also classified into different categories including portioning, hierarchical, density-based, grid-based, fuzzy clustering, etc.

Although a number of clustering algorithms have been proposed, they may not find the best clustering result efficiently and automatically for the given microarray dataset. To find the best clustering result, an important problem involved is how to validate the quality for some clustering result generated by a clustering algorithm. Jain and Dubes [9] divided cluster validation procedures into two main categories: external and internal criterion analysis. External criterion analysis validates a clustering result by comparing it to a given "standard" which is another partition of the data objects. In contrast, internal criterion analysis uses information from within the given data set to represent the goodness of fit between the input dataset and the clustering result.

There are many statistical measures that assess the agreement between an external criterion and a clustering result. For example, Milligan *et al*. [13, 14] evaluated the performance of different clustering algorithms and different statistical measures of agreement on both synthetic and real data. In [8], a number of well-know validity criteria and representative measuring indices were studied further with detail empirical evaluations. The problem of external criterion analysis is that reliable external criteria are rarely available when analysing gene expression data. Therefore, some new measures were proposed for the internal criterion analysis. For example, compactness and isolation of clusters are possible measures of goodness of fit. A measure named Figure of Merit (FOM) was used by Yeung *et al*. [20] for evaluating the quality of clustering on a number of real gene expression datasets.

The main drawback of the existing methods for analysing gene expression pattern is that they can not meet the requirements of automation, high quality and high efficiency at the same time during the analysis process. This motivated this research in designing a novel approach that integrates clustering and validation techniques for mining gene expression such that automation, high quality and high efficiency can be met simultaneously.

3 Proposed Approach

In this section, we first describe the definition of the problem, then we present the details of our approach, including the principles and an approximation method for reducing the computations.

3.1 Problem Definition

The objective of clustering methods is to discover significant groups existed in a dataset. The problem of gene expression clustering can be described briefly as follows. Given a set of m genes with unique identifiers, a vector $E_i = \{E_{i1}, E_{i2}, \ldots, E_{in}\}$ is associated with each gene i, where E_{ij} is a numerical data that represents the response of gene i under condition j. The goal of gene expression clustering is to group together genes with similar expressions over the all conditions. That is, genes with similar corresponding vectors should be classified into the same cluster.

3.2 An Integrated Approach

The main ideas of the proposed approach are as follows. Given a gene expression data, the first step of our approach is to calculate a similarity matrix S in which the entry S_{ij} represents the similarity of the expression patterns for genes i and j. Although a number of alternative measures could be used for calculating the similarity between gene expressions, we use Pearson's correlation coefficients [9] here for its wide application. Note that a similarity matrix needs to be computed and generated only once given a gene expression data. This reduces a lot of computation overhead as incurred by some clustering algorithms that calculate the similarities dynamically.

In the second step, a density-and-affinity based algorithm is applied as the base clustering algorithm. With a specified input parameter, the base clustering algorithm utilizes the similarity matrix S to conduct the clustering task. Thus a clustering result will be produced by the base clustering algorithm based on the given input parameter. A good candidate for the base clustering algorithm is CAST (Cluster Affinity Search Technique) [2], since it generates a clustering result very quickly based only on the value of an input parameter named *affinity threshold t*, where $0 < t < 1$.

In the third step, a validation test is performed to evaluate the quality of the clustering result produced in step two. We adopt *Hubert's Γ statistic* [9] for measuring the quality of produced clustering. Let $X=[X(i, j)]$ and $Y=[Y(i, j)]$ be two $n \times n$ matrix where $X(i, j)$ indicates the similarity of genes i and j, $Y(i, j)$ is defined as follows:

$$Y(i, j) = \begin{cases} 1 & \text{if genes i and j are in same cluster,} \\ 0 & \text{otrwisehe} \end{cases}$$

Hubert's Γ statistic represents the point serial correlation between the matrix X and Y, and is defined as follows:

$$\Gamma = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\frac{X(i,j) - \overline{X}}{\sigma_X} \right) \left(\frac{Y(i,j) - \overline{Y}}{\sigma_Y} \right)$$

where $M = n (n - 1) / 2$ and Γ is between [-1, 1]. Let matrix X be the similarity matrix derived from the gene expression data, matrix Y and *Hubert's Γ statistic* can be calculated easily without much computation overhead. For a clustering result, a higher value of Γ represents the better clustering quality.

With the above steps, it is clear that a good clustering with high quality can be obtained by applying a number of different values for the *affinity threshold t* as input parameters to the CAST algorithm, calculating the *Hubert's Γ statistic* of each clustering result respectively, and choosing the one with the highest value of *Hubert's Γ statistic* as the output. In this way, a local-optimal clustering result may be provided for the users automatically. For example, as shown in Figure 1, the X axis represents the values of *affinity threshold t* input to CAST and the Y axis shows the obtained *Hubert's Γ statistic* for each of the clustering result. The highest peak in the curve corresponds to the best clustering result, which has *Hubert's Γ statistic* value around 0.52 occurred when t is set as 0.25.

In fact, this approach is feasible in practical applications for the following reasons:

1. Once the similarity matrix of the gene expressions was generated at the beginning of execution, CAST executes very fast.
2. The computation of *Hubert's Γ statistic* for each clustering result is easy, too. So the extra computation overhead in doing quality validation will be acceptable.

However, one problem incurred in the above simple approach is how to determine the best value for the *affinity threshold t*. The easiest way is varying the value of *affinity threshold t* with a fixed increment and iterating the executions of CAST by feeding in the series of values as parameter repetitively. For example, we may vary the values of t from 0.05 to 0.95 in steps of 0.05, as shown in Figure 1. For each clustering result, its quality will be measured by using *Hubert's Γ statistic* and the one with the highest measured quality is selected as the best result. We call this approach *CAST-FI* (*Fixed Increment*) in the following discussions. The main disadvantage of *CAST-FI* is that many iterations of computation are required. Therefore, an approximation method will be described for reducing the computation overhead in the next section.

t	0.05		0.20	0.25	0.30	0.35		0.90	0.95
Γ statistic	0.04		0.51	0.52	0.51	0.47		0.06	0.04

⇩ **Best Clustering result**

Figure 1. *Hubert's Γ statistic* vs. values of t.

3.3 Approximation Method

The idea behind the approximation method is to reduce the computations by eliminating unnecessary executions of clustering such as to obtain a "nearly-best" clustering result instead of the optimal one. That is, we try to make the times of executing CAST as less as possible. Therefore, we need to narrow down the range of the parameter *affinity threshold t* effectively. The proposed method works as follows:

1. Initially, a testing range R for setting *affinity threshold t* is set as [0, 1]. We divide R equally into m parts by the points $P_1, P_2, \ldots, P_{m-1}$, where $P_1 < P_2 < \ldots < P_{m-1}$, m□3. Then, the value of each of P_i is taken as the *affinity threshold t* for executing CAST and the *Γ statistic* of the clustering result for each of P_i is calculated. We call this process a "run".

2. When a run of executing the clustering is completed, the clustering at point P_b that produces the highest *Γ statistic* is considered as the best clustering. The testing range R is then replaced by the range $[P_{b-1}, P_{b+1}]$ that contains the point P_b.

3. The above process is repeated until the testing range R is smaller than a threshold δ or the difference between the maximal value and minimal values of the quality is smaller than another threshold σ.

4. The clustering result with the best quality during the tested process is output as the answer.

In this way, we can obtain the clustering result that has a "nearly-best" clustering quality with much less computation. In the next section, through empirical evaluation, we shall evaluate how good the generated clustering result is and to what extent the computations could be reduced by our approach.

4 Applications on Gene Expression Analysis

To validate the feasibility and performance of the proposed approach, we implement the proposed approach in C++ and apply it for analyzing gene expression data. We describe the experimental setup in Section 4.1 and the detailed experimental results on different types of data in Sections 4.2, 4.3 and 4.4.

4.1 Design of Experiments

To evaluate the performance of our approach, we use the microarray expression data of yeast saccharomyces cerevisiae obtained from Lawrence Berkeley National Lab (LBNL) (http://rana.lbl.gov/EisenData.htm). The dataset contains the expressions of 6221 genes under 80 experimental conditions. Based on this dataset, we generate two datasets with different properties for testing. For the first dataset (further named dataset A), we choose 2000 genes from the original dataset randomly. The average similarity of dataset A is 0.137 by using Pearson's correlation coefficient as measurement of similarity. Thus Dataset A represents a low-similarity dataset. Then, in order to generate a dataset with higher similarity, we retrieve a number of genes with high similarity from the original dataset and duplicate these gene expression patterns to generate a dataset of about 1900 genes. Additionally, 100 outliers were mixed with the 1900 genes to form Dataset B of about 2000 genes totally. The average similarity of Dataset B is about 0.696 and thus it represents a high similarity dataset.

We compare our approach with *CAST-FI* and the well-known clustering method, namely *k*-means [9]. For our approach, the parameters m, δ and σ are default as 4, 0.01 and 0.01, respectively. For *k*-means, the value of k is tested in two ways: 1) k is varied from 3 to 21 in step of 2, and 2) k is varied from 3 to 39 in step of 2, respectively. The quality of clustering results was measured by using Hubert's Γ statistic. The experimental results on dataset A and B are described in the following sections, respectively.

4.2 Results on Dataset A

The total execution time and the best clustering quality for the tested methods on Dataset I are listed in Table 1. The notation "*CAST-FI*" indicates the approach running CAST iteratively by varying *affinity threshold t* from 0.05 to 0.95 in fixed increment of 0.05, while the notation "Our Approach" indicates the one described in Section 3 using the proposed computation reduction method.

Table 1. Experimental results (Dataset A).

Methods	Time (sec)	Number of clusters	*Γ* Statistic
Our Approach	27	57	0.514
CAST-FI	246	57	0.514
k-means (*k*=3~21)	404	5	0.447
k-means (*k*=3~39)	1092	5	0.447

It is obvious that our approach and *CAST-FI* outperform *k*-means substantially in both of execution time and clustering quality. In particular, our approach performs 15 times to 40 times faster than *k*-means with k ranged as [3, 21] and [3, 39], respectively. In addition, the results also show that the highest *Γ statistic* value generated by our approach is very close to that of *CAST-FI*, meaning that the clustering quality of our approach is as good as *CAST-FI*. However, our approach is about 8 times faster than *CAST-FI*. Therefore, it is shown that our approach outperforms other clustering methods greatly no matter in quality or computation time.

Table 2 shows the distribution of clusters produced by each tested method. It is shown that *k*-means generated 5 clusters for the best clustering result, while the size of each cluster is ranged between 101 and 400. This phenomenon holds no matter *k* is varied from 3 to 29 or from 3 to 39. However, our approach produced 57 clusters for the best clustering result. In particular, it is clear that 4 main clusters are generated, with two clusters sized between 101 to 400 and another two sized between 401 to 600. Moreover, our approach also generates a number of clusters with small size (1~10 and 11~100), which are mostly outliers (or noise). This means that our approach is superior to *k*-means in filtering out the outliers from the main clusters. This can provide more accurate clustering result and insight for gene expression analysis.

Table 2. Distribution of produced clusters (Dataset A).

Cluster size / Methods	1~10	11~100	101~400	401~600
Our Approach	38	15	2	2
CAST-FI	38	15	2	2
k-means (*k*=3~21)	0	0	5	0
k-means (*k*=3~39)	0	0	5	0

The following observations were made from this experiment:

1. In terms of clustering quality, our approach and *CASI-FI* perform much better than *k*-means, especially in isolating the outliers. This means that the density-and-affinity based methods are superior to partitioning-based methods in clustering low-similarity gene expression data.

2. Our approach executes much faster than *CASI-FI* in discovering the best clustering result, while the resulted clustering quality is very close to that of *CASI-FI*. This illustrates the advantage of the approximation method for computing reduction as described in Section 3.3.

4.3 Results on Dataset B

We conducted the same experiments by replacing dataset A with dataset B, which represents a dataset with higher similarity. Table 3 and Table 4 show the experimental results of the tested methods and the distribution of cluster size under dataset B, respectively. The following observations were made from the empirical results:

1. It is obvious that our approach and *CAST-FI* outperform *k*-means substantially in terms of the clustering quality (Γ *statistic*). Compared to the experimental results on dataset A, the degree of improvement our approach performed over *k*-means in terms of the clustering quality is much higher. In fact, by observing the distribution of size in the generated clusters as shown in Table 4, we found that both our approach and *CAST-FI* produce a main cluster with large size (401-600) and many other small clusters, which are actually outliers. This matches the real distribution of dataset B as described in Section 4.1. In contrast, *k*-means partitions the large cluster into several clusters with uniform size. Consequently, the clustering result distracts with the original data distribution. This indicates that *k*-means can not perform well under high similarity dataset. In particular, it can not identify the outliers correctly.

2. In the aspect of execution time, again our approach is much faster than other methods. Compared to *CAST-FI*, our approach produces clustering quality as good as that by CAST-FI with much shorter execution time. This shows that our approach can still achieve high efficiency and accuracy under high similarity dataset.

Table 3. Experimental results (Dataset B).

Methods	Time (sec)	Number of clusters	Γ Statistic
Our Approach	13	63	0.833
CAST-FI	41	62	0.833
k-means (*k*=2~20)	77	12	0.309
k-means (*k*=2~38)	267	12	0.309

Table 4. Distribution of produced clusters (Dataset B).

Cluster size / Methods	1~10	11~100	101~400	401~600
Our Approach	62	0	0	1
CAST-FI	61	0	0	1
k-means (*k*=2~20)	4	5	3	0
k-means (*k*=2~38)	4	5	3	0

5 Conclusions and Future Work

An integrated approach for mining and validating gene expression patterns is proposed in this paper. The proposed approach can automatically and effectively cluster microarray data generated by multi-condition experiments. Through empirical evaluations on datasets with different degree of similarities, our approach was shown to achieve higher efficiency and clustering quality than other methods. Moreover, the proposed approach can discover the "nearly-best" clustering result without requesting the users to input parameters. Therefore, the proposed approach can provide high degree of automation, efficiency and clustering quality, which are

lacked in other clustering methods for mining gene expression data. Our approach can also be extended to the parallel and distributed systems for achieving higher performance in the future.

In the future, we will further explore the following research issues:

1. Reduce the initial range of input parameter, namely *affinity threshold t*, for executing CAST. This will significantly reduce the computation further once the correct range can be estimated initially.

2. Design a memory-efficient clustering method to be integrated with our iteratively clustering approach. This is especially useful when the number of tested genes in the microarry is large.

3. Extend our approach for the parallel and distributed system environment and evaluate its performance in terms of efficiency and accuracy under various system conditions like varied number of computing units, etc.

6 Acknowledgement

This research was partially supported by National Science Council, R. O. C., under grant NSC 90-2213-E006-132. We would also like to thank the referees for their precious comments and advices.

7 References

[1] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack & A. J. Levine (1999) Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, *Proc. Natl Acad. Sci. USA*, 96, 6745-6750.

[2] Amir Ben-Dor & Zohar Yakhini (1999) Clustering gene expression patterns. *Proc. of the 3rd Annual Int'l Conf. on Computational Molecular Biology (RECOMB '99)*.

[3] A. Brazma, I. Jonassen, J. Vilo, & E. Ukkonen (1998) Predicting gene regulatory elements in silico on a genomic scale. *Genome Research* 8, 1202-1215.

[4] Ming-Syan Chen, Jiawei Han, & Philip S. Yu (1996) Data mining: An Overview from a Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No.6.

[5] J. DeRisi, L. Penland, P. O. Brown, M. L. Bittner, P. S. Meltzer, M. Ray, Y. Chen, Y. A. Su & J. M. Trent (1996) Use of a cDNA microarray to analyze gene expression patterns in human cancer. *Nature Genetics* 14: 457-460

[6] Sudipto Guha, Rajeev Rastogi, & Kyuseok Shim (1998) CURE: An efficient clustering algorithm for large databases. *Proc. of ACM Int'l Conf. on Management of Data*, p. 73-84, New York.

[7] Sudipto Guha, Rajeev Rastogi, & Kyuseok Shim (1999) ROCK: a robust clustering algorithm for categorical attributes. *Proc. of the 15th Int'l Conf. on Data Eng*.

[8] Maria Halkidi, Yannis Batistakis & Michalis Vazirgiannis (2001) On Clustering Validation Techniques. *Journal of Intelligent Information Systems*, Vol. 17, No (2-3), p. 107-145.

[9] Anil K. Jain & Richard C. Dubes (1988) Algorithms for Clustering Data. Prentice Hall.

[10] Teuvo Kohonen (1990) The self-organizing map. *Proc. of the IEEE*, Vol. 78, No 9, p. 1464-1480.

[11] Mark S. Aldenderfer & Roger K. Blashfield (1984) Cluster Analysis. Sage Publications, Inc.

[12] J. B. McQueen (1967) Some Methods of Classification and Analysis of Multivariate Observations. *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, p. 281-297.

[13] G. W. Milligan, S. C. Soon & L. M. Sokol (1983) The effect of cluster size, dimensionality and the number of clusters on recovery of true cluster structure. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 5, p. 40-47.

[14] G. W. Milligan & M. C. Cooper (1986) A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, Vol. 21, p. 441-458.

[15] C. J. Roberts, B. Nelson, M. J. Marton, R. Stoughton, M. R. Meyer, H. A. Bennett, Y. D. He, H. Dai, W. L. Walker, T. R. Hughes, M. Tyers, C. Boone & S.H. Friend (2000) Signaling and circuitry of multiple map pathways revealed by matrix of global gene expression profiles. *Science*, 283, 873-880.

[16] M. Schena, D. Shalon, R. W. Davis & P. O. Brown (1995) Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science* 270: 467-470.

[17] P. T. Spellman, G. Sherlock, M. Q. Zhang, V.R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, & B. Fucher (1998) Comprehensive Identification of Cell Cylce-regulated genes of the yeast *saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell* 9, 3273-3297.

[18] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. S. Lander & T. R. Golub (1999) Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. *Proceedings of the National Academy of Science USA*, Vol. 96, p. 2907-2912.

[19] X. Wen, S. Fuhrman, G. S. Michaels, D. B. Carr, S. Smith, J. L. Barker & R. Somogyi (1998) Neurobiology large-scale temporal gene expression mapping of central nervous system development. *Proc. Natl Acad. Sci. USA*, 95, 334-339.

[20] K. Y. Yeung, D. R. Haynor & W. L. Ruzzo (2001) Validating clustering for gene expression data. *Bioinformatics*, Vol. 17, p. 309-318.

[21] Tian Zhang, Raghu Ramakrishnan & Miron Livny (1996) BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data*, p. 103-114, Montreal, Canada.

Fault Detection and Isolation Using Hybrid Parameter Estimation and Fuzzy Logic Residual Evaluation

Belkacem Athamena
Department of Automatics, University of Biskra, BP 145, Biskra RP, 07000, Algeria.
E-mail: b.athamena@caramail.com

Hadj Ahmed Abbassi
Department of electronics, University of Annaba, BP 12, Annaba, 23000, Algeria.
E-mail: habbassi@wissal.dz

Keywords: Fault detection and isolation, Fuzzy logic, Parameter estimation, Adaptive threshold.
Received: February 11, 2002

Fault diagnosis has become an issue of primary importance in modern process automation as it provides the prerequisites for the task of fault detection. The ability to detect the faults is essential to improve reliability and security of a complex control system. When a physical parameter change due to failure has occurred in a system, the failure effect will hardly be visible in the output performance. Since the failure, effect is reflected as a change in the predictor model. In this paper we describe a completed feasibility study demonstrating the merit of employing hybrid parameter-estimation and fuzzy logic for fault diagnosis. In this scheme, the residual generation is obtained from input-output data process, and identification technique based on ARX model, and the residual evaluation is based on fuzzy logic adaptive threshold method. The proposed fault detection and isolation tool has been tested on a magnetic levitation vehicle system.

1 Introduction

One of the most important goals of intelligent automatic control systems is to increase the reliability, availability, and safety of those systems. A complex automatic system can consist of hundreds of interdependent working elements, which are individually subject to malfunction. Total faults of the systems can cause unacceptable economic loss or hazards to personnel. Therefore, it is essential to provide on-line operating information by a scheme of observation and monitoring which detects faults as they occur, identifies the type of malfunction of faulty components, and compensates for the faults by appropriate actions and management to meet reliability and safety requirements so that the system can indeed continues to operate satisfactorily.

In many application the problem of fault detection and isolation FDI is a crucial issue that has been theoretically and experimentally investigated with different types of approaches, as can be seen from the survey papers (Willsky 1976, Isermann 1984, Basseville 1988, Gertler 1988, Frank 1990) and the books (Basseville & Nikiforov 1993, Chen & Patton 1999, Gertler 1998, Patton & al. 2000, Patton & al. 1989) among other references. It has been widely acknowledged that the FDI problem can be split into two steps: *generation of residuals*, which are ideally close to zero under no-fault conditions, minimally sensitive to noises and disturbances, and maximally sensitive to faults, *and residual evaluation*, namely *design of decision rules* based on these residuals.

In this paper, we study the possible fault symptoms occurring in a magnetic levitation vehicle system MLV. The method proceeds in four stages. First, the MLV model is estimated by hybrid parameter-estimation technique. Then fault symptoms are defined analytically according to physical system features and the residual signal is then designed by the prediction error. After the residual generation, the fundamental problem is residual evaluation, for these applications, is that, even supposing the model to be precise, the measurements are not; thus evaluating precisely the decision threshold value valid for every operating condition is difficult. To go beyond this problem, several solutions have been proposed, for instance, using adaptive threshold evaluation of the residuals.

The paper is organized as follows: In section 2 the hybrid parameter-estimation and the problem formulation are described. Section 3 devotes to the fault diagnosis concept of the fault detection scheme. The design and simulation examples are given in section 4, and the conclusion is drawn in section 5.

2 Hybrid parameter-estimation

The hybrid parameter-estimation method can be briefly described as follows. Consider a single-input single-output system described by a linear differential equation,

$$y(t) = -\sum_{i=1}^{n} a_i y^{(i)}(t) + \sum_{j=0}^{m} b_j u^{(j)}(t) + v(t), \quad n \geq m \qquad (1)$$

where the superscript notations means the time derivative operation, that $y^{(i)}(t) = d^i y(t)/dt^i$ and $y(t)$, $u(t)$ and $v(t)$ are output, input and noise, respectively.

Effectively, we have linear model with regard to the parameters, but impracticable because explanatory

variables $y(t)$ and $u(t)$ are not available (Middleton & Goodwin 1990). The principle is correct, but a previous filtering of data is necessary in order to achieve a transformation of model under a realist form. The methodology is called chain moments of Poisson, which that consists to use a stable nth-order filter. The basic idea of the method is to transform the original system model into an estimated model by introducing a set of identical linear filters, operating on each term in the original model. Let $g(t)$ be the impulse response of the filter, the transformed system model is then given by,

$$\int_0^t y(\tau)g(t-\tau)d\tau = -\sum_{i=1}^{n} a_i \int_0^t y^{(i)}(\tau)g(t-\tau)d\tau + \sum_{j=0}^{m} b_j \int_0^t u^{(j)}(\tau)g(t-\tau)d\tau + \int_0^t v(\tau)g(t-\tau)d\tau \quad (2)$$

where $y^{(i)}(t)$ and $u^{(j)}(t)$ denote the derivatives of order *i* and *j* respectively. Introducing the variables $y_{Fi}(t)$, $u_{Fj}(t)$ and $v_{F0}(t)$, equation (2) can be simplified into,

$$y_{F0}(t) = -\sum_{i=1}^{n} a_i y_{Fi}(t) + \sum_{j=0}^{m} b_j u_{Fj}(t) + v_{F0}(t) \quad (3)$$

In practice, we prefer use a simple structure of $g(t)$, depending of minimum parameters, for this reason, habitually we use,

$$g(t) = \frac{1}{(n-1)!} t^{n-1} e^{\alpha t} \quad (4)$$

The choice of α conditions the bias, but also the convergence of the estimation. We can choose α in manner that $e_i, i = \overline{1,n}$, the filter coefficients approach to the better of $a_i, i = \overline{1,n}$, for example according to the criterion of bandwidth (Athamena & Abbassi 2000) then,

$$y(t) = \sum_{i=1}^{n} (e_i - a_i) y_{Fi}(t) + \sum_{j=0}^{m} b_j u_{Fj}(t) + v_{F0}(t) \quad (5)$$

We obtain a linear model with regard to the parameters by a transformation of the original data to the filtered data, where an analogue relation to the equation (1).

The estimation problem consists of the parameter identification, which appears in the model by the treatment of the input/output data. We consider that θ the parameter vector, which can correctly translate the dynamic behavior of the process, and $\varphi(t)$ the regression vector. The estimation problem is to find a good estimate $\hat{\theta}$ of θ. The common measure of goodness of an estimate in the least squares cost function,

$$c(\hat{\theta}, t) = \frac{1}{2} \int_0^t (y(\tau) - \varphi^T(\tau)\hat{\theta})^2 d\tau \quad (6)$$

The estimation method that we will study in this paper basically depend on our ability to rearrange the model so that the predicted output describable as a linear function of a parameter vector θ: that is, there exists some vector of measured variables, $\varphi(t)$, such that the model output $\hat{y}(t,\theta) \approx \hat{y}(t)$ can be expressed as,

$$\hat{y}(t) = \varphi^T(t)\theta \quad (7)$$

where,

$$\theta = \begin{bmatrix} e_1 - a_1 & \cdots & e_n - a_n & b_0 & \cdots & b_m \end{bmatrix}^T$$
$$\varphi^T(t) = \begin{bmatrix} y_{F1}(t) & \cdots & y_{Fn}(t) & u_{F0}(t) & \cdots & u_{Fm}(t) \end{bmatrix} \quad (8)$$

In this case, we can define the algorithm of the generalized hybrid least squared according to,

$$\delta\hat{\theta}(t) = \frac{\alpha(t)P(t)\varphi(t)(y(t) - \varphi^T(t)\hat{\theta}(t))}{\Gamma(t) + T\varphi^T(t)P(t)\varphi(t)}$$
$$\delta P(t) = \frac{-\alpha(t)P(t)\varphi(t)\varphi^T(t)P(t)}{\Gamma(t) + T\varphi^T(t)P(t)\varphi(t)} + \Omega(t) \quad (9)$$

where δ is the delta operator (Middleton & Goodwin 1990, Athamena & Abbassi 2001) and,

$\alpha(t)$ = A (time-varying) gain, $\alpha(t) \in [0 \quad 1]$.

$\Gamma(t)$ = A (time-varying) normalization term, $\Gamma(t) > 0$.

and, where $\Omega(t)$ represents a modification to the covariance, with: $\Omega(t) = \Omega^T(t) \geq 0$.

For the least square algorithm with forgetting factors, we use,

$$\Omega(t) = \left(\frac{1}{1 - T\lambda}\right)\left(\lambda P(t) - \frac{TP(t)\varphi(t)\varphi^T(t)P(t)}{\Gamma + T\varphi^T(t)P(t)\varphi(t)}\right)$$
$$\alpha(t) = 1 \quad (10)$$

the algorithm also needs initial values $\hat{\theta}(0)$ and $P(0)$. Experience with this simple rule for setting λ shows that a decrease in the value of the forgetting factor leads to two effects:

- The parameter estimates converge to their true values quicker thus, decreasing the faulty alarm delay time.
- But at the expense of increased sensitivity to noise. If λ is much less than 1 the estimate may even oscillates around its true value.

There are various ways around this problem, in this method the constant λ in (10) is replaced by $\lambda(t)$. A typical choice is a recursively given by,

$$\lambda(t) = \lambda_0\lambda(t-1) + (1 - \lambda(0)) \quad (11)$$

typical design values for λ_0 and $\lambda(0)$ are 0.99 and 0.95 respectively. The least square algorithm is used for its speed of convergence, ease of implementation and numerical stability. A large body of research has been devoted to devising choices for the forgetting factor to allow continued adaptively without overdue sensitivity to transient disturbances and without catastrophic numerical effects such as "covariance blow-up".

3 Fault diagnosis concept

The fault diagnosis concept proposed here consists of the basic steps residual generation, residual evaluation and fault alarm presentation as shown in Figure 1 (Athamena & Abbassi 2002).

3.1 Residual generation

Residual generation via hybrid parameter-estimation relies on the principle that possible faults in the monitored process can be associated with specific

parameters and states of a mathematical model of a process given in general by an input-output relation. The main idea is to generate residuals that reflect inconsistencies between nominal and faulty system operations. When faults are present, the residual sequence distribution is changed. Many hypothesis tests can be used to evaluate the residual sequences.

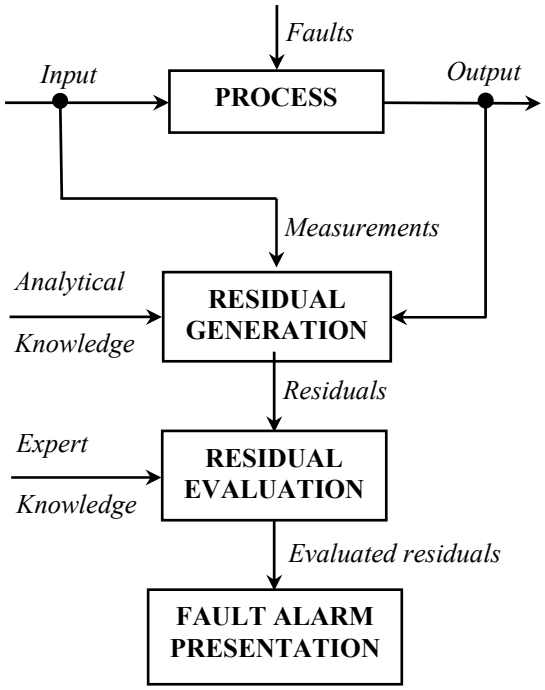


Figure 1: Fault Diagnosis Concept.

In this following, $r_k(t)$ represents the residual in each variable, that is the difference between the measurement parameter vector $\theta_k(t)$ and it's estimated $\hat{\theta}_k(t)$ at each time instant,

$$r_k(t) = \theta_k(t) - \hat{\theta}_k(t), k = \overline{1, n+m+1} \quad (12)$$

if the process is operating normally, the innovation process is zero-mean white noise.

Fault in dynamical systems can be detected with the aid of an innovation sequence that has the property that if the system operates normally the normalized innovation sequence is a Gaussian white noise with zero mean and with a unit covariance matrix. Faults that change the system dynamics affect the characteristics of the normalized innovation sequence by changing its white noise nature, displacing its zero mean, and varying unit covariance matrix. Thus, the problem is how to detect as quickly as possible any change of these parameters from their nominal value.

3.2 Fuzzy logic based decision signal

The residual evaluation is a logic decision making process that transforms quantitative knowledge (residuals) into qualitative knowledge (fault symptoms). The goal is to decide if and where in the process the fault has occurred, with a minimum rate of erroneous decision (false alarms) that are caused by the existing disturbances and modeling uncertainties. In Figure 2, the principle of residual evaluation using fuzzy logic consists of a three-step process. Firstly, the residuals have to be fuzzified, then they have to be evaluated by an inference mechanism using *IF-THEN* rules, and finally they have to be defuzzified.

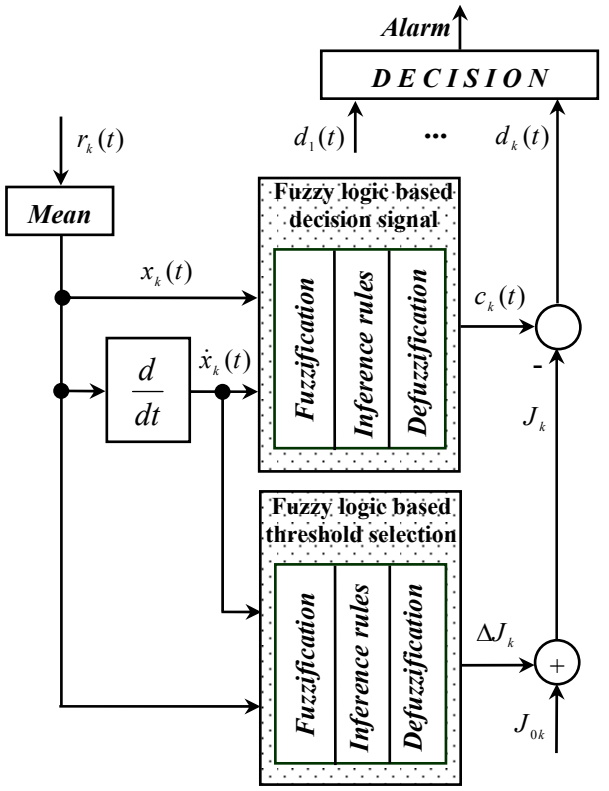


Figure 2: Residual evaluation concept.

The mean value of the residual $r_k(t)$ on a temporal window of p sampling periods T, $x_k(t)$ is given by,

$$x_k(t) = \frac{1}{p}\sum_{j=0}^{p} r_k(t-j) \quad (13)$$

The residual derivative $\dot{x}_k(t)$ will be estimated on the same temporal window by a least square linear approximation,

$$\dot{x}_k(t) = \frac{p\sum_{j=0}^{p} j r_k(t-j) - \sum_{j=0}^{p} j \sum_{j=0}^{p} r_k(t-j)}{p\sum_{j=0}^{p} j^2 - \left(\sum_{j=0}^{p} j\right)^2} \quad (14)$$

The use of mean values over a small temporal window (in the application $p = 8$) somewhat filters the measurement noise and at the same time allows a quick determination of any change in the residuals.

Fuzzification: The fuzzification of the residuals is a mapping of the representation with crisp values into a representation by fuzzy sets. The values $x_k(t)$ and $\dot{x}_k(t)$ are then fuzzified by the fuzzy partitions $X_k = \left(X_i, \mu_{X_i}(x_k(t))\right)$ and $\dot{X}_k = \left(\dot{X}_j, \mu_{\dot{X}_j}(\dot{x}_k(t))\right)$ defined over

the universe of $x_k(t)$ and $\dot{x}_k(t)$, each one composed by five fuzzy sets.

To describe the process, linguistic variables such as "*large-negative*", "*small-positive*", "*zero*" will be used and applied to mean value and residual derivative. To allow an approximate reasoning based on numerical values a "fuzzification" of these values is runaway. Fuzzy sets are built to correspond to each linguistic variable, and membership functions $\mu(0 \le \mu \le 1)$ are defined for those fuzzy sets. The total variations of $x_k(t)$ and $\dot{x}_k(t)$ are split up into five subsets: *negative-high* (NH), *negative-medium* (NM), *zero* (ZE), *positive-medium* (PM) and *positive-high* (PH). The choice of the numerical values for the boundary marks was made using first the simulation results and after that, the experimental results.

Symmetric trapezoidal membership functions are used (Figure 3). This lead to a simple parameterization of each partition with only 4 parameters α_1, α_2, α_3, α_4, corresponding to the trapezoid boundaries.

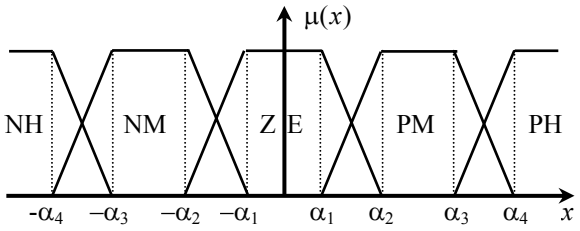


Figure 3: Membership functions of $x_k(t)$ and $\dot{x}_k(t)$.

Inference rules: The common-sense shows clearly that some situations describe by the combination of membership functions of residuals and their derivatives to some fuzzy sets are worse than others. For instance, if the residual is medium positive with a negative derivative, this means that it is decreasing, thus the situation is not so bad, while if the residual is positive high with a positive high derivative, the situation is bad and worsening. For typical situations have been chosen to describe the state of a variable: OK means that the state in *normal*, SP means *suspicious*, AL means *alarming*, and FA means *faulty*.

The 5 fuzzy sets of each partition form 25 combinations, which lead to the decision table found in Table 1. Each element of this table can be interpreted as a fuzzy rule of the type,

$$IF\ x_k(t)\ is\ X_i\ AND\ \dot{x}_k(t)\ is\ \dot{X}_j\ THEN\ state\ is\ S_{ij} \qquad (15)$$

In order to accelerate the processing of this table, it has been modeled as zero order Sugeno fuzzy inference model (Mamdani 1977), which can be viewed as a predefuzzification consequent rule. The rule (15) written as a zero order Sugeno fuzzy rule has the form,

$$IF\ x_k(t)\ is\ X_i\ AND\ \dot{x}_k(t)\ is\ \dot{X}_j\ THEN\ C_{ij} = \Phi_{ij} \qquad (16)$$

Thus the symbolic states {OK, SP, AL, FA} in the table are replaced by numerical constants $\{\Phi_{OK}, \Phi_{SP}, \Phi_{AL}, \Phi_{FA}\}$. These constants have been arbitrarily chosen to $\{\Phi_{OK} = 0,\ \Phi_{SP} = 0.33,\ \Phi_{AL} = 0.66,\ \Phi_{FA} = 1\}$ but their particular value is not important to the decision making process.

The antecedent of the rule (16) represents an intersection of the fuzzy sets, easily handled with an *AND* fuzzy logic operator. The fire strength of a rule associated to the position *ij* of the Table 1, denoted by w_{ij}, is evaluated, at each sampling time, by a T-norms as a product,

$$w_{ij} = \mu_{X_i}(x_k(t)) \wedge \mu_{\dot{X}_j}(\dot{x}_k(t)) = \mu_{X_i}(x_k(t))\mu_{\dot{X}_j}(\dot{x}_k(t)) = w_i \dot{w}_j \qquad (17)$$

Where w_i and $\dot{w}_j$ represent the membership functions of $x_k(t)$ and $\dot{x}_k(t)$ to the respective fuzzy sets.

$x_k(t)$ \ $\dot{x}_k(t)$	**NH**	**NM**	**ZE**	**PM**	**PH**
NH	*FA*	*FA*	*AL*	*SP*	*SP*
NM	*FA*	*AL*	*SP*	*SP*	*SP*
ZE	*SP*	*OK*	*OK*	*OK*	*SP*
PM	*SP*	*SP*	*SP*	*AL*	*FA*
PH	*SP*	*SP*	*AL*	*FA*	*FA*

Table 1: Inference rules.

Defuzzification: Different methods of defuzzification exist, but it seems that none of them satisfies all criteria defined for an "*ideal defuzzification method*". The center of gravity method is; in fact, the most frequently used one for diagnostic system design (Schneider & Frank 1996). As the residuals and their derivative can belong to several fuzzy sets, several elements in the decision table can be valid at the same time; thus, the multiple rule conclusions need to be aggregated. Multiple rules are interpreted as the union of the corresponding fuzzy relations (OR operator). In zero order Sugeno fuzzy model, the output of a rule base as (16) is evaluated as a weighted sum,

$$c_k(t) = \frac{\sum_i \sum_j \Phi_{ij} w_{ij}}{\sum_i \sum_j w_{ij}} \qquad (18)$$

The parameters α_1, and α_2 in Figure 3 that define the bounds of the fuzzy sets ZE and PM (or NM) in the fuzzy partition associated to $x_k(t)$ and $\dot{x}_k(t)$ could be easily chosen from an estimation of the measurements noise variance. The parameters α_3 and α_4 could be taken as the lower and upper thresholds used in classical alarm detection. The means α_4 is the value beyond which the state of the variable is undoubtedly faulty, and α_3 is an intermediary value.

3.3 Fuzzy logic based adaptive threshold

The most simple and straightforward method for fault decision consists in a threshold test of the residual $r_k(t)$ or a measure $g(r_k)$ formed from the residual. If constant thresholds are used one has to cope with the problem of the effects of unknown inputs. If the threshold is chosen too small, false alarms will occur, if the threshold is chosen too large, small faults can not be detected. Therefore, it has recently been shown (Ding & Frank

1991) that it is advantageous to use thresholds that are adapted to the operation of the process. The problem of adaptive threshold logic is illustrated by a typical example in Figure 4. The shape of the threshold follows a certain maneuver of the fault-free process only taking into account the influence of the unknown inputs. Suppose the time evolution of the residual is due to a maneuver of the process in the face of unmatched parameters with a fault at t_F. Evidently, in contrast to a constant threshold, where a false alarm occurs at t_{FA} and fault at t_F cannot be detected, the false alarm can be avoided and the fault at t_F can be detected.

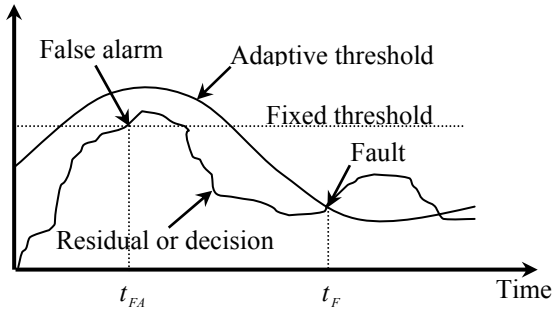


Figure 4: Adaptive threshold test.

A fuzzy-based approach for a robust threshold selection for fault detection has been described by (Frank & Kiupel 1992). The gradual meaning of fuzzy sets has been exploited by defining a threshold through fuzzy membership functions. There are a trapezoidal membership function has been chosen such that the rising and falling edges of the trapeze depend on a parameter incorporating the variance of noise, uncertainty, and disturbances.

The adaptive threshold consists of a predefined value, J_{0k}, and an adaptive term, ΔJ_k, which is determined by heuristic knowledge. This approach has been developed independently by Schneider (Schneider 1993) and Sauter (Sauter & al. 1993). The threshold is adapted depending on the changes of the values of $x_k(t)$ and $\dot{x}_k(t)$ in terms of rules among fuzzy sets that are specified by proper membership function. The resulting relation for the fuzzy threshold adaptation is given by,

$$J_k(x_k,\dot{x}_k) = J_{0k} + \Delta J_k(x_k,\dot{x}_k) \tag{19}$$

The term J_{0k} represents an appropriate threshold for the normal system behavior. The adaptive term, $\Delta J_k(x_k,\dot{x}_k)$, incorporates the effects of modeling errors. The term, $\Delta J_k(x_k,\dot{x}_k)$, has positive as well as negative values such that an adjustable threshold band can be realized which follows the actual residual signal. A schematic diagram of the suggested concept is presented in Figure 2.

The main four steps for the adaptive fuzzy based threshold selection can be stated as:

1. Observation of relations between false alarms and characteristic process conditions.
2. Formulation of rules of thumbs, which are organized by: *IF…THEN…STRUCTURES.*
3. Choice of appropriate fuzzy variables and membership functions.
4. Definition of a fuzzy rule table based on steps 2, 3.

After an initial setup of membership functions and a fuzzy rule base, further knowledge can be incorporated by changing the rules or by introducing new fuzzy variables if necessary. In this way, unstructured disturbances are incrementally included in the decision process. Since this concept is based on linguistic variables no theoretical process knowledge is required but valuable heuristic information can be modified by experienced operational personal.

For simple realization, a standard fuzzy rule is suggested. The values $x_k(t)$ and $\dot{x}_k(t)$ are then fuzzified, each one composed by four fuzzy sets (Figure 5). The linguistic labels of those sets are the common ones: *positive zero* (PZ), *positive small* (PS), *positive medium* (PM), and *positive large* (PL).

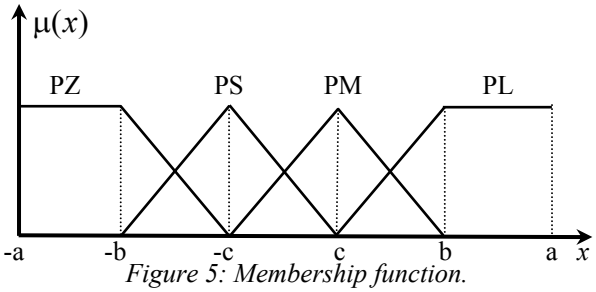


Figure 5: Membership function.

The 4 fuzzy sets of each partition form 16 combinations, which lead to the decision table found in Table 2.

$x_k(t)$ \ $\dot{x}_k(t)$	**PZ**	**PS**	**PM**	**PL**
PZ	*PM*	*PS*	*PM*	*PL*
PS	*PM*	*PS*	*PM*	*PL*
PM	*PS*	*PS*	*PM*	*PL*
PL	*PZ*	*PS*	*PM*	*PL*

Table 2: Inference rules.

Each element of this table can be interpreted as a fuzzy rule of the type,

$$IF\ x_k(t)\ is\ X_i\ AND\ \dot{x}_k(t)\ is\ \dot{X}_j\ THEN\ \Delta J_k(x_k,\dot{x}_k)\ is\ \Delta J_{ij} \tag{20}$$

The heuristics for generating the threshold can be summarized as follows:

- For very small mean value of the residual the threshold has to be increased to a medium level;
- For high residual derivative value the threshold has to be increased considerably;
- For very high residual derivative value the threshold has to be increased drastically; and
- For a very high residual derivative value the threshold has to be increased to a medium level.

The rules are like those described with *max-min* composition method and the center area of defuzzification.

When $c_k(t)$ over-passes a threshold, the isolation procedure is fired. For a proper on-line processing, the case when a fault has been corrected must be detected as well as a defect rise. Thus, two threshold values are used,

one to decide the failure detection, and the other to detect the fault correction, then,

$$\|c_k(t)\| \begin{cases} \le J_k(x_k, \dot{x}_k): & No\ Fault \\ > J_k(x_k, \dot{x}_k): & Fault \end{cases} \quad (21)$$

4 The magnetic levitation vehicle

4.1 System dynamics

In this section, a design example will be presented to illustrate the design procedure of the proposed FDI. Figure 6 shows the cross section of a MLV system. The track is a T-shaped concrete guideway. Electromagnets are distributed along the guideway and along the length of the train in matched pairs. The magnetic attraction of the vertically paired magnets balances the force of gravity and levitates the vehicle above the guideway. The horizontally paired magnets stabilize the vehicle against sideways forces. Forward propulsion is produced by linear induction motor action between train and guideway.

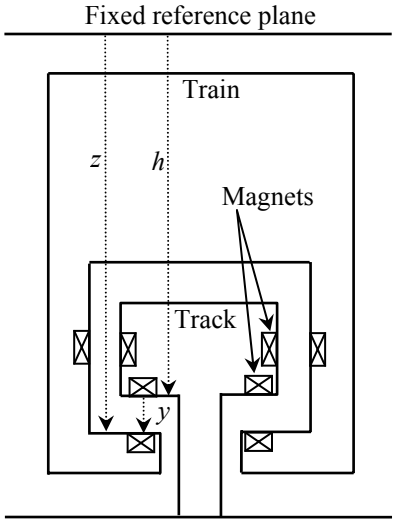


Figure 6: Cross section of a MLV train.

The equations characterizing the train's vertical motion are now being developed according to the law of physics. It is desired to control the gap distance $y(t)$ within a close tolerance in normal operation of the train. The gap distance $y(t)$ between the track and the train magnets is,

$$y(t) = z(t) - h(t) \quad (22)$$

then,

$$\begin{aligned} \dot{y}(t) &= \dot{z}(t) - \dot{h}(t) \\ \ddot{y}(t) &= \ddot{z}(t) - \ddot{h}(t) \end{aligned} \quad (23)$$

where the dots denote time derivatives. The magnet produces a force that is dependent upon residual magnetism and upon the current passing through the magnetizing circuit. For small changes in the magnetizing current $i(t)$ and the gap distance $y(t)$, that force is approximately,

$$f(t) = -Gi(t) + Hy(t) \quad (24)$$

where G constant controls the input-output gain of the open-loop system and H constant controls the poles of the system. That force acts to accelerate the mass M of the train in a vertical direction, so,

$$f(t) = M\ddot{z}(t) = -Gi(t) + Hy(t) \quad (25)$$

For increased current, the distance $z(t)$ diminishes, reducing $y(t)$ as the vehicle is attracted to the guideway.

A network model for the magnetizing circuit is given in Figure 7. This circuit represents a generator driving a coil wrapped around the magnet on the vehicle. In this circuit,

$$Ri(t) + Li(t) - \frac{LH}{G}\dot{y}(t) = v(t) \quad (26)$$

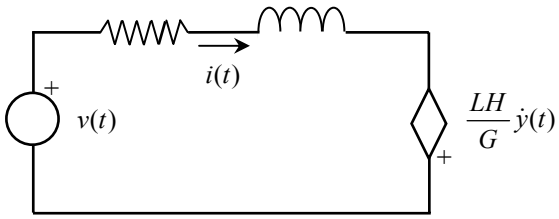


Figure 7: Magnetizing circuit model.

The three state variables $z_1(t) = y(t)$ (*Gap distance*), $z_2(t) = \dot{y}(t)$ (*Gap velocity*) and $z_3(t) = i(t)$ (*Magnetizing current*) are convenient, and in terms of them the vertical motion state equations are,

$$\begin{bmatrix} \dot{z}_1(t) \\ \dot{z}_2(t) \\ \dot{z}_3(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{H}{G} & 0 & -\frac{G}{M} \\ 0 & \frac{H}{G} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} z_1(t) \\ z_2(t) \\ z_3(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \\ \frac{1}{L} & 0 \end{bmatrix} \begin{bmatrix} v(t) \\ f_d(t) \end{bmatrix} \quad (27)$$

where $v(t)$ is the voltage control input and $f_d(t)$ is the force disturbance of guideway irregularities.

If the gap distance $y(t)$ is considered to be the system output, then the state variable output equation is,

$$y(t) = z_1(t) \quad (28)$$

The voltage $v(t)$ is considered to be control input, while guideway irregularities $f_d(t) = \ddot{h}(t)$ constitute a disturbance. The system parameters M, G, L, and R can be derived analytically by static test and dynamic equilibrium of the vehicle.

The open-loop system with $f_d(t) = 0$, described by a linear differential equation,

$$a_3 y^{(3)}(t) + a_2 y^{(2)}(t) + a_1 y^{(1)}(t) + y(t) = b_0 v(t) \quad (29)$$

where,

$$a_1 = -\frac{L(G-M)}{MR}, a_2 = -\frac{G}{H}, a_3 = -\frac{GL}{HR}, b_0 = \frac{G^2}{MRH} \quad (30)$$

Then, the transformed system model is then given by,

$$y(t) = (e_1 - a_1)y_{F1}(t) + (e_2 - a_2)y_{F2}(t) + (e_3 - a_3)y_{F3}(t) + b_0 v_{F0}(t) \quad (31)$$

The model in (31) has the form,

$$y(t) = \varphi^T(t)\theta \quad (32)$$

where,

$$\begin{aligned} \theta &= \begin{bmatrix} e_1 - a_1 & e_2 - a_2 & e_3 - a_3 & b_0 \end{bmatrix}^T \\ \varphi^T(t) &= \begin{bmatrix} y_{F1}(t) & y_{F2}(t) & y_{F3}(t) & v_{F0}(t) \end{bmatrix} \end{aligned} \quad (33)$$

4.2 Fault modeling

Due to the reduced space just four faults have been investigated in this paper. The considered faults are represented in the Table 3.

Fault	***Fault situation***
F_1	Malfunction in the parameter *R*
F_2	Malfunction in the parameter *L*
F_3	Malfunction in the parameter *H*
F_4	Malfunction in the parameter *G*

Table 3: Fault symptoms of MLV system.

It is evident in (30, 33) that if $c_1(t)$, $c_3(t)$ and $c_4(t)$ changes but $c_2(t)$ remain unchanged, this then implies a change in *R*. (Throughout the paper it is assumed that there are never two or more faults occurring simultaneously in the system). Similarly, if both $c_1(t)$ and $c_3(t)$ change, this then implies a change in *L*. If only $c_1(t)$ unchanged, this then implies a change in *H*. If $c_1(t)$, $c_2(t)$, $c_3(t)$ and $c_4(t)$ changes, this then implies a change in *G* (see Table 4). Therefore, faults can be diagnosed by observing changes in $c_k(t)$ in cooperation with the fuzzy residual evaluation based on adaptive threshold method. Furthermore, the size of a fault can be diagnosed if the estimation is precise.

	$c_1(t)$	$c_2(t)$	$c_3(t)$	$c_4(t)$
F_1	1	0	1	1
F_2	1	0	1	0
F_3	0	1	1	1
F_4	1	1	1	1

Table 4: The decision table.

4.3 Experimental results

The effectiveness of method was verified using simulated data. For this purpose, the MLV parameters were chosen as:

$$G = 44NA^{-1}, H = 58000Nm^{-1}, M = 3kg, R = 7\Omega, L = 33mH$$

A 2KHz sampling frequency is considered, two real-time simulations have been carried out. For the choice of the confidence degree, we opt for the value 5% that wants to say that the estimation makes it with a confidence rate of 95%.

Test 1: The first set of faulty data simulates a change in the efficiency of the armature resistance *R*, and a change in the efficiently of the inductance *L*,

$$\Delta R/R = 1.00, t \geq 250; \ \Delta L/L = 1.00, t \geq 350$$

The steady-state values of the estimated model parameters before and after the faults are,

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_0 \end{bmatrix} = \begin{bmatrix} -64.4286 \\ -7.5862\,10^{-4} \\ -0.0036 \\ 0.0016 \end{bmatrix} \xrightarrow{\Delta R} \begin{bmatrix} -32.2143 \\ -7.5862\,10^{-4} \\ -0.0018 \\ 7.9475\,10^{-4} \end{bmatrix} \xrightarrow{\Delta L} \begin{bmatrix} -128.8571 \\ -7.5862\,10^{-4} \\ -0.0072 \\ 0.0016 \end{bmatrix}$$

By applying the RLS estimator with a forgetting factor, the estimates of a_1, a_2, a_3 and b_0 were obtained and converge quickly to their respective true values.

The percentage increase on a_1 is calculated as $\Delta a_1/a_1 = 0.98$ and the percentage increase on a_3 is calculated as $\Delta a_3/a_3 = 0.96$ and the percentage decrease on b_0 is calculated as $\Delta b_0/b_0 = 0.97$. It is observed that the relative change in size of the estimated model parameter is approximately equivalent to the relative change in size of the physical parameter. Therefore, the fault size is diagnosed (Yu 1997). In Figure 8, it can be seen that ΔR causes a significant change in the decisions signal $c_1(t)$, $c_3(t)$ and $c_4(t)$ and ΔL causes a significant change in the decisions signal $c_1(t)$ and $c_2(t)$. So, change in *R* and *L* can be diagnosed respectively.

Test 2: The second set of faulty data simulates a change in *H* and *G* according to,

$$\Delta H/H = 0.10, t \geq 100; \ \Delta G/G = 0.10, t \geq 400$$

The model parameter changes in the steady-state before and after the faults are,

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_0 \end{bmatrix} = \begin{bmatrix} -64.4286 \\ -7.5862\,10^{-4} \\ -0.0036 \\ 0.0016 \end{bmatrix} \xrightarrow{\Delta H} \begin{bmatrix} -64.4286 \\ -6.8966\,10^{-4} \\ -0.0033 \\ 0.0014 \end{bmatrix} \xrightarrow{\Delta G} \begin{bmatrix} -71.3429 \\ -8.3448\,10^{-4} \\ -0.0039 \\ 0.0019 \end{bmatrix}$$

The percentage increase on a_1 is calculated as $\Delta a_2/a_2 = 0.09$ and the percentage increase on a_3 is calculated as $\Delta a_3/a_3 = 0.08$ and the percentage increase on b_0 is calculated as $\Delta b_0/b_0 = 0.07$.

In Figure 9, it can be seen that ΔH causes a significant change in the decisions signal $c_2(t)$, $c_3(t)$ and $c_4(t)$ and ΔG causes a significant change in the decisions signal $c_1(t)$, $c_2(t)$, $c_3(t)$ and $c_4(t)$. So, change in *H* and *G* can be diagnosed respectively.

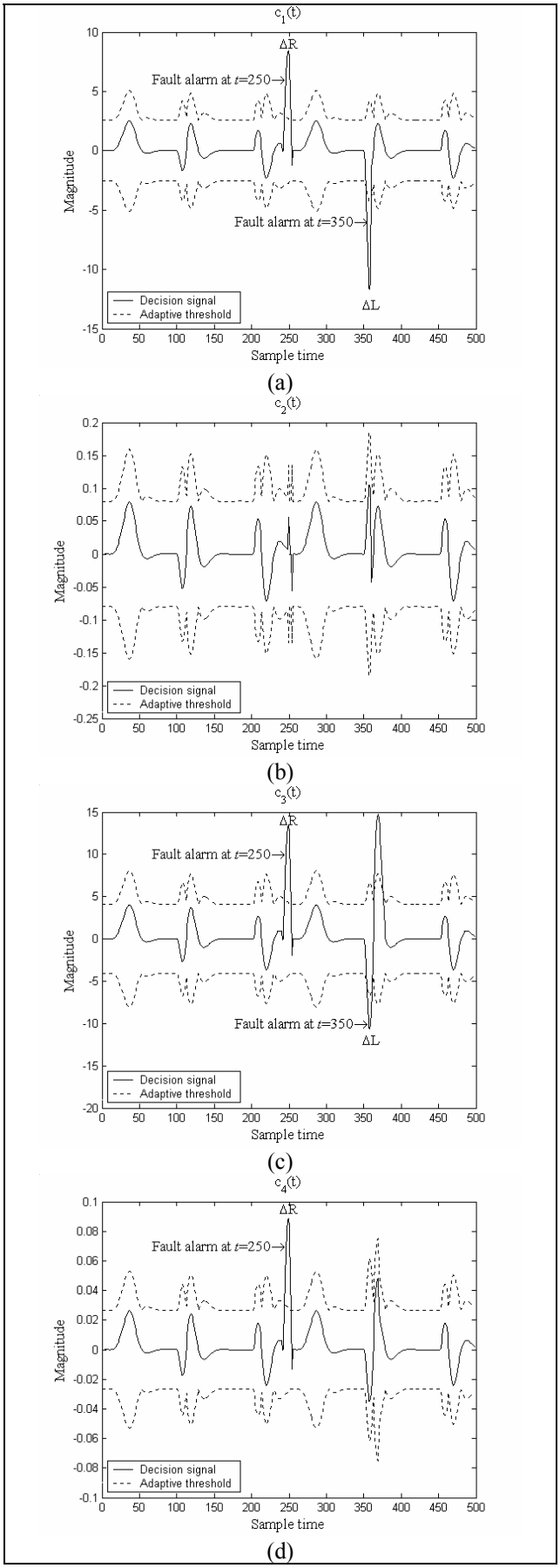


Figure 8: Decision signal and adaptive threshold residual evaluation.

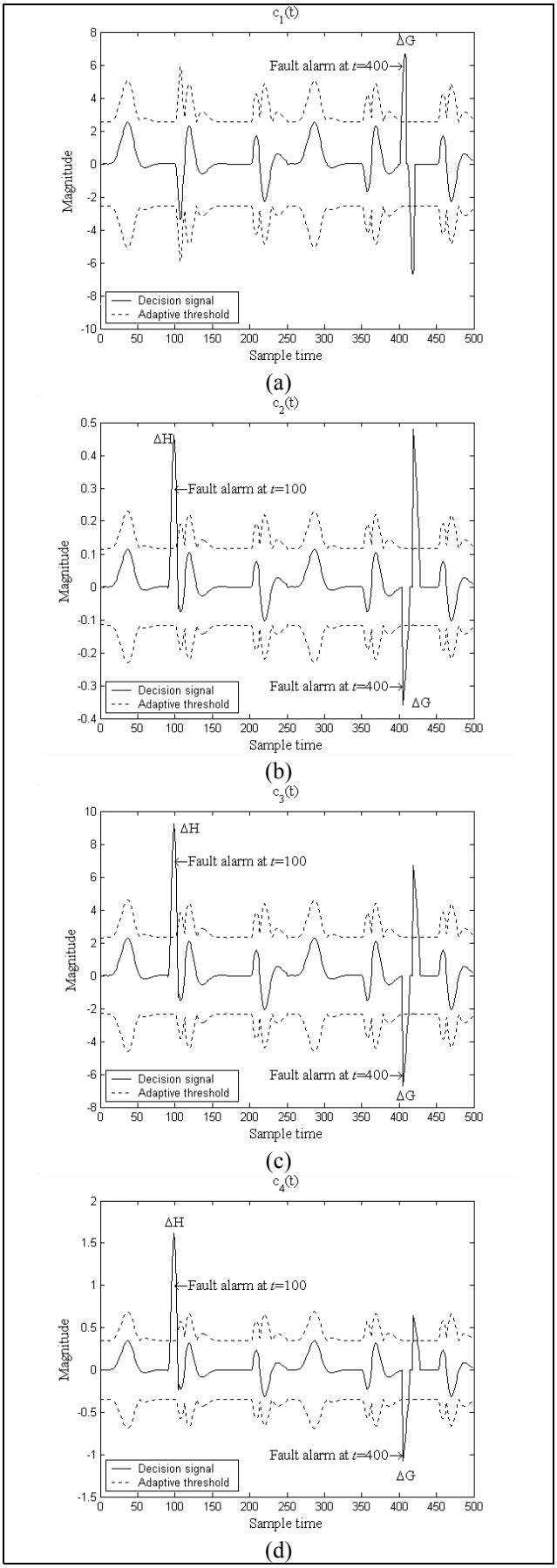


Figure 9: Decision signal and adaptive threshold residual evaluation.

In above two simulations, the changes in the physical parameters are clearly detected and isolated. Note that the model parameter change is delayed from the physical parameter change for all the faults, due to the convergence of the estimates of the model parameters. The maximum delay is about 500 sample intervals, or 2.5 seconds, which is allowable in practice. This lag-time is greatly influenced by the size of the forgetting factor, λ, in the RLS algorithm. It is seen that this fault can be detected at a high robustness against false alarms.

5 Conclusion

In this paper, a completed feasibility study of process fault diagnosis for a magnetic levitation vehicle system using hybrid parameter-estimation and fuzzy logic residual evaluation is presented. The failure effect due to a system parameter change appears as a difference in the prediction error. The fuzzy logic is, of course, particularly tailored for the task of diagnosis. The simulation study suggests that the combination of different methods will be more efficient for fault diagnosis in real industrial systems.

References

Athamena B. & Abbassi H.A. (2000) Fault detection and diagnosis based on fuzzy logic: Application to magnetic levitation system. *International conference on Modelling and simulation, MS'2000*, Las Palmas de Gran Canaria, Spain, 25-27 September.

Athamena B. & Abbassi H.A. (2001) Diagnosis techniques for system faults of industrial processes. *Rencontres francophone sur la logique floue et ses applications, LFA'2001*, Mons, Belgique, 26-27 November.

Athamena B. & Abbassi H.A. (2002) Robust fault detection and isolation in a complex dynamic system. *MMAR 2002, 8th IEEE International Conference on Methods and Models in Automation and Robotics*, 2-5 September 2002, Szczecin, Poland

Basseville M. (1988) Detecting changes in signals and systems-A survey. *Automatica*, 24(3), 309-326.

Basseville M. & Nikiforov I.V. (1993) Detecting of abrupt changes-theory and applications. *Information and system sciences series, Prentice-Hall, Englewood Cliffs*, NJ.

Chen, J. & Patton R.J. (1999) Robust model fault diagnosis for dynamics systems. *Kluwer Academic Publishers*, Boston, MA, USA.

Ding X. & Frank P.M. (1991) Frequency domain approach and threshold selector for robust model-based fault detection and isolation. *Proc. IFAC/IMACS Symp. SAFEPROCESS'91,* Baden-Baden, 307-3012.

Evsukoff A. & Montmain J. (1997) Dynamic model and causal knowledge-based fault detection and isolation. *Proceeding of IFAC-SAFEPROCESS'97*.

Frank P.M. (1990) Fault diagnosis in dynamic system using analytical and knowledge based redundancy- A survey and news results. *Automatica*, 26, 459-474.

Frank P.M. & Kiupel N. (1992) Fuzzy supervision for lean production. *in Proc. 6th Inst. Automat. Robot, IAR Colloquium*, Duisburg, Germany, 19 November.

Gertler J.J. (1988) Survey of model-based detection and isolation in complexes plants. *IEEE Control Systems Mag*., 8(6), 3-11.

Gertler J.J. (1999) Fault detection and diagnosis in engineering systems. *Marcel Dekker*, New York, NY, USA.

Isermann R. (1984) Process fault diagnosis based on modeling and estimation methods-A survey. *Automatica,* 20, 387-404.

Kumamaru K., Söderström T., Sagara S. & Morita K. (1988) On-line fault detection in adaptive control systems by using kullback discrimination index. *IFAC Identification and System Parameter Estimation*, Beijing, 1135-1140.

Mamdani E.H. (1977) The application of fuzzy set theory to control systems-A survey. *In Fuzzy Automata and Decision Process, Eds*. Amsterdam, The Netherlands,77-88.

Middleton R.H. & Goodwin G.C. (1990) Improved finite word length characteristics in digital control using delta operators. *IEEE, Transaction on Automatic Control*, Vol.AC-31, No.11, 1015-1021.

Middleton R.H. & Goodwin G.C. (1990) Digital control and estimation: A unified approach. *Prentice-Hall, Englewood Cliffs*, New Jersey.

Patton R.J., Frank P.M. & Clark R.N. (2000) Issues of fault diagnosis for dynamic systems. *Springer.*

Patton R.J., Frank P.M. & Clark R. (1989) Fault diagnosis in dynamic systems-Theory and application, *International series in systems and control engineering, Prentice-Hall*, London, U.K.

Patton R.J., Chen J. & Lopez Toribio C.J. (1998) Fuzzy observers for non-linear dynamic systems fault diagnosis. *Proceeding of the 37th IEEE Conference on Decision & Control*, Tampa, Florida USA, 84-89.

Sauter D., Dubois G., Levrat E. & Bremont J. (1993) Fault diagnosis in systems using fuzzy logic. *in Proc. EUFIT'93, 1st Europ. Congr. Fuzzy intell. Technol*., Aachen, Germany.

Schneider H. (1993) Implementation of a fuzzy concept for supervision and fault detection of robots. *in Proc. EUFIT'93, 1st Europ. Congr. Fuzzy intell. Technol*., Aachen, Germany.

Schneider H. & Frank P.M. (1996) Observer-based supervision and fault detection in robots using nonlinear and fuzzy logic residual evaluation. *IEEE, Transactions on control systems technology*, 4(3), 274-282.

Willsky A.S. (1976) A survey of design methods for failure detection in dynamic systems. *Automatica,* 12, 601-611.

Yu D. (1997) Fault diagnosis for hydraulic drive system using a Parameter-Estimation Method. *Control Engineering Practice*, 5(9), 1283-1291.

Practical Construction for Multicast Re-keying Schemes Using R-S Code and A-G Code

Chun-yan Bai, Roberta Houston and Gui-liang Feng
The Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA 70504
Email:cxb7146, rah1231, glf@cacs.louisiana.edu

Keywords: Multicast, Re-keying, Reed-Solomon code, Hash function, KDP

Received: August 20, 2002

Multicast Re-keying means the establishment of a new session key for the new subgroup in the multicast system. Practical construction methods for multicast re-keying scheme using Reed-Solomon codes and Algebraic-Geometric codes are presented in this paper with examples to show the detailed constructions. The constructions require no computational assumptions. The storage complexity for group members(Group Controller and other users) and the transmission complexity for the schemes have been reduced to $O(log(n))$ at the same time.

1 Introduction

With the rapid development of networks, the need for high bandwidth, very dynamic and secure group(multicast) communications is increasingly evident in a wide variety of commercial, govermnent, and Internet communities such as video-on-demand, multi-party teleconferencing, stock quote distribution and updating software. Specifically, the security in the multicast communication is the necessity for multiple users who share the same security attributes and communication requirements to securely communicate with each other using a common group session key.

The general goal of secure group communication is to dynamically transmit a message encrypted by the new *session key* over a broadcast channel shared by an exponential number $n = 2^m$ of users so that *all but* some specified small coalition of *k* excluded users can decipher the message, even if these excluded users collude with each other in an arbitrary manner. This is what we call the *broadcast exclusion* problem(also known as the *blacklisting* problem). The establishment of a new *session key* for the new subgroup is called the *Re-keying* of the system.

In the multicast communication system, the group is dynamic, which means that at different time, different subgroups of the initial group is authorized to receive the multicast message because of those dynamically joining and leaving group members. So the secure communication in multicast environment is much more challenging than traditional point-to-point communication and raises numerous new security problems. Examples are the forward secrecy and backward secrecy guarantee. A protocol provides *perfect backward secrecy* if a member joining the group at time t does not gain any information about the content of messages communicated at times $t' < t$. A protocol provides *perfect forward secrecy* if a member leaving the group at time t does not gain any information about the content of messages communicated at time $t' > t$.

Member-joining is easy to handle by just encrypting the new session key with the old session key which is decryptable by all old members and sending the new session key individually to each new member encrypted by their own secret keys. So we just focus on the member-leaving case and assume that there is a group controller(GC) who knows all the system keys in this paper.

The initial study on the secure multicast communication can be traced back to the early 90's [1]. And a lot of works had followed [2,3,4,5,6,7,8,9]. All in all, the work can be divided into two major groups, one of which [2,3,4,5] uses the concept of *key tree structure* to set up the new session key based on the Diffie-Hellman key agreement. In [2], Wallner proposed a scheme which requires only $O(n) = O(n+(n-1))$ keys for GC, $O(log^n) = O(d+1)$ keys for each user and have at most $O(log^n) = O(kd-1)$ transmissions overhead per single eviction. The requirement is further improved in[3,4,5] which greatly reduces the transmission and storage complexity of re-keying schemes. Another stronger property of the tree structured scheme is that it allows the number of excluded users k to be arbitrary, rather than fixed in advance. But some balanced tree structure based schemes have the disadvantage of not providing collusion prevention.

The other group makes use of the *broadcast encryption* idea proposed by Fiat and Naor[6]. The broadcast encryption scheme enables the GC to communicate data secretly to dynamically changing authorized users while preventing any coalition of users to learn anything about the data. Other studies on broadcast encryption schemes can be found in [7,8] and [9]. In [7] , the concept of the Perfect Hash Family(PHF) is reviewed and proved to be useful for the secure new session key distribution. The possibility of

using the error correcting code to construct such a scheme is given there without providing any practical and detailed construction. Hartono *et.al.* [8] borrows the idea of Key Distribution Pattern (KDP), based on which the broadcast encryption scheme that can remove up to t users from a group of n users and is secure against collusion of t malicious users can be set up. How to use the error correcting codes to construct such a KDP is not discussed. In [9], Poovendran and Baras show that by assigning probabilities to member revocations, the optimality, correctness and the system requirements of some of the schemes in [6,7,8] can be systematically studied using information theoretic concepts and also show that the optimal average number of keys per member in a secure multicast scheme is related to the entropy of the member revocation event, thus provides a way for us to inspect each scheme from the theory point of view.

2 Related Work review

Assume that there is a set of users U, a group controller GC and a set K of Key Encrypting Keys (KEK) that is generated and stored by the GC. $Session\ keys$ are used for group member communication. A user u_i will have a subset of Key Encrypting Keys, $K(u_i) \subseteq K$. KEKs are used to update the SK in the event of membership change due to any of the following reasons: (a) a new member admission, (b) expiration of the SK, (c) member compromise, (d) voluntary leave, and (e) member revocation. We only consider the last case, member revocation in this paper.

The secure group communication requires KEKs to securely distribute the updated SK. If every member has an individual public key, for a group consisting of n members, the SK update will involve $O(n)$ encryptions by the GC. The linear increase of the required number of encryptions in group size is not suitable for very large scale applications common in Internet, due to the amount of computational burden on the GC.

Next, we will review two scalable re-keying schemes which can reduce the number of encryptions.

2.1 Re-keying scheme based on PHF

A re-keying scheme called OR scheme in [7] specifies an algorithm by which the GC produces a common session key $k^{U\setminus W}$ for the group $U \setminus W$ without letting those users in W to know the new session key, where $W \subseteq U$. The scheme is as follows:

1. $Key\ initialization$: The GC generates and stores a set K of KEKs and securely gives u_i the set of his KEKs $K(u_i) \subseteq K$.

2. $Broadcast$: To remove a set of users W from U, the GC randomly chooses a session key $k^{U\setminus W}$ and encrypts it with those keys not belonging to W, then broadcasts the encrypted messages to all the users. That is, the GC broadcasts

$$\{E_k(k^{U\setminus W})|\ k \in K, k \notin K(W), K(W) = \cup_{j\in W} K(u_j)\}.$$

3. *Decryption:* Each user $u_i \in U \setminus W$ uses one of his own KEKs $k \in K(u_i)$ to decrypt $E_k(k^{U\setminus W})$ and obtain the new session key $k^{U\setminus W}$.

We review the concept of PHF here for the completeness of this paper. Let n and m be integers such that $2 \leq m \leq n$, $A = \{1, 2, ...n\}$ and $B = \{1, 2, ...m\}$ be two sets. A $hash\ function$ is a function h from A to B $h : A \rightarrow B$. We say a hash function $h : A \rightarrow B$ is perfect on a subset $X \subseteq A$ if h is injective when restricted on X. Let w be an integer such that $2 \leq w \leq m$,and let $H \subseteq \{h : A \rightarrow B\}$. H is called an (n, m, w) *perfect hash family*(PHF) if for any $X \subseteq A$ with $|X| = w$ there exists at least one element $h \in H$ such that h is perfect on X.

It is proven in [7] that if there exists a $PHF(N, n, m, w)$, then there exists a re-keying scheme in which the number of KEKs for each user and the GC are N and Nm respectively and the number of broadcast transmissions to remove up to w users is less than $(m-1)N$.

It is also proven in [7] that an (N, n, d, m) erasure code gives rise to a $PHF(N, n, m, w)$ as long as $N > \begin{pmatrix} w \\ 2 \end{pmatrix}(N-d)$, thus can be used for the construction of the above re-keying scheme. Such a scheme can prevent w users from colluding. The performance of the re-keying scheme based on PHF is determined by the parameter N when w and m are fixed, which should be minimized to reduce the storage and transmission complexity. But the author didn't mention any details on which kind of error correcting code should be used and how it is used for the construction.

2.2 Re-keying scheme based on KDP

In [8], H. Kurnio reviewed the concept of Key distribution Patterns(KDP).

Let $X = \{x_1, x_2, ..., x_n\}$ and $B = \{B_1, B_2, ..., B_N\}$ be a family of subsets of X. The pair (X, B) is called an (n, N, t)-key distribution pattern if

$$|(B_i \cap B_j)\ \setminus\ \cup_{k=1}^{t} B_{s_k})| \geq 1$$

for any $(t+2)$-subset $\{i, j, s_1, ..., s_t\}$ of $\{1, 2, ..., N\}$.

With the idea of KDP, the author presented a theorem to show the existence of a multicast re-keying scheme with dynamic controller based on KDP. But how to effectively construct KDP is still an open problem.

Inspired by the work from [7] and [8], we look at the problem of multicast re-keying from the error-correcting codes point of view in this paper. In order to achieve constructions with feasible storage that do not require computational assumptions, we make an improvement on the constraints that must be satisfied to construct the broadcast encryption scheme in[7,8] by avoiding the requirement of

being PHF and KDP. Based on the OR model mentioned above and assumed a system with GC, we give two practical construction of schemes based on Reed-Solomon codes and avoid any computational assumptions. Conditions underlining the constructions are also given together with examples to show the detail constructions.

Kumar *et.al.* [10] also consider the blacklisting problem through error-correcting codes, but their method is quite different from ours.

3 Multicast Re-keying Scheme based on R-S code

3.1 Background on code

Let $GF(q)$ be a finite field.

Definition 3.1 (Linear Code) *An $[m, k, d]$ linear code is a k-dimensional subspace $V_{m,k}$ of m-dimensional linear space V_m over GF_q, where the minimum Hamming distance between any pair of elements is d.*

Reed-Solomon code is an important kind of linear block BCH codes which had been widely used in such areas as space communication systems, spread-spectrum communication systems and computer storage systems.

Definition 3.2 ($\text{Reed} - \text{Solomon}$ Code) *Let $x_1, ..., x_m \in GF(q)$ be distinct and $k > 0$. The $(m, k)_q$ Reed-Solomon code is given by the subspace $\{(f(x_1), ..., f(x_m))|f \in GF_{q,k}\}$, where $GF_{q,k}$ denote the set of polynomials on $GF(q)$ of degree less than k.*

R-S code is a Maximum Distance Separable (MDS) code, which means that the error-correcting capability of the R-S code can reach the Singleton bound. The R-S code has the property that the $(m, k)_q$ R-S code is an $[m, k, m - k + 1]_q$ linear code and it requires that $m \leq q$.

3.2 R-S code based construction

3.2.1 First construction

Theorem 3.1 *Let (N, k, d) be a Reed-Solomon code over $GF(q)$, where N is the length of the codewords, k is the length of the information bits and d is the minimum distance of the code. The number of the codewords $n = q^k$. Let W be a subset of $\{1, 2, ..., n\}$ with $|W| = w$. Then such an error-correcting code can be used to construct a multicast encryption scheme as long as it satisfies that*

$$N > w * (N - d).$$

Proof: Let T be the set of codewords of a (N, k, d) code, $|T| = n$. We write each element of T as $(c_{i1}, c_{i2}, ..., c_{iN})$ with $c_{ij} \in \{1, 2, ..., q\}$, where $1 \leq i \leq n, 1 \leq j \leq N$ and n is the number of codewords. For each j we define a function h_j from $A = \{1, ..., n\}$ to $B = \{1, ..., q\}$ by $h_j(i) = c_{ij}$ and let $H = \{h_j|j = 1, ...N\}$.

In the key initialization phase, The GC generates and stores a set of Nq keys defined as $K = \{k_{(h,b)} \mid h \in H, b \in B\}$. For a user $u_i, 1 \leq i \leq n$, GC secretly gives u_i the set of N Key Encryption Keys $K(u_i) = \{k_{(h,h(i))}|h \in H\}$.

In the broadcast stage of removing a set of users W from $U, |W| \leq w$, the GC randomly select a new session key and encrypt it with those KEKs that do not belong to W, then broadcast the encrypted messages to all the users. So those users that have been removed can not use their own KEKs to decrypt and obtain the new session key.

As to the decryption phase, we need to prove that any user u_i that does not belong to W has at least one key to decrypt and obtain the new session key.

Let $W = \{u_{i1}, ..., u_{iw}\}$. Since the minimum distance of the code is d, for any given pair of elements $x_1, x_2 \in U$, there are at most $N - d$ functions from H such that the values of these $N - d$ functions evaluated on x_1 and x_2 are the same. For any user $u_i \notin W$, it has at most $N - d$ functions that is the same as u_{i1}, at most $N - d$ same functions as u_{i2}, ... and at most $N - d$ same functions as u_{iw}. The worst case is that the same $N - d$ functions that u_i has with u_{i1} is different from those $N - d$ functions that u_i has with u_{i2}, which is different from those $N - d$ functions that u_i has with u_{i3}, ... That is, all the w (N-d) functions are different. So we conclude that if $N > w * (N - d)$, then u_i has at least one function that is different from all those functions belong to W. That is, there exists a function $h_\alpha \in H$ such that $\{h_\alpha(j)|j = i_1, i_2, ..., i_w, i\}$ are all distinct. It follows that $k_{(h_\alpha, h_\alpha(i))}$ is in $K(u_i) \subseteq K(U \setminus W)$, so u_i can decrypt the encrypted message and obtain the new session key $k^{U \setminus W}$. □

The theorem holds for any set L of members who wants to leave the original group as long as $|L| \leq w$.

Example 3.1 *Take the $(N, k, d) = (4, 2, 3)$ RS code over finite field $GF(4) = GF(2^2) = \{0, 1, \alpha, \alpha^2\}$. The primitive element α is the root of $x^2 + x + 1 = 0$. From theorem 3.1 we know that, if*

$$N - w(N - d) > 0,$$

then there exists a broadcast encryption scheme based on such a RS code, which means that $w < \frac{N}{N-d} = \frac{4}{4-3} = 4$.

Since $k = 2$, the information sequence is

$$\overline{m} = (m_1, m_2).$$

The codewords, that is KEKs for all users corresponding to all possible information sequence is shown in Table 1.

3.2.2 Discussion

1. From [7], it is also known that any (N,k,d) error correcting code gives rise to a PHF(N,n,m,w) which is proven to be effective to set up the multicast encryption scheme.

	m_2	m_1	h(0)	h(1)	$h(\alpha)$	$h(\alpha^2)$
u_1	0	0	0	0	0	0
u_2	0	1	1	1	1	1
u_3	0	α	α	α	α	α
u_4	0	α^2	α^2	α^2	α^2	α^2
u_5	1	0	0	1	α	α^2
u_6	1	1	1	0	α^2	α
u_7	1	α	α	α^2	0	1
u_8	1	α^2	α^2	α	1	0
u_9	α	0	0	α	α^2	1
u_{10}	α	1	1	α^2	α	0
u_{11}	α	α	α	0	1	α^2
u_{12}	α	α^2	α^2	1	0	α
u_{13}	α^2	0	0	α^2	1	α
u_{14}	α^2	1	1	α	0	α^2
u_{15}	α^2	α	α	1	α^2	0
u_{16}	α^2	α^2	α^2	0	α	1

Table 1: User KEKs constructed from (4,2,3) R-S code

There, it requires that the minimum distance of the error-correcting code d' satisfies that

$$d' > \frac{\left(\begin{pmatrix} w \\ 2 \end{pmatrix} - 1\right) N}{\begin{pmatrix} w \\ 2 \end{pmatrix}}.$$

That is,the minimum d' that satisfies the above inequality is

$$d' = \left\lfloor \frac{\left(\begin{pmatrix} w \\ 2 \end{pmatrix} - 1\right) N}{\begin{pmatrix} w \\ 2 \end{pmatrix}} \right\rfloor + 1.$$

While here, from the above theorem, since

$$N > w * (N - d),$$

the minimum distance d has to satisfy that

$$d > \left(1 - \frac{1}{w}\right) N.$$

That is, the minimum d that satisfies the above inequality is

$$d = \left\lfloor \frac{w-1}{w} N \right\rfloor + 1.$$

Because $d \leq d'$, the requirement for constructing the broadcast encrypting scheme using R-S code had been reduced since it is more easier to find such a RS code, which allows us to increase the length of the information bit k when the code length is fixed and further more to reduce the requirement for the bandwidth.

2. For any $[n, k, d]_q$ R-S code over finite field F_q, when $N = q$, $k = log_q n$, where n is the number of codewords,

$$d = N - k + 1 = q - log_q n + 1,$$

then from

$$N < w * (N - d),$$

we get

$$w < \frac{N}{N-d} = \frac{q}{log_q n - 1}.$$

Example 3.2 *Take an $[8, 3, 6]_8$ R-S code over finite field*

$$GF(8) = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$$

as an example. Since $q = 8, N = q = 8, k = 3, d = N - k + 1 = 8 - 3 + 1 = 6$, then the number of codewords $n = 8^3 = 512$ and

$$w < \frac{N}{N-d} = \frac{q}{log_q n - 1} = \frac{8}{3-1} = 4.$$

So the $[8, 3, 6]_8$ R-S code over finite field $GF(8)$ can be used to construct the secure broadcast scheme as long as the number of members who want to quit is less than or equal to 3 where $N > w(N - d) \leftrightarrow 8 > 3(8 - 6)$.

3.2.3 Extension of the scheme

In order to improve the communication efficiency, the OR scheme we discussed can be slightly modified with erasure code such that the bandwidth used by the GC for broadcasting information can be reduced.

An $[n, k, m]$ erasure code is a special class of error-correcting codes that allow recovery of a message if part of its messages($\leq (n - m)$) are damaged or erased during the transmission. An erasure code can be constructed using Reed-Solomon code over finite field $GF(q)$. The decoding procedure uses k pairs of $(e_i, p_v(e_i))$ to recover the original k information messages, where e_i is one of the field element over $GF(q)$ and $p(x) = v_0 + v_1(x) + ... + v_{k-1}(x^{k-1})$ is the polynomial for the R-S encoding.

The broadcast encryption scheme described in Theorem 3.1 can be modified as follows. In the broadcast phase, before broadcasting the new session key $k^{U \setminus W}$, an encoding procedure is first applied to the new session key. The new session key $k^{U \setminus W}$ was divided into t pieces $k^{U \setminus W} = (k_1^{U \setminus W}, k_2^{U \setminus W}, ..., k_t^{U \setminus W})$, then encodes them using $[Nm, t, \alpha]$ erasure code to obtain the codeword $C(k^{U \setminus W}) = (c_1, c_2, ..., c_{Nm})$. The GC uses all the KEKs that do not belong to the users of W to encrypt the corresponding components of $C(k^{U \setminus W})$ and broadcasts the encrypted messages to all the users. That is, the GC broadcasts

$$\{E_{k_i}(c_i) \mid k_i \in K, k_i \notin K(W)\}.$$

As long as each non-excluded user has at least α keys that can decrypt α messages of $\{E_{k_i}(c_i)\}$, he can then apply the erasure code to obtain the new session key. While for those users in W, same as before, they can not find the session keys.

Theorem 3.2 *The above scheme works as long as the following inequality holds:*

$$N - w(N - d) \geq \alpha.$$

For an $[n, k, m]$ erasure code over $GF(q)$, we expect k to be as large as possible in order to minimize the extra bandwidth n/k for the transmission. Actually, the basic scheme we discussed in Theorem 3.1 is a special case of using $[n, 1, 1]$ erasure code for the construction.

Example 3.3 *Consider the same example as in Example 3.1: the RS code $(N, k, d) = (4, 2, 3)$ over finite field $GF(4) = GF(2^2) = \{0, 1, \alpha, \alpha^2\}$. The primitive element α is the root of $x^2 + x + 1 = 0$. The KEKs for all users corresponding to all possible information sequence is shown in Table 1.*

For the above scheme to work, it needs that

$$N - w(N - d) \geq \alpha,$$

that is,

$$4 - w(4 - 3) \geq \alpha,$$

that is,

$$4 - w \geq \alpha.$$

For $\alpha = 1$, w can be 1,2 or 3. For $\alpha = 2$, w can be 1 or 2. And for $\alpha = 3$, w can only be 1. We take $w = 2$ and $\alpha = 2$ as an example.

We divide the new session key $k^{U\setminus W}$ into two parts $k^{U\setminus W} = (k_0^{U\setminus W}, k_1^{U\setminus W})$, then encodes $k^{U\setminus W}$ using a [16,2,2] erasure code to obtain a codeword

$$C(k^{U\setminus W}) = (c_1(0), c_2(1), c_3(\alpha), ..., c_{16}(\alpha^{14})),$$

where

$$c_i(e_i) = p(e_i), e_i \in GF(16)$$

and

$$p(x) = k_0 + k_1 x.$$

Suppose any two users $W = \{u_{s_1}, u_{s_2}\}, |W| = 2$ want to leave the group, the GC uses all the KEKs that do not belong to this two users to encrypt all these 16 pieces of encoded keys and broadcasts the encrypted messages to all the users. Then each user that is not in W has at least 2 keys to decrypt 2 messages, thus can recover the original new session key.

3.3 Second R-S code based construction

In [8], Hartono proposed a broadcast encryption scheme based on the KDP(Key Distribution Pattern) which can be used for dynamic GC. If the GC is fixed in the system and is trustable, then the condition for the scheme can be improved to make it work for general case.

3.3.1 Scheme description

Theorem 3.3 *Let $X = \{x_1, x_2, ...x_{n^*}\}$ be a set of KEKs, $U = \{U_1, U_2, ..., U_n\}$ be the set of $users'$ KEKs, which is a family of subset of X, that is, for $\forall\, i$, $U_i \subseteq X$. Let $W = \{U_{s_1}, U_{s_2}, ...U_{s_w}\}$ be a subset of U with $|W| = w$. If for $\forall\, i$, it satisfies that:*

$$|U_i \setminus \cup_{k=1}^{w} U_{s_k}| \geq 1,$$

Then a broadcast encryption scheme can be constructed which can remove up to w users from a group of n users.

In this scheme, The GC generates and stores a set X of KEKs in the key initialization phase, and sends each user u_i a subset $U_i \subseteq U$ of X as the user's KEKs. When a set of users W want to quit from the group, the GC selects a new session key $k^{U\setminus W}$ and encrypts the session key with all KEKs except those belong to users in W, that is, GC broadcasts $\{E_{k_r}(k^{U\setminus W}) \mid k_r \in X \setminus (U_{s_1} \cup U_{s_2} ... \cup U_{s_w})\}$. So, those users in W can not decrypt the encrypted message. While, since for $\forall u_i$ that $U_i \in X \setminus W$,

$$|U_i \setminus \cup_{k=1}^{w} U_{s_k}| \geq 1,$$

it has at least one key that does not belong to W, so it can decrypt $E_{k_r}(k^{U\setminus W})$ and obtain the new session key $k^{U\setminus W}$.

From the theorem we know that for any given w and n, we should make n^* as small as possible. Same, for any given w and n^*, we hope n to be as large as possible.

Next, we will show how to use Reed-Solomon code to construct the KEK set X and U and how the scheme works.

3.3.2 R-S code based construction

We take the R-S code (N, k, d) over the finite field $GF(q) = \{0, 1, \alpha, ...\alpha^{q-2}\}$, The number of users $n = q^k$. The RS codeword $\bar{c}$ of length N is generated from k information symbols taken from the finite field $GF(q)$ through polynomial

$$h(x) = m_0 + m_1 x + m_2 x^2 + ...$$
$$+ m_{k-2} x^{k-2} + m_{k-1} x^{k-1},$$

where

$$\overline{m} = (m_0, m_1, ...m_{k-1})$$

and

$$\bar{c} = (c_0, c_1, ..., c_{q-1})$$
$$= (h(0), h(1), h(\alpha), h(\alpha^2), ..., h(\alpha^{q-2})).$$

For each user u_i, the KEK set that corresponds to the k information symbols

$$\overline{m_i} = (m_{i1}, m_{i2}, ...m_{ik})$$

is

$$U_i = \{\big(h_i(0), h_i(1), h_i(\alpha), ..., h_i(\alpha^{q-2})\big)\},$$

where $|U_i| = q = N$. So, the KEK set for all users is

$$U = \cup_{i=1}^{n} U_i.$$

The total KEK set X for GC is

$$X = \{X_i,\ i = 1, 2, ..., n^*\},$$

where $n^* = N * q = q^2$, and

$$X_i = \{(h, \beta) \mid h \in \{h(0), h(\alpha), ..., h(\alpha^{q-1})\},$$

$$\beta \in GF(q).\}$$

Next, we will use an example to show the exact procedure on the RS-code construction of the scheme.

Example 3.4 *Take the same example as in Example 3.1: that is the* $(N, k, d) = (4, 2, 3)$ *RS code over finite field* $GF(4) = GF(2^2) = \{0, 1, \alpha, \alpha^2\}$. *The primitive element* α *is the root of* $x^2 + x + 1 = 0$. *The KEKs for all users corresponding to all possible information sequence is shown in Table 1. After extending the users' KEKs by use of the way shown in section 3.3.2, we obtain the KEKs set* $X = \{X_i, i = 1, 2, ..., 16\}$ *as shown in Table 2.*

From Table 2 we can see that,

$$U_1 = \{x_1, x_5, x_9, x_{13}\}$$

$$= \{(h_1, 0), (h_2, 0), (h_3, 0), (h_4, 0)\}$$

$$U_2 = \{x_2, x_6, x_{10}, x_{14}\}$$

$$= \{(h_1, 1), (h_2, 1), (h_3, 1), (h_4, 1)\}$$

$$U_3 = \{x_3, x_7, x_{11}, x_{15}\}$$

$$= \{(h_1, \alpha), (h_2, \alpha), (h_3, \alpha), (h_4, \alpha)\}$$

$$U_4 = \{x_4, x_8, x_{12}, x_{16}\}$$

$$= \{(h_1, \alpha^2), (h_2, \alpha^2), (h_3, \alpha^2), (h_4, \alpha^2)\}$$

$$U_5 = \{x_1, x_6, x_{11}, x_{16}\}$$

$$= \{(h_1, 0), (h_2, 1), (h_3, \alpha), (h_4, \alpha^2)\}$$

$$...$$

$$U_{16} = \{x_4, x_5, x_{11}, x_{14}\}$$

$$= \{(h_1, \alpha^2), (h_2, 0), (h_3, \alpha), (h_4, 1)\}.$$

All the KEKs hold by the GC is given by:

$$U = \cup_{i=1}^{16} U_i.$$

Suppose there are $w = 2$ *users who want to quit from the group* $\{u_1, u_2, ..., u_{16}\}$, *say users* $W = \{u_7, u_8\}$, *we can check that for each user* $u_i \notin W$,

$$|U_i \setminus \cup_{k=1}^{w} U_{s_k}| \geq 1.$$

For example,

$$\left|U_1 \setminus \cup_{k=1}^{2} u_{s_k}\right| = |\{x_1, x_5\}| = 2 \geq 1.$$

So such a set of KEKs can be used to implement the broadcast encryption scheme.

4 Construction of the Scheme using A-G code

Since the R-S code over $GF(q)$ requires the length of the codewords $N \leq q$, we can not make the codeword longer than q. Using Algebraic-geometric code(A-G code), the scheme can be extended to the case when codeword length $N > q$. Next we will show an example on how to use A-G code to construct the OR model for the multicast re-keying scheme.

4.1 A-G code

For those who are interested in more details about A-G code, please refer to the paper [11] and [12].

4.2 Example of A-G code based multicast re-keying scheme

Let us consider the Hermitian code over $GF(4) = GF(2^2)$ with $k = 2$. The Hermitian curve over $GF(2^2)$ is $x^3 + y^2 + y = 0$. The curve has rational points:

$$\{(0,0), (0,1), (1,\alpha), (1,\alpha^2),$$

$$(\alpha,\alpha), (\alpha,\alpha^2), (\alpha^2,\alpha), (\alpha^2,\alpha^2)\}$$

$$=^{\triangle} \{((x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4),$$

$$(x_5,y_5), (x_6,y_6), (x_7,y_7), (x_8,y_8)\}.$$

Let code polynomial be $c(x) = m_0 + m_1 x$, it has 16 codewords:

$$\overline{c} = (c(x_1,y_1), c(x_2,y_2), c(x_3,y_3), c(x_4,y_4),$$

$$c(x_5,y_5), c(x_6,y_6), c(x_7,y_7), c(x_8,y_8))$$

All the codewords, that is, the KEKs set are shown in Table 3.

In this example, $c(x)$ has at most 2 zero points, $d = 8 - 2 = 6$. since $q = 4, N = 8, d = 6$, the number of users $n = q^2 = 16$, the number of keys is $N * q = 8 * 4 = 32$.

For the OR multicast re-keying scheme to work, it requires that

$$N > w(N - d),$$

that is,

$$8 > w(8 - 6),$$

so w can be 2 or 3. Since w can be 3, from $N - w(N - d) = \alpha$, we know that α can be 2.

5 Conclusions

In this paper, two practical constructions for Multicast Rekeying Schemes using Reed-Solomon Codes are given with examples to show the detailed construction procedure. Because it has many properties we expected, RS code provides us a practical way to construct the multicast re-keying

	h_1	h_1	h_1	h_1	h_2	h_2	h_2	h_2	h_3	h_3	h_3	h_3	h_4	h_4	h_4	h_4
	0	1	α	α^2	0	1	α	α^2	0	1	α	α^2	0	1	α	α^2
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
u_1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
u_2	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
u_3	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
u_4	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
u_5	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
u_6	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0
u_7	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0
u_8	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0
u_9	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0
u_{10}	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0
u_{11}	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1
u_{12}	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0
u_{13}	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0
u_{14}	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1
u_{15}	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0
u_{16}	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0

Table 2: Construction of Re-keying scheme with (4,2,3) R-S code.

	m_2	m_1	h(0)	h(1)	$h(\alpha)$	$h(\alpha^2)$				
u_1	0	0	0	0	0	0	0	0	0	0
u_2	0	1	0	0	1	1	α	α	α^2	α^2
u_3	0	α	0	0	α	α	α^2	α^2	1	1
u_4	0	α^2	0	0	α^2	α^2	1	1	α	α
u_5	1	0	1	1	1	1	1	1	1	1
u_6	1	1	1	1	0	0	α^2	α^2	α	α
u_7	1	α	1	1	α^2	α^2	α	α	0	0
u_8	1	α^2	1	1	α	α	0	0	α^2	α^2
u_9	α	0	α	α	α	α	α	α	α	α
u_{10}	α	1	α	α	α^2	α^2	0	0	1	1
u_{11}	α	α	α	α	0	0	1	1	α^2	α^2
u_{12}	α	α^2	α	α	1	1	α^2	α^2	0	0
u_{13}	α^2	0	α^2	α^2	α^2	α^2	α^2	α^2	α	α
u_{14}	α^2	1	α^2	α^2	α	α	1	1	0	0
u_{15}	α^2	α	α^2	α^2	1	1	0	0	α	α
u_{16}	α^2	α^2	α^2	α^2	0	0	α	α	1	1

Table 3: User KEKs constructed from (8,2,6) A-G code.

scheme efficiently. The storage complexity and transmission complexity have been reduced at the same time, which is another advantage of the method proposed in this paper.

Because this paper is only an initial work for using RS-code in constructing the re-keying scheme, a lot of work is being done and will be done in the future such as how to improve the communication transmission efficiency by encoding the new session key with error correcting code first, how to deal with multiuser and multistage leaving from the group, how to handle when a new user is joining, but the members in the group has reached the maximum, how to apply AG code instead of RS code in the construction to improve the performance, how to make the GC be a group member also, how to extend these two schemes to apply for the distributed environment and so on.

References

[1] S.Berkovits. How to Broadcast a Secret. *Advances in Cryptology - EUROCRYPT'91, Lecture Notes in Computer Science 547*, p535-541, 1991;

[2] D.M.Wallner, E.J.Harder and R.C.Agee. Key Management for Multicast: Issues and Architectures. *Intenet Draft*, September, 1999;

[3] C.K.Wong, M.Gouda an dS.S.Lam, Secure Group Communication using Key graphs. *Proceedings of SIGCOMM'98*, p68-79, 1998;

[4] A.Perrig, Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. *CrypTec'99*, Hongkong, December, 1999;

[5] I.Chang, R.Engel, *et.al.*, Key Management for Secure Internet Multicast using Boolean Fnction Minimization Techniques. *INFOCOM'99*,September, 1999;

[6] A.Fiat and M.Naor, Broadcast Encryption. *Advances in Cryptology - CRYPT'92, Lecture Notes in Computer Science*, vol.773, p481-491, 1993;

[7] S.Naini R. and H.Wang, New Constructions for Multicast Re-Keying Schemes using Perfect Hash Families. *7th ACM Conference on Computer and Communication Security*, ACM Press,p228-234,2000;

[8] H.Harnio, R.S.Naini, W.Susilo and H. Wang, Key Management for Secure Multicast with Dynamic Controller. *Information Security and Privacy, 5th Australasian Conference, ACISP'00, Lecture Notes in Computer Science 1841*, p178-190,2000; September, 1998;

[9] R.Poovendran and J.S.Baras, An Information Theoretic Analysis of Rooted-Tree Based Secure Multicast Key Distribution Schemes. *IEEE Transactions on Information Theory*, May, 1999;

[10] R.Kumar, S.Rajagophlan and A.Sahai, Coding constructions for blacklisting problems without computational assumptions. *Advances in Cryptology - CRYPTO'99*, Lecture Notes in Computer Science, p609-623, 1999;

[11] G.L. Feng and T.R.N. Rao, Decoding Algebraic Geometric Codes up to the Design Minimum Distance. *IEEE Transaction on Information Theory*, vol.1, No.1, p37-45, Jan. 1993;

[12] G.L. Feng, V.K. Wei, T.R.N. Rao, and K.K. Tzeng, Simplified Understanding and Efficient Decoding of a Class of Algebraic-Geometric Codes. *IEEE Transaction on Information Theory*, vol.40, No.4, p981-1002, Jul. 1994;

Building and Managing Software Reuse Libraries

Zina Houhamdi
Computer Science Institute, University of Biskra, BP 145, Biskra, 07000, Algeria.
E-mail: z_houhamdi@yahoo.fr

Keywords: Reuse library, Taxonomy, Classification, Reuse Description Formalism, Software defects.

Received : October 1, 2002

Software reuse has been claimed to be one of the most promising approaches to enhance programmer productivity and software quality. One of the problems to be addresses to achieve high software reuse is organizing databases of software experience, in which information on software products and processes is stored and organized to enhance reuse.
The Reuse Description Formalism (RDF) is a generalization of the faceted index approach to classification. It was initially designed as a tool to help increase reusability of software components at the code level (e.g. functions or subroutines). The goal of this study is to show that RDF can also be used effectively to represent and reuse other types of software knowledge. The emphasis here is on those proprieties of RDF that facilitates the representation of these objects.
This paper demonstrates RDF's representation power by constructing sample classification taxonomy for software defects, and explains how this taxonomy can be used by the system tester to predict the types of defects associated with software components.

1 Introduction

Current software reuse systems based on the faceted index approach [14] to classification suffer from one or more of the following problems [3,9]: they are applicable to a restricted set of domains; they posses poor retrieval mechanisms; their classification schemes are not extensible; and/or they lack mechanisms for ensuring the consistency of library definitions. The primary contribution of this study is the design and implementation of the Reuse Description Formalism, which overcomes these problems.

- *RDF* is applicable to a wide range of software and non-software domains. The *RDF* specification language is capable of representing not only software components at the code level, but it is also capable of representing more abstract or complex software entities such as projects, defects, or processes. What is more, these software entities can all be made part of one software library and can be arranged in semantic nets using various types of relations such as "is-a", "component-of", and "members-of" [4].
- *RDF* provides an extensible representation scheme. A software reuse library system must be flexible enough to allow representation schemes to evolve as the needs and level of expertise in an organization increases. The *RDF* specification language provides several alternatives to extend or adjust a taxonomy so as to allow the incorporation of new objects into the library without having to classify all other objects [5].
- *RDF* provides a consistency verification mechanism. Most software reuse library systems are based on representation models, which must satisfy certain basic predicates for the library to be in a consistent state. The *RDF* specification language includes an "assertion" mechanism whose purpose is to help specify and ensure the consistency of the object descriptions contained in a library.

In short, *RDF* addresses the main limitations of current faceted classification systems by extending their representation model.
The remaining of this paper presents a detailed definition of the *RDF* system. It introduces the concepts behind *RDF*'s representation and similarity models by developing a sample software reuse library. These concepts were formalized [10].
To create and organize reuse library, an extensive domain analysis must be performed beforehand [13]. This analysis must produce a classification scheme (including attributes and their types) as well as an approximate measure of similarity between objects.
This section develops a small software library to classify operations to manipulate data structures consisting of repeated element (e.g., stacks, trees, Hash tables). For representation purposes we start with a trivial library and enhance it as more features of *RDF* are introduced.

2 Creating taxonomy

Booch [2] classifies operations over a data structure in the following three classes, based on how the structure is accessed.

- Constructors: operations that alter the data structure.
- Selectors: operations that evaluate the data structure.
- Iterators: operations that visit all element of the structure.

We can describe this simple classification scheme by defining an attribute called function as follows:

Attribute function : {construct, select, iterate};

Another attribute for classification of operations is execution time as a function of the size of data structure.

```
Attribute timing: {constant, log, linear, loglinear, quadratic,
                   slow};
```

Attributes function and timing define a simple classification scheme that can be used to describe four operations for stack manipulation. Each of these descriptions is called *instance*.

```
Push = [function = constructor & timing = constant];
Pop  = [function = constructor & timing = constant];
Top  = [function = select      & timing = constant];
New  = [function = constructor & timing = constant];
```

This section has introduced two basic concepts of *RDF* language: attributes and instances. The type associated with both attributes is an enumeration of terms. Each instance defines the attribute values of a particular data structure operation.

3 Extending Taxonomy

The characterization of the functionality of operation presented above is too coarse. In fact, the descriptions of push, pop and new are identical. This section refines this characterization by extending the classification scheme. There are at last three approaches to do this.

- Add or replace terms in the type of attribute.
- Add more attributes.
- Describe attribute values in terms of more primitive attributes.

The first two approaches are common practice while designing a taxonomy and the only alternatives a library designer has with other classification systems such as AIRS or faceted classification system. The third approach is unique to *RDF*, and allows the construction of hierarchical classification system.

3.1 Adding values to a type

In this approach, the classification scheme is refined by including additional values to the type of an attribute. In particular, we add new terms to the functionality attribute. In the context of data structures consisting of repeated elements, the constructor term will be replaced by three new terms create, insert, and remove. With this new definition we can now tell push from pop and tell those from new. The updated definitions are as follows:

```
Attribute function : {create, insert, remove, select, iterate};
Push = [function = insert  & timing = constant];
Pop  = [function = remove  & timing = constant];
Top  = [function = select  & timing = constant];
New  = [function = create  & timing = constant];
```

This drawback of this approach is that instance definitions had to be manually modified (e.g., changing construct by the corresponding new term in each instance). Moreover, these extensions create flat taxonomies with few attributes and many terms, instead of hierarchies.

3.2 Adding attributes

In *RDF*, it is possible to define a new attribute and then use it to refine the classification of selected instances. Unlike other faceted classification system, this new attribute does not have to be used in all instances. Hence, the addition of attributes requires modifying only those instances for which the new attribute is meaningful and important.

For example, we extend the taxonomy by adding a new attribute called *exception*. This attribute is used to describe those operations that can signal a fatal exception such as a stack overflow or underflow. The following definitions are added or modified in our library:

```
Attribute exception : {underflow, overflow};
Push = [function = insert & timing = constant & exception =
        overflow];
Pop  = [function = remove & timing = constant & exception
        = underflow];
```

Only those operations that can generate an exception (push and pop) have been described using the attribute exception. The remaining in the library (top and new) were not modified and, therefore, have no defined value for the attribute exception. It can be argued that the attribute exception could have been defined with an additional term called *noexception* to describe those operations that do not generate exceptions. In this solution, all instances would been defined using the same set of attributes and therefore a system like AIRS could still be used to model our taxonomy. Although this argument is valid in the current example, in fact that *RDF* can handle descriptions with different sets of attributes in particularly important in the case of libraries containing objects of different classes such as "project", "systems", "packages", and "operations". The attributes of these sample classes are most probably disjunct, but they can all be classified in a single library.

3.3 Describing values of an attribute

RDF provides a new approach to extend a classification scheme: describe all terms of an attribute using more primitive attributes. This process is illustrated by refining again the functionality attribute. Within the domain of data structure consisting of repeated elements, the functionality is described in term of three new attributes: access (whether the data structure is written or only read), target (which elements are affected), and newsize (how the number of elements varies).

```
Attribute access  : {write, read};
Attribute target  : {leftmost,rightmost,keyed,any,all,none};
Attribute newsize : {increase, decrease, reset, same};
```

These new attributes are used to define each of terms that belong to the attribute functionality.

```
Create  = [in constructors & newsize=reset & target=none];
Insert  = [in constructors & newsize = increase];
Remove  = [in constructors & newsize = decrease];
```

```
Select    = [in selectors];
Iterate   = [in iterators];
```

Where constructor, selectors, and iterators each define a class of instances. The class mechanism is used both as an abstract mechanism and, also, as an abbreviation for expressions. These classes are defined as follows:

```
Constructors = class (access = write);
Selectors    = class (access = read & newsize = same);
Iterators    = class (target = all);
```

The definition of the attribute functionality can now be changed, because its element no longer belong to enumeration type to a class of instances, namely the class of instances defined in terms of one or more of the attributes access, target, and newsize.

```
Attribute function : class (has access | has target | has
                            newsize);
```

Since all former terms of attribute function are defined, instances described using these values (e.g., push) do not need to be redefined. That is, this extension of the classification system does not affect the classification of objects already in the library.
This extended classification scheme allows us to define new categories of functionality. For example, we can define modify as a possible value of functionality, and also describe more specific iterators.

```
Modify           = [in modifiers];
Passive_iterate  = [in iterators & in selectors];
Active_iterate   = [in iterators & in constructors];
Modify_iterate   = [in iterators & in modifiers];
Modifiers        = class (access = write & newsize = same);
```

Where modifiers is the class of all operations that update elements in the data structure.
In summary, the process required to extend a classification scheme by redefining the terms of the attribute is as follows:

1. Select an attribute *a* whose terms are to be refined. Let T be the type of *a*. In the example, *a* = functionality and T = {create, insert, remove, select, iterate}.
2. Perform a domain analysis on the domain of the terms of a. From this analysis, define a set A of new attributes that describe terms in T, and determine the type for each attribute in A. In the example, A = {access, target, newsize} with their corresponding term enumerations.
3. Redefine attribute a. possible values for a are not terms as before (type T is no longer part of the library), but instances that belong to a class defined using the attributes in A.
4. Define each former term $t \in T$ as an instance using the attributes in A, following the same procedure used to describe data structure operations.
5. If needed, other values for a can be described. This values can be specializations of former terms (e.g., passive_iterate) or they can represent new concepts (e.g., modify).

In principle, this process of refinement can be done indefinitely providing deep hierarchical taxonomies, but there is a point in which using this formalism is no longer useful (e.g., do not use *RDF* to describe detailed functionality, including pre- and post-conditions).

4 Creating object hierarchies

Reusable software usually consists of packages or modules, made from operations and their packages. We want to represent this modular structure, but we do not want to force any granularity of reuse. That is, we want to have a library consisting of packages and operations, assuming that both complete packages and isolated operations will be reused. The following declarations define the kinds of reusable software components for a library of data structure packages. Because a package can have several subunits, the subunits attribute has a set type.

```
Attribute subunits : set of components;
Attribute parent   : packages;
Components         = class (in packages | in operations);
Packages           = class (has subunits);
Operations         = class (has function | has timing);
```

Two other attributes for packages are defined: maxsize (whether there are limits in the number of elements of the structure) and control (whether concurrent access is supported).

```
Attribute maxsize : {bounded, limited, unbounded};
Attribute control : {sequential, concurrent};
```

With these declarations, a stack package comprising the operations already described can be defined using one extra attribute (parent). The implementation has no preset bound on size and does not provide support for concurrency.

```
Stack = [subunit = set (parent = stack) & maxsize =
         unbounded & control = sequential];
Push  = [parent = stack & function = insert & timing =
         constant & exception = overflow];
Pop   = [parent = stack & function = remove & timing =
         constant & exception = underflow];
Top   = [parent = stack & function = select & timing =
         constant];
New   = [parent = stack & function = create & timing =
         constant];
```

Where the construct "set (parent = stack) denotes the set of all instances defined in the library for which the attribute parent is equal to stack, in other words, the set {pop, push, top, new}.

5 Dependencies among attributes

All classification schemes assume that certain semantic relations between attributes values are being maintained. For this purpose, *RDF* provides a mechanism that uses assertions to define semantic constrains between attribute values. For example, consider the case of attributes describing the functionality of an operation. If the data structure is not written then there is no size change, and if the structure is reset then there is no specific target. These two relations can be expressed as follows:

Assertion access = read ⇒ newsize = same;
Assertion newsize = reset ⇒ target = none;

In addition, the attribute maxsize and control are only relevant for packages, and all units that declare a package as their parent must indeed be subunits of the package.

Assertion has maxsize | has control ⇒ in package;
Assertion in packages ⇒ subunits (parent = *self*);

The keyword *self* denotes the instance being analyzed for compliance with the assertion.

6 Defining synonyms

One of the difficulties of describing operations given our current taxonomy is remembering the precise terms used in the library. Besides, certain concepts can be given or referenced by more than one name. The introduction of synonyms for terms has been suggested as a partial solution to this problem.

One could declare that distance between two terms is zero, making them synonyms from the point of view of queries based on similarity. However, queries based on exact matches will considered them different. In *RDF* is possible to declare an identifier i_1 to be a synonym of an identifier i_2 by simply declaring $i_1 = i_2$. For example:

Update = write; Preserve = read;

These definitions introduce the synonyms update and preserve for the terms write and read of attribute access, respectively.

7 Queries and comparing objects

In order to find reusable software components in the library of packages and operations; it is necessary to define the distance values associated with the terms of enumerations types.

This allows *RDF* to compute distances not only between these terms, but also between instances defined using these terms. Distances between terms are defined with a distance clause. For example attribute access and newsize and their distance clauses are given below. The distances shown here are just sample values. {The process of assigning distances is not described in this paper because the emphasis is not on how to define similarity distances between object}.

Attribute access : {write, read}
Distance {write → read: 4 , read → write: 6};

Attribute newsize : {increase, decrease, reset, same}
Distance {increase → decrease: 5, same: 7, decrease → increase: 5, reset: 3, reset → same: 10, same → reset: 10};

By transitivity, we can determine other distance not explicitly given. For example, the distance from increase to reset is 5 + 3 = 8, and the distance from decrease to same is 12. Note that a bigger value for this distance (13) can be obtained going from decrease to reset to same, but *RDF* always uses the smallest value.

Basically, the distance between two instances is computed by adding the distances of their corresponding attribute values. For example, the distance from remove to select is 16, given by the distance from write to read (4) plus the distance from decrease to same (12).

Remove = [access = write & newsize = decrease];
16 = 4 + 12
Select = [access = read & newsize = same];

Distances between instances are used by *RDF* to select reuse candidates from a library. This selection is performed using the query command. For example, the following query finds components that are similar to an operation that retrieves an arbitrary element from a data structure in at most logarithmic time.

Query function = [in selectors & target = any] & timing = log;

Consider another example. Find a data structure with three operations: one to initialize, one to insert an element, and one to traverse the structure without modifying it; concurrent control is not needed, but the structure must be able to handle an unbounded number of elements.

Query maxsize = unbounded & control = sequential &
Subunits = {[function = create], [function = insert], [function = passive_iterate]};

In this query, only the functionality of the operations has been specified. Attribute timing is not defined; meaning that any value for timing is equally acceptable in the retrieved operations.

8 Sample *RDF* taxonomy

RDF was initially designed as a tool to help increase reusability of software components at the *code level* (e.g. functions or subroutines). The goal of this section is to show that *RDF* can also be used effectively to represent and reuse other types of software knowledge. This section includes a taxonomy for representing software defects, and explains how *RDF* library of software defects can help a system tester.

One obvious necessity of software systems is the ability to function without defects. Traditional software construction processes have specific subprocesses to detect defects (e.g., "unit test", and "acceptance test"). However, detecting faults is not enough: to reduce the

number of defects associated with a product and its development process requires the ability to explain and predict them. The ability to explain a defect helps to find its source, thus reducing the coast associated with its correction. In addition, being able to predict defects in a software system helps to select processes, methods and tools to avoid defects of a particular kind, reducing the need for later detection and correction procedures. Prediction also helps to improve the effectiveness of testing mechanisms by increasing the chances of finding defects.
In order to explain and predict software defects, we need to characterize the different kinds of defects associated with a particular software environment and project [1].

3.1 Characterizing defects using *RDF*

A software product can be defined by two distinct types of entities [1,15]: data and processes. The first attribute we use to discriminate among defects is whether they are directly associated with processes or with documents. If a defect is related to document, it is called a fault. If it is related to process, it is called either a failure or an error: failures are associated with processes that are performed automatically and errors are associated with human processes.
The attribute entity classifies the kind of entity (either data or process) in which the defect occurs. The attribute creator classifies the creator or agent of that entity (either computer or human). These attributes are used to define faults, errors, and failures.

```
Attribute entity    : {data, process};
Attribute creator   : {computer, human};
Defects             = class (has entity | has creator);
Faults              = class (entity = data);
Failures    = class (entity = process & creator = computer);
Errors      = class (entity = process & creator = human);
```

Cause of defects. Failure, faults and errors are interrelated. Failures are caused by one or more faults (system failures are also caused by environmental accidents; here we only consider software related failures). For example, a failure during the execution of a program is caused by a fault in the program. Faults in a document are the consequence of defects in the processes that create the document or in the date used by these processes. For example, failure in a software tool can produce a fault in a document. The cause attribute describes these relationships. Because we do not model human processes, this attribute does not apply to errors.

```
Attribute cause        : set of defects;
Assertion has cause ⇒ in failures |in faults;
```

Severity of a defect. Another way to characterize defects is by their severity: this information helps prioritize activities aimed at correcting defects. We distinguish four levels of severity: fatal (stops production or development completely), critical (impacts production or development significantly), noncritical (prevents full use of features), and minor.

```
Attribute severity : {fatal, critical, noncritical, minor};
```

Defects and the Lifecycle. We are interest in determining when and where a defect enters the system and when it is detected. Because the phases of the lifecycle are related to documents (e.g., the requirements phase is related to the requirement document), we use phases to measure the time at which errors and failures occurs as well as to determine the (kind of) document in which a fault occurs. The occurrence attribute relates a defect to phase at which it is detected. We explicitly declare the phase type that is used in these two attributes.

```
Type phase = {requirement, specification, design, coding,
      unit_test, integration, operation, integration_test,
      acceptance_test, maintenance};
Attribute occurrence : phase;
Attribute detection  : phase;
```

So far we have defined attributes to characterize defects in general. The remaining analysis defines specific kinds of failures, faults, and errors.

Kinds of failures. A failure occurs during the execution of either the software product or a software tool. Our focus is on failures associated with the execution of a particular kind of software product: implementation of data structures.

```
Attribute failure_kind : {overflow, underflow, illegal_access,
    wrong_output, infinite_loop, tool_failure};
Assertion has failure_kind ⇒ in failures;
```

Kinds of faults. Faults are defects in documents: they occur in executable documents (i.e., code) and also in other types of documents. Again, our focus is on documents interpreted by the computer, so we consider only faults on those documents.

```
Attribute fault_kind : {control_flow, algebraic_computation,
    data_use, data_initialization, data_definition, interface };
Assertion has fault_kind ⇒ in faults;
```

In general it is difficult to isolate defects in documents. However, if a particular area in a document contains a defect, one is interested in knowing whether something is missing (omission) or something is wrong (commission). We use the fault_mode attribute to distinguish between these two cases.

```
Attribute fault_mode        : {omission, commission};
Assertion has fault_mode ⇒ in faults;
```

Kinds of errors. Defects introduced by humans (i.e., errors) are ultimately the cause of the most other type of defects in a software product; hence understanding their nature is critical. On the other hand, a complete

characterization of errors involves modeling human processes, which is out of the scope of this work. We simply characterize errors by the particular domain that is misunderstood or misused, using the error_kind attribute.

Attribute error_kind : {appilication_area, problem_solution, syntax, semantics, environment, clerical};
Assertion has error_kind ⇒in errors;

3.2 Sample descriptions

The following examples of defects and their characterization use the proposed classification scheme. The particular software project is the construction of a package to manipulate Hash tables.

Case 1. Consider a programmer coding a particular function, which according to the specifications must receive as, input two integer arguments. The programmer understands exactly what must be implemented, but mistakenly declares the function with only one formal argument. This fault is detected while reading code during unit testing. These defects are classified as follows:

Fault_1 = [in fault & occurrence = coding & detection = unit_test & severity = critical & cause = {error1} & fault_mode = omission & fault_kind = interface];
Error1 = [in error & error_kind = clerical];

Case 2. Consider the case that deletions in a Hash table do not always reclaim storage. This causes a system crash during operation due to an overflow in a Hash table; the problem is corrected promptly by reformatting the table. The specific problem is that a code optimizer swapped two statements. These defects are classified as follows:

Failure_2 = [in failures & severity = noncritical & occurrence = operation & cause = {Swapped_stmt} & failure_kind = overflow];
Swapped_stmt = [in faults & severity = critical & occurrence = coding & detection = operation & cause = {Failure_op} & fault_kind = control_flow & fault_mode = commission];
Failure_op = [in failures & occurrence = coding & detection = operation & failure_kind = tool_failure];

3.3 Explaining and predicting defects

Having a database with software components, software defects, and their interrelations are useful to explain and predict defects. These explanations/predictions are not automatic: they are done by a person who obtains relevant information using queries to the database. (We assume that distances between terms of all attributes are defined.)

The following is a description of a failure that has been diagnosed as an overflow in a data structure; this failure occurred during integration test.

Overflow_fail = [in failures & severity = fatal & occurrence = integration-test & failure_kind = overflow];

We do know the kind of fault that caused overflow_fail, so we query the database for faults that have caused failures using the following query command.

Query in faults & occurrence = coding & cause = {overflow_fail}.

To predict defects in packages, defects descriptions must be integrated with package descriptions in a single database. We relate packages with their faults (and thus indirectly with errors and failures) by adding attributes to both packages and faults. The docum attribute for faults is the package in which the fault occurs; the fault_set attribute for package describes the set of know faults.

Attribute docum : Packages;
Assertion has docum ⇒ in faults;
Attribute fault_set : set of faults;
Assertion has fault_set ⇒ in Packages & fault_set = set (docum = self);

Assume that we want to predict the kinds of defects that may be associated with the hashing data structure package. The following query retrieve packages that are similar to the Hash package. The subunits are assumed to be already defined.

Query maxsize = bounded & control = sequential & subunits = {hash_create, hash_insert, hash_lookup, hash_delete};

Assuming that similar packages will have similar defects, we can use the faults of the retrieval packages to predict the faults that may occur in the Hash package.

9 Conclusion

In summary, we have presented a software reuse library system called *RDF* and show how its representation model overcome the limitations of current reuse library systems based on faceted representations of objects [3,8]. *RDF* overcomes part of the limitations of current faceted system by extending the their representation model. Two main concepts form the core of *RDF*'s representation model: instance and classes. Instances are descriptions of reusable objects, while classes represent collections of instances with a set of common properties. Objects are described in terms of attributes and associated values. Unlike faceted classification, which is limited to having only terms as attribute (facet) values, *RDF* allows attributes values to be instances and even sets of instances.

This generalization can be used to create one-to-one, one-to-many, and many-to-many relations between different object classes within a library. In other words, *RDF*'s specification language [5] is powerful enough to represent a wide variety of software domains, ranging from standard software components such as data structure packages and their operations, to more complex

domains such as software defects and software process models. In addition, *RDF* language provides facilities for ensuring the consistency of the libraries.

We have already study tree other domains to demonstrate *RDF*'s representation power by representing taxonomy definitions of various software domains. First, it includes taxonomies for describing components of a commercial software library called the EVB GRACE library [6] and a library for Command, Control, and Information Systems (CCIS) developed at Contel Technology Center [9]. Second, it includes a taxonomy for describing software evaluation models using GQM (Goal/Question/Metric) paradigm [7]. Finally, it presents taxonomy for describing software process models.

Yet, no evaluation has been performed on RDF's similarity-based retrieval mechanism. Towards this end, we are currently developing a reuse software library–based on information contained in the software engineering laboratory (SEL) database [11]. This database contains thousands of records containing functional and structural descriptions, a well as statistical data, related to hundreds of projects developed at the NASA Goddard Space Flight Center. In addition, this database contains information regarding the origin of the project components [12], which indicates whether they were implemented from scratch or by reusing other components at NASA. This reuse history will allow us to evaluate our similarity-based retrieval mechanism by comparing the reuse candidates it proposes with the ones that were actually used at NASA.

References

[1]. V. Basili and Rombach (1987) Tailoring the Software Process to Project Goals and Environments. *In proceedings of the 9th International Conference on Software Engineering*, IEEE Computer Society Press, California, pages 345-357.

[2]. G. Booch (1987) Software Components with Ada, *Benjamin-Cumming Publishing Company*, Menlo Park, California.

[3]. Z. Houhamdi and S. Ghoul (2201) A Reuse Description Formalism, *ACS/IEEE International Conference on Computer Systems and Applications, AICCSA'2001, Lebanese American University*, Beirut, Lebanon, pp. 25--32.

[4]. Z. Houhamdi and S. Ghoul (2001) A Classification System for software reuse, *Fifth International Symposium on Programming System, ISPS2001, USTHB Computer science Institute,* Algiers, Algeria ,pp. 339—345.

[5]. Z. Houhamdi (2001) A Specification language for software reuse, *CSS/IEEE Alexandria Chapter. 11th International Conference On computers: Theory and Application, ICCTA2001, Head of Electrical Control,* Alexandria, Egypt, pp. 125-133.

[6]. Z. Houhamdi (2001) Developing a Reuse Library, *CSS/IEEE Alexandria Chapter. 11th International Conference On computers: Theory and Application, ICCTA2001, Head of Electrical Control*, Alexandria, Egypt, pp. 134--148.

[7]. Z. Houhamdi (2001) An adaptative approach to reuse software, *SCS/IEEE 2001. The third Middle East Symposium on Simulation and Modeling, MESM'2001, Amman University*, Amman, Jordan, pp.108--113 .

[8]. Z. Houhamdi (2002) Software Reuse: a new classification approach, *The International Symposium on Innovation in Information and Communication Technology, ISIICT'2001, Philadelphia University*, Amman, Jordan, pp. 247--258.

[9]. Z. Houhamdi and S. Ghoul (2001) Classifying software for reusability. *Mail of technical and scientific knowledge.* Periodic magazine of the university of Biskra, Algeria, N°01, pp.41-47.

[10]. Z. Houhamdi and S. Ghoul (2002) RDF : A Formalism for reusing software. *The South African Computer Journal, SART/SACJ, N° 29*, (accepted).

[11]. R. Kester (1990) SEL Ada reuse analysis and representation, *Technical Report, NASA Space Flight Center*, Greenbelt, Maryland.

[12]. Software engineering Laboratory (SEL) database Organization and User's guide (1990), *NASA Goddard Space Flight Center, Greenbelt, Maryland.*, revision 1 edition.

[13]. R. Prieto-Diaz (1987) Domain analysis for software reusability, *In proceedings of the 11th international Computer Software and applications Conference (COMPSA'87). IEEE Computer Society Press*, pp. 23--29.

[14]. R. Prieto-Diaz (1991) Implementing faceted classification for software reuse, *Communication of the ACM* pp. 88--97.

[15]. P.A. Straub (1992) The nature of Bias and Defects in the Software Specification Process. *Ph.D. thesis, computer Science Department*, University of Maryland

Deriving Self-Stabilizing Protocols for Services Specified in LOTOS

Monika Kapus-Kolar
Jožef Stefan Institute, POB 3000, SI-1001 Ljubljana, Slovenia
monika.kapus-kolar@ijs.si

Keywords: distributed service implementation, automated protocol derivation, LOTOS

Received: September 6, 2002

A transformation is proposed which, given a specification of the required external behaviour of a distributed server and a partitioning of the specified service actions among the server components, derives a behaviour of individual components implementing the service. The adopted specification language is close to Basic LOTOS. Unlike in other protocol derivation algorithms based on LOTOS-like languages, distributed conflicts in the given service are allowed, and resolved by self-stabilization of the derived protocol.

1 Introduction

In top-down distributed systems design, one of the most difficult transformations is decomposition of a process into a set of co-operating subprocesses. Such a transformation is considered correct if it preserves, to the required degree, those actions of the process which are considered essential. Such actions are often referred to as the *service* that the process offers to its environment, i.e. the process is observed in the role of a *server*.

A service consists of atomic service actions, of which the most important are *service primitives*, i.e. atomic interactions between the server and its users, executed in *service access points*. In addition, one might decide to introduce some *hidden service actions*, to represent various important events within the server.

When decomposing a server, the first step is to decide on its internal architecture. It can be represented as a set of *server components* (e.g. one component per service access point), with channels for their communication. We shall assume that all components are on the same hierarchical level, for a multi-level architecture can always be obtained by gradual decomposition.

The next step is to assign individual service actions to individual server components, paying attention to the location and capability of components.

The final step is to specify details of the inter-component communication, i.e. to derive an efficient *protocol* implementing the service, where efficiency is measured in terms of the communication load. While the first two steps require creative decisions, protocol derivation can be *automated*. Given a formal specification of the architecture of a server, of its service and of its distribution, one can mechanically decide on the protocol exchanges necessary to implement the specified distributed causal relations and choices between service actions.

A protocol is typically much more complex than the service it implements. Besides, one usually does not care about the exact nature of an otherwise satisfactory protocol. Therefore, algorithms for automated protocol derivation are most welcome! They automate exactly that part of server decomposition which is the most difficult for a human, requiring simultaneous reasoning about the numerous co-operating parties.

Even if one decides for automated protocol derivation, it remains possible to strongly influence the resulting protocol, by introducing dummy hidden service actions. For example, introducing a pair of consecutive service actions executed by two different server components introduces a protocol message from the first to the second component. Prefixing each of a set of alternatives by a service action at a particular component makes the choice local to the component. In other words, instead of spending time on protocol design, one should rather concentrate on detailed service design, specifying all important dynamic decisions as explicit service actions [16]. By various allocations of the actions to server components, service implementations with various degrees of centralization are obtained.

A prerequisite for automated protocol derivation is that the service is specified in a formal language. It is desirable that the derived behaviours of individual server components are specified in the same language as the service, so that the same algorithm can be used for further decomposition.

It is desirable that a protocol derivation algorithm is to a large extent compositional, so that it can cope with large service specifications, provided that they are well structured. Moreover, a compositional algorithm reflects the service structure in the derived protocol specification, increasing the service designers' confidence into the automatically generated implementation.

It is difficult to construct a general protocol derivation algorithm with high-quality results and low complexity. Typical algorithms work on small classes of service specifications.

Protocol synthesis has been subject to intensive research since the middle eighties. An exhaustive survey can be found in [26], so we provide no systematic review of the

existing methods and refer to them only where necessary for comparison with the proposed solutions.

The protocol derivation transformation proposed in our paper is an enhancement of that in [10]. As in [10], we assume that a server consists of an *arbitrary fixed number of components* exchanging the necessary protocol messages *asynchronously*, over reliable, unbounded, initially empty first-in-first-out (FIFO) channels with a finite, but unknown transit delay. The adopted specification language is a syntactically simplified sublanguage of LOTOS [7, 2], a standard process-algebraic language intended primarily for specification of concurrent and reactive systems. Service primitives are not allowed to carry parameters, neither do we allow specification of real-time constraints. However, the principles for enhancing a basic protocol derivation method to cope with data and real time are well known [11, 12, 23].

For a service containing distributed conflicts, a precise implementation takes care that they never cause divergence in service execution. Firstly one should try to make all conflicts local to individual components, by inserting auxiliary hidden service actions, but that is acceptable only as long as no external service choice is undesirably converted into an internal server choice. For the remaining distributed conflicts, divergence prevention requires extensive inter-component communication [9, 20, 21]. Although even such protocols can be derived compositionally [17], the communication costs they introduce are usually acceptable only if exact service implementation is crucial or during the periods when server users compete strongly for the service. In a typical situation, the probability of a distributed conflict is so low that divergence should rather be resolved than prevented.

In LOTOS, there are two process composition operators allowing specification of service actions in distributed conflict, the operator of choice and the operator of disabling. In [10], only local choice is allowed. For disabling, the derived protocols are supposed to *self-stabilize after divergence*, but the proposed solution is not correct in the general case [15]. Besides, [10] has problems with implementation of parallel composition [15]. In an unpublished response to [15], Bochmann and Higashino proposed some solutions for the problems, but have not integrated them into their protocol derivation algorithm and have not been able to specify the solution for disabling in LOTOS.

We specify self-stabilization upon disabling purely in the adopted LOTOS-like language, and also suggest how to implement distributed choice. Further improvements over [10] are implementation solutions for processes with successful termination as a decisive event, for processes which might enter inaction without first declaring successful termination, for combining terminating and non-terminating alternatives, for process disabling with multiple initiators, and for interaction hiding and renaming. The proposed solutions can be seen also as an improvement over [3], another algorithm for the purpose in which we have identified a bug [15].

Name of the construct	Syntax
Specification	$w ::= \mathbf{spec}\ b\ \mathbf{where}\ D\ \mathbf{endspec}$
	$D ::= \underline{\text{set}}\ \underline{\text{of}}\ d$
Process definition	$d ::= p(x)\ \mathbf{is}\ b \mid p\ \mathbf{is}\ b$
Process name	$p ::= \mathit{ProcIdentifier}$
Parameter name	$x ::= \mathit{ParIdentifier}$
Behaviour	$b ::=$
Inaction	$\mathbf{stop}$
Successful termination	$\mid \delta$
Sequential composition	$\mid b_1 \gg b_2$
Action prefix	$\mid a; b_2$
Choice	$\mid b_1 [] b_2$
Parallel composition	$\mid b_1 \mid [G] \mid b_2$
Disabling	$\mid b_1 [> b_2$
Hiding	$\mid \mathbf{hide}\ G\ \mathbf{in}\ b_1\ \mathbf{endhide}$
Renaming	$\mid \mathbf{ren}\ R\ \mathbf{in}\ b_1\ \mathbf{endren}$
Process instantiation	$\mid p(v) \mid p$
	$G ::= \underline{\text{set}}\ \underline{\text{of}}\ g$
Interaction gate	$g ::= s \mid h$
Data value	$v ::= \underline{\text{term}}\ \underline{\text{of}}\ \underline{\text{type}}\ n^*$
Index	$n ::= 1 \mid 2$
	$R ::= \underline{\text{set}}\ \underline{\text{of}}\ r$
Gate renaming	$r ::= g'/g$
Action	$a ::= \mathbf{i} \mid s \mid h \mid ho$
Service primitive	$s ::= u^c$
Service-primitive type	$u ::= \mathit{PrimIdentifier}$
Server component	$c ::= \mathit{CompIdentifier}$
Auxiliary gate	$h ::= \mathbf{s}^c_{c'} \mid \mathbf{r}^c_{c'} \mid \mathbf{a}^n_{c'} \mid \mathbf{b}_{c'} \mid \mathbf{t}$
Data offer	$o ::= !v \mid ?v \mid ?x{:}v$

Table 1: The adopted specification language

The paper is organized as follows. Section 2 introduces the adopted specification language and its service specification sublanguage, some building blocks for the derived protocol specifications, and the adopted protocol correctness criterion. Section 3 describes the adopted principles of protocol derivation. The derivation is guided by various service specification attributes. In Section 4, we introduce rules for attribute evaluation and suggest how to obtain a well-formed service specification. Section 5 comprises discussion and conclusions.

2 Preliminaries

2.1 Specification language and its service specification sublanguage

The language employed, defined in Table 1 in a Backus-Naur-like form, is an abstract representation of some LOTOS constructs, in the exclusive setting of the protocol derivation problem. Not shown in the table are parentheses for control of parsing, the syntax for sets, and shorthands.

A b denotes a behaviour, i.e. a process exhibiting it, for instance a server as a whole, an individual server component, a service part or some other partial server behaviour. For a particular server, let $\mathcal{C}$ denote the universe of its components.

$\mathbf{spec}\ \varepsilon\ \mathbf{where}\ D\ \mathbf{endspec} = \mathbf{spec}\ \delta\ \mathbf{where}\ D\ \mathbf{endspec}$	
$\varepsilon \lvert[G]\rvert b = b \lvert[G]\rvert \varepsilon = b$	$a;\varepsilon = a;\delta$
$\varepsilon \gg b = b \gg \varepsilon = b$	$\varepsilon; b = b$
$\mathbf{hide}\ G\ \mathbf{in}\ \varepsilon\ \mathbf{endhide} = \varepsilon$	$\mathbf{ren}\ R\ \mathbf{in}\ \varepsilon\ \mathbf{endren} = \varepsilon$

Table 2: Absorption rules for ε

stop denotes inaction of the specified process.

δ denotes successful termination.

In some cases, the protocol derivation mapping defined below introduces an ε specifying execution of no actions. ε is similar to δ, because execution of no actions is successful by definition. With the help of the absorption rules in Table 2, it will be possible to make the derived specifications free of ε.

i denotes an anonymous internal action of the specified process. Besides internal actions, processes execute interactions with their environment. Such an external action is primarily denoted by the interaction gate on which it occurs. If it is a service primitive, it is specified as a u^c and denotes a type u interaction between server component c and a service user. If it is an action on an auxiliary gate h, it might be associated with a data offer o, that has to match with the data offer of the process environment. The only data that our processes can handle are strings of zero or more elements 1 and/or 2.

A component c can send messages to another component c' over gate $\mathbf{s}_{c'}^{c}$, while c' receives them over gate $\mathbf{r}_{c}^{c'}$. For specific purposes, c' will sometimes call the gate $\mathbf{a}_{c}^{n}$ (accept), where n will be a partial context identifier. If c' is unable to immediately handle a message received on gate $\mathbf{r}_{c}^{c'}$, it will store it into a FIFO buffer and subsequently claim it on an internal gate $\mathbf{b}_{c}$. Gate **t** will always be an internal gate of a server component, serving for hidden interaction of its parts.

A data offer "$!v$" denotes exactly the data value specified by the term v. A data offer "$?x : v$" or "$?v$" denotes any data value which has a prefix specified by v. When the interaction occurs, one of the values legal for the data offer is selected, and if variable x is specified, stored into it for future use.

"$b_1 \gg b_2$" denotes a process first behaving as b_1, and after its successful termination as b_2, where δ of b_1 is interpreted in "$b_1 \gg b_2$" as **i**. "$a; b_2$" is the special case of the sequential composition where b_1 is an individual action, so that no **i** is needed for transfer of control to b_2.

"$b_1 [] b_2$" denotes a process ready to behave as b_1 or as b_2. Sometimes we will use "[]" as a prefix operator, where choice from an empty set of processes is equivalent to **stop**.

"$b_1 \lvert[G]\rvert b_2$" denotes parallel composition of processes b_1 and b_2, where G specifies the degree and form of their synchronization. An action on a gate listed in G or a δ can only be executed as a common action of the two processes, while the processes execute other actions independently. The usual shorthand for "$\lvert[]\rvert$" is "$\lvert\lvert\lvert$". Sometimes we will use "$\lvert\lvert\lvert$" as a prefix operator, where parallel composition of an empty set of processes specifies an ε.

No.	e
(1)	w::= $\mathbf{spec}\ b\ \mathbf{where}\ D\ \mathbf{endspec}$
(2)	$d ::= p\ \mathbf{is}\ b$
(3)	$b ::= \mathbf{stop}$
(4)	$b ::= \delta$
(5)	$b ::= b_1 \gg b_2$
(6)	$b ::= a; b_2$
(7)	$b ::= b_1 \lvert[S]\rvert b_2$
(8)	$b ::= b_1 [] b_2$
(9)	$b ::= b_1 [> b_2$
(10)	$b ::= \mathbf{hide}\ S\ \mathbf{in}\ b_1\ \mathbf{endhide}$
(11)	$b ::= \mathbf{ren}\ R\ \mathbf{in}\ b_1\ \mathbf{endren}$
(12)	$b ::= p$
(13)	$a ::= s \mid \mathbf{i}$
	$S ::= \underline{\text{set}}\ \underline{\text{of}}\ s$
	$r ::= u_2^c / u_1^c$

Table 3: Service specification sublanguage

"$b_1 [> b_2$" denotes a process with behaviour b_1 potentially disabled upon the start of process b_2. While b_1 is still active, the process might terminate by executing δ in b_1.

"$\mathbf{hide}\ G\ \mathbf{in}\ b_1\ \mathbf{endhide}$" denotes a process behaving as b_1 with its actions on the gates listed in G hidden from its environment. For the environment, the hidden actions are equivalent to **i**.

"$\mathbf{ren}\ R\ \mathbf{in}\ b_1\ \mathbf{endren}$" denotes a process behaving as b_1 with its visible gates (and thereby the actions on them) renamed as specified in R, where in an r, the first and the second item respectively define the new and the old name.

Explicit processes can be defined and instantiated, possibly with an input parameter. In the original LOTOS syntax, explicit processes are defined on formal gates, that are associated with actual gates upon process instantiation. In our simplified language, gate instantiation can be expressed as renaming of the gates on which a process is originally defined applied to the particular process instance.

A specification w defines a behaviour b and the processes instantiated in it, except for the processes predefined in Section 2.2. If D is empty, "$\mathbf{where}\ D$" may be omitted. If it is a service specification (Table 3), then 1) any specified action must be a service primitive or an **i**, 2) gate renaming is allowed only locally to individual server components, and 3) all the explicitly specified processes must be without parameters. Some rows in Table 3 are numbered, so that the corresponding rows in some of the remaining tables can refer to them. In all our example service specifications, every **i** and every δ is furnished with a superscript denoting the server component responsible for it.

The relation used throughout the paper for judging equivalence of behaviours is *observational equivalence* "$\approx$" [2], i.e. we are interested only into the external behaviour of processes, that is in the actions which they make available for synchronization with their environment (all actions except **i** and actions transformed into **i** by hiding).

2.2 Some building blocks for protocol specifications

The contribution of our paper lies in functions for generating protocol specifications in the proposed language. These specifications will be based on some characteristic patterns, for generation of which we define some auxiliary functions (Table 4).

$\mathbf{S}_c(C,v) := \|\|\|_{c' \in (C \setminus \{c\})} \mathbf{s}^c_{c'}!v$
$\mathbf{R}_c(C,v) := \|\|\|_{c' \in (C \setminus \{c\})} \mathbf{r}^c_{c'}!v$
$\mathbf{E}_c(C,C',v) :=$ (if $(c \in C)$ then $\mathbf{S}_c(C',v)$ else ε endif $\|\|\|$ if $(c \in C')$ then $\mathbf{R}_c(C,v)$ else ε endif)
$\mathbf{P}_c(S) := \{u^c \mid (u^c \in S)\}$
$\mathbf{P}_c(R) := \{(u'^c/u^c) \mid ((u'^c/u^c) \in R)\}$

Table 4: Auxiliary specification-generating functions

$\mathbf{S}_c(C,v)$ generates a specification of parallel sending of protocol message v from component c to each member of C other than c. Likewise, $\mathbf{R}_c(C,v)$ specifies parallel receiving of v at c from each member of C other than c.

$\mathbf{E}_c(C,C',v)$ specifies exchange of message v in such a way that each component in C' receives it from every component in C other than itself.

$\mathbf{P}_c(S)$ and $\mathbf{P}_c(R)$ are projection functions. $\mathbf{P}_c(S)$ extracts from S the service primitives belonging to component c, while $\mathbf{P}_c(R)$ extracts from R the renamings of such primitives.

We also assume that there are three predefined processes. Processes "Loop" and "Loop(v)" execute an infinite series of "g" or "g?v" actions, respectively. Shorthands for instantiation of the processes on a gate g for a prefix v are "$\mathrm{Loop}(g)$" and "$\mathrm{Loop}(g?v)$", respectively.

Process "FIFO(v)" is an unbounded FIFO buffer ready to store messages with prefix "v" and to terminate whenever empty. A shorthand for instantiaton of the process on an input gate g_1 and an output gate g_2 for a prefix v is "$\mathrm{FIFO}(g_1,g_2,v)$". To specify that a $\mathrm{FIFO}(g_1,g_2,v)$ should accept all kinds of messages, one sets v to an empty string, that we denote by ε. Such are the buffers pairwise connecting server components. They constitute the communication medium, defined as

Medium **is** $\|\|\|_{c \neq c'} \mathrm{FIFO}(\mathbf{s}^c_{c'}, \mathbf{r}^{c'}_c, \varepsilon)$

2.3 Protocol correctness criterion

Given a service behaviour b, we derive a b_c for each individual component c. The protocol must satisfy the minimal correctness criterion that every protocol message sent is also received. We further expect that in the absence of distributed conflicts, the server behaves towards its users precisely as required (see Table 5). Note that "$\approx (b \gg \delta)$" might also be sufficient, because successful termination of a distributed server, as an act of multiple server components, does not qualify as one of the regular service actions, i.e. service actions assigned to individual components.

If b contains distributed conflicts, precise service execution is expected only for those server runs which do not reveal any of the conflicts. When divergence in service execution occurs, the server should continue to support only the direction of service execution with the highest pre-assigned priority, while the directions competing with it must be abandoned as quickly as possible.

$(\mathrm{Service} \approx b) \vee ((\|\mathcal{C}\| > 1) \wedge (\mathrm{Service} \approx (b \gg \delta)))$
where $\mathrm{Service} =$ **hide** G **in** $(\|\|\|_{c \in \mathcal{C}} b_c) \|[G]\| \mathrm{Medium}$ **endhide**
$G = \cup_{c \neq c'} \{\mathbf{s}^c_{c'}, \mathbf{r}^{c'}_c\}$

Table 5: Precise service implementation

For a "$b_1 [> b_2$", it is appropriate that b_2 has a higher priority than b_1. We adopt this arrangement also for "$b_1 [] b_2$". There are, however, two exceptions. If the server components responsible for the start of b_2 manage to agree on successful termination of b_1 before b_2 starts, b_2 must be abandoned. In the case of "$b_1 [] b_2$", b_2 must be abandoned already when the components manage to agree on the start of b_1.

3 Principles of protocol derivation

3.1 Service attributes and the concept of a well-formed service specification

When mapping a service specification subexpression into its counterparts at individual server components, one refers to its various attributes. A subexpression attribute reveals some property of the subexpression itself or some property of the context in which it is embedded. Computation of service attributes is discussed in Section 4.1.

There is always a dilemma whether to conceive a very general mapping, i.e. a mapping with very few restrictions on the attributes, or a simple mapping with a very restricted applicability. We take the following pragmatic approach.

Above all, we try to avoid restrictions on the specification style (see [28] for a survey of the most typical styles) because, even if a service specification can be restyled automatically, the derived protocol specification will reflect the new style, and as such be hardly comprehensible to the designers of the original specification.

On the other hand, we rely without hesitation on restrictions which can be met simply by introducing some additional hidden service actions. Such insertion can always be automated and causes no restructuring of the service specification. Besides, there is usually more than one way to satisfy a restriction by action insertion. By choosing one way or another, it is possible to influence the derived protocol, i.e. its efficiency and the role of individual server components. Hence by relying strongly on such restrictions, we not only simplify the protocol derivation mapping, but also make space for protocol customization.

A service specification satisfying all the prescribed restrictions is a *well-formed specification*. We postpone suggestions for obtaining such a specification to Section 4.2.

3.2 Compositional approach to service implementation

When mapping a service specification in a compositional way, we map each of its constituent process specifications, including the main service process. Mapping a specification of a process p, we map specifications b of the individual parts of the behaviour specified by its body.

During service execution, each instantiation of such a p gives rise to a new instance of the behaviour specified by such a b. Each such instance is an individual service part and, as such, expected to be implemented in an independent way. In other words, such an instance represents a special context, that first of all needs a dynamically unique identifier. The identifier can then be included in all protocol messages belonging to the particular instance, to make its distributed implementation communication-closed. The simplest way to produce such an identifier is to concatenate (specified by operator "$\cdot$") z, the dynamically unique identifier of the particular instance of p, and $CI(b)$, the dynamically unique context identifier of b within the body of p [14].

Mapping a specification of a process p onto a c results in a specification of a local process p with a formal parameter "z". When the local process is instantiated, "z" propagates into its body the identifier of the particular process instance, so that it can be used in the associated protocol messages. The main service process is instantiated only once, so its "z" can be assigned statically. For a dynamically created process instance, "z" is the identifier of its instantiation. Those properties are reflected in Table 6, more precisely described below.

(1)	$\mathbf{T}_c(w,z) := \mathbf{spec}\ \mathbf{Term}_c(b,z)$ **where** $\{\mathbf{T}_c(d)\|(d \in D)\}$ **endspec**
(2)	$\mathbf{T}_c(d) := p$(z) **is** $\mathbf{Term}_c(b_1$, z)
(12)	$\mathbf{T'}_c(b,z) := p(z{\cdot}CI(b))$

Table 6: Mapping $\mathbf{T}$ for a service specification and mapping $\mathbf{T}'$ for process instantiation

$\mathbf{T}_c(b,z)$ will be the basic function for mapping a service part b onto a component c within a context z. Sometimes the implementation of a b generated by mapping $\mathbf{T}$ will be enriched with some additional protocol messages reporting its successful termination to server components not yet knowing it. The mapping which generates such enriched implementation will be called $\mathbf{Term}_c(b,z)$. Mapping $\mathbf{T}$ of a structured b combines the mappings **Term** of its constituent parts.

For a b, it might be that a c has no duties in its distributed implementation, i.e. that c is not a participating component of b (formally $\neg PC_c(b)$, i.e. not a member of $PC(b)$). In such a case, $\mathbf{T}_c(b,z)$ will be ε or $\mathbf{stop}$, while in the case of $PC_c(b)$, $\mathbf{T}_c(b,z)$ will more precisely be called $\mathbf{T'}_c(b,z)$.

In the following, let $\mathbf{Term}(b,z)$ denote a **Term** implementation of b, i.e. all $\mathbf{Term}_c(b,z)$ plus the protocol channels. Likewise, $\mathbf{T}(b,z)$ denotes a $\mathbf{T}$ implementation.

In an environment of competing service parts, it is important to have a simple characterization of all protocol messages belonging to a particular part b. In a $\mathbf{T}(b,z)$, such a message will carry either identifier $CI(b)$ or identifier $CI(b')$ of a subpart b' of b. To indicate that messages of the second type also belong to b, $CI(b')$ will in all cases have $CI(b)$ as a prefix. In a $\mathbf{Term}(b,z)$, the additionally introduced messages will carry identifier $CI^+(b)$. As $\mathbf{T}(b,z)$ is a part of $\mathbf{Term}(b,z)$, $CI(b)$ will have $CI^+(b)$ as a prefix. So it will be possible to specify readiness to receive any message belonging to a $\mathbf{Term}(b,z)$ simply by $?z{\cdot}CI^+(b)$ in the receptions.

The basic building blocks of context identifiers, hence also of protocol messages, are 1 and 2, because they refer typically to parts b_1 and b_2 of a b. That is, of course, not the only possible choice. By changing 2 to 0, for example, one could obtain pure binary identifiers. In any case, it is important that the number of different messages on individual channels is kept low, for message encodings can then be short. For that reason, messages (i.e. the context identifiers they contain) are re-used extensively, except where that could corrupt their dynamic uniqueness.

Example 1 *For the example service in Table 7, it is crucial that the implementations of the two concurrent instances of process Proc use different protocol messages. Likewise it is important that protocol messages are re-used, because Proc is instantiated an infinite number of times.*

The reception buffers of the three components (see Section 3.9) are not shown in the example, to make the specifications more readable. The buffers are not crucial for deadlock prevention, anyhow.

3.3 Termination types

For a b representing the entire service that is being implemented, it is evident that its successful termination (if any) must be implemented as δ (or as its ε equivalent) at each of the server components. In other words, each $\mathbf{Term}_c(b,z)$ must be terminating, i.e. each c must be a terminating component of b for mapping **Term**, formally $TC_c^+(b)$, i.e. c must be an element of $TC^+(b)$.

If a b is not the last part of the service, $TC_c^+(b)$ is not mandatory. It is sometimes better to let $\mathbf{Term}_c(b,z)$ finish by **stop** instead, i.e. $\neg TC_c^+(b)$ [14]. Such inaction at c is later disrupted by activities of c outside $\mathbf{Term}_c(b,z)$. If b never successfully terminates, formally $\neg TM(b)$, $\neg TC_c^+(b)$ is the only option.

If $TC_c^+(b)$, one has to decide whether c should detect or declare termination of b already within the $\mathbf{T}_c(b,z)$ part of $\mathbf{Term}_c(b,z)$, i.e. whether $TC_c^+(b)$ should imply $TC_c(b)$, i.e. that c is an element of $TC(b)$. If $TC_c^+(b)$ but not $TC_c(b)$, formally $RT_c(b)$, c terminates $\mathbf{Term}_c(b,z)$ upon receiving termination reports "$z{\cdot}CI^+(b)$" from all the ending components of $\mathbf{T}(b,z)$ [14] (see Table 8). Where the danger exists of such a report being received already within $\mathbf{T}_c(b,z)$, care is taken that it is different from any message referred to within $\mathbf{T}_c(b,z)$. Hence protocol $\mathbf{Term}(b,z)$

$w = \mathbf{spec\ ren}\ \mathrm{a}^\alpha/\mathrm{A}^\alpha, \mathrm{b}^\gamma/\mathrm{B}^\gamma, \mathrm{c}^\beta/\mathrm{C}^\beta\ \mathbf{in}\ \mathrm{Proc}\ \mathbf{endren}\ |||\ \mathbf{ren}\ \mathrm{d}^\alpha/\mathrm{A}^\alpha, \mathrm{e}^\gamma/\mathrm{B}^\gamma, \mathrm{f}^\beta/\mathrm{C}^\beta\ \mathbf{in}\ \mathrm{Proc}\ \mathbf{endren}$
$\quad \mathbf{where}\ \mathrm{Proc}\ \mathbf{is}\ (((\mathrm{A}^\alpha; \delta^\alpha)|||(\mathrm{B}^\gamma; \delta^\gamma)) \gg (\mathrm{C}^\beta; \mathrm{Proc}))\ \mathbf{endspec}$
$\mathbf{T}_\alpha(w, \varepsilon) \approx \mathbf{spec\ ren}\ \mathrm{a}^\alpha/\mathrm{A}^\alpha\ \mathbf{in}\ \mathrm{Proc}(1)\ \mathbf{endren}\ |||\ \mathbf{ren}\ \mathrm{d}^\alpha/\mathrm{A}^\alpha\ \mathbf{in}\ \mathrm{Proc}(2)\ \mathbf{endren}$
$\quad \mathbf{where}\ \mathrm{Proc(z)}\ \mathbf{is}\ (\mathrm{A}^\alpha; \mathbf{s}^\alpha_\beta!\mathrm{z}; \mathbf{r}^\alpha_\beta!\mathrm{z}; \mathrm{Proc(z)})\ \mathbf{endspec}$
$\mathbf{T}_\beta(w, \varepsilon) \approx \mathbf{spec\ ren}\ \mathrm{c}^\beta/\mathrm{C}^\beta\ \mathbf{in}\ \mathrm{Proc}(1)\ \mathbf{endren}\ |||\ \mathbf{ren}\ \mathrm{f}^\beta/\mathrm{C}^\beta\ \mathbf{in}\ \mathrm{Proc}(2)\ \mathbf{endren}$
$\quad \mathbf{where}\ \mathrm{Proc(z)}\ \mathbf{is}\ (((\mathbf{r}^\beta_\alpha!\mathrm{z}; \delta)|||(\mathbf{r}^\beta_\gamma!\mathrm{z}; \delta)) \gg \mathrm{C}^\beta; ((\mathbf{s}^\beta_\alpha!\mathrm{z}; \delta)|||(\mathbf{s}^\beta_\gamma!\mathrm{z}; \delta)) \gg \mathrm{Proc(z)})\ \mathbf{endspec}$
$\mathbf{T}_\gamma(w, \varepsilon) \approx \mathbf{spec\ ren}\ \mathrm{b}^\gamma/\mathrm{B}^\gamma\ \mathbf{in}\ \mathrm{Proc}(1)\ \mathbf{endren}\ |||\ \mathbf{ren}\ \mathrm{e}^\gamma/\mathrm{B}^\gamma\ \mathbf{in}\ \mathrm{Proc}(2)\ \mathbf{endren}$
$\quad \mathbf{where}\ \mathrm{Proc(z)}\ \mathbf{is}\ (\mathrm{B}^\gamma; \mathbf{s}^\gamma_\beta!\mathrm{z}; \mathbf{r}^\gamma_\beta!\mathrm{z}; \mathrm{Proc(z)})\ \mathbf{endspec}$

Table 7: An example of multiple process instantiation

$\mathbf{T}_c(b, z) :=$ if $PC_c(b)$ then $\mathbf{T}'_c(b, z)$ else if $TC_c(b)$ then ε else **stop** endif endif
$\mathbf{Term}_c(b, z) :=$ if $TC^+_c(b)$ then if $TC_c(b)$ then $(\mathbf{T}_c(b, z)$ if $EC_c(b)$ then $\gg \mathbf{S}_c((TC^+(b) \setminus TC(b)), z{\cdot}CI^+(b))$ endif)
else $((\mathbf{T}_c(b, z)[>\delta)|||\mathbf{R}_c(EC(b), z{\cdot}CI^+(b)))$ endif
else $\mathbf{T}_c(b, z)$ endif

Table 8: Functions **T** and **Term**

has two phases, namely protocol $\mathbf{T}(b, z)$ and exchange of termination reports.

A c is an ending component of b for mapping **T**, formally $EC_c(b)$, i.e. c is a member of $EC(b)$, if it might be the last component to execute an action within $\mathbf{T}(b, z)$. If $EC_c(b)$, c must, of course, declare termination already within $\mathbf{T}_c(b, z)$, i.e. $EC_c(b)$ by definition implies $TC_c(b)$, and thereby $TC^+_c(b)$.

In many cases, we are free to decide whether $TC^+_c(b)$ should imply $TC_c(b)$ or not, but it is not always directly evident how our decision would influence the overall number of the involved protocol messages. Therefore we follow the classical solution that $TC^+_c(b)$ should always imply $TC_c(b)$ (i.e. $\neg RT_c(b)$), except where that would lead to an erroneous service implementation (discussed in the operator-specific sections). If there are no such cases, mapping **Term** systematically reduces to mapping **T**, i.e. there is a single mapping function, like in the earlier approaches [3, 10].

If $\neg PC_c(b)$, $TC_c(b)$ will always be equal to $TC^+_c(b)$, reducing $\mathbf{Term}_c(b, z)$ to a mere ε or **stop** (see function **T** in Table 8). Hence the components participating in the distributed implementation of a b remain those listed in $PC(b)$, even if we enhance the mapping function from **T** to **Term**.

For a protocol $\mathbf{T}(b, z)$, we define that it successfully terminates when all $\mathbf{T}_c(b, z)$ with $TC_c(b)$ successfully terminate. Likewise, successful termination of $\mathbf{Term}(b, z)$ requires successful termination of all $\mathbf{Term}_c(b, z)$ with $TC^+_c(b)$.

3.4 Implementation of inaction

A **stop** has no participating component, so the first rule in Table 8 implies that every server component implements it as a **stop**.

3.5 Implementation of successful termination

In some cases, it is crucial to have in mind that successful termination δ is also a kind of an action. These are the cases where it is in a decisive position, like an initial δ in a "$b_1[]b_2$" or the δ of b_1 or an initial δ of b_2 in a "$b_1[>b_2$" [14]. So one selects, as convenient, for each δ a server component responsible for its execution, its only participating component. Mapping $\mathbf{T}'$ for the component is a δ (Table 9).

(4) $\mathbf{T}'_c(b, z) := \delta$

Table 9: Mapping $\mathbf{T}'$ for successful termination

3.6 Implementation of hiding and renaming

The only properties of actions within a service part b that influence protocol message exchange are their position within b and their assignment to server components. That is not changed by hiding or local renaming, so implementation of those operations is trivial (Table 10).

(10) $\mathbf{T}'_c(b, z) := \mathbf{hide}\ \mathbf{P}_c(S)\ \mathbf{in}\ \mathbf{Term}_c(b_1, z)\ \mathbf{endhide}$
(11) $\mathbf{T}'_c(b, z) := \mathbf{ren}\ \mathbf{P}_c(R)\ \mathbf{in}\ \mathbf{Term}_c(b_1, z)\ \mathbf{endren}$

Table 10: Mapping $\mathbf{T}'$ for hiding and renaming

3.7 Implementation of action prefix

To map an "$a; b_2$" onto a participant c (Table 11), one first needs $\mathbf{P}_c(a)$, the projection of a. If c is not the executor of a, i.e. its only participant, the projection is empty. If a is a service primitive, its executor is evident from its identifier. If it is an **i**, one selects its executor as convenient.

If a component c might be the first to execute an action within $\mathbf{Term}(b_2, z)$, it is a starting component of b_2, formally $SC_c(b_2)$, i.e. c is a member of $SC(b_2)$. Such a c is responsible for preventing a premature start of

(13) $\mathbf{P}_c(a) :=$ if $PC_c(a)$ then a else ε endif
(6) $\mathbf{T'}_c(b,z) := (\mathbf{P}_c(a); \mathbf{E}_c(PC(a), SC(b_2), z{\cdot}CI(b)) \gg \mathbf{Term}_c(b_2,z))$

Table 11: Mapping $\mathbf{T}'$ for action prefix

$\mathbf{Term}(b_2,z)$, i.e. it must not start $\mathbf{Term}_c(b_2,z)$ until it executes a or receives a report "$z \cdot CI(b)$" on it. Hence protocol $\mathbf{T}(b,z)$ has three phases, namely execution of a, exchange of reports on a, and protocol $\mathbf{Term}(b_2,z)$.

3.8 Implementation of sequential composition

For a b specified as "$b_1 \gg b_2$", we require that b_1, at least sometimes, successfully terminates, because otherwise b_2 would be irrelevant.

Protocol $\mathbf{T}(b,z)$ (Table 12) has three phases, namely protocol $\mathbf{Term}(b_1,z)$, exchange of reports "$z \cdot CI(b)$" on its termination, and protocol $\mathbf{Term}(b_2,z)$. Where danger exists that a message belonging to the second phase is received already within a $\mathbf{Term}_c(b_1,z)$, care is taken that it is different from any message referred to within $\mathbf{Term}_c(b_1,z)$. It is crucial that every c with duties within the second or the third phase terminates $\mathbf{Term}_c(b_1,z)$ in all the terminating runs of b_1, i.e. that $TC_c^+(b_1)$ is true.

(5) $\mathbf{T'}_c(b,z) := (\mathbf{Term}_c(b_1,z) \gg \mathbf{E}_c(EC^+(b_1), SC(b_2), z{\cdot}CI(b)) \gg \mathbf{Term}_c(b_2,z))$

Table 12: Mapping $\mathbf{T}'$ for sequential composition

As in the case of action prefix, reports on termination of the first phase are sent to the starting components of b_2, but now their senders are the ending components of $\mathbf{Term}(b_1,z)$ [19]. A c is an ending component of b_1 for mapping **Term**, formally $EC_c^+(b_1)$, i.e. c is a member of $EC^+(b_1)$, if it might be the last component to execute an action within $\mathbf{Term}(b_1,z)$. It is crucial that a terminating b_1 has at least one ending component, and that in every non-terminating run of such a b_1, there is at least one ending component c not terminating $\mathbf{Term}_c(b_1,z)$, so that start of $\mathbf{Term}(b_2,z)$ is prevented.

We want the second phase (i.e. termination reporting) to completely isolate $\mathbf{Term}(b_2,z)$ from $\mathbf{Term}(b_1,z)$, so that protocol messages from $\mathbf{Term}(b_1,z)$ and termination reports may be re-used within $\mathbf{Term}(b_2,z)$. That is particularly important for implementation of iteration and tail recursion, as in Example 2. To achieve the isolation, we take care that upon the start of $\mathbf{Term}(b_2,z)$, components receiving within it no longer want to receive within $\mathbf{Term}(b_1,z)$.

Example 2 *In Table 13, we implement a service consisting of two consecutive parts. It might happen that the first part does not terminate, but a premature start of the second part is nevertheless prevented.*

3.9 Implementation of parallel composition

For a b specified as "$b_1|[S]|b_2$", we assume that all actions specified in b_1 or b_2, including δ, are actually executable within b, i.e. that they are all relevant.

Protocol $\mathbf{T}(b,z)$ (Table 14) consists basically of protocols $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$ running in parallel and locally synchronized on service primitives from S.

If there are any distributed conflicts in b_1 and/or b_2, formally $AD(b)$, $\mathbf{Term}(b_1,z)$ and/or $\mathbf{Term}(b_2,z)$ are typically imprecise implementations of b_1 and b_2, unable to synchronize properly on S. So if S is non-empty, $AD(b)$ is forbidden.

If S is empty, b_1 and b_2 are nevertheless synchronized on their successful termination (if any). If termination of b is subject to a distributed conflict within b_1 and/or b_2, formally $TD(b)$, negotiation of more than one component is required within $\mathbf{Term}(b_1,z)$ and/or $\mathbf{Term}(b_2,z)$. That is unacceptable, for such termination is a decisive termination (see below). So $TD(b)$ is forbidden.

For independent concurrent execution of $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$, it should be sufficient to take care that their protocol message spaces are disjoint [10]. Unfortunately, it turns out that on a shared channel, unprompt reception in one of the protocols might hinder reception in the other. In the case of a non-empty S, that might even lead to a deadlock [15].

Kant and Higashino suggested that each c could solve the problem by prompt reception of messages into a pool, for further consumption by $\mathbf{Term}_c(b_1,z)$ or $\mathbf{Term}_c(b_2,z)$. So in Table 14, we introduce for each part $\mathbf{Term}_c(b_n,z)$ for each channel from a c' to c that is shared (formally $SH_{c',c}(b)$), a FIFO buffer for incoming messages. Such a buffer is, unlike $\mathbf{Term}_c(b_n,z)$, always ready to receive from the channel on gate $\mathbf{r}^c_{c'}$, thereby removing the possibility of blocking. $\mathbf{Term}_c(b_n,z)$ can subsequently claim the received messages from the buffer on a hidden gate $\mathbf{b}_{c'}$. As demonstrated in the following example, such buffers might be necessary even if S is empty. On the other hand, buffers are often redundant, but that is hard to establish.

Example 3 *In the first part of Table 15, there is a parallel composition implemented properly.*

In the second part, the reception buffers are omitted, and there is a scenario "$\mathrm{a}^\alpha; \mathbf{s}^\alpha_\beta!1; \mathrm{d}^\alpha; \mathbf{s}^\alpha_\beta!2$*" leading to a dead-*

$w = \mathbf{spec}\ ((\mathrm{a}^\alpha; \mathrm{Proc})[](\mathrm{b}^\alpha; \delta^\beta)) \gg (\mathrm{b}^\gamma; \delta^\gamma)$ **where** Proc **is** $(\mathrm{c}^\beta; \mathrm{c}^\alpha; \mathrm{Proc})$ **endspec**
$\mathbf{T}_\alpha(w,1) \approx \mathbf{spec}\ (\mathrm{a}^\alpha; \mathbf{s}^\alpha_\beta!11; \mathrm{Proc})[](\mathrm{b}^\alpha; \mathbf{s}^\alpha_\beta!12; \delta)$ **where** Proc **is** $(\mathbf{r}^\alpha_\beta!11; \mathrm{c}^\alpha; \mathbf{s}^\alpha_\beta!11; \mathrm{Proc})$ **endspec**
$\mathbf{T}_\beta(w,1) \approx \mathbf{spec}\ ((\mathbf{r}^\beta_\alpha!11; \mathrm{Proc})[](\mathbf{r}^\beta_\alpha!12; \delta)) \gg \mathbf{s}^\beta_\gamma!1; \delta$ **where** Proc **is** $(\mathrm{c}^\beta; \mathbf{s}^\beta_\alpha!11; \mathbf{r}^\beta_\alpha!11; \mathrm{Proc})$ **endspec**
$\mathbf{T}_\gamma(w,1) \approx \mathbf{spec}\ \mathbf{r}^\gamma_\beta!1; \mathrm{b}^\gamma; \delta$ **endspec**

Table 13: An example combining finite and infinite alternatives

(7) $\mathbf{T'}_c(b,z) := (\mathbf{Par}_{c,1}\|[\mathbf{P}_c(S)]\|\mathbf{Par}_{c,2})$ where $\mathbf{Par}_{c,n} :=$ **hide** $\{\mathbf{b}_{c'}\|SH_{c',c}(b)\}$ **in ren** $\{(\mathbf{b}_{c'}/\mathbf{r}^c_{c'})\|SH_{c',c}(b)\}$ **in** $\mathbf{Term}_c(b_n,z)$ **endren** $\|[\{\mathbf{b}_{c'}\|SH_{c',c}(b)\}]\|\ \|\|_{SH_{c',c}(b)}\mathrm{FIFO}(\mathbf{r}^c_{c'},\mathbf{b}_{c'},z{\cdot}CI^+(b_n))$ **endhide**

Table 14: Mapping $\mathbf{T'}$ for parallel composition

$w =$ **spec** $(((\mathrm{a}^\alpha;\delta^\alpha)\|\|\|(\mathrm{b}^\beta;\delta^\beta))\gg(\mathrm{c}^\beta;\delta^\beta))\|[\mathrm{b}^\beta]\|(\mathrm{d}^\alpha;\mathrm{b}^\beta;\delta^\beta)$ **endspec**
$\mathbf{T}_\alpha(w,\varepsilon)\approx$ **spec** $(\mathrm{a}^\alpha;\mathbf{s}^\alpha_\beta!1;\delta)\|\|\|(\mathrm{d}^\alpha;\mathbf{s}^\alpha_\beta!2;\delta)$ **endspec**
$\mathbf{T}_\beta(w,\varepsilon)\approx$ **spec hide** $\mathbf{b}_\alpha$ **in** $(\mathrm{b}^\beta;\mathbf{b}_\alpha!1;\mathrm{c}^\beta;\delta)\|[\mathbf{b}_\alpha]\|\mathrm{FIFO}(\mathbf{r}^\beta_\alpha,\mathbf{b}_\alpha,1)$ **endhide** $\|[\mathrm{b}^\beta]\|$**hide** $\mathbf{b}_\alpha$ **in** $(\mathbf{b}_\alpha!2;\mathrm{b}^\beta;\delta)\|[\mathbf{b}_\alpha]\|\mathrm{FIFO}(\mathbf{r}^\beta_\alpha,\mathbf{b}_\alpha,2)$ **endhide endspec**
$\mathbf{T}_\alpha(w,\varepsilon)\approx$ **spec** $(\mathrm{a}^\alpha;\mathbf{s}^\alpha_\beta!1;\delta)\|\|\|(\mathrm{d}^\alpha;\mathbf{s}^\alpha_\beta!2;\delta)$ **endspec**
$\mathbf{T}_\beta(w,\varepsilon)\approx$ **spec** $(\mathrm{b}^\beta;\mathbf{r}^\beta_\alpha!1;\mathrm{c}^\beta;\delta)\|[\mathrm{b}^\beta]\|(\mathbf{r}^\beta_\alpha!2;\mathrm{b}^\beta;\delta)$ **endspec**
$w =$ **spec** $(((\mathrm{a}^\alpha;\delta^\alpha)\|\|\|(\mathrm{b}^\beta;\delta^\beta))\gg(\mathrm{c}^\beta;\delta^\beta))\|\|\|(\mathrm{d}^\alpha;\mathrm{e}^\beta;\delta^\beta)$ **endspec**
$\mathbf{T}_\alpha(w,\varepsilon)\approx$ **spec** $(\mathrm{a}^\alpha;\mathbf{s}^\alpha_\beta!1;\delta)\|\|\|(\mathrm{d}^\alpha;\mathbf{s}^\alpha_\beta!2;\delta)$ **endspec**
$\mathbf{T}_\beta(w,\varepsilon)\approx$ **spec** $(\mathrm{b}^\beta;\mathbf{r}^\beta_\alpha!1;\mathrm{c}^\beta;\delta)\|\|\|(\mathbf{r}^\beta_\alpha!2;\mathrm{e}^\beta;\delta)$ **endspec**

Table 15: An example of parallel composition requiring buffered reception

$w =$ **spec** $(\delta^\alpha[>(\mathrm{a}^\alpha;\mathrm{b}^\beta;\delta^\alpha))\|[\mathrm{a}^\alpha]\|(\delta^\alpha[](\mathbf{i}^\alpha;\mathrm{a}^\alpha;\delta^\alpha))$ **endspec**
$\mathbf{T}_\alpha(w,1)\approx$ **spec** $((\delta[>(\mathrm{a}^\alpha;\mathbf{s}^\alpha_\beta!11;\mathbf{r}^\alpha_\beta!11;\delta))$ $\|[\mathrm{a}^\alpha]\|(\delta[](\mathbf{i};\mathrm{a}^\alpha;\delta)))$ $\gg\mathbf{s}^\alpha_\beta!1;\delta$ **endspec**
$\mathbf{T}_\beta(w,1)\approx$ **spec** $((\mathbf{r}^\beta_\alpha!11;\mathrm{b}^\beta;\mathbf{s}^\beta_\alpha!11;$ **stop**$)[>\delta)$ $\|\|(\mathbf{r}^\beta_\alpha!1;\delta)$ **endspec**

Table 16: An example of decisive and synchronized termination

lock, because message 2 is not the first in the channel.

In the third part, we no longer require that the two concurrent parts are synchronized on b^β*. We also rename the second* b^β *into* e^β*, to distinguish it from the first one. The above scenario no longer leads to a deadlock, but its destination state erroneously requires that* b^β *is executed before* e^β*. Again, reception buffers would help.*

For a b specified as "$b_1|[S]|b_2$", successful termination of $\mathbf{T}(b,z)$ requires successful termination of $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$. If such termination is decisive for one or both of the component protocols, i.e. represents a δ in a decisive position within b_1 or b_2, formally $DT(b)$, its implementation is problematic [14, 15]. It has been suggested that such a δ should be put under control of a single server component, its pre-assigned executor, responsible both for its decisive role and for its synchronization role [14]. If successful termination of $\mathbf{T}(b,z)$ is to be a matter of a single component, the latter must be the only member of $TC(b)$, and consequently the only member of $EC(b)$, $TC^+(b_1)$, $TC^+(b_2)$, $EC(b_1)$ and $EC(b_2)$.

Example 4 *An example of decisive and synchronized termination is given in Table 16. Termination of* b *has been put under exclusive control of component* α*, while component* β *receives only a report of it.*

3.10 Implementation of choice

For a b specified as "$b_1[]b_2$", we assume that there are service actions (at least a δ) in both alternatives, so that both are relevant. The operator introduces distributed conflicts, formally $DC(b)$, if b has more than one starting component.

Protocol $\mathbf{T}(b,z)$ combines protocols $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$. b_2 is the higher-priority alternative, so $\mathbf{Term}(b_2,z)$ upon its start always quickly disables $\mathbf{Term}(b_1,z)$, even if $\mathbf{Term}(b_1,z)$ has already started. On the other hand, when a component detects the start of $\mathbf{Term}(b_1,z)$, it tries to prevent starting of $\mathbf{Term}(b_2,z)$, but might be unsuccessful.

Until one of the alternatives is abandoned, protocols $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$ run in parallel, so we require that their protocol message sets are disjoint.

Within $\mathbf{Term}(b_1,z)$, any starting action must be promptly reported to any starting component c of b_2, formally $SR_c(b_1)$, to inform it that execution of b_2 should not start unless it already has. Analogously, we require $SR_c(b_2)$ for any starting component c of b_1. If $DC(b)$, any component might already be executing b_1 when $\mathbf{Term}(b_2,z)$ starts, so we require $SR_c(b_2)$ also for the non-starting participants of b_1, to make them quickly abandon execution of b_1. Note that the executor of an action is informed of the action by the action itself.

If not earlier, a participant c abandons $\mathbf{Term}_c(b_2,z)$ upon successful termination of $\mathbf{Term}_c(b_1,z)$, if any. At that moment, it must already be obvious that $\mathbf{Term}(b_2,z)$ will never start, i.e. every starting component of b_2 must have already executed an action within $\mathbf{Term}(b_1,z)$, thereby refusing to be an initiator of $\mathbf{Term}(b_2,z)$. In other words, such a starting component c' must guard the termination at c, formally $GT^+_{c,c'}(b_1)$.

If not earlier, a participant c abandons $\mathbf{Term}_c(b_1,z)$ upon successful termination of $\mathbf{Term}_c(b_2,z)$, if any. At that moment, c must already have detected the start of $\mathbf{Term}(b_2,z)$, and that is true if and only if c is a participating component of b_2.

(8) $\mathbf{T'}_c(b,z) :=$ if $\neg DC(b)$ then $(\mathbf{Term}_c(b_1,z)[]\mathbf{Term}_c(b_2,z))$
else **ren** $\cup_{n=1,2}$ $(\{(u^c/u^c_n)|(u^c \in AS_c(b_n))\} + \{(\mathbf{r}^c_{c'}/\mathbf{a}^n_{c'})|CH^+_{c',c}(b_n)\})$ **in hide t in**
$((\mathbf{Const}_{c,1}|[\mathbf{StGt}_{c,2} + \mathbf{RecGt}_{c,2} + \{\mathbf{t}\}]|\mathbf{Const}_{c,2})$
$|[\mathbf{StGt}_{c,1} + \mathbf{RecGt}_{c,1} + \mathbf{StGt}_{c,2} + \mathbf{RecGt}_{c,2}]|\mathbf{Const}_{c,3})$
$|[\mathbf{RecGt}_{c,1} + \{\mathbf{a}^2_{c'}|CH^+_{c',c}(b_1)\}]|\mathbf{Const}_{c,4}$
endhide endren
where $\mathbf{Const}_{c,1} := (((\mathbf{Task}_{c,1} \gg \mathbf{t}; \mathbf{stop})[> (\mathbf{OneStRec}_{c,2} \gg (\mathbf{AllStRec}_{c,2}|||\mathbf{AllRec}_{c,1})))[> \delta)$
where $\mathbf{Task}_{c,1} :=$ **ren** $\{(u^c_1/u^c)|(u^c \in AS_c(b_1))\} + \{(\mathbf{a}^1_{c'}/\mathbf{r}^c_{c'})|CH^+_{c',c}(b_1)\}$ **in** $\mathbf{Par}_{c,1}$ **endren**
where $\mathbf{Par}_{c,1} :=$ see Table 14
$\mathbf{Const}_{c,2} := (\mathbf{Task}_{c,2}[](\mathbf{t}; \delta))$
where $\mathbf{Task}_{c,2} :=$ **ren** $\{(u^c_2/u^c)|(u^c \in AS_c(b_2))\} + \{(\mathbf{a}^2_{c'}/\mathbf{r}^c_{c'})|CH^+_{c',c}(b_2)\}$
in $\mathbf{Term}_c(b_2,z)$ **endren**
$\mathbf{Const}_{c,3} := (((\mathbf{OneStRec}_{c,2} \gg (\mathbf{AllStRec}_{c,2}|||\mathbf{AllRec}_{c,1}))[]$
$(\mathbf{OneStRec}_{c,1} \gg (\mathbf{AllStRec}_{c,1}$
$[> (\mathbf{OneRec}_{c,2} \gg (\mathbf{AllStRec}_{c,2}|||\mathbf{AllRec}_{c,1})))))$
$[> \delta)$
$\mathbf{Const}_{c,4} := ((|||_{CH^+_{c',c}(b_1)}(\mathrm{Loop}(\mathbf{a}^1_{c'}?z{\cdot}CI^+(b_1))[> \mathrm{Loop}(\mathbf{a}^2_{c'}?z{\cdot}CI^+(b_2))))[> \delta)$
$\mathbf{StGt}_{c,n} := \{u^c_n|(u^c \in SS_c(b_n))\}$
$\mathbf{RecGt}_{c,n} := \{\mathbf{a}^n_{c'}|CH^+_{c',c}(b_n)\}$
$\mathbf{OneRec}_{c,n} := ([]_{g \in \mathbf{RecGt}_{c,n}}(g?z{\cdot}CI^+(b_n); \delta))$
$\mathbf{OneStRec}_{c,n} := (([]_{g \in \mathbf{StGt}_{c,n}}(g; \delta))[]\mathbf{OneRec}_{c,n})$
$\mathbf{AllRec}_{c,n} := (\mathbf{stop}|||(|||_{g \in \mathbf{RecGt}_{c,n}}\mathrm{Loop}(g?z{\cdot}CI^+(b_n))))$
$\mathbf{AllStRec}_{c,n} := ((|||_{g \in \mathbf{StGt}_{c,n}}\mathrm{Loop}(g))|||\mathbf{AllRec}_{c,n})$ endif

Table 17: Mapping $\mathbf{T'}$ for choice

A participant c combines $\mathbf{Term}_c(b_1,z)$ and $\mathbf{Term}_c(b_2,z)$ as specified in Table 17. If $\neg DC(b)$, $\mathbf{Term}(b_1,z)$ is known to be the selected alternative as soon as it starts, so every c is allowed to execute $\mathbf{Term}_c(b_1,z)$ and $\mathbf{Term}_c(b_2,z)$ as alternatives.

If $DC(b)$, $\mathbf{Term}_c(b_1,z)$ and $\mathbf{Term}_c(b_2,z)$ must be combined in such a complicated way that no LOTOS operator can express it directly. So we resort to the so called *constraint-oriented specification style* [28]. This is the style in which two or more parallel processes synchronize on the actions they collectively control, and each process imposes its own constraints on the execution of the actions, so that they are enabled only when so allowed by all the processes referring to them.

A $\mathbf{T'}_c(b,z)$ consists of four constraints. $\mathbf{Const}_{c,1}$ and $\mathbf{Const}_{c,2}$ are respectively responsible for execution of $\mathbf{Term}_c(b_1,z)$ and $\mathbf{Term}_c(b_2,z)$, while $\mathbf{Const}_{c,3}$ and $\mathbf{Const}_{c,4}$ serve for their additional co-ordination.

In the first place, we must be aware that in the case of $DC(b)$, protocols $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$ are actually executed in parallel for some time, so every shared incoming channel in principle requires an input buffer for $\mathbf{Term}_c(b_1,z)$ and an input buffer for $\mathbf{Term}_c(b_2,z)$ (see Section 3.9). But as no c' ever transmits to c within $\mathbf{Term}_{c'}(b_1,z)$ after it has transmitted to c within $\mathbf{Term}_{c'}(b_2,z)$, input buffers for prompt reception are necessary only for $\mathbf{Term}_c(b_1,z)$. So we enhance $\mathbf{Term}_c(b_1,z)$ into $\mathbf{Par}_{c,1}$, as described in Table 14, though the buffers are usually redundant.

Internally to $\mathbf{T'}_c(b,z)$, we rename every service primitive u^c in $\mathbf{Term}_c(b_1,z)$ (i.e. in $\mathbf{Par}_{c,1}$) into u^c_1. Likewise, we internally rename every service primitive u^c in $\mathbf{Term}_c(b_2,z)$ into u^c_2. Besides, we internally to $\mathbf{T'}_c(b,z)$ split every reception gate $\mathbf{r}^c_{c'}$ into gates $\mathbf{a}^1_{c'}$ and $\mathbf{a}^2_{c'}$, where messages for $\mathbf{Term}_c(b_1,z)$ are, according to their contents, routed to the first gate, and messages for $\mathbf{Term}_c(b_2,z)$ to the second gate. The renamings are guided by service attributes $AS_c(b_n)$ (lists all the service actions of b_n at c) and $CH^+_{c',c}(b_n)$ (true if the channel from c' to c is employed within $\mathbf{Term}(b_n,z)$).

Applying all the above renamings to $\mathbf{Par}_{c,1}$ and $\mathbf{Term}_c(b_2,z)$, we obtain processes $\mathbf{Task}_{c,1}$ and $\mathbf{Task}_{c,2}$, respectively, that have disjoint sets of service primitives and reception gates. Every action within $\mathbf{T'}_c(b,z)$ is an action of $\mathbf{Task}_{c,1}$ or an action of $\mathbf{Task}_{c,2}$, except that there is also an action on a hidden gate **t** serving for synchronization of $\mathbf{Const}_{c,1}$ and $\mathbf{Const}_{c,2}$ upon successful termination of $\mathbf{Task}_{c,1}$.

The critical actions of $\mathbf{Task}_{c,1}$ are its starting actions. They must influence execution of $\mathbf{Task}_{c,2}$, so they are subject to synchronization between $\mathbf{Const}_{c,1}$ and $\mathbf{Const}_{c,3}$. A starting action of $\mathbf{Task}_{c,1}$ is a starting service action of b_1 at c, i.e. a member of $SS_c(b_1)$, or a reception. If it is a member of $SS_c(b_1)$, it might also be an **i** or a δ, i.e. not suitable for synchronization, so we in principle require that every member of $SS_c(b_1)$ is a service primitive. If c is not a starting component of b_2, $\mathbf{Const}_{c,3}$ is redundant, hence the requirement is not necessary.

The critical actions of $\mathbf{Task}_{c,2}$ are its starting actions. They must in principle influence execution of $\mathbf{Task}_{c,1}$, so they are subject to synchronization between $\mathbf{Const}_{c,1}$ and $\mathbf{Const}_{c,2}$. A starting action of $\mathbf{Task}_{c,2}$ is a member of $SS_c(b_2)$ or a reception. If disruption of $\mathbf{Task}_{c,1}$ is necessary, i.e. if $PC_c(b_1)$, we require that every member of

$w = \mathbf{spec}\ ((\mathrm{a}^{\alpha};\delta)|||(\mathrm{b}^{\beta};\delta))[]((\mathrm{c}^{\gamma};\delta)|||(\mathrm{b}^{\beta};\delta))\ \mathbf{endspec}$
$w_1 = \mathbf{spec}\ ((\mathrm{a}^{\alpha};(\delta^{\beta}|||\delta^{\gamma}))||||(\mathrm{b}^{\beta};\delta^{\gamma}))[]((\mathrm{c}^{\gamma};(\delta^{\alpha}|||\delta^{\beta}))||||(\mathrm{b}^{\beta};(\delta^{\alpha}|||\delta^{\gamma})))\ \mathbf{endspec}$
$w_2 = \mathbf{spec}\ ((\mathrm{a}^{\alpha};(\delta^{\beta}|||\delta^{\gamma}))||||(\mathrm{b}^{\beta};\mathbf{i}^{\gamma};(\delta^{\alpha}|||\delta^{\beta})))[]((\mathrm{c}^{\gamma};(\delta^{\alpha}|||\delta^{\beta}))||||(\mathrm{b}^{\beta};(\delta^{\alpha}|||\delta^{\gamma})))\ \mathbf{endspec}$
$w_3 = \mathbf{spec}\ ((\mathrm{a}^{\alpha};(\delta^{\beta}|||\delta^{\gamma}))||||(\mathrm{b}^{\beta};\mathbf{i}^{\gamma};(\delta^{\alpha}|||\delta^{\beta})))[]((\mathrm{c}^{\gamma};((\mathbf{i}^{\alpha};\delta^{\gamma})|||\delta^{\beta}))||||(\mathrm{b}^{\beta};((\mathbf{i}^{\alpha};\delta^{\beta})|||\delta^{\gamma})))\ \mathbf{endspec}$

$\mathbf{T}_{\alpha}(w_3,\varepsilon) \approx \mathbf{spec\ ren}\ \mathbf{r}^{\alpha}_{\gamma}/\mathbf{a}^{1}_{\gamma}, \mathbf{r}^{\alpha}_{\beta}/\mathbf{a}^{2}_{\beta}, \mathbf{r}^{\alpha}_{\gamma}/\mathbf{a}^{2}_{\gamma}\ \mathbf{in\ hide\ t\ in}$
$(((\boxed{(\mathbf{hide}\ \mathbf{b}_{\gamma}\ \mathbf{in}\ ((\mathrm{a}^{\alpha};((\mathbf{s}^{\alpha}_{\beta}!1;\delta)|||(\mathbf{s}^{\alpha}_{\gamma}!1;\delta)))||||(\mathbf{b}_{\gamma}!1;\delta))|[\mathbf{b}_{\gamma}]|\mathrm{FIFO}(\mathbf{a}^{1}_{\gamma},\mathbf{b}_{\gamma},1)\ \mathbf{endhide}\gg\mathbf{t};\mathbf{stop})}$
$[>(((\mathbf{a}^{2}_{\beta}?2;\delta)[](\mathbf{a}^{2}_{\gamma}?2;\delta))\gg(\mathrm{Loop}(\mathbf{a}^{2}_{\beta}?2)|||\mathrm{Loop}(\mathbf{a}^{2}_{\gamma}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1))))[>\delta)$
$|[\mathbf{a}^{2}_{\beta},\mathbf{a}^{2}_{\gamma},\mathbf{t}]|(\boxed{((\mathbf{a}^{2}_{\gamma}!2;\mathbf{s}^{\alpha}_{\gamma}!2;\delta)||||(\mathbf{a}^{2}_{\beta}!2;\mathbf{s}^{\alpha}_{\beta}!2;\delta))}\ [](\mathbf{t};\delta))$
$|[\mathbf{a}^{1}_{\gamma},\mathbf{a}^{2}_{\gamma}]|((\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1)[>\mathrm{Loop}(\mathbf{a}^{2}_{\gamma}?2))[>\delta)$
$\mathbf{endhide\ endren\ endspec}$

$\mathbf{T}_{\beta}(w_3,\varepsilon) \approx \mathbf{spec\ ren}\ \mathrm{b}^{\beta}/\mathrm{b}^{\beta}_{1}, \mathbf{r}^{\beta}_{\alpha}/\mathbf{a}^{1}_{\alpha}, \mathbf{r}^{\beta}_{\gamma}/\mathbf{a}^{1}_{\gamma}, \mathrm{b}^{\beta}/\mathrm{b}^{\beta}_{2}, \mathbf{r}^{\beta}_{\alpha}/\mathbf{a}^{2}_{\alpha}, \mathbf{r}^{\beta}_{\gamma}/\mathbf{a}^{2}_{\gamma}\ \mathbf{in\ hide\ t\ in}$
$((((\boxed{(\mathbf{hide}\ \mathbf{b}_{\alpha},\mathbf{b}_{\gamma}\ \mathbf{in}\ ((\mathbf{b}_{\alpha}!1;\delta)||||(\mathrm{b}^{\beta}_{1};\mathbf{s}^{\beta}_{\gamma}!1;\mathbf{b}_{\gamma}!1;\delta))$
$|[\mathbf{b}_{\alpha},\mathbf{b}_{\gamma}]|(\mathrm{FIFO}(\mathbf{a}^{1}_{\alpha},\mathbf{b}_{\alpha},1)|||\mathrm{FIFO}(\mathbf{a}^{1}_{\gamma},\mathbf{b}_{\gamma},1))\ \mathbf{endhide}\gg\mathbf{t};\mathbf{stop})}$
$[>(((\mathrm{b}^{\beta}_{2};\delta)[](\mathbf{a}^{2}_{\gamma}?2;\delta))\gg(\mathrm{Loop}(\mathrm{b}^{\beta}_{2})|||\mathrm{Loop}(\mathbf{a}^{2}_{\gamma}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1))))[>\delta)$
$|[\mathrm{b}^{\beta}_{2},\mathbf{a}^{2}_{\gamma},\mathbf{t}]|(\boxed{((\mathbf{a}^{2}_{\gamma}!2;\delta)||||(\mathrm{b}^{\beta}_{2};((\mathbf{s}^{\beta}_{\alpha}!2;\delta)||||(\mathbf{s}^{\beta}_{\gamma}!2;\delta))\gg\mathbf{a}^{2}_{\alpha}!2;\delta))}\ [](\mathbf{t};\delta)))$
$|[\mathrm{b}^{\beta}_{1},\mathbf{a}^{1}_{\alpha},\mathbf{a}^{1}_{\gamma},\mathrm{b}^{\beta}_{2},\mathbf{a}^{2}_{\gamma}]|(((((\mathrm{b}^{\beta}_{2};\delta)[](\mathbf{a}^{2}_{\gamma}?2;\delta))\gg(\mathrm{Loop}(\mathrm{b}^{\beta}_{2})|||\mathrm{Loop}(\mathbf{a}^{2}_{\gamma}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1)))[]$
$(((\mathrm{b}^{\beta}_{1};\delta)[](\mathbf{a}^{1}_{\alpha}?1;\delta)[](\mathbf{a}^{1}_{\gamma}?1;\delta))\gg$
$((\mathrm{Loop}(\mathrm{b}^{\beta}_{1})|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1))$
$[>(\mathbf{a}^{2}_{\gamma}?2;(\mathrm{Loop}(\mathrm{b}^{\beta}_{2})|||\mathrm{Loop}(\mathbf{a}^{2}_{\gamma}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1))))))$
$[>\delta))$
$|[\mathbf{a}^{1}_{\alpha},\mathbf{a}^{1}_{\gamma},\mathbf{a}^{2}_{\alpha},\mathbf{a}^{2}_{\gamma}]|(((\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)[>\mathrm{Loop}(\mathbf{a}^{2}_{\alpha}?2))||||(\mathrm{Loop}(\mathbf{a}^{1}_{\gamma}?1)[>\mathrm{Loop}(\mathbf{a}^{2}_{\gamma}?2)))[>\delta)$
$\mathbf{endhide\ endren\ endspec}$

$\mathbf{T}_{\gamma}(w_3,\varepsilon) \approx \mathbf{spec\ ren}\ \mathbf{r}^{\gamma}_{\alpha}/\mathbf{a}^{1}_{\alpha}, \mathbf{r}^{\gamma}_{\beta}/\mathbf{a}^{1}_{\beta}, \mathbf{r}^{\gamma}_{\alpha}/\mathbf{a}^{2}_{\alpha}, \mathbf{r}^{\gamma}_{\beta}/\mathbf{a}^{2}_{\beta}\ \mathbf{in\ hide\ t\ in}$
$((((\boxed{(\mathbf{hide}\ \mathbf{b}_{\alpha},\mathbf{b}_{\beta}\ \mathbf{in}\ ((\mathbf{b}_{\alpha}!1;\delta)||||(\mathbf{b}_{\beta}!1;((\mathbf{s}^{\gamma}_{\alpha}!1;\delta)||||(\mathbf{s}^{\gamma}_{\beta}!1;\delta))))$
$|[\mathbf{b}_{\alpha},\mathbf{b}_{\beta}]|(\mathrm{FIFO}(\mathbf{a}^{1}_{\alpha},\mathbf{b}_{\alpha},1)|||\mathrm{FIFO}(\mathbf{a}^{1}_{\beta},\mathbf{b}_{\beta},1))\ \mathbf{endhide}\gg\mathbf{t};\mathbf{stop})}$
$[>(((\mathrm{c}^{\gamma};\delta)[](\mathbf{a}^{2}_{\beta}?2;\delta))\gg(\mathrm{Loop}(\mathrm{c}^{\gamma})|||\mathrm{Loop}(\mathbf{a}^{2}_{\beta}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\beta}?1))))[>\delta)$
$|[\mathrm{c}^{\gamma},\mathbf{a}^{2}_{\beta},\mathbf{t}]|(\boxed{((\mathrm{c}^{\gamma};((\mathbf{s}^{\gamma}_{\alpha}!2;\delta)||||(\mathbf{s}^{\gamma}_{\beta}!2;\delta))\gg\mathbf{a}^{2}_{\alpha}!2;\delta)||||(\mathbf{a}^{2}_{\beta}!2;\delta))}\ [](\mathbf{t};\delta)))$
$|[\mathbf{a}^{1}_{\alpha},\mathbf{a}^{1}_{\beta},\mathrm{c}^{\gamma},\mathbf{a}^{2}_{\beta}]|(((((\mathrm{c}^{\gamma};\delta)[](\mathbf{a}^{2}_{\beta}?2;\delta))\gg(\mathrm{Loop}(\mathrm{c}^{\gamma})|||\mathrm{Loop}(\mathbf{a}^{2}_{\beta}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\beta}?1)))[]$
$(((\mathbf{a}^{1}_{\alpha}?1;\delta)[](\mathbf{a}^{1}_{\beta}?1;\delta))\gg$
$((\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\beta}?1))$
$[>(\mathbf{a}^{2}_{\beta}?2;(\mathrm{Loop}(\mathrm{c}^{\gamma})|||\mathrm{Loop}(\mathbf{a}^{2}_{\beta}?2)|||\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)|||\mathrm{Loop}(\mathbf{a}^{1}_{\beta}?1))))))$
$[>\delta))$
$|[\mathbf{a}^{1}_{\alpha},\mathbf{a}^{1}_{\beta},\mathbf{a}^{2}_{\alpha},\mathbf{a}^{2}_{\beta}]|(((\mathrm{Loop}(\mathbf{a}^{1}_{\alpha}?1)[>\mathrm{Loop}(\mathbf{a}^{2}_{\alpha}?2))||||(\mathrm{Loop}(\mathbf{a}^{1}_{\beta}?1)[>\mathrm{Loop}(\mathbf{a}^{2}_{\beta}?2)))[>\delta)$
$\mathbf{endhide\ endren\ endspec}$

Table 18: An example of distributed choice

(9) $\mathbf{T'}_{c}(b,z) :=$ <u>if</u> $\neg DC(b)$ <u>then</u> $(\mathbf{Term}_{c}(b_1,z)[>\mathbf{Term}_{c}(b_2,z))$
<u>else</u> $\mathbf{ren}\ \cup_{n=1,2}(\{(u^{c}/u^{c}_{n})|(u^{c}\in AS_{c}(b_n))\}+\{(\mathbf{r}^{c}_{c'}/\mathbf{a}^{n}_{c'})|CH^{+}_{c',c}(b_n)\})\ \mathbf{in\ hide\ t\ in}$
$((\mathbf{Const}_{c,1}|[\mathbf{StGt}_{c,2}+\mathbf{RecGt}_{c,2}+\{\mathbf{t}\}]|\mathbf{Const}_{c,2})$
$|[\mathbf{StGt}_{c,2}+\mathbf{RecGt}_{c,2}+\{\mathbf{p}^{c}_{1}\}]|\mathbf{Const}_{c,3})$
$|[\mathbf{RecGt}_{c,1}+\{\mathbf{a}^{2}_{c'}|CH^{+}_{c',c}(b_1)\}]|\mathbf{Const}_{c,4}$
$\mathbf{endhide\ endren}$
<u>where</u> $\mathbf{Const}_{c,3} := ((\mathbf{AllStRec}_{c,2}[](\mathbf{p}^{c}_{1};\mathbf{OneRec}_{c,2}\gg\mathbf{AllStRec}_{c,2}))[>\delta)$
the rest of definitions as in Table 17 <u>endif</u>

Table 19: Mapping $\mathbf{T'}$ for disabling

$SS_{c}(b_2)$ is a service primitive.

The gates on which the starting service primitives and receptions within a $\mathbf{Task}_{c,n}$ occur are listed in $\mathbf{StGt}_{c,n}$ and $\mathbf{RecGt}_{c,n}$, respectively. $\mathbf{OneStRec}_{c,n}$ specifies a process ready to synchronize on one action of $\mathbf{Task}_{c,n}$ on gates from $\mathbf{StGt}_{c,n}$ and $\mathbf{RecGt}_{c,n}$. $\mathbf{AllStRec}_{c,n}$ specifies a processes ready to synchronize on all such actions. Processes $\mathbf{OneRec}_{c,n}$ and $\mathbf{AllRec}_{c,n}$ are analogous to $\mathbf{OneStRec}_{c,n}$ and $\mathbf{AllStRec}_{c,n}$, respectively, except that they refer only to receptions.

$\mathbf{Const}_{c,1}$ prescribes the following: 1) Basically, execute $\mathbf{Task}_{c,1}$ and indicate its successful termination by a $\mathbf{t}$. 2) If $\mathbf{Task}_{c,2}$ starts in the meantime (that will always be before $\mathbf{t}$), stop the basic activity, but remain ready for reception of protocol messages sent to $\mathbf{Task}_{c,1}$. 3) Always be ready to terminate, though $\mathbf{Const}_{c,2}$ will ensure that that

$w = \mathbf{spec}\ ((\mathrm{a}^\alpha;\delta)|||(\mathrm{b}^\beta;\delta))[>((\mathrm{c}^\alpha;\delta)|||(\mathrm{b}^\beta;\delta))\ \mathbf{endspec}$

$w_1 = \mathbf{spec\ hide}\ \mathbf{p}^\alpha,\mathbf{p}^\beta\ \mathbf{in}\ (((\mathrm{a}^\alpha;\delta^\alpha)|||(\mathrm{b}^\beta;\delta^\beta))\gg((\mathbf{p}^\alpha;\delta^\alpha)|||(\mathbf{p}^\beta;\delta^\beta))\gg(\delta^\alpha|||\delta^\beta))[>((\mathrm{c}^\alpha;\delta^\beta)|||(\mathrm{b}^\beta;\delta^\alpha))\ \mathbf{endhide}\ \mathbf{endspec}$

$\mathbf{T}_\alpha(w_1,\varepsilon) \approx \mathbf{spec\ hide}\ \mathbf{p}^\alpha,\mathbf{t}\ \mathbf{in\ ren}\ \mathbf{r}^\alpha_\beta/\mathbf{a}^1_\beta,\mathbf{r}^\alpha_\beta/\mathbf{a}^2_\beta\ \mathbf{in}$

$((((\boxed{(\mathbf{hide}\ \mathbf{b}_\beta\ \mathbf{in}\ (\mathrm{a}^\alpha;((\mathbf{s}^\alpha_\beta!1;\delta)|||(\mathbf{b}_\beta!1;\delta))\gg\mathbf{p}^\alpha;((\mathbf{s}^\alpha_\beta!1;\delta)|||(\mathbf{b}_\beta!1;\delta)))\ |[\mathbf{b}_\beta]|\mathrm{FIFO}(\mathbf{a}^1_\beta,\mathbf{b}_\beta,1)\ \mathbf{endhide}\gg\mathbf{t};\mathbf{stop})})$

$[>(((\mathrm{c}^\alpha;\delta)[](\mathbf{a}^2_\beta?2;\delta))\gg(\mathrm{Loop}(\mathrm{c}^\alpha)|||\mathrm{Loop}(\mathbf{a}^2_\beta?2)|||\mathrm{Loop}(\mathbf{a}^1_\beta?1))))[>\delta)$

$|[\mathrm{c}^\alpha,\mathbf{a}^2_\beta,\mathbf{t}]|(\boxed{((\mathrm{c}^\alpha;\mathbf{s}^\alpha_\beta!2;\delta)|||(\mathbf{a}^2_\beta!2;\delta))}\ [](\mathbf{t};\delta)))$

$|[\mathbf{p}^\alpha,\mathrm{c}^\alpha,\mathbf{a}^2_\beta,\mathbf{t}]|(((\mathrm{Loop}(\mathrm{c}^\alpha)|||\mathrm{Loop}(\mathbf{a}^2_\beta?2))[](\mathbf{p}^\alpha;\mathbf{a}^2_\beta?2;(\mathrm{Loop}(\mathrm{c}^\alpha)|||\mathrm{Loop}(\mathbf{a}^2_\beta?2))))[>\delta))$

$|[\mathbf{a}^1_\beta,\mathbf{a}^2_\beta]|((\mathrm{Loop}(\mathbf{a}^1_\beta?1)[>\mathrm{Loop}(\mathbf{a}^2_\beta?2))[>\delta)\ \mathbf{endren\ endhide\ endspec}$

$\mathbf{T}_\beta(w_1,\varepsilon) \approx \mathbf{spec\ hide}\ \mathbf{p}^\beta,\mathbf{t}\ \mathbf{in\ ren}\ \mathrm{b}^\beta/\mathrm{b}^\beta_1,\mathbf{r}^\beta_\alpha/\mathbf{a}^1_\alpha,\mathrm{b}^\beta/\mathrm{b}^\beta_2,\mathbf{r}^\beta_\beta/\mathbf{a}^2_\alpha\ \mathbf{in}$

$((((\boxed{(\mathbf{hide}\ \mathbf{b}_\alpha\ \mathbf{in}\ (\mathrm{b}^\beta_1;((\mathbf{s}^\beta_\alpha!1;\delta)|||(\mathbf{b}_\alpha!1;\delta))\gg\mathbf{p}^\beta;((\mathbf{s}^\beta_\alpha!1;\delta)|||(\mathbf{b}_\alpha!1;\delta)))\ |[\mathbf{b}_\alpha]|\mathrm{FIFO}(\mathbf{a}^1_\alpha,\mathbf{b}_\alpha,1)\ \mathbf{endhide}\gg\mathbf{t};\mathbf{stop})})$

$[>(((\mathrm{b}^\beta_2;\delta)[](\mathbf{a}^2_\alpha?2;\delta))\gg(\mathrm{Loop}(\mathrm{b}^\beta_2)|||\mathrm{Loop}(\mathbf{a}^2_\alpha?2)|||\mathrm{Loop}(\mathbf{a}^1_\alpha?1))))[>\delta)$

$|[\mathrm{b}^\beta_2,\mathbf{a}^2_\alpha,\mathbf{t}]|(\boxed{((\mathbf{a}^2_\alpha!2;\delta)|||(\mathrm{b}^\beta_2;\mathbf{s}^\beta_\alpha!2;\delta))}\ [](\mathbf{t};\delta)))$

$|[\mathbf{p}^\beta,\mathrm{b}^\beta_2,\mathbf{a}^2_\alpha,\mathbf{t}]|(((\mathrm{Loop}(\mathrm{b}^\beta_2)|||\mathrm{Loop}(\mathbf{a}^2_\alpha?2))[](\mathbf{p}^\beta;\mathbf{a}^2_\alpha?2;(\mathrm{Loop}(\mathrm{b}^\beta_2)|||\mathrm{Loop}(\mathbf{a}^2_\alpha?2))))[>\delta))$

$|[\mathbf{a}^1_\alpha,\mathbf{a}^2_\alpha]|((\mathrm{Loop}(\mathbf{a}^1_\alpha?1)[>\mathrm{Loop}(\mathbf{a}^2_\alpha?2))[>\delta)\ \mathbf{endren\ endhide\ endspec}$

Table 20: An example of distributed disabling

will happen only after successful termination of $\mathbf{Task}_{c,1}$ or $\mathbf{Task}_{c,2}$.

$\mathbf{Const}_{c,2}$ prescribes the following: Execute $\mathbf{Task}_{c,2}$ or terminate upon a $\mathbf{t}$ indicating that $\mathbf{Task}_{c,1}$ has successfully terminated.

$\mathbf{Const}_{c,3}$ in addition prescribes that in the case that the first action belongs to $\mathbf{Task}_{c,1}$, $\mathbf{Task}_{c,2}$ may start only upon a reception, i.e. upon detecting that $\mathbf{Term}(b_2,z)$ has already started at a remote site.

With the described measures for prompt start reporting and for prevention of premature local termination, $\mathbf{T'}_c(b,z)$ will progress towards completion of $\mathbf{Task}_{c,1}$ or $\mathbf{Task}_{c,2}$ as appropriate.

There is, however, still a problem to solve. $\mathbf{Task}_{c,2}$ must not terminate while c may still expect messages sent to $\mathbf{Task}_{c,1}$. So we require that $\mathbf{Task}_{c,2}$ (i.e. $\mathbf{Term}_c(b_2,z)$) never successfully terminates without receiving on each of the channels on which $\mathbf{Term}_c(b_1,z)$ receives. Upon a reception within $\mathbf{Term}_c(b_2,z)$, c knows that on the channel, there will be no more messages for $\mathbf{Term}_c(b_1,z)$. For some channels, the requirement might be redundant.

It is convenient if c indeed promptly becomes unwilling to receive on gates in $\mathbf{RecGt}_{c,1}$, to improve the possibility of re-use of protocol messages belonging to $\mathbf{Term}_c(b_1,z)$. Therefore we introduce $\mathbf{Const}_{c,4}$. An analogous constraint for protocol messages belonging to $\mathbf{Term}_c(b_2,z)$ would also be desirable, but we have found its automatic specification too difficult.

Example 5 *An example of distributed choice is given in Table 18. The original service specification w is gradually transformed into a well-formed specification, following suggestions from Section 4.2. w_1 secures prompt reporting of each individual starting service action. w_2 in addition secures that no component terminates the first alternative until it is selected by components β and γ, the starting components of the second alternative. w_3 in addition secures that every channel employed for the first alternative is also employed for the second one.*

In each individual component specification, the first and the second alternative are highlighted by a box. When divergence occurs, components execute the first alternative, but gradually switch to the other. We see that every protocol message of the first alternative is a 1, and every message of the second one is a 2. All the specified FIFO buffers are redundant.

3.11 Implementation of disabling

For a b specified as "$b_1[> b_2$", we assume that there are service actions (at least a δ) in both parts, so that both are relevant. The operator does not introduce distributed conflicts, formally $\neg DC(b)$, if there is a c which is the only participating component of b_1 and also the only starting component of b_2.

Protocols $\mathbf{Term}(b_1,z)$ and $\mathbf{Term}(b_2,z)$ are combined as for "$b_1[]b_2$", except that $\mathbf{Term}(b_2,z)$ is allowed to start as long as there is a starting component c of b_2 which has not yet detected that b_1 is successfully terminating and confirmed this knowledge by executing a special-purpose service primitive $\mathbf{p}^c$ in b_1.

A participant c combines $\mathbf{Term}_c(b_1,z)$ and $\mathbf{Term}_c(b_2,z)$ as specified in Table 19. If $\neg DC(b)$, activation of $\mathbf{Term}(b_2,z)$ is a local matter of the starting component of b_2. For any other c, $\mathbf{Term}_c(b_1,z)$ is equivalent to **stop**, i.e. the component just waits for an eventual start of $\mathbf{Term}_c(b_2,z)$.

If $DC(b)$, we require that b_1 consists of a regular part b_3 followed by a dummy part b_4 indicating its successful termination (if $\neg TM(b_1)$, b_4 is never activated, and as such not specified), i.e. we pretend that the service we are implementing is actually "$b_3[> b_2$". More precisely, we require

$b_4 = ((|||_{SC_c(b_2)}(\mathbf{p}^c;\delta^c)) \gg (|||_{TC_c^+(b_1)}\delta^c))$

where **p** primitives are supposed to be hidden on a higher service level and not among the visible primitives of b_3. Note that we also prescribe the executor of each individual δ. Since $DC(b)$ and $TM(b_1)$ imply that b in no way synchronizes with concurrent service parts, any $\mathbf{p}^c$ may be regarded entirely as an internal action of $\mathbf{T'}_c(b,z)$.

For such a b_1, protocol $\mathbf{Term}(b_1,z)$ consists of two phases. The first phase is $\mathbf{Term}(b_3,z)$ followed by reporting of successful termination to all the starting components of b_4, i.e. exactly to the starting components of b_2. In other words, the components are, as required, promptly informed when starting of $\mathbf{Term}(b_2,z)$ becomes undesirable. If the first phase successfully terminates before $\mathbf{Term}(b_2,z)$ starts, $\mathbf{T}(b,z)$ starts executing the usual distributed implementation of a well-formed "$b_4[]b_2$". If the start of $\mathbf{Term}(b_2,z)$ is sufficiently delayed, the executed alternative is b_4, i.e. b_1 is not disrupted by b_2. In any case, no participant abandons $\mathbf{Term}(b_2,z)$ until every starting component c of b_2 has executed a $\mathbf{p}^c$, i.e. refused to be an initiator of $\mathbf{Term}(b_2,z)$.

Comparing $\mathbf{T'}_c(b_1[> \ b_2,z)$ with $\mathbf{T'}_c(b_1[]b_2,z)$, we see that, instead of waiting for the starting actions of $\mathbf{Term}_c(b_1,z)$, $\mathbf{Const}_{c,3}$ now waits for the only $\mathbf{p}^c$ in $\mathbf{Term}_c(b_1,z)$, if any. Consequently, instead of synchronizing on the gates in $\mathbf{StGt}_{c,1}$ and $\mathbf{RecGt}_{c,1}$, $\mathbf{Const}_{c,1}$ and $\mathbf{Const}_{c,3}$ have to synchronize just on $\mathbf{p}_1^c$, hence $\mathbf{Const}_{c,3}$ is much easier to specify.

Example 6 *An example of distributed disabling is given in Table 20. To obtain a well-formed service specification, we furnish the first part with the required hidden* **p** *actions, and make sure that the starting actions of the second part are promptly reported and that both protocol channels are used for the part.*

4 Computation and tuning of service attributes

4.1 Attribute evaluation rules

The attributes in Table 21 provide information on service actions and their executors. SS_c and AS_c respectively list for an a, b or p its starting service actions and all its service actions at c. SC_c and PC_c respectively indicate for an a or b that c is its starting component or its participating component.

The attributes in Table 22 provide information on successful terminations. TM, IT and DT respectively indicate for a b or p that it might successfully terminate, that it might terminate initially, or that the termination might be decisive.

The attributes in Table 23 provide information on distributed conflicts. DC indicates for a b that distributed conflicts are introduced by its top-level composition operator. AD and TD respectively indicate for a b or p whether there are any distributed conflicts in it and whether there are distributed conflicts involving its successful termination.

The attribute SR_c in Table 24 indicates for a b or p that its start must be promptly reported to c.

The attribute EC_c in Table 25 indicates for a b or p that c is its ending component for mapping **T**. EC_c^+ is the analogue for mapping **Term**.

No.	SS_c	No.	SS_c
(2)	$SS_c(p) = SS_c(b)$	(4)	$SS_c(b) = \{\delta \mid PC_c(b)\}$
(3)	$SS_c(b) = \emptyset$	(6)	$SS_c(b) = SS_c(a)$
(12)	$SS_c(b) = SS_c(p)$	(13)	$SS_c(a) = \{a \mid PC_c(a)\}$
(7)	$SS_c(b) = ((SS_c(b_1) \setminus S) \cup (SS_c(b_2) \setminus S) \cup (SS_c(b_1) \cap SS_c(b_2) \cap S))$		
(8,9)	$SS_c(b) = (SS_c(b_1) \cup SS_c(b_2))$		
(5)	$SS_c(b) = ((SS_c(b_1) \setminus \{\delta\}) \cup \{\mathbf{i} \mid (\delta \in SS_c(b_1))\})$		
(10)	$SS_c(b) = ((SS_c(b_1) \setminus S) \cup \{\mathbf{i} \mid ((SS_c(b_1) \cap S) \neq \emptyset)\})$		
(11)	$SS_c(b) = ((SS_c(b_1) \setminus \{s \mid \exists (s'/s) \in R\}) \cup \{s' \mid \exists s \in SS_c(b_1) : ((s'/s) \in R)\})$		
No.	AS_c	No.	AS_c
(2)	$AS_c(p) = AS_c(b)$	(4)	$AS_c(b) = SS_c(b)$
(3)	$AS_c(b) = \emptyset$	(12)	$AS_c(b) = AS_c(p)$
(5)	$AS_c(b) = ((AS_c(b_1) \setminus \{\delta\}) \cup \{\mathbf{i}\} \cup AS_c(b_2))$		
(6)	$AS_c(b) = (SS_c(a) \cup AS_c(b_2))$		
(7–9)	$AS_c(b) = (AS_c(b_1) \cup AS_c(b_2))$		
(10)	$AS_c(b) = ((AS_c(b_1) \setminus S) \cup \{\mathbf{i} \mid ((AS_c(b_1) \cap S) \neq \emptyset)\})$		
(11)	$AS_c(b) = ((AS_c(b_1) \setminus \{s \mid \exists (s'/s) \in R\}) \cup \{s' \mid \exists s \in AS_c(b_1) : ((s'/s) \in R)\})$		
(3–12)	$(SC_c(b) = (SS_c(b) \neq \emptyset)) \wedge (PC_c(b) = (AS_c(b) \neq \emptyset))$		
(4)	$\exists c : (PC(b) = \{c\})$		
(13)	$(\exists c : (PC(a) = \{c\})) \wedge ((\exists u : (a = u^c)) \Rightarrow PC_c(a))$		

Table 21: Service actions and their executors

No.	DT	No.	DT
(2)	$DT(p) = DT(b)$	(10,11)	$DT(b) = DT(b_1)$
(3,4)	$DT(b) = false$	(12)	$DT(b) = DT(p)$
(5,6)	$DT(b) = DT(b_2)$	(7)	$DT(b) = (DT(b_1) \vee DT(b_2))$
(8)	$DT(b) = (DT(b_1) \vee DT(b_2) \vee IT(b))$		
(9)	$DT(b) = (TM(b_1) \vee IT(b_2) \vee DT(b_2))$		
(3–12)	$TM(b) = \exists c : (\delta \in AS_c(b))$		
(3–12)	$IT(b) = \exists c : (\delta \in SS_c(b))$		

Table 22: Successful terminations

No.	AD	No.	AD
(2)	$AD(p) = AD(b)$	(3,4)	$AD(b) = false$
(6)	$AD(b) = AD(b_2)$	(5,7)	$AD(b) = (AD(b_1) \vee AD(b_2))$
(12)	$AD(b) = AD(p)$	(10,11)	$AD(b) = AD(b_1)$
(8,9)	$AD(b) = (AD(b_1) \vee AD(b_2) \vee DC(b))$		
No.	TD	No.	TD
(2)	$TD(p) = TD(b)$	(10,11)	$TD(b) = TD(b_1)$
(3,4)	$TD(b) = false$	(12)	$TD(b) = TD(p)$
(5,6)	$TD(b) = TD(b_2)$	(7)	$TD(b) = (TD(b_1) \vee TD(b_2))$
(8)	$TD(b) = (TD(b_1) \vee TD(b_2) \vee (DC(b) \wedge IT(b)))$		
(9)	$TD(b) = (TD(b_2) \vee (DC(b) \wedge (TM(b_1) \vee IT(b_2))))$		
(8)	$DC(b) := (\lvert SC(b) \rvert > 1)$		
(9)	$DC(b) := (\lvert PC(b_1) \cup SC(b_2) \rvert > 1)$		

Table 23: Distributed conflicts

No. SR_c	No.	SR_c
(1) $SR_c(b) = false$	(5,7,9–11)	$SR_c(b_1) = SR_c(b)$
(2) $SR_c(b) = SR_c(p)$	(5,6)	$SR_c(b_2) = false$
(7) $SR_c(b_2) = SR_c(b)$	(12)	$SR_c(p) = (SR_c(p) \vee SR_c(b))$
(8) $SR_c(b_1) = (SR_c(b) \vee SC_c(b_2))$		
(8) $SR_c(b_2) = (SR_c(b) \vee SC_c(b_1) \vee (DC(b) \wedge PC_c(b_1)))$		
(9) $SR_c(b_2) = (SR_c(b) \vee PC_c(b_1))$		

Table 24: Start reporting

No.	EC_c	No.	EC_c
(2)	$EC_c(p) = EC_c^+(b)$	(5,6)	$EC_c(b) = EC_c^+(b_2)$
(3)	$EC_c(b) = false$	(10,11)	$EC_c(b) = EC_c^+(b_1)$
(4)	$EC_c(b) = PC_c(b)$	(12)	$EC_c(b) = EC_c(p)$
(7–9)	$EC_c(b) = (EC_c^+(b_1) \vee EC_c^+(b_2))$		
(3–12)	$EC_c^+(b) = ((EC_c(b) \wedge \nexists c' : RT_{c'}(b)) \vee RT_c(b))$		

Table 25: Ending components

No.	TC_c^+	No.	TC_c^+
(1)	$TC_c^+(b) = TM(b)$	(2)	$TC_c^+(b) = TC_c^+(p)$
(5)	$TC_c^+(b_1) = (EC_c(b_1) \vee PC_c(b_2) \vee TC_c(b))$		
(5–9)	$TC_c^+(b_2) = (TC_c(b) \wedge PC_c(b) \wedge TM(b_2))$		
(7–11)	$TC_c^+(b_1) = (TC_c(b) \wedge PC_c(b) \wedge TM(b_1))$		
(12)	$TC_c^+(p) = (TC_c^+(p) \wedge TC_c(b))$		
No.	TC_c		
(3–6,10,11)	$TC_c(b) = TC_c^+(b)$		
(7)	$TC_c(b) = (TC_c^+(b) \wedge (EC_c(b) \vee \neg PC_c(b) \vee (\neg DT(b) \wedge \nexists c' : SH_{c',c}(b))))$		
(8,9)	$TC_c(b) = (TC_c^+(b) \wedge (((\forall c' \in SC(b_2) : GT_{c,c'}^+(b_1)) \wedge (\neg DC(b) \vee ((EC_c(b) \vee \neg TM(b_1)) \wedge \nexists c' : (CH_{c',c}^+(b_1) \wedge \neg CT_{c',c}^+(b_2)))) \wedge (\neg TM(b_2) \vee PC_c(b_2))) \vee \neg PC_c(b)))$		
(12)	$TC_c(b) = (TC_c^+(b) \wedge (TC_c^+(p) \vee \neg PC_c(b)))$		
(3–12)	$RT_c(b) = (TC_c^+(b) \wedge \neg TC_c(b))$		

Table 26: Termination types

The attributes in Table 26 provide information on termination types. TC_c and TC_c^+ respectively indicate for a b or p that c is its terminating component for mapping **T** or **Term**. RT_c indicates for a b that c detects its termination upon receiving a special report on it.

The attributes in Table 27 provide information on utilization of protocol channels. $CH_{c,c'}$ and $CH_{c,c'}^+$ respectively indicate for a b or p that mapping **T** or **Term** introduces protocol messages on the channel from c to c'. $CT_{c,c'}$ and $CT_{c,c'}^+$ respectively indicate that the channel is used in every successfully terminating run. For a b consisting of two competing parts, $SH_{c,c'}$ indicates if the channel is shared.

The attributes $GT_{c,c'}$ and $GT_{c,c'}^+$ in Table 28 respectively indicate for a b or p that in mapping **T** or **Term**, its successful termination at c is guarded by c'.

By the rules in Table 29, we choose for a b such identifiers CI and CI^+ that all protocol messages introduced by mapping **T** or **Term**, respectively, are dynamically unique.

No. $CH_{c,c'}$	No.	$CH_{c,c'}$
(2) $CH_{c,c'}(p) = CH_{c,c'}^+(b)$	(10,11)	$CH_{c,c'}(b) = CH_{c,c'}^+(b_1)$
(3,4) $CH_{c,c'}(b) = false$	(12)	$CH_{c,c'}(b) = CH_{c,c'}(p)$
(5) $CH_{c,c'}(b) = (CH_{c,c'}^+(b_1) \vee CH_{c,c'}^+(b_2) \vee ((c \neq c') \wedge EC_c^+(b_1) \wedge SC_{c'}(b_2)))$		
(6) $CH_{c,c'}(b) = (CH_{c,c'}^+(b_2) \vee ((c \neq c') \wedge PC_c(a) \wedge SC_{c'}(b_2)))$		
(7–9) $CH_{c,c'}(b) = (CH_{c,c'}^+(b_1) \vee CH_{c,c'}^+(b_2))$		
No. $CT_{c,c'}$	No.	$CT_{c,c'}$
(2) $CT_{c,c'}(p) = CT_{c,c'}^+(b)$	(10,11)	$CT_{c,c'}(b) = CT_{c,c'}^+(b_1)$
(3) $CT_{c,c'}(b) = true$	(12)	$CT_{c,c'}(b) = CT_{c,c'}(p)$
(4) $CT_{c,c'}(b) = false$		
(5) $CT_{c,c'}(b) = (CT_{c,c'}^+(b_1) \vee CT_{c,c'}^+(b_2) \vee ((c \neq c') \wedge EC_c^+(b_1) \wedge SC_{c'}(b_2)))$		
(6) $CT_{c,c'}(b) = (CT_{c,c'}^+(b_2) \vee ((c \neq c') \wedge PC_c(a) \wedge SC_{c'}(b_2)))$		
(7) $CT_{c,c'}(b) = (CT_{c,c'}^+(b_1) \vee CT_{c,c'}^+(b_2))$		
(8,9) $CT_{c,c'}(b) = (CT_{c,c'}^+(b_1) \wedge CT_{c,c'}^+(b_2))$		
(3–12) $CH_{c,c'}^+(b) = (CH_{c,c'}(b) \vee (EC_c(b) \wedge RT_{c'}(b)))$		
(3–12) $CT_{c,c'}^+(b) = (CT_{c,c'}(b) \vee (EC_c(b) \wedge RT_{c'}(b)))$		
(7–9) $SH_{c,c'}(b) = (CH_{c,c'}^+(b_1) \wedge CH_{c,c'}^+(b_2))$		

Table 27: Channel utilization

No. $GT_{c,c'}$	No.	$GT_{c,c'}$
(2) $GT_{c,c'}(p) = GT_{c,c'}^+(b)$	(10,11)	$GT_{c,c'}(b) = GT_{c,c'}^+(b_1)$
(3) $GT_{c,c'}(b) = true$	(12)	$GT_{c,c'}(b) = GT_{c,c'}(p)$
(4) $GT_{c,c'}(b) = (\neg TC_c(b) \vee ((c = c') \wedge PC_c(b)))$		
(5) $GT_{c,c'}(b) = (GT_{c,c'}^+(b_1) \vee GT_{c,c'}^+(b_2) \vee (PC_c(b_2) \wedge \exists c'' : (EC_{c''}^+(b_1) \wedge GT_{c'',c'}^+(b_1))))$		
(6) $GT_{c,c'}(b) = ((PC_{c'}(a) \wedge ((c = c') \vee PC_c(b_2))) \vee GT_{c,c'}^+(b_2))$		
(7) $GT_{c,c'}(b) = (GT_{c,c'}^+(b_1) \vee GT_{c,c'}^+(b_2))$		
(8,9) $GT_{c,c'}(b) = (GT_{c,c'}^+(b_1) \wedge GT_{c,c'}^+(b_2))$		
(3–12) $GT_{c,c'}^+(b) = (\neg TC_c^+(b) \vee (TC_c(b) \wedge GT_{c,c'}(b)) \vee (\neg TC_c(b) \wedge \exists c'' : (EC_{c''}(b) \wedge GT_{c'',c'}(b))))$		

Table 28: Termination guarding

No. CI^+	No.	CI^+
(1,2) $CI^+(b) = \varepsilon$	(5,6)	$CI^+(b_2) = CI(b)$
(5,10,11) $CI^+(b_1) = CI(b)$		
(7–9) if $\exists c, c' : SH_{c,c'}(b)$ then $CI^+(b_1) = CI(b) \cdot 1$, $CI^+(b_2) = CI(b) \cdot 2$ else $CI^+(b_1) = CI^+(b_2) = CI(b)$ endif		
No. CI		
(3–6,10,11) $CI(b) = CI^+(b)$		
(7–9) if $(((CI^+(b_1) \neq CI(b)) \wedge (CI^+(b_2) \neq CI(b))) \vee \nexists c, c' : (TC_c^+(b) \wedge RT_{c'}(b) \wedge CH_{c,c'}(b)))$ then $CI(b) = CI^+(b)$ else $CI(b) = CI^+(b) \cdot 1$ endif		
(12) if $\nexists c, c' : (TC_c^+(b) \wedge RT_{c'}(b) \wedge CH_{c,c'}(b))$ then $CI(b) = CI^+(b)$ else $CI(b) = CI^+(b) \cdot 1$ endif		

Table 29: Context identifiers

Attribute evaluation rules for a service specification con-

stitute a system of equations which might have more than one solution for the attributes of the explicitly defined processes. One should always maximize their attribute TC^+, while other attributes must be minimized.

4.2 Additional restrictions and their satisfaction

Table 30 summarizes the additional restrictions introduced so far for a well-formed service specification.

The first three restrictions state that no irrelevant service part may be specified. The restriction for parallel composition is actually more rigorous than its approximation in Table 30 (see Section 3.9).

The next two restrictions refer to the ending components of a b. Usually they can be satisfied simply by proper choice of executors for individual δ in b, but not always. It might be that a "$b_1[]b_2$" or a "$b_1[> b_2$" is terminating, but no c qualifies for its ending component, because a $GT^+_{c,c'}(b_1)$ or $PC_c(b_2)$ or a $CT^+_{c',c}(b_2)$ is not true as required. $GT^+_{c,c'}(b_1)$ can be satisfied by securing that in the terminating runs of b_1, the last (possibly dummy) action at c always comes after a (possibly dummy) action at c'. For $PC_c(b_2)$, it suffices to insert into b_2 a dummy action at c. For $CT^+_{c',c}(b_2)$, it helps to introduce into every terminating run of b_2 an action at c prefixed by an action at c'.

The next two restrictions require that there are hidden **p** primitives at certain places in the service specification. If **p** primitives are already used for other purposes, any other reserved service primitive type will do.

The next restriction states that a b with distributed conflicts must not synchronize with a concurrent service part, in order to avoid deadlock resulting from imprecise implementation of b. However, if the concurrent service part is sufficiently flexible (like, for example, a skilled user of an imprecisely implemented service), there will be no deadlock and the restriction may be ignored.

The next two restrictions secure prompt start reporting. An ordinary action a is always specified in a context "$a; b_2$". A report recipient c must be the executor of a or a starting component of b_2, so that the message will be generated to implement the action-prefix operator. If a c is a missing starting component of b_2, that can be solved by introducing into b_2 a dummy starting service action at c. For reporting of a δ, there is no such b_2 following, so we have only the first option.

In a general case, execution of a disruptive b might start by concurrent execution and reporting of several starting actions. To avoid as much as possible such multiple reporting of the start of b, it is advisable to rewrite the specification of b into the action-prefix form (as required in [10] for b_2 in a "$b_1[> b_2$"), i.e. make sure that $AP(b)$ (defined in Table 31).

The last two restrictions state that a service action in a particular position must not be an **i** or a δ. If it is an **i**, change it into a service primitive and hide it on a higher level. If it is a δ, prefix it with a subsequently hidden service primitive. For both cases, $DC(b)$ implies that b runs in such a context that the transformation is irrelevant.

(5)	$TM(b_1)$
(7)	$((\cup_{c\in\mathcal{C}} AS_c(b_1)) \cap (S + \{\delta\}))$
	$= ((\cup_{c\in\mathcal{C}} AS_c(b_2)) \cap (S + \{\delta\}))$
(8,9)	$(\lvert PC(b_1)\rvert > 0) \wedge (\lvert PC(b_2)\rvert > 0)$
(7)	$DT(b) \Rightarrow (\lvert EC(b)\rvert = 1)$
(3–12)	$EC_c(b) \Rightarrow TC_c(b)$
(1)	$\nexists c : (\mathbf{p}^c \in AS_c(b))$
(9)	$DC(b) \Rightarrow \exists b_3 :$
	$((b_1 = (b_3$ if $TM(b_1)$ then
	$\gg(\lvert\lvert\lvert_{SC_c(b_2)}(\mathbf{p}^c; \delta^c)) \gg (\lvert\lvert\lvert_{TC^+_c(b_1)} \delta^c)$
	endif)
	$\wedge \nexists c : (\mathbf{p}^c \in AS_c(b_3)))$
(7)	$((S \neq \emptyset) \Rightarrow \neg AD(b)) \wedge \neg TD(b)$
(4)	$SR_c(b) \Rightarrow PC_c(b)$
(6)	$SR_c(b) \Rightarrow (PC_c(a) \vee SC_c(b_2))$
(8)	$DC(b) \Rightarrow (SC_c(b_2) \Rightarrow (((\{\mathbf{i}, \delta\} \cap SS_c(b_1)) = \emptyset))$
(8,9)	$DC(b) \Rightarrow (PC_c(b_1) \Rightarrow (((\{\mathbf{i}, \delta\} \cap SS_c(b_2)) = \emptyset))$

Table 30: Restrictions

No.	AP	No.	AP
(2)	$AP(p) = AP(b)$	(3,4,6)	$AP(b) = true$
(7,9)	$AP(b) = false$	(8)	$AP(b) = (AP(b_1) \wedge AP(b_2))$
(12)	$AP(b) = AP(p)$	(5,10,11)	$AP(b) = AP(b_1)$

Table 31: Action-prefix form

5 Discussion and conclusions

5.1 Correctness

A formal proof of the protocol derivation method is given in [18], and briefly outlined below.

For every service part b, the only property that really matters is correctness of its $\mathbf{T}'$ and $\mathbf{Term}$ implementations for the context in which it is embedded, where a $\mathbf{T}'$ implementation consists of the members of $PC(b)$, while a $\mathbf{Term}$ implementation might also involve some other server components. However, when proving the property, we also assume over twenty auxiliary properties of the implementations.

All the properties are proven by induction on the service structure. Most of them are synthesized properties. We prove them for the $\mathbf{T}'$ implementations of **stop** and δ. For every composite b (i.e. for every service composition operator), we prove that if **Term** implementations of the constituent service parts possess the properties, the $\mathbf{T}'$ implementation of b possesses their analogues. In addition we prove that if the $\mathbf{T}'$ implementation of a b possesses the properties, its $\mathbf{Term}$ implementations possess their analogues. For the few inherited properties, the proof goes in the reverse direction. By proving the main property for the main service process, we prove that the entire service is properly implemented.

5.2 Message complexity

The operators potentially introducing protocol messages are the operators of sequence, choice and disabling. It is often possible to reduce the number of such operators by restructuring the service specification, i.e. by making its inherent parallelism more explicit. If such restyling of the service (and consequently of the protocol) is not unacceptable for readability reasons, it can greatly reduce the message complexity, and can even be automated [25]. One should also strive for optimal insertion of dummy service actions and optimal assignment of hidden service actions to server components.

Anyway, some of the messages introduced by our protocol derivation mapping are redundant.

- In some cases, it would be possible to omit a message based on the observation that for the service part b_1 to which it belongs, it sequences two service actions which are already sequenced for a concurrent service part b_2 synchronized on them [13].
- It would be possible to further optimize terminations of implementations of individual service parts, and their reporting in individual runs [14, 24].
- When implementing a "$b_1 [] b_2$", one could make better use of the fact that only the initial parts of b_1 and b_2 are concurrent.
- When implementing a "$b_1 [> b_2$", one could make better use of the fact that only the initial part of b_2 is concurrent to b_1.

With more extensive re-use of messages, their encodings could be shorter, but messages would no longer directly identify the service part to which they belong, leading to more complicated protocol specifications.

5.3 Comparison with similar methods

The popular formal technique for specifying self-stabilizing protocols have long been finite state machines (FSMs) [6, 27, 22]. With their explicit representation of states, they are very convenient for the purpose. Namely, when a process proceeds along a selected path in the transition graph representing its FSM, the fact that it ignores messages belonging to the abandoned paths can be specified simply by furnishing each state on the selected path with loops representing reception of such messages. In a process-algebraic language like LOTOS, there is no explicit notion of states, so specification of self-stabilization is a tricky task.

There are two basic approaches to deriving self-stabilizing protocols. In the older approach [6, 27], a protocol is first derived for the ideal case with no divergences and subsequently furnished with the reception-ignoring loops. The derivation algorithm in [22], like ours, handles the ideal and the non-ideal cases in an integrated manner, and is consequently much less complex. Moreover, the algorithm derives protocols in a compositional way, supporting implementation of sequence, choice and iteration. For those operators, the structure of services is quite well reflected in the derived protocols. Unfortunately, FSMs are less suited for explicit specification of more complex operators, particularly for such introducing concurrency. We have solved the problem by switching to the more expressive LOTOS.

We know no comparable LOTOS-based protocol derivation transformation. Some hidden divergence is allowed in [1], but it is resolved with the help of global controllers.

5.4 Handling of data

We intend to extend our method to service actions associated with data [5, 11], to approach the ideal that the service specification language should be the same as the protocol specification language. The strategy for flexible integrated handling of messages implementing proper ordering of actions and those carrying data is simple [11]: 1) In the service, identify the points where inter-component exchange of data would be desirable. 2) At each point, introduce a (possibly dummy) action of the data sender followed by a (possibly dummy) action of the data recipient, so that there will be an action-ordering message between the two components. 3) Let the message carry the data. In our case, data could also be carried in a message reporting termination of a b to a c with $RT_c(b)$.

Data exchange is also desirable as a powerful means for compositional service specification. Whenever the more specific operators (e.g. sequential composition, choice and disabling) do not suffice for describing a particular kind of composition of a set of service parts, one can still run the parts in parallel and let them exchange and process information on their respective states.

5.5 Handling of quantitative temporal constraints

Once being able to handle service actions with data, one can easily implement quantitative temporal constraints [12, 23]. Such a constraint specifies the allowed time gap between two service actions. So the time when the first action is executed is just another piece of data generated by the first action and needed for timely execution of the second one. Temporal constraints can also be employed for preventing distributed conflicts and for further optimization of protocol traffic [23].

5.6 The problem of co-ordinated self-stabilization

The most difficult challenge for future research seems to be implementation of self-stabilization after divergence in synchronized service parts. The problem is important because synchronized processes are the core of the constraint-

oriented specification style, that is indispensable for expressing more exotic forms of service composition. To solve it in a general case, one would need a protocol incorporating negotiation of implementations of concurrent service parts, so an enhancement along the lines of [29] could help.

5.7 Conclusions

Automatic implementation of self-stabilization after divergence is an important achievement in LOTOS-based protocol derivation, because many realistic services contain distributed conflicts (e.g. a connection establishment service with both parties as possible initiators). In the era of service integration, the problem is even more acute, because one often wishes to combine services which are not exactly compatible. Take for example feature interactions in telecommunications, which can be nicely detected and managed based on specifications in LOTOS [4]. With the possibility of compositional derivation of self-stabilizing protocols, it suffices to specify dynamic management of such interactions on the service level.

In our future work, we will focus on protocol derivation in E-LOTOS [8], the enhanced successor of LOTOS, because it supports specification of real-time aspects.

References

[1] Bista BB, Takahashi K, Shiratori N: A compositional approach for constructing communication services and protocols. IEICE Transactions on Fundamentals E82-A(11):2546–2557 (1999)

[2] Bolognesi T, Brinksma E: Introduction to the ISO specification language LOTOS. Computer Networks and ISDN Systems 14(1):25–59 (1987)

[3] Brinksma E, Langerak R: Functionality decomposition by compositional correctness preserving transformation. South African Computer Journal 13:2–13 (1995)

[4] Dietrich F, Hubaux J-P: Formal methods for communication services: meeting the industry expectations. Computer Networks 38(1):99–120 (2002)

[5] Gotzhein R, Bochmann Gv: Deriving protocol specifications from service specifications including parameters. ACM Transactions on Computer Systems 8(4):255–283 (1990)

[6] Gouda MG, Yu YT: Synthesis of communicating finite-state machines with guaranteed progress. IEEE Trans. on Communications COM-32(7):779–788 (1984)

[7] ISO/IEC: Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. IS 8807, 1989

[8] ISO/IEC: Information Technology - Enhancements to LOTOS (E-LOTOS). IS 15473, 2001

[9] Kahlouche H, Girardot JJ: A stepwise refinement based approach for synthesizing protocol specifications in an interpreted Petri net model. Proceedings of IEEE INFOCOM'96, pp 1165–1173, 1996

[10] Kant C, Higashino T, Bochmann Gv: Deriving protocol specifications from service specifications written in LOTOS. Distributed Computing 10(1):29-47 (1996)

[11] Kapus-Kolar M: Deriving protocol specifications from service specifications including parameters. Microprocessing and Microprogramming 32:731–738 (1991)

[12] Kapus-Kolar M: Deriving protocol specifications from service specifications with heterogeneous timing requirements. Proceedings SERTS'91. IEE, London 1991, pp 266–270

[13] Kapus-Kolar M: On context-sensitive service-based protocol derivation. Proceedings of MELECON'96. IEEE Computer Society Press 1996, pp 955–958

[14] Kapus-Kolar M: More efficient functionality decomposition in LOTOS. Informatica (Ljubljana) 23(2):259–273 (1999)

[15] Kapus-Kolar M: Comments on deriving protocol specifications from service specifications written in LOTOS. Distributed Computing 12(4):175–177 (1999)

[16] Kapus-Kolar M: Service-based synthesis of two-party protocols. Elektrotehniški vestnik 67(3):153–161 (2000)

[17] Kapus-Kolar M: Global conflict resolution in automated service-based protocol synthesis. South African Computer Journal 27:34–48 (2001)

[18] Kapus-Kolar M: Deriving self-stabilizing protocols for services specified in LOTOS. Technical Report #8476, Jožef Stefan Institute, Ljubljana, 2003

[19] Khendek F, Bochmann Gv, Kant C: New results on deriving protocol specifications from service specifications. Proceedings of ACM SIGCOMM'89, pp 136–145, 1989

[20] Langerak R: Decomposition of functionality: A correctness-preserving LOTOS transformation. Protocol Specification, Testing and Verification X. North-Holland, Amsterdam 1990, pp 203–218

[21] Naik K, Cheng Z, Wei DSL: Distributed implementation of the disabling operator in LOTOS. Information and Software Technology 41(3):123–130 (1999)

[22] Nakamura M, Kakuda Y, Kikuno T: On constructing communication protocols from component - based service specifications. Computer Communications 19(14):1200–1215 (1996)

[23] Nakata A, Higashino T, Taniguchi K: Protocol synthesis from timed and structured specifications. Proceedings of ICNP'95. IEEE Computer Society Press 1995, pp 74–81

[24] Nakata A, Higashino T, Taniguchi K: Protocol synthesis from context-free processes using event structures. Proceedings RTCSA'98. IEEE Computer Society Press 1998, pp 173–180

[25] Pavón Gomez S, Hulström M, Quemada J, de Frutos D, Ortega Mallen Y: Inverse expansion. Formal Description Techniques IV. North-Holland, Amsterdam 1992, pp 297-312

[26] Saleh K: Synthesis of communication protocols: An annotated bibliography. Computer Communication Review 26(5):40–59 (1996)

[27] Saleh K, Probert RL: An extended service-based method for the synthesis of protocols. Proceedings of the Sixth Bilkent Intern. Symp. on Computer and Information Sciences. Elsevier, Amsterdam 1991, pp 547–557

[28] Vissers CA, Scollo G, Sinderen Mv: Specification styles in distributed systems design and verification. Theoretical Computer Science 89:179–206 (1991)

[29] Yasumoto K, Higashino T, Taniguchi K: A compiler to implement LOTOS specifications in distributed environments. Computer Networks 36(2–3):291–310 (2001)

Embedding Complete Binary Trees into Faulty Flexible Hypercubes with Unbounded Expansion

Jen-Chih Lin and Steven K.C. Lo
Department of Information Management
Jin-Wen Institute of Technology,
No. 99, An-Chung Rd., Hsin-Tien City,
Taipei, Taiwan, R.O.C.
E-mail: yachih@ms13.hinet.net
Department of Computer Science and Information Engineering,
Tamkang University, Tamsui, Taiwan, R.O.C.
Email: kclo@cs.tku.edu.tw

Keywords: Flexible Hypercube, hypercube, embedding, complete binary tree

Received: July 1, 2002

We develop novel algorithms to facilitate the embedding job when the Flexible Hypercube contains faulty nodes. We present strategies for reconfiguring a complete binary tree into a flexible hypercube with n-expansion. These embedding algorithms show a complete binary tree can be embedded into a faulty flexible hypercube with load 1, congestion 1 and dilation 4 such that $O(n^2 - m^2)$ faults can be tolerated, where $(n-1)$ is the dimension of a Flexible Hypercube and $(m-1)$ is the height of a complete binary tree. These methodologies are proven and these algorithms are present to save them.

1 Introduction

In the study of parallel computing, networks of processors are often organized into various configuration such as trees, rings, linear arrays, meshes and hypercubes. These configurations can be represented as graphs. If the properties and structures of underlying graph used effectively, the computation and communication speeds can often removed.

Among the various interconnection networks that have been studied and built, hypercube networks have received much attention. This attention is mainly due to the hypercube advantages of rich interconnection, routing simplicity, and embedding capabilities. However, due to the power-of-2 size and logarithmic degree, hypercubes suffer two major disadvantages, namely, high cost extensibility and large internal fragmentation in partitioning. In order to conquer the difficulties associated with hypercubes and these generalizations of the hypercubes, the Flexible Hypercube[5] has been proposed during past years. The Flexible Hypercube unlike both the supercube[14] and the hypercube, may be expanded (or designed) in a number of possible configurations while guaranteeing the same basic fault-tolerant properties and without a change in the communication. The existence of hypercube subgraphs in the Flexible Hypercube ensures that hypercube embedding algorithms developed for the hypercube may also be utilized in the Flexible Hypercube. The flexibility in node placement may possibly be utilized to aid in supporting a specific embedding. The Flexible Hypercube, while maintaining the fault-tolerance of the other topologies and the ease of communication, allows the placement of new nodes at any currently unused addresses in the system.

Graph embedding problems have applications in a wide variety of computational situations. For example, the flow of information in a parallel algorithm defines a program graph, and embedding this into a network tells us how to organize the computation on the network. Other problems that can be formulated as graph embedding problems are laying out circuits on chips, representing data structures in computer memory, and finding efficient program control structures.

The power of a message-passing parallel computer depends on the topology chosen for underlying interconnection network, which can be modeled as undirected graph. Different graphs have been proposed as static interconnection topology for multiprocessors. They include linear arrays, rings, meshes, complete binary trees mesh of trees, de Bruijn networks, and so on. Therefore, we model both the parallel algorithm and the parallel machine as graphs. Given two graphs, $G(V, E)$ and $G'(V', E')$, embedding[9] the guest graph G into the host graph G' maps each vertex in the set V into a vertex (or a set of vertices) in the set V' and each edge in the set E into an edge (or a set of edges) in the set E'. Let these nodes in a graph correspond to processors and edges to communication links in an interconnection network. Embedding one graph into another is important because an algorithm may have been designed for a specific interconnection network, and it may be necessary to adapt it to another network. Four costs associated with graph embedding are dilation, expansion, load and congestion. The maximum amount that we must stretch any edge to achieve an embedding is called the di-

lation of the embedding. By expansion, we mean the ratio of the number of nodes in the host graph to the number of nodes in the graph that is being embedded. The congestion of an embedding is the maximum number of edges of the guest graph that are embedded using any single edge of the host graph. The load of an embedding is the maximum number of nodes of the guest graph that are embedded in any single node of the host graph. An efficient simulation of one network on another network requires that these four costs be as small as possible. However, for most embedding problems, it is impossible to obtain an embedding that minimizes these costs simultaneously. Therefore, some tradeoffs among these costs must be made.

One approach to achieve faulty-tolerance in hypercubes is to introduce spare nodes or links[4, 12], so that hypercube structure can still be maintained when nodes fail. This approach can be expensive and it is difficult to make hardware modifications on those machines already in the market place. Another approach exploits the inherent redundant nodes or links in hypercube to achieve fault tolerance[6, 15]; that is no extra nodes or links are added to alter the structure of hypercube, but instead use the unused nodes as spares. In this dissertation, we consider only the second type of fault-tolerance design in faulty hypercube-derived computers.

In a multiprocessor system, we follow two fault models defined in [6] and [11]. The first model assumes that, in a faulty node, the computational function of the node is lost while the communication function remains intact; this is the *partial faulty* model. The second model assumes that, in a faulty node, the communication function is lost too; this is the *total faulty* model. In this dissertation, our model is the partial faulty model. That is, when the computation nodes are faulty, the communication links are well and only the faulty nodes are remapped.

The paper presents novel algorithms to facilitate the embedding job when the Flexible Hypercube contains faulty nodes. Of particular concern are the network structures of the Flexible Hypercube that balance the load before as well as after faults start to degrade the performance of the Flexible Hypercube. To obtain replaceable nodes of faulty nodes, 2-expansion is permitted such that up to $(n-2)$ faults can be tolerated with congestion 1, dilation 4 and load 1, where $(n-1)$ is the dimension of a Flexible Hypercube. Results presented herein demonstrate that embedding methods are optimized. Furthermore, we present strategies for reconfiguring a complete binary tree into a Flexible Hypercube with n-expansion. These embedding algorithms show a complete binary tree can be embedded into a faulty flexible hypercube with load 1, congestion 1 and dilation 4 such that $O(n^2-m^2)$ faults can be tolerated, where $(n-1)$ is the dimension of a flexible hypercube and $(m-1)$ is the height of a complete binary tree.

The remainder of this paper is organized as follows. In the next section, some notations and definitions will be introduced. At the same time, we describe how to embed a complete binary tree into a Flexible Hypercube with 2-expansion. In section 3, we embed a complete binary tree into a Flexible Hypercube with n-expansion under partial faulty model. Finally, we conclude this paper.

2 Preliminary

The Flexible Hypercube is constructed by any number of nodes and based on a hypercube. A Flexible Hypercube, denoted by FH_N, is defined as an undirected graph $FH_N = (V, E)$, where V is the set of processors (called nodes) and E is the set of bidirectional communication links between the processors (called edges). In an n-dimensional Flexible Hypercube with N nodes where $2^n \leq N < 2^{n+1}$ (n is a positive integer), each node can be expressed by an $(n+1)$-bit binary string $i_n...i_0$ where $i_p \in \{0,1\}$ and $0 \leq p \leq n$.

Definition 1 *[8] A $(2^n - t)$-node Flexible Hypercube is a lack of t nodes, which are referred to herein as virtual nodes. For any virtual node y, denoted as $I(x)$ where x is any node of the Flexible Hypercube, if the function $I(x)$ exists, then $x_{n-1} = \overline{y_{n-1}}$, and $x_i = y_i$ for $0 \leq i \leq n-2$.*

Definition 2 *[8] The Hamming distance of two nodes x and y, denoted by $HD(x, y)$, is the number of 1's in the bit set of resulting sequence of the bitwise XOR of x and y.*

Definition 3 *[8] For any two nodes x and y in a supercube, let $x = x_{n-1} \ldots x_0$, $y = y_{n-1} \ldots y_0$, then $Dim(x, y) = \{ i$ in $(0 \ldots n-1) \mid x_i \neq y_i \}$.*

Definition 4 *[5] Suppose $FH_N = (V, E)$ is an $(n-1)$-dimensional Flexible Hypercube, then the node sets H_1, H_2, V_1, V_2, V_3 are defined as follows*

1. $H_1 = \{x \mid x \in V$ and $x_{n-1} = 0\}$,
2. $H_2 = \{x \mid x \in V$ and $(x_{n-1} = 1$ or $(I(x) \notin V)\}$,
3. $V_1 = H_1 - H_2$
4. $V_2 = H_1 \cap H_2$
5. $V_3 = H_2 - H_1$

Definition 5 *[5] Suppose $FH_N = (V, E)$ is an $(n-1)$-dimensional Flexible Hypercube, then the edge set E is the union of E_1, E_2, E_3 and E_4, where*

1. $E_1 = \{(x, y) \mid x, y \in H_1$ and $HD(x, y) = 1\}$,
2. $E_2 = \{(x, y) \mid x, y \in V_3$ and $HD(x, y) = 1\}$,
3. $E_3 = \{(x, y) \mid x \in V_3, y \in V_1$ and $HD(x, y) = 1\}$,
4. $E_4 = \{(x, y) \mid x \in V_3, y \in V_2$ and $HD(x, y) = 2\}$.

Figure 1: A Flexible Hypercube contains 14-node

Addresses of nodes in a Flexible Hypercube are constructed as follows. As discussed above, addresses consist of binary strings of n-bits. The first 2^{n-1} addresses correspond to nodes in H_1 and must be the binary representations of 0 through $2^{n-1}-1$. Each of the remaining nodes (up to $2^{n-1}-1$ nodes) in the set $V_3 = H_2 - H_1$ may be placed adjacent to any node x in H_1 and is given the address $I(x)$. Any node in H_1 is a Hamming distance of 1 from at most one node in V_3. This method of node addressing effectively relaxes the constraint that all nodes in the network must be numbered consecutively. This is unique among the hypercube topologies mentioned above. Notably, supercubes and hypercubes are both special cases of Flexible Hypercubes. In addition to expanding the Flexible Hypercube incrementally, it can also be expanded flexibly with respect to the placement of new nodes in the system while maintaining fault-tolerance. When a new node is added to a Flexible Hypercube system, n new connections should be added and at most $(n-1)$ existing edges must be removed.

An inevitable consequence of the flexibility of construction and the fault tolerance of the Flexible Hypercube is an uneven distribution of the utilized communication ports over system nodes. Although the Flexible Hypercube loses its property of regularity, more links help obtain the replacement nodes of the faulty nodes of the Flexible Hypercube. The Flexible Hypercube with 14-node is shown in Figure 1. In the Figure 1, $H_1 = \{0,1,2,3,4,5,6,7\}$, $H_2 = \{1,3,8,10,12,13,14,15\}$, $V_1 = \{0,2,4,5,6,7\}$, $V_2 = \{1,3\}$ and $V_3 = \{8,10,12,13,14,15\}$.

Lemma 1 *[1] A double-rooted complete binary tree can be contained in a hypercube with dilation* 2*, congestion* 1*, expansion* 1 *and load* 1.

Lemma 2 *There exists an embedding of a complete binary tree T_h in a $(2^{h+1}-2)$-node Flexible Hypercube.*

Proof. The total number of nodes of a complete binary tree T_h is 2^{h-1}. The nodes set of H_1 of the Flexible Hypercube is a hypercube and it has 2^h nodes. We infer the method of the embedding by lemma 1. There exists an embedding of DT_h in a 2^h-node hypercube. The expansion is $(2^{h+1}-2)/(2^h-1) = 2$. Therefore, the T_h can be embedded into a $(2^{h+1}-2)$-node Flexible Hypercube with dilation 2, congestion 1, load 1 and expansion 2. ■

Lemma 3 *[10]A $(2^{h+1}-2)$-node Flexible Hypercube contains an embedding of T_h.*

Proof. The total number of nodes in a complete binary tree DT_h is 2^h. Because the node set of H_1 of the Flexible Hypercube is a hypercube, it has 2^h nodes. We can infer that the embedding method is from lemma 1. There exists an embedding of DT_h in a 2^h-node hypercube. The expansion is $(2^{h+1}-2)/2^h - 1 = 2$. Therefore, T_h can be embedded into a $(2^{h+1}-2)$-node Flexible Hypercube with dilation 2, congestion 1, load 1 and expansion 2. ■

Lemma 4 *[10]There exists an embedding of T_h in a $(2^{h+1}-2)$-node Flexible Hypercube with dilation* 4*, congestion* 1*, expansion* 2*, load* 1 *and $O(n)$ faults.*

Lemma 5 *[10]The embedding methods in the Flexible Hypercube are optimized mainly for balancing the processor and communication link loads.*

3 N-Expansion Embedding

Now, we extend the result from 2-expansion to n-expansion. In other words, we eliminate the limitation of expansion. We assume the total number of nodes of Flexible Hypercube FH_N is N, $2^{n-1} \le N < 2^n$ and the total number of nodes of a complete binary tree T_m of height $(m-1)$ is $2^m - 1$.

Lemma 6 *A complete binary tree of height $(m-1)$ can be embedded into a $(2^n - t)$-node Flexible Hypercube $(0 \le t \le 2^{n-1}, m < n)$ with dilation* 2 *and load* 1.

Proof. The result is trivial from lemma 1. ■

We present these algorithms as follows:

$\mathbf{Algorithm\ replacing-method}$:

1. *if* the root r is faulty *then*
  1.1 search the other root node r'
  1.2 *if* the other root node r' is faulty *then*
    1.2.1 return the root r
    1.2.2 $replacing - rule(r)$
  1.3 *else*
    1.3.1 node r is replaced by node r'.
    1.3.2 exit the $algorithm\ \ replacing - method$

2 *if* the other node x is faulty *then*
  2.1 $replacing - rule(x)$

$\mathbf{Algorithm\ Replacing-rule(x)}$

1 $i = 0$; $j = 0$

2 *while* $i \le (n - m + 1)$ *do*
  2.1 we can search the node ϖ

/* $\varpi \in V$, $HD(x,\varpi) = 1$, $Dim(x,\varpi) = m+i$*/.

2.2 *if* node ϖ is not a virtual node and it is free then

2.2.1 node x is replaced by node ϖ

2.2.2 remove all of nodes in a queue

2.2.3 exit the *while-loop*

2.3 $put(\varpi, i+m-1)$ in a queue

2.4 $i = i+1$; $j = j+1$

2.5 end;

3 *while* the queue is not empty *do*

3.1 remove the first pair (α,β) from the queue

3.2 *if* $\alpha \in V_1$ *then*

3.2.1 $i = 0$

3.2.2 *while* $i \leq (\beta+1)$ *do*

3.2.2.1 we can search the node λ

/* $\lambda \in V$, $HD(\alpha,\lambda) = 1$, $Dim(\alpha,\lambda) = i$*/.

3.2.2.2 *if* node λ is not a virtual node and it is free then

3.2.2.2.1 node x is replaced by node λ

3.2.2.2.2 exit the *while-loop*

3.2.2.3 $i = i+1$; $j = j+1$

3.2.2.4 end;

3.3 else $v - replacing - rule(\alpha,\beta)$

3.4 end;

4 *if* $j = [(n-m)(n+m+1)/2]$ then

4.1 declare the replaceable node of searching is faulty.

4.2 exit the $replacing - rule(x)$

Algorithm v − replacing − rule(α,β)

1 $i = 0$

2 *while* $i \leq (\beta+1)$ *do*

2.1 we can search the node k

/* $k \in V_3$, $HD(\alpha,k) = 2$, $Dim(\alpha,k) = (\beta-1,i)$, $E(\alpha,k) \in E_3$*/.

2.2 *if* node k is not a virtual node and it is free *then*

2.2.1 node r is replaced by node k

2.2.2 exit the *while-loop*

2.3 $i = i+1$; $j = j+1$

2.4 end;

The searching path of the replacing node of a Flexible Hypercube is shown as follows.

$node\ 0 = 0X_{n-2}X_{n-3}...X_{m+1}X_mX_{m-1}...X_1X_0$

$node\ 1 = 0X_{n-2}X_{n-3}...X_{m+1}X'_mX_{m-1}...X_1X_0$

$node\ 2 = 0X_{n-2}X_{n-3}\cdots X'_{m+1}X_mX_{m-1}\cdots X_1X_0$

$\vdots$

$node\ (n-m-1) = 0X'_{n-2}X_{n-3}...X_{m+1}X_mX_{m-1}...X_1X_0$

$node\ (n-m) = 1X_{n-2}X_{n-3}...X_{m+1}X_mX_{m-1}...X_1X_0$

$node\ (n-m+1) = 0X_{n-2}X_{n-3}...X'_mX_{m-1}...X_1X'_0$

$node\ (n-m+2) = 0X_{n-2}X_{n-1}...X'_mX_{m-1}...X'_1X_0$

$\vdots$

$node\ (n-m+m) = 0X_{n-2}X_{n-1}...X'_mX'_{m-1}...X_1X_0$

Figure 2: Embed a T_2 into a FH_{14}

Figure 3: The root node 0 is faulty

$node\ (n-m+m+1) = 0X_{n-2}X_{n-1}...X'_{m+1}X_mX_{m-1}...X_1X'_0$

$\vdots$

$node\ [(n-m)(n+m+1)/2] = 1X'_{n-2}X_{n-1}...X_mX_{m-1}...X_1X_0$

We illustratetwo examples of finding a replaceable node as shown in Figure 2 to Figure 4.

Theorem 7 *The ending of searching path includes at least* $\{[(n-m)(n+m+1)]/2 - t\}$ *nodes.*

Proof. By lemma 3, we can embed a complete binary tree into a Flexible Hypercube from node 0 to node (2^m-1), which can be expressed by a m-bit binary string $i_{m-1}...i_0$ where $i_p \in \{0,1\}$. First, we can change a bit in a sequence from bit m to bit $(n-1)$ and push the node in the queue. We can get $(n-m)$ different nodes. Second, we pop the node from the queue. From the first node we can change a bit in a sequence from bit 0 to bit $(m-1)$, and we can get m different nodes. Then, we can change a bit in a sequence from bit 0 to bit m from the second node and we can get $(m+1)$ different nodes. Until the queue is empty we can get the sum of searching of nodes is $[m+(m+1)+...+(n-1)]$. The ending of searching path includes $(n-m)+[m+(m+1)+...+(n-1)] = (n-m)+[(n-m)(n+m-1)]/2 = [(n-m)(n+m+1)]/2$ nodes. We assume we have t virtual

Figure 4: The other node 2 is faulty

nodes. Therefore, in the worst case we can search at least $\{[(n-m)(n+m+1)]/2-t\}$ nodes. By [13] and [14], we infer the edges of the replacing-method exist and none of node has a duplicate searching. ■

Theorem 8 *If the root of the tree is faulty and the number of faulty nodes is less than* $\{[(n-m)(n+m+1)]/2+1-t\}$ *, we can find the replaceable node of node in* $\{[(n-m)(n+m+1)]/2+1\}$ *iterations.*

Proof. We assume that we can not find the replaceable node of the faulty node. That is, all of nodes on the searching path are already used or fault. So, at least we can search the other root node r' and $\{[(n-m)(n+m+1)]/2-t\}$ nodes. In the worst case, the searching path includes t virtual nodes. Therefore, we can search $\{[(n-m)(n+m+1)]/2+1-t\}$ nodes in $[(n-m)(n+m+1)]/2+1]$ iterations. Because the number of faulty nodes is less than $\{[(n-m)(n+m+1)]/2+1-t\}$, we can find the replaceable node. The originally assumption is wrong. We can find the replaceable node of the root r in $\{[(n-m)(n+m+1)]/2+1\}$ iterations. ■

Theorem 9 *If a node of a subtree is faulty and the number of faulty nodes is less than* $\{[(n-m)(n+m+1)]/2-t\}$*, we can find the replaceable node of faulty node in* $\{[(n-m)(n+m+1)]/2\}$ *iterations.*

Proof. We assume that we can not find the replaceable node of the faulty node. That is, all of nodes on the searching path are already used or fault. So, at least we can search the $\{[(n-m)(n+m+1)]/2-t\}$ nodes. In the worst case the searching path is including t virtual nodes at most. Therefore, we can search $\{[(n-m)(n+m+1)]/2-t\}$ nodes in $[(n-m)(n+m+1)]/2]$ iterations. Because the number of faulty nodes is less than $\{[(n-m)(n+m+1)]/2-t\}$, we can find the replaceable node. The originally assumption is wrong. We can find the replaceable node of the other node x in $\{[(n-m)(n+m+1)]/2\}$ iterations. ■

Theorem 10 $O(n^2-m^2)$ *faults can be tolerated.*

Proof. By theorem 8, there are $\{[(n-m)(n+m+1)]/2+1-t\}$ faults can be tolerated. By theorem 9, there are $\{[(n-m)(n+m+1)]/2-t\}$ faults can be tolerance. To sum up, we can show that there exist $O(n^2-m^2)$ faults can be tolerated.

Theorem 11 *These results hold dilation* 4*, congestion* 1 *and load* 1.

Proof. We show that we can embed a complete binary tree of height m into a (2^n-t)-node Flexible Hypercube using nodes of $V_1 \cup V_2$ with dilation 2.

Case 1. First, If a node x of a subtree is faulty, we can search the node ϖ, $HD(x,\varpi)=1$ by the $replacing-rule(x)$. Second, If the node ϖ is used or fault, we can search the other nodes ϖ, $HD(\varpi,\lambda)=1$ by the $replacing-rule(x)$. At last, we can get the dilation 2 in the worst case.

Case 2. First, if the root node r is faulty, we can search the other root node r'. If the other root node r' is faulty, we can search the node ϖ, $HD(x,\varpi)=1$ by the $replacing-rule(x)$. If the node ϖ is used or fault, we can search the other nodes ϖ, $H.D.(\varpi,\lambda)=1$ by the $replacing-rule(x)$. At last, we can get the dilation 2 in the worst case.

Because every replaceable path is only one path by the $algorithm\ replacing-method$, we can get congestion 1 and load 1. Therefore, when the root node and spacer node are faulty, it is a worst case the dilation $=2+2=4$. However, the dilation is $=1+2=3$ in others condition. The other costs associated with graph mapping are congestion 1 and load 1. ■

4 Conclusion

In this paper, we develop new algorithms to facilitate the embedding complete binary tree. We infer n-expansion from 2-expansion. Our results demonstrate that $O(n^2-m^2)$ faults can be tolerated. Also, the methodology is proven and an algorithm is presented to solve them. These existent parallel algorithms on complete binary tree architectures to be easily transformed to or implemented on Flexible Hypercube architectures with load 1, congestion 1 and dilation 4.

After any arbitrarily complete binary tree structures can be reconfiguring in a Flexible Hypercube with faulty nodes, we are also interested in the mapping of an arbitrary binary tree and multi-dimensional meshes into a Flexible Hypercube with faulty nodes. In addition, several variations of the hypercube structure have been proposed and investigated in recent years to overcome the shortcomings of the topology of the hypercube. In the future, we will develop these algorithms to facilitate the embedding job in other hypercube-derived computers.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: numerical methods*, Prentice Hall, Englewood Ciffs, New Jersey, 1989.

[2] L. N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hypercube Structure for a Computer Networks," *IEEE Trans. Comput.*, Vol. C-33, pp. 323-333, 1984.

[3] V. Chaudhary and J. K. Aggarwal, "Generalized Mapping of Parallel Algorithms onto Parallel Architectures," *Proc. International Conf. on Parallel Processing*, pp. 137-141, 1990.

[4] S. Dutt, and J. P. Hayes, "An automorphism approach to the design of fault-tolerance Multiprocessor," *Proc. 19^{th} International Symp. on Fault-Tolerant Computing*, 1989.

[5] T. Hameenanttila, X.-L. Guan, J. D. Carothers and J.-X. Chen, "The Flexible Hypercube: A New Fault-Tolerant Architecture for Parallel Computing," *J. Parallel and Distributed Comput.*, Vol. 37, pp. 213-220, 1996.

[6] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a Hypercube in the Presence of Faults," *ACM Theory of Computing*, pp. 274-284, 1987.

[7] H. P. Katseff, "Incomplete Hypercube," *IEEE Trans. Comput.*, Vol. C-37, pp. 604-607, 1988.

[8] H.-C. Keh and J.-C. Lin, "On fault-tolerant embedding of Hamiltonian cycles, linear arrays and rings in a Flexible Hypercube," *Parallel Computing*, pp. 769-781, Vol. 26, 2000.

[9] J.-C. Lin and H.-C. Keh, "Reconfiguration of Complete Binary Trees in Full IEH Graphs and Faulty Hypercube," *International Journal of High performance Computing Applications*, Vol. 15, No. 1, pp. 55-63, 2001.

[10] J.-C. Lin, T.-H. Chi, H.-C. Keh and A.-H. A. Liou, "Embedding of Complete Binary Tree with 2-expansion in a Faulty Flexible Hypercube," *Journal of Systems Architecture*, Vol. 47, No. 6, pp.543-548, 2001.

[11] F. J. Provost and R. Melhem, "A Distributed Algorithm for Embedding Trees in Hypercubes with Modifications for Run-Time Fault Tolerance," *J. Parallel Distributed Comput.*,Vol. 14, pp. 85-89, 1992.

[12] D. A. Rennels, "On implementing Fault-tolerance in binary hypercubes," *Proc. 16^{th} International Symp. on Fault-tolerant Computing*, pp. 344-349, 1986.

[13] Y. Saad and M. H. Schultz, "Topological Properties of Hypercube," *IEEE Trans. Comput.*, Vol. 37, No. 7, pp. 867-872, 1988.

[14] A. Sen, "Supercube: An Optimally Fault Tolerant Network Architecture," *Acta Informatica*, Vol. 26, pp. 741-748, 1989.

[15] S. B. Tien, C. S. Raghavendra, and M. A. Sridhar, "Generalized Hypercubes and Hyperbus structure for a computer network,"*Hawaii International Conf. on System Science*, pp. 91-100, 1990.

Supporting the Development of Time-Triggered Co-Operatively Scheduled (TTCS) Embedded Software Using Design Patterns

Michael J. Pont
Embedded Systems Laboratory, Department of Engineering,
University of Leicester, University Road, LEICESTER, LE1 7RH, United Kingdom.
M.Pont@le.ac.uk

Keywords: Pattern, Design Pattern, Embedded System, Co-operative, Time-Triggered, Microcontroller

Received: July 16, 2002

We are concerned in this paper with the use of "design patterns" to facilitate the development of software for embedded systems. The particular focus is on embedded software with a time-triggered architecture, using co-operative task scheduling. Such "TTCS" software is known to have very predictable behaviour: such a characteristic is highly desirable in many applications, including (but not restricted to) those with safety-related or safety-critical functions. In practice, TTCS archi-tectures are less widely employed than might be expected, not least because care must be taken during the design and implementation of such systems if the theoretically-predicted behaviour is to be obtained. In this paper, we seek to demonstrate that the use of appropriate patterns can greatly simplify the process of creating effective TTCS software.

1. Introduction

As the title suggests, we are concerned in this paper with the development of software for embedded systems. Typical application areas for this type of software range from passenger cars and aircraft through to common domestic equipment, such as washing machines and microwave ovens.

The particular focus of the work discussed here is on the use of "patterns" to design and implement software for embedded systems. Current work on patterns was originally inspired by the publications of Christopher Alexander and his colleagues (e.g. Alexander *et al.*, 1977; Alexander, 1979). Alexander is an architect who described what he called "a pattern language" relating various architectural problems (in buildings) to good design solutions. He defines patterns as "a three-part rule, which expresses a relation between a certain context, a problem, and a solution" (Alexander, 1979, p.247).

Alexander's concept of descriptive problem-solution mappings was adopted by Ward Cunningham and Kent Beck who used this approach as the basis for a small pattern language intended to provide guidance to novice Smalltalk programmers (Cunningham and Beck, 1987). This work was in turn built upon by Erich Gamma and colleagues who, in 1995, published an influential book on general-purpose object-oriented software patterns (Gamma *et al.*, 1995). Since the mid 1990s, the development of pattern-based design techniques has become an important area of research in the software-engineering community. Gradually, the focus has shifted from the use, assessment and refinement of individual patterns, to the creation of complete pattern languages, in areas including telecommunications systems (Rising, 2001), and systems with hardware constraints (Noble and Weir, 2001).

Despite the fact that pattern-based (software) design techniques were initially developed to match the needs of the developers of desktop systems, we argue in this paper that pattern-based design has the potential to become an particularly useful adjunct to existing techniques for developing *embedded* systems. To support this argument, we employ a realistic case study to illustrate how patterns can be applied in a typical embedded project.

We begin the main part of the paper by considering some of the important characteristics of embedded software in greater detail.

2. Designing "co-operative" software

Embedded software is often described in terms of communicating tasks (e.g. Nissanke, 1997; Shaw, 2001). The various possible system architectures may then be characterised in terms of these tasks: for example, if the tasks are invoked by aperiodic events (typically implemented as hardware interrupts) the system may be described as 'event triggered' (Nissanke, 1997). Alternatively, if all the tasks are invoked periodically (say every 10 ms), under the control of a timer, then the system may be described as 'time triggered' (Kopetz, 1997). The nature of the tasks themselves is also significant. If the tasks, once invoked, can pre-empt (or interrupt) other tasks, then the system is said to be 'pre-emptive'; if tasks cannot be interrupted, the system is said to be co-operative.

Various studies have demonstrated that, compared to pre-emptive schedulers, co-operative schedulers have a number of desirable features, particularly for use in safety-related systems (Allworth, 1981; Ward, 1991; Nissanke, 1997; Bate, 2000). Set against this is the fact that the creation of TTCS architectures requires careful design and implementation if the theoretically-predicted improvements in system reliability are to be realised in practice (e.g. Pont, 2001).

The main concern expressed about the use of co-operative scheduling is that long tasks will have an impact on the responsiveness of the system. This issue is succinctly summarised by Allworth: *"[The] main drawback with this [co-operative] approach is that while the current process is running, the system is not responsive to changes in the environment. Therefore, system processes must be extremely brief if the real-time response [of the] system is not to be impaired."* (Allworth, 1981).

Concerns of this nature are justified: any co-operative system that has been designed without considering issues of task duration is likely to prove extremely unreliable. However, there are a number of different techniques that may be employed in order to ameliorate such problems. For example, there are some basic 'brute force' solutions:

- By using a faster processor, or a faster system oscillator, we can reduce the duration of 'long' tasks.

Other alternatives include:

- Splitting up 'long tasks' (triggered infrequently) into shorter 'multi-stage' tasks (triggered frequently), so that the processor activity can be more evenly distributed.
- Using 'time out' mechanisms to ensure that tasks always complete within their alloted time.
- Employing a 'hybrid' scheduler, thereby retaining most of the desirable features of the (pure) co-operative scheduler, while allowing a single long (pre-emptible) task to be executed.
- Making use of an additional processor, and a 'shared-clock' scheduler, to obtain a true multi-tasking capability.

In the right circumstances, each of these ideas can prove effective. However, such observations do not, on their own, make it very much easier for developers to deploy TTCS architectures. Instead, what is needed is a means of what we might call 'recycling design experience': specifically, we would like to find a way of allowing less experienced software engineers to incorporate successful solutions from previous TTCS designs in their own systems.

This is - of course - precisely the type of problem which pattern-based design is intended to address (e.g. Gamma *et al.*, 1995).

3. Patterns for embedded systems

In 1996 we began to assemble a collection of patterns to support the development of TTCS embedded systems. We have now described more than seventy patterns (see Pont, 1998; Pont *et al.*, 1999; Pont, 2001; Pont and Banner, in press; Pont and Ong, in press), which we will refer to here as the 'PTTES collection'.

To illustrate what is involved in pattern-based design, we have reproduced one of the components from the PTTES collection in an appendix to this paper. The pattern we have chosen is **MULTI-STATE TASK**. Please note that to meet the size constraints of this paper, the pattern has been edited slightly: however, the key features have been retained.

As you examine this pattern, please note the following:

- The core of the pattern is a link between a particular problem (in a given context), and a solution to this problem, as originally laid out by Alexander (1979). Note that the solution is not necessarily unique, and many patterns (with different names) may share the same context and problem statements.
- It is sometimes assumed that a (software) pattern is simply a code library. It should be clear from **MULTI-STATE TASK** that this is not the case. Of course, some code is included: however, the pattern also includes a broad discussion of the problem area, a presentation of a solution, and a discussion of the consequences of applying this solution.
- Like most of the PTTES patterns, **MULTI-STATE TASK** has links to 'related patterns and alternative solutions'. This is one way of helping the user of the patterns to complete a complex design, and / or to help highlight alternative design solutions.
- While the basic pattern structure used will be familiar to users of "desktop" patterns (e.g. see Gamma *et al.*, 1995), sections of particular relevance to embedded developers are also included. For example, hardware resource implications, and safety implications, are explicitly addressed.

In practice, while **MULTI-STATE TASK** is useful in its own right, it is rare to use only a single pattern to develop any system; indeed, even where a single pattern is implemented, various other patterns may be considered as different design options are reviewed.

For example, as we noted earlier in this paper, TTCS systems that are designed without due consideration being given to task durations are likely to prove extremely unreliable. The following patterns directly address such issues:

- The processor patterns (**STANDARD 8051**, **SMALL 8051**, **EXTENDED 8051**) allow selection of a processor with performance levels appropriate for the application.
- The oscillator patterns (**CRYSTAL OSCILLATOR** and **CERAMIC RESONATOR**) allow an appropriate choice of oscillator type and oscillator frequency to be made, taking into account system performance (and, hence, task duration), power-supply requirements, and other relevant factors.
- The various Shared-Clock schedulers (**SCC SCHEDULER**, **SCI SCHEDULER (DATA)**, **SCI SCHEDULER (TICK)**, **SCU SCHEDULER (LOCAL)**, **SCU SCHEDULER (RS-232)**, **SCU SCHEDULER (RS-485)**) describe how to schedule tasks on multiple processors, which still maintaining a time-triggered system architecture.
- Using one of the Shared-Clock schedulers as a foundation, the pattern **LONG TASK** describes how to migrate longer tasks onto another processor without compromising the basic time-triggered architecture.
- **LOOP TIMEOUT** and **HARDWARE TIMEOUT** describe the design of timeout mechanisms which may be used to ensure that tasks complete within their alloted time.
- **MULTI-STAGE TASK** discusses how to split up a long, infrequently-triggered task into a short task, which will be called more frequently. **PC LINK (RS232)** and **LCD CHARACTER PANEL** both implement this architecture.
- **HYBRID SCHEDULER** describes a scheduler that has most of the desirable features of the (pure) co-operative scheduler, but allows a single long (pre-emptible) task to be executed.

4. Applying the patterns

In order to illustrate why we believe that patterns are likely to prove particularly beneficial to developers of embedded systems, we will consider the design of an "embedded" cruise-control system (CCS) for a passenger car.

4.1 System requirements

A CCS is often used to illustrate the use of real-time software design methodologies (for example, see Hatley and Pirbhai, 1987; Awad *et al.*, 1996). Such a system is usually assumed to be required to take over the task of maintaining the vehicle at a constant speed even while negotiating a varying terrain, involving, for example, hills or corners in the road. Subject to certain conditions (typically that the vehicle is in top gear and exceeding a preset minimum speed), the cruise control is further assumed to be engaged by the driver via 'cruise button' adjacent to the steering wheel, and disengaged by touching the brake pedal.

More specifically, we will assume that the CCS (illustrated in Figure 1) is required to operate as follows:

- When the key is turned in the car ignition, the CCS will be activated. When initially activated, the CCS is in 'Idle' state.
- In **Idle** state, no changes to the throttle setting will be made. The system remains in this state until the user presses the 'Cruise' switch adjacent to the steering wheel: the system then emits one brief 'beep', and enters 'Initialization' state.
- In **Initialization** state, the CCS will determine the current vehicle speed and gear setting. If the vehicle is [a] exceeding MINIMUM_SPEED by at least 5 mph; [b] is no more than 5 mph less than MAXIMUM_SPEED; [c] is in top gear; and [d] the brake pedal is not depressed, the system will emit two brief 'beeps' and enter 'Cruising' state. If these conditions are not met, the system will emit one sustained 'beep' and return to 'Idle' state.
- On entry to **Cruising** state, the system will measure the current speed: this represents the speed at which the user wishes to travel (referred to here as DESIRED_SPEED). The CCS will attempt to adjust the throttle setting in order to maintain the vehicle within +/- 2 mph of DESIRED SPEED at all times. If at any time [1] the speed of the vehicle exceeds MAXIMUM_SPEED, or [2] the speed of the vehicle drops below MINIMUM_SPEED, or [3] the Cruise switch is pressed, or [4] the brake pedal is pressed, then the CCS will emit two sustained 'beeps' and then return to Idle state.
- Like many automotive systems, the application will be used in range of vehicles using the Controller Area Network (CAN) bus (see Lawrenz, 1997, for details of CAN). Appropriate use of this bus should be considered as part of the design process.

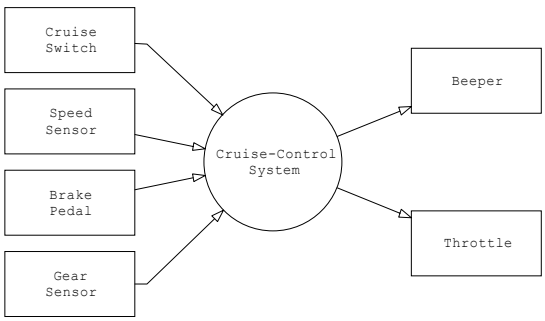


Figure 1: A Context diagram representing the CCS we will explore in this example.

Overall, while our system is somewhat simplified, it will be adequate for our purposes here.

4.2 Start with one node (or less)

As the cost of microcontroller hardware continues to fall, the use of more than processor is becoming increasingly common. For example, a typical automotive environment now contains more than 40 embedded processors (Leen *et al.*, 1999).

In this case, as we noted in the initial specification, it is highly likely that the CCS would be implemented as a multi-processor design, linked over the CAN bus. While, as we will discuss later in this example, the PTTES collection includes support for CAN, we generally advocate an incremental approach to the development of multi-processor systems. Specifically, we usually begin construction of systems using a single-processor prototype; in some circumstances (where the processing required is particularly complex) we may use a desktop PC for some of the prototyping (Pont, in preparation).

Informally, we can say that the aim of this approach is "to get a basic system running as quickly as possible, and then - gradually - refine it". It should be noted that this type of incremental development approach has an important role in recent "extreme programming" methodologies (Beck, 2000). As we will demonstrate, one consequence of the use of a consistent pattern language is that the conversion from single-processor designs to multi-processor designs is greatly simplified.

In this case, we will begin the system development using a single (embedded) processor.

4.3 Work in from the outside

The software 'glue' used to link embedded processors with external components (ranging from switch, keypads, LEDs and high-power AC or DC loads) is a key part of all software designs. Identifying and applying patterns that can match these requirements will, in most applications, constitute a large part of the design effort.

We will consider the interface software, and hardware, required to match the design in this section.

Switch interfaces

In the case of the CCS, we need to link the processor to three switches: one for the Cruise request (to indicate that the use wishes to engage or disengage the CCS), one for the brake sensor (to disengage the CCS), and one from the gearbox (to determine whether the vehicle is in top gear).

From developers without experience in embedded systems, the design of a switch interface can seem rather trivial. However, issues such as switch bounce and the need to consider the impact of electrostatic discharge (ESD) can make the design of reliable switch interface rather more involved. There are therefore four different patterns in the PTTES collection to support the design of switch interfaces. Inspection of the various switch patterns will reveal that, of these, **SWITCH INTERFACE (HARDWARE)** will probably prove most appropriate in these circumstances.

Buzzer interface

We need to control a small buzzer, which - according to the specification - will be sounded to indicate the state of the system.

For these purposes, a small piezo-electric buzzer will be appropriate: these generate a high-volume output at low voltages (3V - 5V), and low currents (around 10 mA). Reviewing the various DC load patterns in the PTTES collection, it is clear that the port pins on a typical microcontroller can be set at values of either 0V or 5V under software control. Each pin can typically sink (or source) a current of around 10 mA. With care, the port may be used to directly drive low-power DC loads, such as the buzzer we require here: **NAKED LOAD** describes how to achieve this safely.

Note that **NAKED LOAD** is concerned only with the hardware aspects of the LED Interface: however, the 'Related Patterns' section of **NAKED LOAD** emphasises the link to the pattern **PORT I/O**, where the relevant software issues are considered.

Throttle interface

To design the throttle interface, we will assume that the throttle will be controlled by a solenoid, and that the throttle position will be varied by means of the DC drive voltage.

To generate the variable DC voltage required, the pattern **HARDWARE PWM** can be used to support the design of a pulse-width modulated output.

In this case (unlike the 'beeper'), the current and voltage requirements will far exceed the very limited capability of most microcontroller port pins: some form of driver circuit will therefore be required. Seven different patterns for controlling DC loads are presented in the PTTES collection: of these, **MOSFET DRIVER** will probably be the most appropriate for use here.

Measuring the speed of the vehicle

As the final part of the interface design, we need to find a means of measuring the current speed of the vehicle. As the basis of this, we will assume the presence of a standard pulse transducer on one or more wheels of the vehicle: this transducer will be assumed to generate a sequence of square-wave pulses, with a frequency (or pulse rate) proportional to the vehicle speed.

Two patterns are provided in the PTTES collection which will directly support the processing of signals from such a transducer: **HARDWARE PULSE COUNT** and **SOFTWARE PULSE COUNT**. Either pattern could form the basis of a successful design in this case.

4.4 The control algorithm

When the user presses the cruise switch, the CCS much check to see that the speed and gear conditions are met. If they are not, then the system will remain under manual speed control.

If the pre-conditions are met, the job of the CCS is to record the current vehicle speed and make appropriate adjustments to the current throttle setting, in order to ensure that - as far as possible - this speed is maintained.

Implicit in the specification is that the driver will - reasonably - expect the system to operate as follows:

- If the vehicle encounters a disturbance (for example, the car drives up a steep hill) the vehicle will - inevitably - slow down. The CCS must not take "a long time" (more than a few seconds) to return the vehicle to the required speed.
- The specification says that, in "steady state" conditions (for example, on a flat, straight, road), the CCS must maintain precisely the required speed (+/- 2 mph). In addition, we assume that the speed must not "oscillate" (for example, change repeatedly from 1 mph too fast to 1 mph too slow, etc).

To meet these requirements, we need to consider the control algorithm that will be used to keep the speed at the required level while the vehicle is in the Cruise state.

Of the various possible control algorithms we could employ, Proportional Integral Differential control is the most widely used: an inspection of the pattern **PID CONTROLLER** suggests that this algorithm will be appropriate in this application. It also provides implementation details for a suitable controller, and guidance on the setting of the P, I and D parameters.

4.5 The software architecture

At this stage, having reviewed the relevant interface and control patterns, we are in a position to identify the basic tasks that will be performed by the CCS:

- The various switches (cruise, brake, gear) will be polled regularly (typically every 50 ms, or so).
- The buzzer will be sounded (as required).
- The vehicle speed will be measured (every 100ms will probably be sufficient; tests on a prototype are the only reliable way of confirming this).
- The new throttle setting will be calculated, once every 100 ms (see above), using the PID control algorithm (when the vehicle is cruising).
- The throttle sending will be varied, again every 100 ms (when the vehicle is cruising).

As with most of the (single-processor) designs created using the PTTES collection, the pattern **CO-OPERATIVE SCHEDULER** (described in detail in Pont, 2001) will provide the core of the system architecture for the CCS. Briefly, this pattern describes how to schedule tasks to run periodically, at pre-defined times. The "operating system" that results is created entirely in the C programming language, and is highly portable.

Please note that we assume that the CCS will be initialised every time the car is used, and will remain inactive until the Cruise switch is pressed. The result will be a three-state design, which may well benefit from the use of the architecture described in **MULTI-STATE TASK** (see Appendix).

4.6 Moving to a multi-processor design

After appropriate prototyping and testing has been conducted using the single-processor prototype, then a multi-processor prototype will be constructed.

If we review the various multi-processor patterns in the PTTES collection, **SCC SCHEDULER** seems to be the basis of the most appropriate design. This pattern describes how multiple processors can be linked using a Controller Area Network (CAN) protocol, as required by the CCS specification.

Various possible multi-processor designs could be considered for this system. For example, the sensing of vehicle speed could take place on one node, with the control algorithm implemented on a second node, and throttle control carried out by a third node. This might prove to be a particularly flexible arrangement because - in some vehicles in a range - it may well be that it is possible to obtain data about the vehicle speed from an existing sensor (over the CAN bus), and / or that the throttle actuator is already in use as part of the (manual) vehicle speed control. The different nodes (Speed Node, Control Node, Throttle Node) may therefore not be all required on all vehicles.

Whatever final design is chosen, the common (TTCS) nature of all the patterns in the collection mean that it is generally very easy to move tasks between nodes as different designs are investigated.

5. Conclusion

At the start of this paper, we suggested that that pattern-based design has the potential to become an particularly useful adjunct to existing techniques for developing embedded systems. Having sought to illustrate how patterns can be used to support the development of an embedded CCS, we return to consider this issue.

Existing design techniques for all forms of software-rich systems include "structured" approaches (e.g. DeMarco, 1978; Hatley and Pirbhai, 1987) and the "Unified Modelling Language" (UML; Fowler and Scott, 2000). Such techniques provide effective, standard, notations

for recording design decisions: however, they do not provide any means of substituting for the lack of experience on the part of a particular designer. The consequence is not difficult to predict, and is summarised succinctly in this quotation from an experienced developer of embedded systems: "It's ludicrous the way we software people reinvent the wheel with every project" (Ganssle, 1992).

At the most basic level, patterns allow us to address such problems by promoting the re-use of good designs decisions.

The effect of patterns-based design is - we would argue - likely to be particularly evident in the embedded sector, for reasons that are illustrated in the CCS example consider earlier in the paper. Like many embedded applications, the successful development of the CCS system (without any patterns) requires knowledge and / or experience in many different areas, including programming, electronics, the CAN bus, mathematics, basic signal processing and control systems. The wide range of fields required to complete this development is, while not unknown, certainly much less common in the "desktop" sector. Pattern-based design allows us to present the information required to develop such multi-disciplinary systems in a very effective way.

To conclude, we should emphasise that software patterns should not be seen as an attempt to produce a panacea or what Brooks (1986) calls a 'silver bullet' for the problems of embedded software design or implementation. Patterns may assist in the rapid development and testing of appropriate designs, but it is not feasible to provide all software engineers or their managers, irrespective of background or training, with sufficient knowledge of relevant fields to ensure that they can, for example, create appropriate designs for aircraft flight control systems or fault diagnosis systems based on sliding-mode observers. However, what we may be able to achieve is to make software managers, and the teams they manage, better able to recognise projects in which it would be advisable to appoint (say) an artificial intelligence, signal processing or control expert from within the company on the project team, or to employ an outside consultant to fulfil such a rôle.

Acknowledgement

The author is grateful to the anonymous reviewers for their constructive comments on the first draft of this paper.

References

Alexander, C. (1979) *"The Timeless Way of Building"*, Oxford University Press, NY.

Alexander, C., Ishikawa, S., Silverstein, M. with Jacobson, M. Fisksdahl-King, I., Angel, S. (1977) *"A pattern language"*, Oxford University Press, NY.

Allworth, S.T. (1981) *"An Introduction to Real-Time Software Design"*, Macmillan, London.

Awad, M., Kuusela, J. and Ziegler, J. (1996) *"Object-oriented technology for real-time systems"*, Prentice-Hall, New Jersey, USA.

Bate, I. (2000) "Introduction to scheduling and timing analysis", in *"The Use of Ada in Real-Time System"* (6 April, 2000). IEE Conference Publication 00/034.

Beck, K. (2000) *"Extreme Programming Explained"*, Addison Wesley.

Brooks, F.P. (1986) "No silver bullet - essence and accidents of software engineering," in H.J. Kugler (Ed.) *Information Processing 86*, Elsevier Science, Amsterdam. Pp.1069-1076.

Cunningham, W. and Beck, K. (1987) "Using pattern languages for object-oriented programs", Proceedings of OOPSLA'87, Orlando, Florida.

DeMarco, T. (1978) *"Structured analysis and system specification",* Prentice Hall, New Jersey.

Fowler, M. and Scott, K. (2000) *"UML Distilled" (2nd Edition)*, Addison-Wesley, Reading, MA.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *"Design patterns: Elements of reusable object-oriented software"*, Addison-Wesley, Reading, MA.

Ganssle, J. (1992) *"The art of programming embedded systems"*, Academic Press, San Diego, USA.

Hatley, D.J. and Pirbhai, I.A. (1987) *"Strategies for real-time system specification",* Dorset House.

Hatley, D.J. and Pirbhai, I.A. (1987) *"Strategies for real-time system specification",* Dorset House.

Lawrenz, W. (1997) *"CAN System Engineering"*, Springer.

Leen, G., Heffernan, D. and Dunne, A. (1999) "Digital networks in the automotive vehicle", *Computing and Control*, 10(6): 257-266.

Nissanke, N. (1997) *"Realtime Systems"*, Prentice-Hall.

Noble, J. and Weir, C. *"Small Memory Software"*, Addison Wesley, 2001.

Pont, M.J. (1998) "Control system design using real-time design patterns", Proceedings of Control '98 (Swansea, UK), September 1998, pp.1078-1083.

Pont, M.J. (2001) *"Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of microcontrollers"*, ACM Press / Addison-Wesley, UK

Pont, M.J. (in preparation) "Supporting Extreme Embedded Programming (XEP) using patterns", to be submitted to EuroPLoP 2003.

Pont, M.J. and Banner, M.P. (in press) "Designing embedded systems using patterns", to appear in *Journal of Systems and Software*.

Pont, M.J. and Ong, H.L.R. (in press) "Using watchdog timers to improve the reliability of TTCS embedded systems: Seven new patterns and a case study", to

appear in the proceedings of VikingPLOP 2002, Denmark, September 2002.

Pont, M.J., Li, Y., Parikh, C.R. and Wong, C.P. (1999) "The design of embedded systems using software patterns", Proceedings of Condition Monitoring 1999 [Swansea, UK, April 12-15, 1999] pp.221-236.

Rising, L., [Ed.] (2001) "*Design Patterns in Communications Software*", Oxford University Press.

Shaw, A.C. (2001) *"Real-Time Systems and Software"*, John Wiley & Sons, New York.

Ward, N. J. (1991) "The static analysis of a safety-critical avionics control system", in Corbyn, D.E. and Bray, N. P. (Eds.) "*Air Transport Safety: Proceedings of the Safety and Reliability Society Spring Conference, 1991"* Published by SaRS, Ltd.

Appendix

We present an abbreviated version of the pattern **MULTI-STATE TASK** (from Pont, 2001) in this appendix.

MULTI-STATE TASK

Context

- You are developing software for an embedded application.
- The application has a time-triggered architecture, constructed using a scheduler.

Problem

How do you replace multiple tasks in an application with a single task that performs different activities depending on the current state of the system (and why is it - sometimes - a good ideas to do so)?

Background

[Some "background" material is included in the full version of this pattern. It is omitted here.]

Solution

MULTI-STAGE TASK encapsulates a system architecture that is apparent in many well-designed embedded applications.

To understand the need for this architecture, consider a simple washing-machine control system (Figure MST-1).

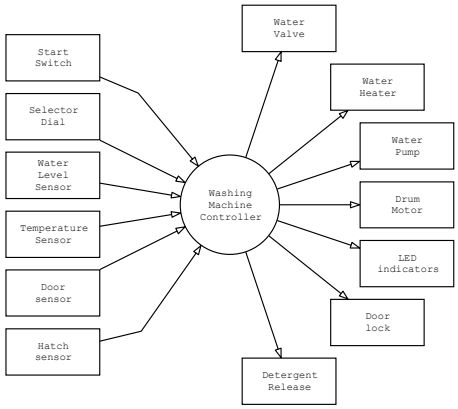


Figure MST-1: Outline design.

Here is a brief description of the way in which we expect the system to operate:

1. The user selects a wash program (e.g. 'Wool', 'Cotton') on the selector dial.
2. The user presses the 'Start' switch.
3. The door lock is engaged.
4. The water valve is opened to allow water into the wash drum.
5. If the wash program involves detergent, the detergent hatch is opened. When the detergent has been released, the detergent hatch is closed.
6. When the 'full water level' is sensed, the water valve is closed.
7. If the wash program involves warm water, the water heater is switched on. When the water reaches the correct temperature, the water heater is switched off.
8. The washer motor is turned on to rotate the drum. The motor then goes through a series of movements, both forward and reverse (at various speeds) to wash the clothes. (The precise set of movements carried out depends on the wash program that the user has selected.) At the end of the wash cycle, the motor is stopped.
9. The pump is switched on to drain the drum. When the drum is empty, the pump is switched off.

The description is simplified for the purposes of this example, but it will be adequate for our purposes here.

Based on the above description we will try to identify some of the functions that will be required to implement this system. A provisional list might be as follows:

- `Read_Selector_Dial()`
- `Read_Start_Switch()`
- `Read_Water_Level()`
- `Read_Water_Temperature()`
- `Control_Detergent_Hatch()`
- `Control_Door_Lock()`
- `Control_Motor()`
- `Control_Pump()`
- `Control_Water_Heater()`
- `Control_Water_Valve()`

Now, suppose we wish to identify the **tasks** to be scheduled (co-operatively) in order to implement this application. Based on the above list, it may be tempting to conclude that each of the functions listed above should become a task in the system. However, while it would be possible to work in this way, **this would be likely to lead to a complex and cumbersome system implementation**.

To see why this is so, take one example: the function `Control_Water_Heater()`. We want to heat the water only at particular times during the wash cycle. Therefore, if we want to treat this as a task and schedule it - say every 100 ms - we need to creation an implementation something like the following:

```
void TASK_Control_Water_Heater(void)
   {
   if (Switch_on_water_heater_G == 1)
      {
      Water_heater = ON;
```

```
      return;
      }

   // Switch off heater
   Water_pin = OFF;
   }
```

What this task does when it is executed is to check a flag: if it is necessary to heat the water, it starts to do so: otherwise, it stops the heating process.

There are two problems with creating the program in this way:

- We are going to end up with large numbers of tasks (very large numbers in a more substantial application), most of which - like this task - actually do very little. In applications without external memory this is a particular problem, because each task will consume some of the limited memory (RAM) resources.
- It is not at all clear which, if any, of these tasks will actually set the flag (`Switch_on_water_heater_G`), or the other similar flags that will be required in the other tasks in this application.

In practice, what we require in this and many similar applications is a single 'System Update' task: this, as we will see is a task that will be regularly scheduled and will, where necessary, call functions - like `Control_Water_Heater()` as and when required.

In the washing machine, this system update task will look something like the code in the Listing MST-1.

```
void Update(void)
   {
   static tWord Time_in_state;

   switch (System_state_G)
   {
   case START:
      {
      // Lock the door
      Control_Door_Lock(ON);

      // Start filling the drum
      Control_Water_Valve(ON);

      // Release the detergent (if any)
      if (Detergent_G[Program_G] == 1)
         {
         Control_Detergent_Hatch(ON);
         }

      // Ready to go to next state
      System_state_G = FILL_DRUM;
      Time_in_state_G = 0;

      break;
      }

   case FILL_DRUM:
      {
      // Remain in state until drum is full
      // NOTE: Timeout facility included
      if (++Time_in_state_G >= MAX_FILL_TIME)
         {
         // Drum should be fully by now...
         System_state_G = ERROR;
         }

      // Check the water level
      if (Read_Water_Level() == DRUM_FILLED)
         {
         // Does we require hot water?
         if (Hot_Water_G[Program_G] == 1)
            {
            Control_Water_Heater(ON);

            // Ready to go to next state
            System_state_G = HEAT_WATER;
            Time_in_state_G = 0;
            }
         else
            {
            // Using cold water only
            // Ready to go to next state
            System_state_G = WASH_01;
            Time_in_state_G = 0;
            }
         }
      break;
      }
...
   }
```

Listing MST-1: Part of a possible implementation of the single task used to implement a washing-machine control system.

Listing MST-1 is a representative example of a **MULTI-STAGE TASK**.

We can describe the simplest form of this architecture as follows:

- The system involves the use of a number of different functions
- The functions are always called in the same sequence.
- The functions are called from a single task, as required.

Note that variations on this theme are also common: for example, the functions may not always be called in the same sequence: the precise sequence followed (and the particular set of functions called) will frequently depend on user preferences, or on some other system inputs.

Hardware resource implications

This architecture makes very efficient use of system resources.

Reliability and safety implications

There are no specific reliability or safety implications.

Portability

This high-level pattern is highly portable.

Overall strengths and weaknesses

☺ **MULTI-STAGE TASK** encapsulates a simple architecture that matches the needs of many embedded applications

Related patterns and alternative solutions

MULTI-STAGE TASK combined with **ONE-TASK SCHEDULER** [Pont, 2001, p.749] - and / or with **ONE-YEAR SCHEDULER** [Pont, 2001, p.755] provides a very simple and efficient system architecture with minimal CPU, memory and power requirements.

Example: Traffic Lights

[A detailed example is included in the full version of this pattern. It is omitted here.]

The GAT Approach to Specifying Mixed Systems

Jean-Claude Royer
Département d'Informatique de l'Ecole des Mines de Nantes,
4, rue Alfred Kastler. B.P. 20722 F-44307 NANTES Cedex 3
Phone: +33 2 51 85 82 05, Fax: +33 2 51 85 82 49
E-mail: Jean-Claude.Royer@emn.fr

Keywords: Dynamic Behaviour, Algebraic Data Type, Symbolic Transition System, Synchronous Product, Concurrency, Communication, Formal Specification, Mixed System.

Received: March 10, 2002

This paper outlines a practical use of algebraic specifications for the development of heterogeneous software systems. This kind of systems mixes several viewpoints, e.g. static, functional and dynamic aspects. Writing, from scratch, an algebraic specification for such systems is quite difficult, so we developed the concept of Graphic Abstract Data Type (GAT). In this paper we present a method to build an algebraic specification of a sequential system via a symbolic transition system (STS). The STS models both the dynamic aspects and the static aspects of the system. The STS is also the basis of an algorithm that computes the functional aspects of the system (an algebraic specification). Computing the specification is partly automatic, this improves the compatibility between the aspects. This approach is extended to concurrent and communicating systems by the use of a synchronous product of STSs. We proved that the STS is an abstract interpretation of the generated specification. We demonstrate that the set of axiom may be transformed into a terminating term rewriting system. Then from the generation method of the specification the properties of consistency and completeness are got and this ensures the existence of a partial initial algebra. We showed that the synchronous product of GATs preserves the state predicates, the preconditions and the definedness predicate of the components. We also give sufficient conditions to get the GAT determinism and the GAT compactness of the product of two GATs.

1 Introduction

Modelling heterogeneous, or mixed, software systems requires the integration of several paradigms. These paradigms relate, at least, to three of the main aspects of systems: the static, the functional and the dynamic aspects. Static aspects deal with the signatures, the types and the relations between types. Functional aspects describe the semantics for operations or explicit some conditions and invariants. Dynamic aspects focus on the so-called dynamic behaviour of systems. It is related to concurrency, synchronizations and communications. The main issues with mixed systems are to ensure the consistency between the different aspects and to provide assistance for specifying systems and proving properties.

Algebraic specifications of abstract data types [BWP84, Wir90] are suited for the specification of both static and functional aspects of a system. Algebraic specifications are modular (a collection of data types) and abstract (the properties are set by axioms). Numerous tools and techniques can help proving the specification properties. The algebraic techniques are less straight applicable to dynamic systems because data types evolve, communicate and act concurrently [ABR99]. In this area, we suggested [AR00] the Graphic Abstract Data Type (GAT) model: a symbolic transition system that helps one to build algebraic specifications of components. This model is quite operational and fully integrated into conventional algebraic specifications. As in process algebra two kinds of components are distinguished: sequential components and concurrent components. For each component two views are considered: the dynamic view and the functional view. The dynamic view describes the static and dynamic aspects of the component, it uses a notion of *Finite and Symbolic Transition System* (STS). The functional view is an algebraic specification of a partial abstract data type [BWP84, Wir90]. We define a notion of compatibility between a STS and an algebraic specification to express the consistency between the two views. The STS defines a partial equivalence relation over the data type.

One problem with such an approach is to prove dynamic properties. We address this in [Roy01a], our solution is based on algebraic temporal operators and techniques (possibly automatic) to prove such properties with a general theorem prover.

This paper describes a survey of the GAT approach principles and properties. The notions introduced in this paper have been used for several case studies [AR98, AR99, AR00, Roy01a]. It also has inspired the KORRIGAN model and methods for LOTOS and SDL [PCR99, CPR99]. Last, some of these concepts have been used to improve object-oriented methods like OMT [ABR95] and UML

[PRR02, Roy02].

Section 2 presents a middle size case study: a vending machine. Then, in Section 3, we describe partial abstract data types, our notion of STS and the links between these concepts in the GAT approach. Section 4 details the extraction of the formal specification of our vending machine case study. In Section 5 we justify several properties about our specifications: the STS interpretation, the termination of the term rewriting system, the hierarchical consistency and completeness of the specification and some properties about the product of two GATs. The Section 6 is dedicated to related works, last we conclude and point out future work.

2 The Vending Machine Case Study

In order to specify a system, a (formal) language is required but also an adequate method. We consider that a preliminary analysis was done and produced a system architecture. This decomposition can be obtained using some already known methods. For instance, methods for LOTOS [Tur93, PCR99] are relevant here.

We deal with a vending machine (a French one) which accepts coins, gets out change and delivers a drink. To simplify, it only accepts coins of one, two or five Francs and gets out only coins of one Franc. The user gives coins, one at a time, and when the sum is sufficient enough he may choose a drink. If this drink is in the stock then the user gets it, else he has to do another choice. The vending machine cannot allow choices if it does not have enough money to get out change to the user. The price of the different kinds of drink are not supposed to be the same but the maximum cost of one drink is assumed to be of five Francs.

We consider the architecture depicted in the Figure 1. The vending machine has two concurrent and communicating parts: a cash changer (the `CC` box) and a drink distributer (the `DD` box). Each part is a component specialized in a set of activities. The `CC` and `DD` components are sequential and the `VM` overall machine is concurrent. The meanings of the gates for communications are explained below. The `GIVE` gate receives coins from the user (one at a time). The `GET` gate is for getting out coins. There are three different cases. It may either get out all the money after a cancellation, or get out the overflow when the user gives too much than required, or get out the change for the difference between the given sum and the price of the drink. `CANCEL` is used to interrupt the transaction, and is a ternary synchronization. `OK` denotes that the sum is sufficient to choose a drink. `CHOOSE` allows one to choose a drink. `GETOUT` means that the `DD` component returns the cost of the chosen drink to the `CC` component. `DRINK` delivers the chosen drink to the user.

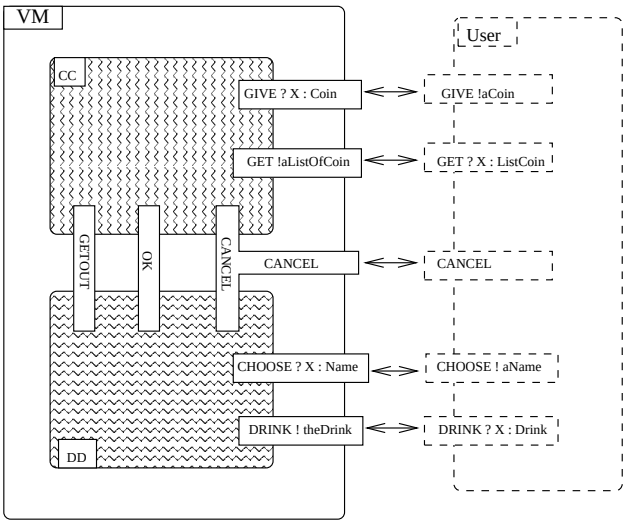


Figure 1: The `Vending Machine` Architecture

3 GAT Principles

This Section introduces the concepts of *Symbolic Transition System* (STS) and *Graphical Abstract Data Type* (GAT).

A GAT for a component is a STS with some static informations and an algebraic specification of a data type. We distinguish two kinds of GAT components: sequential components and concurrent components. For each GAT component we consider two views: the dynamic view and the functional view. The dynamic view is a STS: a finite set of states and a finite set of labelled transitions. Classic finite transition systems, or Labelled Transition Systems (LTSs) have labels which are closed terms. Unlike LTS, our STS labels are operation calls with variables and guards. This concept is related to machines where states and transitions are not necessarily unique objects. A state may represent a set of either finite or infinite objects and a transition collects several state changes. This kind of state machine avoids the state and transition explosion problems and makes dynamic behaviours more readable. The functional view is an algebraic specification of a partial abstract data type [BWP84].

In the GAT process specification we suggest to start from the dynamic view of the components since it is the most abstract view. First, the specifier declares the operations, the conditions and the states of the component. These informations are represented graphically using a STS. Second, the semantics of the operations are provided (in the functional view) by an algebraic specification. Instead of writing this algebraic specification from scratch, we propose a guideline. The core of our extracting method is the AG-derivation algorithm [AR99], which guides the axiom generation using the STS. It provides an operational specification style where axioms may be transformed into left-to-right conditional rewriting rules. In case of concurrent and communicating components, the synchronous product of STSs is used before generating the axioms. Figure 2 de-

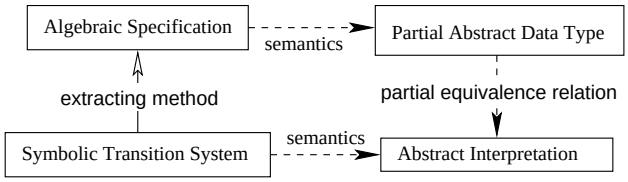


Figure 2: The GAT Semantics

scribes an overview of the GAT process and its semantics. An algebraic specification is extracted from a STS and its semantics is a partial abstract data type. The STS represents a graphic view of a partial equivalence relation over the data type.

3.1 Partial Abstract Data Type

We consider partial abstract data types, because our STSs need partial operations. We consider initial semantics due to its close relation with proofs and deduction. To gain expressiveness we consider hierarchical presentations of algebraic specifications, with constructors, hidden symbols and definedness predicates. The notations below come from [Wir90, BWP84]. Some of our hypothesis may be relaxed but this is beyond the scope of this paper.

A *signature* $\Sigma = (So, F)$ is a tuple where So is a set of sorts and F a So-indexed family of function symbols such that F is equipped with a mapping $type : F \rightarrow So^* \times So$. When there is no ambiguity, we identify f and $type(f)$. As usual, arity, argument types and range type are defined upon f.

Definition 3.1 *A hierarchical presentation for a type TI, called the Type of Interest, is a $Spec_{TI}$ tuple*

$$< \Sigma_{TI},\ H\Sigma_{TI},\ E_{TI},\ \Gamma_{TI},\ Def_{TI},\ Spec_P >$$

where:

- *Σ_{TI} is the visible (public) signature ($<\{TI\},\ F_{TI}>$), where each $f \in F_{TI}$ has at least one occurrence of TI.*
- *$H\Sigma_{TI}$ is the set of hidden symbols (sorts and functions $< HS_{TI},\ HF_{TI} >$), $f \in HF_{TI}$ has at least one occurrence of TI.*
- *E_{TI} is a set of conditional axioms, each axioms has at least one occurrence of an operation from F_{TI} or HF_{TI}.*
- *Γ_{TI} the set of constructors, we require that constructors of visible sorts are visible functions ($\Gamma_{TI} \subseteq F_{TI}$).*
- *Def_{TI} is the set of definedness formulas, they denote when the result of an operation on a term built from sort TI is defined.*
- *$Spec_P$ the primitive presentation part (a hierarchical presentation or an empty one).*

This definition allows several layers with only one new defined sort, it may easily extend to several new sorts in the same layer. We assume that $Spec_{TI}$ is always an enrichment of boolean.

An *internal operation* has TI as resulting type. An *external operation* (or *observer*) does not have TI as resulting type. A *basis* internal operation has not any parameter of type TI. We also distinguish *constructor* or *generator* as the subset of the internal operations sufficient to generate all the value of the data type.

As in [Wir90], we associate to $Spec_{TI}$ a *hierarchical specification*

$$SP = < \Sigma,\ H\Sigma,\ E,\ Cons,\ D,\ P >$$

with respectively signature, hidden signature, axioms, constructors, definedness predicate and primitive part. The primitive part P is the specification associated to the primitive presentation part $Spec_P$. We define $\Sigma = \bigcup_s \Sigma_s, H\Sigma = \Sigma \cup \bigcup_s H\Sigma_s,\ E = \bigcup_s E_s,\ Cons = \bigcup_s \Gamma_s,\ D = \bigcup_s Def_s$ where $\bigcup_s$ is done for every sort $s \in So$. The set of sort So is the set of all the sorts defined in the hierarchical presentation.

For every sort $s \in So$, $T(\Sigma, X)_s$ is the set of terms with variables of sort s. The set of terms without variables or *ground terms* of sort s is $T(\Sigma)_s$. Given $s \in S_P$, S_P is the set of primitive sort, the term $t \in T(\Sigma, X)_s$ is said to be of *primitive sort* if $t \in T(\Sigma_P, X)_s$. We note $H\Sigma_{Cons} = < HS,\ Cons >$ the constructor signature.

A model of the specification is a partial algebra A satisfying the axioms, and we note t^A the interpretation of the symbol t in A. As usual A_s is the carrier set of values of sort s.

A $H\Sigma$-algebra A is *reachable* w.r.t $Cons$ if for each $s \in HS$ and each carrier element $a \in A_s$, $\exists t \in T(H\Sigma_{Cons}) \wedge a = t^A$. $Gen_{Cons}(H\Sigma)$ is the set of reachable $H\Sigma$-algebras and $Gen_{Cons}(H\Sigma, E)$ is the set of reachable $H\Sigma$-algebras which are models of $< H\Sigma,\ E >$. Let $\Sigma \subseteq \Sigma'$ and A a Σ'-algebra, $A_{|\Sigma}$ is the Σ-restriction of A.

Partial algebras are algebras where the functions may be partial. Functions are assumed to be strict. A *partial algebra* A is a total algebra such that the interpretation of any term t, of sort s, in A is defined if and only if A satisfies a definedness formula $D_s(t)$. The $=$ symbol stands for strong equality (*i.e.* the two values are both defined and equal or they are both undefined) and $\stackrel{e}{=}$ stands for existential equality (*i.e.* the two values are both defined and equal). The use of definedness predicates implies that such partial algebras satisfy $D(true) \wedge D(false) \wedge true \neq false$. All the equations occurring in the GAT axioms are restricted to existential ones ($\stackrel{e}{=}$). Notions of homomorphisms and valuations may be defined, note that variable quantifications range over defined values ($D_s(X)$). We consider total Σ-homomorphism, *i.e.* a So-family of partial functions $h : A \longrightarrow B$ such that $\forall f : s_1,\ ...,\ s_n \longrightarrow s$ and $\forall a_i \in A_{s_i}$ then $D(f^A(a_1,\ ...,\ a_n)) => D(f^B(h_{s_1}(a_1),\ ...,\ h_{s_n}(a_n)))$

and $h(f^A(a_1, ..., a_n)) = f^B(h_{s_1}(a_1), ..., h_{s_n}(a_n))$.

$Formula(\Sigma, X)$ is the set of formulas built over Σ and a set of variables X. It contains Σ-equations, boolean constructions and first-order quantifiers.

$$\Phi, \Psi ::= \quad t = t' \mid \neg\Phi \mid \Phi \wedge \Psi \mid \Phi \vee \Psi \mid \Phi => \Psi \\ \mid (\forall x : s.\Phi) \mid (\exists x : s.\Phi)$$

Let $v : X \longrightarrow A$ a total valuation, the satisfaction of a formula $f \in Formula(\Sigma, X)$ is noted $A, v \models f$. Dropping the valuation, $A \models f$ means that the satisfaction is true for all total valuations. We restrict to positive conditional axioms: $\bigwedge_{1 \leq i \leq l} u_i \stackrel{e}{=} v_i \implies C$, where C has the form $t = t'$ or $D(t)$. Deduction will be noted $E \vdash t = u$.

$Mod(SP)$ (the models of SP) is the class of reachable and partial Σ-algebras which are restriction of a partial $H\Sigma$-algebra satisfying E and such that the restriction to primitive signature is a model of the primitive specification part.

$$Mod(SP) = \quad \{A \in Gen(\Sigma) \\ \mid \exists B \in Gen_{Cons}(H\Sigma, E), \\ B_{|\Sigma} = A \wedge A_{|\Sigma_P} \in Mod(P)\}$$

The chosen semantics of such a specification is initial.

$$I(SP) = \{I \in Mod(SP) \mid I \text{ is initial in } Mod(SP)\}$$

A partial algebra I is *initial* in a class of algebras if and only if there exists a unique total homomorphism from I to every algebra in the class.

3.2 Consistency and Completeness

The two following definitions are constraints on hierarchical specifications.

Definition 3.2 *A specification is hierarchically consistent if and only if*

- $E \vdash true \neq false$,
- $\forall t_P \in T(\Sigma_P)_s$, $E \vdash D(t_P) => E_P \vdash D(t_P)$, *and*
- $\forall t_P, t'_P \in T(\Sigma_P)_s$, $E \vdash t_P = t'_P => E_P \vdash t_P = t'_P$.

Definition 3.3 *A specification is sufficiently complete if* $\forall t \in T(\Sigma)_s$, $s \in S_P$, $E \vdash D(t) => \exists t_P \in T(\Sigma_P)_s$ *such that* $E \vdash t = t_P$.

These two properties ensure, in case of positive conditional axioms, the initial algebra existence [BWP84].

3.3 Deduction System

We are interested in sensible signatures [HO80]: $\forall s \in So$, $T(H\Sigma_{Cons})_s \neq \emptyset$, *i.e.* each sort contains at least one ground constructor term. It is a sufficient condition to ensure the existence of a reachable algebra. In this context sound and complete deduction systems exist for equational or conditional deduction. Conditional calculus for the partial framework may be found in [BWP84, AC95, CR97]. The $\vdash$ conditional borrows from [CMR99]. Let Φ a set of positive conditional axioms, ϕ, ψ, ϵ with possible subscript are positive conditional axioms. The rules are described in Figure 3. By adding infinite induction to the conditional

$$\frac{}{\Phi \vdash \phi} \text{ if } \phi \in \Phi$$

$$\frac{}{\Phi \vdash x \stackrel{e}{=} y => y \stackrel{e}{=} x}$$

$$\frac{\Phi \vdash t_1 \stackrel{e}{=} t_2}{\Phi \vdash D(t)} \ \forall\, t \text{ subtem of } t_1 \text{ and } t_2$$

$$\frac{\Phi \vdash \phi}{\Phi \vdash (\bigwedge_{x \in X_s} x \stackrel{e}{=} \Theta(x)) => \phi[\Theta]} \\ \text{for } \theta : X \rightarrow |T_{\Sigma(Y)}| \text{ with } D(\phi[\Theta])$$

$$\frac{\Phi_1 \vdash \phi_1 \wedge ... \wedge \phi_n => \psi_i \quad \Phi_2 \vdash \psi_1 \wedge ... \wedge \psi_k => \epsilon}{\Phi_1 \cup \Phi_2 \vdash (\ \psi_1 \wedge ... \wedge \psi_{i-1} \wedge \ \phi_1 \wedge ... \phi_n \wedge \ \psi_{i+1} \wedge ... \wedge \psi_k\ ) => \epsilon}$$

$$\frac{\Phi \vdash \forall Y.\phi}{\Phi \vdash (\bigwedge_{y \in Y_s, s \in S} D(y)) => \phi}$$

$$\frac{\Phi \vdash \phi}{\Phi[\Theta] \vdash \phi[\Theta]} \\ \text{for } \theta : X \rightarrow |T_{\Sigma(Y)}| \\ \text{with } D(\phi[\Theta]) \text{ and } D(\Phi[\Theta])$$

Figure 3: The Conditional Deduction System

deduction system we get the inductive theory, the calculus will be noted $\vdash_i$ [Wir90]. We rather use a presentation with generators as in [GG88], *structural induction* for TI is defined as follows. Assume x is a variable, $b_1, ..., b_n$ are basis generators and $g_1, ..., g_m$ are recursive generators with only one argument of sort TI (this restriction is easily removed). $c : TI$ is a fresh constant not occurring in α, $g_k(c)$ is a call of g_k with $c : TI$ and fresh variables for other arguments.

$$\frac{\Phi \vdash_i \phi[b_k/x]\ 1 \leq k \leq n \quad \Phi \cup \{\phi[c/x]\} \vdash_i \phi[g_k(c)/x]\ 1 \leq k \leq m}{\Phi \vdash_i \phi}$$

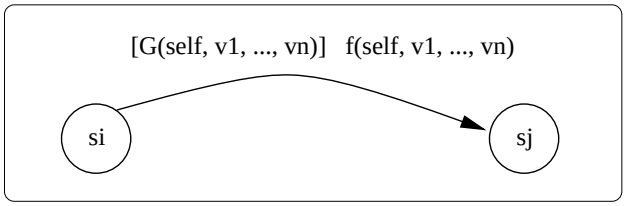


Figure 4: Edge of the STS

3.4 Symbolic Transition System

The finite state machine formalism is well-known by practitioners. It is well-suited to the description of interactions and controls. One problem with such a formalism is the great number of states and transitions. For instance, one can combine states into super-states or aggregate states as in [Har87]. However when the system has not a finite or bounded number of state one must use more powerful concepts. It often happens if one has a mixed system with both control and data types. We define the notion of finite and symbolic transition system. This notion arises also from the need of a full semantics for language like LOTOS [STB97] and in the AltaRica formalism [APGR00]. See Figure 5 page 95 for an example. Our STSs also provide super-states (see [AR99]).

Let $St = \{s_i \ 1 \leq i \leq n\}$ a set of identifiers called *the set of states*. A symbolic transition systems is a finite set of states St and a finite set of labelled transitions Tr. The Figure 4 illustrates the graphical presentation of such a transition, `G` is the guard and $f(self, v_1, ..., v_n)$ the term label. $self$ in this figure denotes a variable associated to TI, and $v_i : R_i$ are variables. Note also that we allow receipt variables both in guard and in term labels. Variables occurring in the guard and in the term label are not necessarily the same, but to simplify we consider the same set of variables in both terms. Symbols and terms occurring in the STS must be interpreted in the context of the algebraic specification. The notation $A_{TI} \cap D^A_{TI}$ denotes the subset of the defined values of the carrier set. We explicitly uses the definedness predicate for TI even if it is not required, however, to simplify notations we assume $D_s(t)$ for every primitive term.

Definition 3.4 *Given a $Spec_{TI}$ specification and A a model of it; we define an associated symbolic transition system as a couple (St, Tr):*

1. *the states are $St = \{s_i, \ 1 \leq i \leq n\}$, each s_i denotes a subset of A_{TI} and $A_{TI} \cap D^A_{TI} = \bigoplus_{1 \leq i \leq n} s_i$;*
2. *the set of initial states is a subset of St;*
3. *the set of transition Tr is a subset of $St \times St$.*
4. *the transitions Tr of the STS must verify the following interpretation formulas: an edge, from state s_i to state s_j, is labelled by `[G(self, X1, ..., Xn)] f(self, X1, ..., Xn)` if and only if*

$$\forall v \in A_{TI} \cap D^A_{TI}, \ \forall u_i \in A_{R_i}, \ if \ v \in s_i \ \wedge \ G^A(v, u_1, ..., u_n) \ then \ f^A(v, u_1, ..., u_n) \in s_j$$

The transitions correspond to internal operations of TI with an interpretation formula based on state predicates. The term f can be any algebraic term, the equality of terms needs typing information since we have operator overloading. Our notion is more general than the symbolic transition graph defined in [HL95]. We have more general states (not only tuples of conditions) and we have no restriction on variables occurring on transitions.

Definition 3.5 *A STS has maximal transitions if and only if for every term label f it exists at most only one transition $(s, t) \in Tr$ labelled by f.*

From now on we consider STS with maximal transitions. This does not decrease the expressiveness because any STS may be transformed into a STS with maximal transitions by collecting guards of the transitions with the same label from s to t.

3.5 GAT Definition

A Graphic Abstract data Type description is an abstract specification of a data type using a STS (STS_{TI}), a hierarchical presentation as in Section 3.1 ($Spec_{TI}$) and an associated equivalence relation.

$$GAT_{TI} = (STS_{TI}, \approx_{TI}, Spec_{TI})$$

Definition 3.6 *We define the $\approx_{TI}$ partial equivalence relation as:*

$$\forall v, v' \in A_{TI} \cap D^A_{TI}, \ v \approx_{TI} v' \Leftrightarrow (\exists! s_i \in St, \ v \in s_i \wedge v' \in s_i)$$

Let $\{P_{s_i}\}_{1 \leq i \leq n}$ a finite set of boolean functions called *state predicates*. These functions are interpreted as the characteristic functions of the subsets s_i ($P^A_{s_i}(v) \Leftrightarrow v \in s_i$). Each $P^A_{s_i}$ is the characteristic function of an equivalence class of values of $A_{TI} \cap D^A_{TI}$ for any partial algebra model of $Spec_{TI}$.

Lemma 3.7 *$\{P_{s_i}\}_{1 \leq i \leq n}$ verifies the following properties:*

$$\begin{array}{l} \textit{exclusivity:} \\ \forall s_i, s_j, \ s_i \neq s_j => \neg(P_{s_i} \ \wedge \ P_{s_j}) \end{array} \quad (1)$$

$$\textit{complementarity: } D_{TI} = \bigvee_{1 \leq i \leq n} P_{s_i} \quad (2)$$

and conversely, if $\{P_{s_i}\}_{1 \leq i \leq n}$ is a set of state predicate which verifies the two above properties then it defines a partial equivalence relation:

$$\begin{array}{c} \textit{if } D_{TI}(v) \ \wedge \ D_{TIf}(v') \textit{ then} \\ v \approx_{TI} v' <=> \exists s_i \ P_{s_i}(v) \wedge P_{s_i}(v') \end{array}$$

3.6 Notations and Hypotheses

In the sequel we use the following notations: $self : TI$ denotes a variable of type TI, D_{TI} is the definedness predicate for TI, P_{s_i} are the state predicates, $precond_{op}$ is the precondition of the op operation, G will be a guard, $*$ is a tuple of variables, op_B (respectively op_R) denotes a basis internal operation labelling an initial transition (respectively a recursive one labelling a non initial transition). A transition from a source state to a target state will be noted

$$source \xrightarrow{[G(self,*)]op_R(self,*)} target$$

Note that some of the following definitions are higher-order definitions since sometimes they are defined relatively to a state name or a function name. But they are assumed to be expanded into a finite set of first-order formulas.

We consider a *GAT determinism* property; it means that if there are two transitions starting from a given state and with the same label then their guards are exclusive.

Definition 3.8 *A GAT is determinism if and only if for every couple of transitions labelled by the same term either their source states are distinct or their guards are exclusive.*

We also need finitely generated values, *i.e* every values can be denoted by a finite sequence of generators [Wir90]. Then an important hypothesis is about *the generator choice*, here we assume that each TI internal operation is a generator[1]. The reason is that a generator will denote a temporal logic instant, which is assumed distinct from another one. From a practical and operational point of view it does not complicate or grow too much the specification.

The notion of reachability may be adapted for GAT in the following way. A s state will be *strict* if it contains at least one finitely generated value ($\exists self : T(\Sigma_{cons})_{TI},\ D_{TI}(self) \wedge P_s(self)$).

Definition 3.9 *A GAT is compact if and only if $\forall s \in S$ s is strict, where S is the set of states of the STS.*

The previous property states that each state represents at least one finitely generated value. One static and necessary condition to ensure compactness is: *every states may be reached from an initial state*. In the presence of guards, the strictness property is generally undecidable. But this is not a hard requirement because the specifier may ensure it using a similar technique as in [Roy01a].

3.7 GAT Auxiliary Operations

We present in this Section a summary of operations generated by the GAT method. The formulas are the same for sequential or concurrent components. All these formulas may be generated automatically from the STS description. We consider them as total boolean functions inductively defined on the STS.

The definedness predicate is inductively defined by:

$$D_{TI}(op_B(*)) \mathrel{\widehat{=}} precond_{op_B}(*)$$
$$D_{TI}(op_R(self,*)) \mathrel{\widehat{=}} precond_{op_R}(self,*) \wedge D_{TI}(self)$$

This predicate denotes if a given term represents or not a value of the partial algebra.

The operation preconditions have the from:

$$precond_{op_B}(*) \mathrel{\widehat{=}} \bigvee_{\xrightarrow{[G(*)]op_B(*)} target} G(*)$$
$$precond_{op_R}(self,*) \mathrel{\widehat{=}} \bigvee_{\substack{[G(self,*)]op_R(self,*) \\ source \to target}} P_{source}(self) \wedge G(self,*)$$

The precondition for an observer are defined with the same formulas than in the case of a recursive generator.

The last family is the *state predicates*:

$$P_{target}(op_B(*)) \mathrel{\widehat{=}} \bigvee_{\xrightarrow{[G(*)]op_B(*)} target} G(*)$$
$$P_{target}(op_R(self,*)) \mathrel{\widehat{=}} \bigvee_{\substack{[G(self,*)]op_R(self,*) \\ source \to target}} P_{source}(self) \wedge G(self,*)$$

These definitions cope with the need of operation strictness.

3.8 Hierarchical Presentation of a GAT

The hierarchical presentation associated to a GAT is defined as follows.

$$Spec_{TI} = < \Sigma_{TI},\ H\Sigma_{TI},\ E_{TI},\ \Gamma_{TI},\ Def_{TI},\ Spec_P >$$

where

- $\Sigma_{TI} = < \{TI\},\ F_{TI} >$. The set of function F_{TI} contains: proper operations of the STS, the preconditions, the state predicates and the guards.
- $H\Sigma_{TI} = < HS_{TI},\ HF_{TI} >$: sometimes hidden operations are useful for example in the case of data type `Changer` to compute the `toGet` result.
- E_{TI} is compound from the definitions of the preconditions, the state predicates and the guards; in addition it contains the axioms computed by the AG-derivation algorithm.
- Γ_{TI} is the set of generator.
- Def_{TI} contains the definition of the definedness predicate D_{TI} and definition formulas for the observers. These definition formulas are computed by $D(obs(self,*)) = precond_{obs}(self,*) \wedge D_{TI}(self)$.
- $Spec_P$ is the context of the TI definition.

[1] This hypothesis is not required by the GAT extracting method but it comes from the aim to express temporal properties associated with transitions of the STS.

4 Formal Specification of the Vending Machine

In this Section we describe the formal specification of the different components either sequential or concurrent. A comprehensive specification of the case study may be found in [Roy01b].

4.1 The Cash Changer Component

The graphic presentation of the STS for the `CC` component is described in Figure 5. This is basically a guarded finite state machine with some notations to represent operation signature. A transition labelled by an operation name represents the effect of an event occurring on the component. The data type associated to the `CC` component is named `Changer`. Its algebraic specification has a signature and positive conditional axioms. The STS describes the signature following the notations explained below. Solid arrows denote internal operations (`give`, `ok`, `getOut`, `newChanger`, `cancel`). A basis internal operation is depicted with a dashed arrow (`newChanger`). An observer is drawn with a dotted arrow (`toGet`).

The data type must contain all the operations defined in Section 3.7 related to the definedness and the abstract implementation of the state machine. It also contains the operations described in the STS, and additionally we have in the signature of the `Changer` type observers required for guards and communications. We also add the

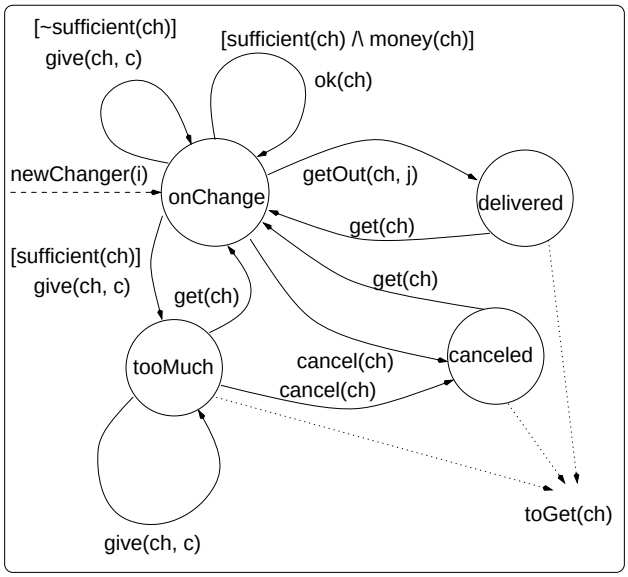


Figure 5: The `CC` STS

`getAll`, `getChange`, and `getOverflow` total (hidden) observers to define the three functionalities of the `toGet` observer.

The axioms for the definedness, preconditions, and state predicates are automatically computed with the formulas of Section 3.7. We use the extracting AG-derivation principles to produce an algebraic specification of the other operations. The general form of the axioms is a positive conditional equation:

`condition => f(self, v`$_1$ `...  v`$_n$`)` $\stackrel{e}{=}$ `rt`

where `f` is an operation name, `v`$_1$ `...  v`$_n$ is a tuple of variable, and `self` a variable of type `Changer`. To extract axioms we use the AG-derivation principles which computes automatically the condition and the left-hand side conclusion term. User interactions are required to get the `rt` right-hand side term. If the user cannot give this term, this means that the value of `rt` depends on the `self` history, *i.e.* the sequences of generator reaching the current state. An AG-derivation of `self` is a substitution of `self` by a generator term in the axiom of `f`. Below are the steps to build the specification of the `toGet` observer. This operation is partial, hence only the states `toomuch`, `canceled`, and `delivered` must be analysed. For example

```
% tooMuch state
tooMuch(self) => toGet(self) = ?
```

For each state we try to give a right-hand side conclusion term. For example in this case we write:

```
% the tooMuch state
tooMuch(self) =>
      toGet(self) = getOverflow(self)

% the canceled state
canceled(self) =>
          toGet(self) = getAll(self)
```

If it is not possible to give an answer the algorithm replaces the `self` variable by the operation calls reaching this state. The conditional part changes according to this replacement and the specifier must provide either the right-hand side terms or the process continue. The derivation process stops either with a non recursive call or in a state already visited. For example with the `delivered` state we have only one total transition reaching this state.

```
% the delivered state
delivered(self) => toGet(self) = ?

% one level of AG-derivation
onChange(self) => toGet(getOut(self, j))
                  = getChange(self, j)
```

More details about the AG-derivation algorithm may be found in [AR99]. We assume that this algorithm ensures the following properties. If $f(t,*) \in T(\Sigma)_P$ and $D_{TI}(t)$ then it exists at least one axiom which may rewrite the term $f(t,*)$. A term t built by the GAT method is defined as soon as variables occurring in it are defined ($\forall r,\ D(r) => D(t[r/x])$).

4.2 The DD Component

The same process is achieved for the other sequential component. We have the STS of Figure 6 and we get an algebraic specification for the associated `Distributer` data type.

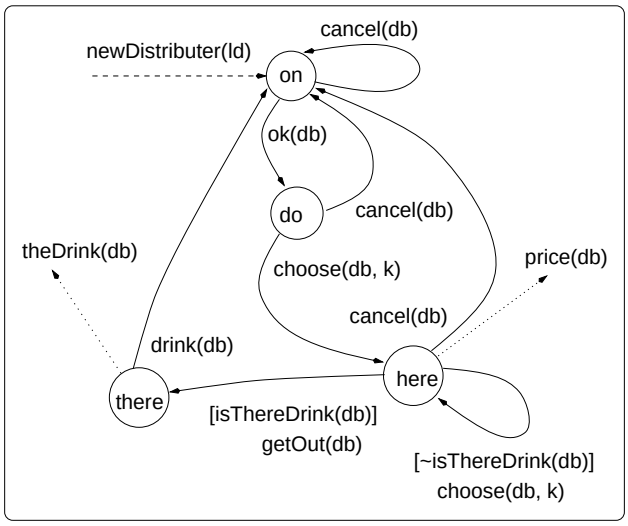


Figure 6: The DD STS

4.3 Concurrent and Communicating GATs

In this Section we describe the composition scheme for components in order to handle concurrency and communications. For example, the VM machine in Figure 1 is composed of a CC and a DD parts. We herein consider a binary product and we also restrict our presentation to one emission and one receipt, however, our constructions extend to nary product and to several communications. The synchronization list denotes the actions which are required to synchronize in each component. Here, there are synchronizations on the `ok, cancel` and `getOut` actions. The semantics of synchronization is obtained from the synchronous product of STSs in a similar way than for the synchronous product of automata [Arn94]. Firstly, we built the free product of the two STSs. Secondly we get out the pair of transitions which are not allowed by the list of synchronizations. Last, the synchronizations are enriched by communications. An algebraic specification is eventually built from the computed STS. Thus both synchronization and communication are integrated in an algebraic style.

4.3.1 Synchronization and Communications

Let $Top = (S^t,\ S_0^t,\ T^t)$ and $Down = (S^d,\ S_0^d,\ T^d)$ be two STSs. A synchronization list V gives the pairs of synchronous actions. The actions not in this list are asynchronous. In our example, the synchronization list is $V =$ `[(ok, ok), (cancel, cancel), (getOut, getOut])`. The synchronous vector is the complete list of actions of the product: `{(ok, ok), (cancel, cancel), (getOut, getOut), (give,` ϵ`),` `(get,` ϵ`), (`ϵ`, drink), (`ϵ`, choose)}`. ϵ denotes no action on the corresponding component. This rule is similar to the LOTOS one, other rules may be possible, for example the CCS rule.

During a synchronization, some values may be emitted or received. Communications may occur during synchronizations in the way depicted in Figure 7. We use `?` to denote a receipt and `!` for an emission. The terms `top(self)` and `down(self)` are algebraic terms denoting the corresponding component processes. It also represents the states $s1$ and $s2$ of the STSs. `actT` (respectively `actD`) is an operation of the top (respectively down) component, and are synchronous actions. A value

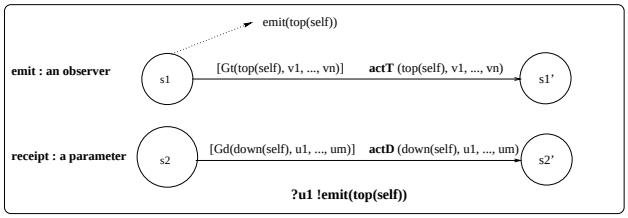


Figure 7: Implementation of Communications

is emitted by an observer and received by the mean of a variable. In the Figure 7 example, the emitted value is `emit(top(self))` and receipt is done with the $\mathtt{u}_1$ variable. A transition of the product is a couple of transitions and the associated guard is the conjunction of the component guards. Thus we get the following condition for the transition expression from (s_1, s_2) to (s_1', s_2'):

```
[Gt(top(self), v1, ..., vn) ∧
Gd(down(self), emit(top(self)),
               u2, ..., um)
```

4.3.2 Synchronous Product of STSs

A table of the legal transitions is built from the synchronization rules (Fig. 8). A transition expression belongs to the table if it is consistent with the synchronization rule. The resulting STS for the product is $(S,\ S_0,\ T)$ where $S_0 = S_0^t \times S_0^d$ and $S \subseteq S^t \times S^d$, S is the subset of states of the free product reachable from S_0 using only the legal transitions. We compute a table of the transitions of the product using a simple algorithm. This table has four

Source State	(s_1, s_2)	...
Transition	`(actT(top(self), v1...vn),` `actD(down(self),` `emit(top(self)), u1...um))`	...
Target State	(s_1', s_2')	...
Guard	`[Gt(top(self), v1...vn)` ∧ `Gd(down(self), u1...um)`	...

Figure 8: The Transition Table

lines and each column represents a source state, a possible transition, the target state and the guard to trigger the transition. The couples of initial states of the components are the initial states of the product. To build the table, the couples of initial states of the product are put in the source state line. Then, the transition expressions starting from a source state and consistent with the synchronization rules are added. The target state and the condition of this transition are also set in the corresponding lines.

4.3.3 The Associated Algebraic Specification

The construction of the corresponding algebraic specification is done in two steps. The first step is automatic and computes a reduced algebraic specification. The resulting specification is sometimes too simple for specifying more complex systems, but we can extend it. It is possible to define other operations (internal or external) in a functional style over the current specification. A second step completes the specification, it requires user-interactions. The STS associated to the product and the AG-derivation is also useful in this case.

Contrary to the approach described in [AR00] the product type is built using a bit different way. This way is more abstract and natural, it is also simpler and more suited to dynamic checking. The most important advantage is that it is uniform and it easily extends to nary product of GATs. This seems a bit hard to write by hand but it can be automatically generated by an extension of the CLAP tools [CPR01].

The type associated to the product is the product of the types associated to the components. $Product$ is the sort associated to the product of the Top sort and the $Down$ sort. To get a GAT, we associate to each pair of actions in the synchronous product of STS an operation name. For example the couple `(actT, actD)` will be named `act`. The profile of this operation is obtained by merging the two component operation profiles coping with emissions and receipts. Because it is a GAT we apply the extracting principles to get an algebraic specification for type $Product$. The definedness predicate, the preconditions and the state predicates are generated in the same way than for the sequential case, see Section 3.7. The rest of the specification is an observational specification of the selectors `top` and `down`. The general principle is illustrated on the example of Figure 7.

```
top(act(self)) = actT(top(self), v1...vn)
down(act(self)) = actD( down(self),
          emit(top(self)), u2...um)
```

Note that these axioms express synchronization between `actT` and `actD` and communication from the `top` part to the `down` part. The above axioms mean that observing the `top` component of a system after an `act` action is an `actT` action on the `top` component. During this `act` action the `DD` component executes an `actD` action. During this synchronization the value `emit(top(self))` is sent to the `down` component. The synchronization between two components may be formalized as a bijective mapping between two execution paths, but we do not detail this here.

4.4 The `VM` Component

Once the `CC` and `DD` components are specified, we build the GAT for the whole `VM` machine (Fig. 9). First, we build the synchronous product of the two previous STSs, this gives us the global dynamic behaviour of the `VM` machine. The `Machine` data type associated with the `VM` component is based on the product of the component data types `Changer` and `Distributer`. The shorthand for guards in Figure 9 are: `S` for `sufficient(theCC(self))`, `M` for `money(theCC(self))`, `IS` for `isThereDrink(theDD(self))`. Basis constructor profiles of the component are defined as the merging of the basis constructor profiles of the components. For the `Machine` we have only one basis constructor with profile `newMachine : Natural, List[Drink] -> Machine`. Two selectors, the `theCC` and `theDD` operations, are defined for the `Machine` to retrieve the `Changer` and `Distributer` component data types. Their profiles are `theCC : Machine -> Changer` and `theDD : Machine -> Distributer`. We associate an operation name to each pair of action in the product. For example the `(getOut, getOut)` synchronous action is also named `getout` (overloading is allowed) and has profile `Machine -> Machine`. Below is the example of the `getout` global synchronization with its communication.

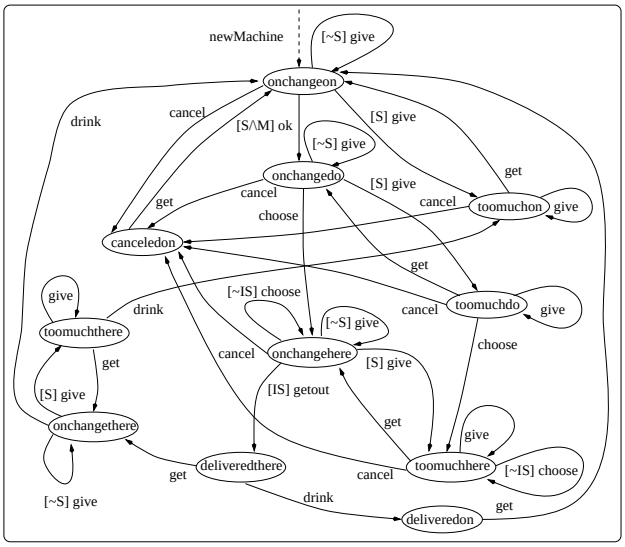


Figure 9: The `VM` STS

```
theCC(getout(self)) =
  getOut(theCC(self), price(theDD(self)))
theDD(getout(self)) = getOut(theDD(self))
```

5 GAT Properties

This Section presents and justifies some properties of the specifications generated by the GAT principles. The computation of the algebraic specification is partly automatic, which is an advantage for non expert specifiers. The resulting specification has also interesting properties.

5.1 The STS Interpretation

In this section we prove some general properties linking our generated algebraic specification and the symbolic system.

We must prove that our extracting method builds an algebraic specification which satisfies the interpretation formulas associated with the transitions (clause C of Definition 3.4).

Fact 5.1 *The extracting GAT method ensures that the interpretation formulas associated with the transitions of the STS are true in the algebraic specification.*

In the two cases (sequential or concurrent) the definition of the state predicate associated to a transition like in Figure 4 is: $P_{s_j}(f(self, v_1, ..., v_n)) = (G(self, v_1, ..., v_n) \wedge P_{s_i}(self)) \vee$ ... Then the formula $G(self, v_1, ..., v_n) \wedge P_{s_i}(self) => P_{s_j}(f(self, v_1, ..., v_n))$ is true.

Lemma 3.7 defines properties (1 and 2) linking definedness predicates with state predicates. The following theorem proves that our extracting method also ensures that the STS defines a partial equivalence relation.

Theorem 5.1 *The state predicates of a deterministic GAT are exclusive and complementary, they define a partial equivalence relation over the values of the data type.*

The two proofs may be done by structural induction on the GAT. Let $v = op_B(v_1, ..., v_n)$ then $\forall s_i \neq s_j$, $\neg(P_{s_i}(op_B(v_1, ..., v_n)) \wedge P_{s_j}(op_B(v_1, ..., v_n)))$ because of the GAT determinism which implies that guards are exclusive. The same analysis is also true if $v = op_R(self, v_1, ..., v_n)$.

The second property is trivial for the op_B case. For the op_R case we have $D_{TI}(op_R(self, v_1, ..., v_n)) = precond_{op_R}(self, v_1, ..., v_n) \wedge D_{TI}(self) = \bigvee_{\substack{[G]op_R \\ source \to target}} P_{source}(self) \wedge G(self, v_1, ..., v_n) \wedge D_{TI}(self)$. The other part is equal to $\bigvee_{1 \leq i \leq n} P_{s_i}(op_R(self, v_1, ..., v_n)) = \bigvee_{1 \leq i \leq n} \bigvee_{\substack{[G]op_R \\ s \to s_i}} P_{s_i}(self) \wedge G(self, v_1, ..., v_n) \wedge D_{TI}(self)$. The two expressions are equal. Note that the complementarity property implies that $\forall s_i \in S$, $P_{s_i} => D_{TI}$. Note also that the below lemma is a mean to check the GAT determinism property.

Lemma 5.2 *If a GAT is compact and its state predicates are exclusive then it is GAT determinism.*

If there is only one state there is not two distinct transitions with the same label then the GAT is deterministic. Let $s_i \neq s_j$ if the STS is not GAT determinism it exists two transitions labelled by the same operation, starting from the same state and with non exclusive guards. If t and t' are the target states of these transitions then $t \neq t'$ since transitions are maximal. Thus it exists a term v such that $P_t(v) \wedge P_{t'}(v)$ and this is not possible.

Our experiences with GAT specification show that writing errors or erroneous definitions of guards arise very often. The exclusivity and complementarity properties are always ensured by means of our axiom generation but assuming some hypothesis. To prove these properties with a tool is a first means to check some problems in the algebraic specification (for example the GAT determinism of the STS). Our experimentations reveal that these proofs are really relevant to detect bad algebraic definitions.

5.2 Termination of the Term Rewriting System

We may implement the specification, transforming axioms into left-to-right rewriting rules. Results on modularity of termination are pertinent here [Der95, FJ95, Rao95, AG00]. There are many works around termination of rewriting systems, but only few of them are relevant to our context, because we have a conditional and hierarchical system. One successful approach is [FJ95]. The principle is to define a hierarchy (or definition ordering) for the set of operation and to use a notion of alien-decreasing system. The alien-decreasing property being a bit technical, we avoid the details here (see [FJ95] and [AR99]). However several difficulties have to be solved before an application in our context:

- To handle mutually recursive definitions and conditional rules is not natural because one has to modify the original specification. This may greatly disturb a non specialist.
- The alien-decreasing property is rather strict and it is easy lost if we add a non alien-decreasing rule.
- Status computation needs an algorithm that is not yet existing.

We investigate other approaches related to modular termination. One theorem of [Der95] applies when we have a non conditional system. Unfortunately there is no theorem related to the conditional case. A general theorem of Gramlich [Gra95] is also important in our context. A rewrite system is *overlay* if every (conditional) critical pairs are obtained by overlapping left-hand sides of rules at top position. AG-derivation prohibits a proper subterm to match a term, thus there is a superposition only if left-hand side conclusion terms are equal. In case of such conditional pairs, the GAT determinism property of the STS ensures that the conditions are exclusive, hence, our system is overlay. These critical pairs are not "properly critical", they are *infeasible*[2] in the sense of [Gra95]. Gramlich's theorem

[2] An infeasible conditional critical pair is obviously joinable.

states: *Any conditional term rewriting system which is an innermost terminating overlay system such that all conditional critical pairs are joinable is terminating and confluent*. This theorem shows that, to get termination, it is sufficient to prove innermost termination. K. Rao [Rao95] defines an approach based on this theorem. He proves a useful result, which unfortunately does not cope with conditional rules. Arts and Giesl, in [AG00], also propose a criterion to prove innermost termination.

As one may see, our systems have many properties (left-linearity, constructor system, conditional and hierarchical, amongst others). We investigate for a proper approach. We suggest to use Σ_{depth} which is a decreasing order [Klo92] based on the depth of the constructor in the term. Each symbol has a weight, for a constructor it is its depth in the term. A term like `sum(give(ch, c))` has $2*weight(sum)$ and `add(c, ch)` has weight 1. With this order we can orient an axiom like: `toomuch(ch) => toGet(give(ch, c)) = overflow(give(ch, c))`. Whenever this fails another level of AG-derivation increases the left conclusion part but may decrease the right-hand side and the condition terms. A last problem to solve is about functional extensions which have no constructor in the left-hand side conclusion term. For instance `money(ch) = inf(ch, stock(ch))`. The solution is to replace, in other rules, the `money` call and to consider such operation with the highest priority than the rest of the rules.

Several experiments, using Larch Prover [GH93], confirm that the termination property is true with our specifications. We experiment with nearly ten systems from 50 to 300 rules with the *dsmpos* ordering (a registered simplification ordering). However, this may require some minor modifications of the specifications. The most often to change the definition order of rules and to replace calls of the functional extensions is sufficient. Sometimes an additional level of AG-derivation is needed or an explicit change of the operator status for the dsmpos ordering.

5.3 Consistency and Completeness

Because of our generating method, once termination is ensured we get consistency and sufficient completeness. The two main reasons are: we get an overlay-confluent rewriting system since the STS is GAT determinism and generated axioms respect the principle of sufficient completeness. These properties and the use of positive conditional axioms ensure the existence of a partial initial algebra [BWP84].

Theorem 5.3 *The specification associated with a GAT is hierarchically consistent and sufficiently complete, then an initial partial algebra exists.*

As seen in the previous Section, if we prove our conditional system terminating then it is convergent and normal forms are unique. Consistency is ensured ($E \vdash true \neq false$) since normal forms are unique and this inequation is true for Boolean. Since we have an enrichment of Boolean, a predefined term is either defined or undefined within E (this is also true within E_P). To prove $E \vdash t_P = t'_P$ we consider two cases: either the two terms are defined in E, from previous point they are both defined in E_P and equal in E_P because of unique normal forms. If the two terms are not defined in E the same is also true in E_P and strong equality holds between them.

There are two ways to prove the sufficient completeness property for a specification: using algorithms (for example [Kou85]) or using an axiom writing method that guarantees sufficient completeness. This later approach is used in Bidoit's work ([Bid82]) and reused here. Let t_P be a term of a predefined sort, if $E \vdash D(t_P)$ then we have $E_P \vdash D_P(t_P)$ from the hierarchical consistency. It is then sufficient to show the existence of a rule to rewrite this term. We consider terms with general form $f(t, *)$ where $f \in \Sigma_P$ and $t \in T(\Sigma)_{TI}$. If $t \in T(\Sigma)_{TI} \wedge D_{TI}(t)$, from properties 1 and 2, then there exists a (single) state s such that $P_s(t)$. A property of the AG-derivation algorithm is to produce a rule which rewrites the term $f(t, *)$ (see Section 4.1).

5.4 Properties of the Product

We prove several results showing that the auxiliary operations for the product are naturally split on the components. To simplify, we only consider two cases one with synchronization and a communication and another one without synchronization. We prove the first theorem about the definition of the state predicates.

Theorem 5.4 *Let the synchronous product of two GAT components: if t (respectively d) is the state of the top component (respectively the down component) then $P_{(t,d)}(self) = P_t(top(self)) \wedge P_d(down(self))$.*

The proof of this theorem is done by induction on the generators of the product. If $self$ is a $op_B(v_1, ..., v_n, u_1, ..., u_m)$ term then $P_{(t,d)}(op_B(v_1, ..., v_n, u_1, ..., u_m)) = \bigvee_{\xrightarrow{[G_t \wedge G_d](op_{B_t}, op_{B_d})}(t,d)} G_t(op_{B_t}(v_1, ..., v_n)) \wedge G_d(op_{B_d}(u_1, ..., u_m))$, and separating top and down expressions we get:

$P_{(t,d)}(op_B(v_1, ..., v_n, u_1, ..., u_m)) = (\bigvee_{\xrightarrow{[G_t]op_{B_t}}t} G_t(op_{B_t}(v_1, ..., v_n))) \bigwedge (\bigvee_{\xrightarrow{[G_d]op_{B_d}}d} G_d(op_{B_d}(u_1, ..., u_m))) = P_t(op_{B_t}(v_1, ..., v_n)) \wedge P_d(op_{B_d}(u_1, ..., u_m))$.

For the case of an op_R with a synchronization and a communication: $P_{(t',d')}(op_R(self, v1, ..., v_n, u_2, ..., u_m)) = \bigvee_{(t,d)\rightarrow(t',d')}^{[G]op_R} P_{(t,d)}(self) \wedge G(self, v1, ..., v_n, u_2, ..., u_m)$

by induction hypothesis we get:

$\bigvee_{\substack{[G(self)]op_R \\ (t,d)\rightarrow(t',d')}} P_t(top(self)) \wedge P_d(down(self))) \wedge G_t(top(self), v1, ..., v_n) \wedge G_d(down(self), emit(top(self)), u_2, ..., u_m)$.
The following expression $P'_t(top(op_R(self, v1, ..., v_n, u_2, ..., u_m))) \wedge P'_d(down(op_R(self, v1, ..., v_n, u_2, ..., u_m)))$ reduces to $P'_t(op_{R_t}(top(self), v1, ..., v_n)) \wedge P'_d(op_{R_d}(down(self), emit(top(self)), u_2, ..., u_m)) = (\bigvee_{\substack{[G_t(self)]op_{R_t} \\ t\rightarrow t'}} P_t(top(self)) \wedge G_t(top(self), v1, ..., v_n)) \wedge (\bigvee_{\substack{[G_d(self)]op_{R_d} \\ d\rightarrow d'}} P_d(down(self)) \wedge G_d(down(self), emit(top(self)), u_2, ..., u_m))$ and the two unions are equal. A simpler decomposition is true in the case of an asynchronous recursive call.

Theorem 5.5 *With the same conditions as in the previous theorem: if op_R is a synchronization with a communication as in Figure 7 then $precond_{op_R}(self, v_1, ..., v_n, u_2, ..., u_m) = precond_{op_{R_t}}(top(self), v1, ..., v_n) \wedge precond_{op_{R_d}}(down(self), emit(top(self)), u_2, ..., u_m)$; if op_R is an asynchronous call then $precond_{op_R}(self, v_1, ..., v_n) = precond_{op_{R_t}}(top(self), v1, ..., v_n) \wedge D_{Down}(down(self))$.*

The proof of this theorem is similar to the previous one. If $self$ is a $op_B(v_1, ..., v_n, u_1, ..., u_m)$ term then we have $precond_{op_B}(v_1, ..., v_n, u_1, ..., u_m) = precond_{op_{B_t}}(v1, ..., v_n) \wedge precond_{op_{B_d}}(u_1, ..., u_m)$. $D_{Product}(op_B(v_1, ..., v_n, u_1, ..., u_m)) = precond_{op_B}(v_1, ..., v_n, u_1, ..., u_m) = precond_{op_{B_t}}(v1, ..., v_n) \wedge precond_{op_{B_d}}(u_1, ..., u_m) = D_{Top}(op_{B_t}(v1, ..., v_n)) \wedge D_{Down}(op_{B_d}(u_1, ..., u_m))$. For the case of an op_R constructor with synchronization and communication the same way is successful taking into account the induction hypothesis. $precond_{op_R}(self, v1, ..., v_n, u_2, ..., u_m) = \bigvee_{\substack{[G]op_R \\ (t,d)\rightarrow(t',d')}} P_{(t,d)}(self) \wedge G(self, v1, ..., v_n, u_2, ..., u_m)$ and a similar decomposition as in the previous theorem gives the result. If op_R is an asynchronous call $precond_{op_R}(self, v1, ..., v_n) = \bigvee_{\substack{[G]op_R \\ (t,d')\rightarrow(t',d')}} P_{(t,d')}(self) \wedge G(self, v1, ..., v_n) = \bigvee_{\substack{[G]op_R \\ (t,d')\rightarrow(t',d')}} P_t(top(self)) \wedge G_t(self, v1, ..., v_n)) \wedge P'_d(down(self))$. We have $precond_{op_{R_t}}(self, v1, ..., v_n) = \bigvee_{\substack{[G_t]op_{R_t} \\ t\rightarrow t'}} P_t(left(self)) \wedge G_t(self, v1, ..., v_n))$ and $D_{Down}(down(self)) = \bigvee_{d'} P'_d(down(self))$ from property 2. We may verify the equality of the two expressions.

Theorem 5.6 *With the conditions of theorem 5.4, $D_{Product}(self) = D_{Top}(top(self)) \wedge D_{Down}(down(self))$.*

If $self$ is a $op_B(v_1, ..., v_n, u_1, ..., u_m)$ term then: $D_{Product}(op_B(v_1, ..., v_n, u_1, ..., u_m)) = precond_{op_B}(v_1, ..., v_n, u_1, ..., u_m) = precond_{op_{B_t}}(v1, ..., v_n) \wedge precond_{op_{B_d}}(u_1, ..., u_m) = D_{Top}(op_{B_t}(v1, ..., v_n)) \wedge D_{Down}(op_{B_d}(u_1, ..., u_m))$. For the case of an op_R with synchronization and communication: $D_{Product}(op_R(self, v_1, ..., v_n, u_2, ..., u_m)) = precond_{op_R}(self, v_1, ..., v_n, u_2, ..., u_m) \wedge D_{Product}(self) = precond_{op_{R_t}}(top(self), v_1, ..., v_n) \wedge precond_{op_{R_d}}(down(self), emit(top(self)), u_2, ..., u_m) \wedge D_{Top}(top(self)) \wedge D_{Down}(down(self)) = D_{Top}(op_{R_t}(top(self), v_1, ..., v_n)) \wedge D_{Down}(op_{R_d}(down(self), emit(top(self)), u_2, ..., u_m))$. The case of an asynchronous call is similar but simpler.

The three previous results provide a simpler way to generate the auxiliary operations and it has also some interest for automated proofs.

Theorem 5.7 *The synchronous product of two deterministic GATs is a deterministic GAT.*

This result simply comes from the fact that a state of the product is a product of the component states and a guard of the product is the conjunction of two component guards. If we consider two transitions with the same label then each of them is the aggregation of two component transitions which are exclusive. The reverse property is not true.

The compactness for a concurrent component does not follow from the compactness of its parts. One sufficient condition to ensure compactness is the notion of *transition compactness*. A STS is transition compact if and only if every transition may be triggered at least once.

Lemma 5.8 *Let's consider two transition compact GATs with no receipt in guards then their product is a transition compact GAT.*

In the general case we have $G(self, v_1, ..., v_n, u_2, ..., u_m) = G_t(top(self), v_1, ..., v_n) \wedge G_d(down(self), emit(top(self)), u_2, ..., u_m)$. If there is no receipt in guards then if G_t and G_d can be triggered it is also true for G. This is an important case which often arises in practice, for instance with our vending machine. The reverse way of this lemma is false.

6 Related Works

We give some links to works related to mixed formal specifications. Our approach is mainly related to LOTOS, LTL and AD-DT. Complementary informations may be found in [ABR99].

An important concern, in our approach, is to provide help to the specifier to write the algebraic specification.

We defined guidelines and tools to the specifiers. Our approach includes several automatic steps in the specification process, which makes it usable by a non expert specifier. The user gets several properties (like consistency and completeness) without many difficulties. Our approach also improves the consistency/compatibility between the different descriptions. These concerns are neither addressed in LOTOS nor in AD-DT.

As in LOTOS, we focus on system with dynamic and functional parts. The main difference is about the semantics which is uniform with partial abstract data types. However, standard semantics of LOTOS does not take into account full data types but only ground expressions. Standard LOTOS semantics is restricted to finite state machines. Symbolic semantics for Full LOTOS is a way to overcome this limitation [KT97]. Dynamic properties are proved on the STS with the help of an oracle on data types. Proofs in our context use only one formalism and one environment. A successful approach is the FULL modal logic for LOTOS described in [MC01]. It is based on STS and it defines a bisimulation. It is related to our logic but there are several differences. We have no restriction on data types and recursion but FULL has. Our temporal logic may be reduced to first-order one this is not yet the case for FULL. We think that a complete approach of the semantics of LOTOS is possible with GATs.

Our work on specification is related to LTL and dynamic data type. Labelled Transition Logic (LTL) [RL97] is a formalism for specifying concurrent reactive systems. We agree with the authors on the real application of formal methods. We also aim at providing guidelines, friendly presentation and tools for non-academic people. We use STS and partial algebras, whereas they use labelled transition systems and first-order structures. Both concurrent semantics are based on an extension of the synchronous product of automata. A LTL transition is a conditional axiom linking two constructor terms denoting states. Thus the source state and the target state of a transition in LTL are terms. A GAT state has generally not a simple term representation. An important difference, from a methodological point of view, is that we use a graphic dynamic description, *a priori* to help the computation of the algebraic specification. We also get important properties about our STSs and algebraic specifications. In the LTL approach the graphic description is built *a posteriori* from the algebraic specification.

Both our approach and abstract dynamic data types (AD-DT) [CR97]) use partial abstract data type. In AD-DT the specification is twofold: algebraic axioms with a ternary predicate for the transition system. Our approach is simpler because we have only one layer in the algebraic specification. In [CR97], there is a general algebraic approach of temporal logic. This is a more ambitious and powerful approach, this is also has the drawback of a non complete logic. But a restricted approach to positive conditional dynamic formulas is proposed and the deduction system is proved to be sound and complete. Our approach is based on first-order logic and temporal algebraic operators. We have shown that a subset of CTL [Eme90] reduced to our framework. It seems that the two restricted approaches have the same expressiveness.

7 Future Work and Conclusion

We provide an algebraic approach for mixed system with an homogeneous semantics based on partial algebra. This approach may be partly automated and tools have been experimented [CPR01]. It handles any kinds of data type, either finite or unbound. It provides abstraction and readability of state-transition systems and a kind of separation of concerns.

We proved that our method ensure more easily consistency and completeness of the generated specification. The main difficulty is to prove the rewriting system terminating and in this case we proposed some successful ways. Our specifications define automatically operation preconditions which is a well-known concept in the areas of programming and specification. We proved several natural properties linking the auxiliary operations of the component product to the related operations of components. This additionally provides a simpler way to generate these operations and advantages for automated proofs.

We added an original approach to writing and proving temporal logic properties. It is uniform because data and temporal formulas are first-order formulas. It is related to classic temporal logic like CTL*, and it allows also past operator. We have done some experimentations with deadlocks and strategy to automate some proofs already exist.

Several theoretical questions remain. One area of interest is to study bisimulation over STS and means to prove them in our context. We think that our approach is right to provide a full semantics for LOTOS, this is one of our future goal. For a more practical point of view we plan to use PVS rather than Larch Prover. The main reasons are: it supports model checking and a higher-order logic.

References

[ABR95] Pascal André, Franck Barbier, and Jean-Claude Royer. Introducing Formalism in Object-Oriented Analysis and Design: an Experimentation. *Computer Science and Technique*, 14(8):973–1005, 1995. in French, ISSN 0752-4072.

[ABR99] Egidio Astesiano, Manfred Broy, and Gianna Reggio. *Algebraic Specification of Concurrent Systems*, pages 467–520. IFIP State-of-the-Art Reports. Springer Verlag, 1999. ISBN 3-540-63772-9.

[AC95] Egidio Astesiano and Maura Cerioli. Free objects and equational deduction for partial conditional specifications. *Theoretical Computer Science*, 152(1):91–138, 11 December 1995.

[AG00] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, April 2000.

[APGR00] Arnold, Point, Griffault, and Rauzy. The altarica formalism for describing concurrent systems. *FUNDINF: Fundamenta Informatica*, 34:109–124, 2000.

[AR98] Pascal André and Jean-Claude Royer. The Invoicing System: Using GAT. In Michel Allemand, Christian Attiogbé, and Henri Habrias, editors, *Comparing Systems Specification Techniques: "What questions are prompted by ones particular method of specification?"*, ISBN 2-906082-29-5, pages 381–395, Nantes, France, 1998.

[AR99] Pascal André and Jean-Claude Royer. Building Executable Data Types from Dynamic Descriptions. Rapport de recherche, IRIN, 1999.

[AR00] Pascal André and Jean-Claude Royer. An Algebraic Approach to the Specification of Heterogeneous Software Systems. Rapport de recherche, IRIN, 2000. http://www.sciences.univ-nantes.fr/irin/Vie/RR/RR-IRIN-007.ps, presented at WADT'99.

[Arn94] André Arnold. *Finite Transition Systems*. International Series in Computer Science. Prentice-Hall, 1994. ISBN 0-13-092990-5.

[Bid82] Michel Bidoit. Types abstraits algébriques : spécifications structurées et présentations gracieuses. *Colloque de l'AFCET : les mathématiques de l'informatique Paris*, pages 347–357, 1982.

[BWP84] Manfred Broy, Martin Wirsing, and Claude Pair. A Systematic Study of Models of Abstract Data Types. *Theoretical Computer Science*, 33:139–174, 1984.

[CMR99] M. Cerioli, T. Mossakowski, and H. Reichel. From Total Equational to Partial Conditional. In H.J. Kreowski, B. Krieg-Brueckner, and E. Astesiano, editors, *Algebraic Foundation of Information Systems Specification*, chapter 3, pages 31–104. Springer Verlag, 1999.

[CPR99] Christine Choppy, Pascal Poizat, and Jean-Claude Royer. From Informal Requirements to COOP: a Concurrent Automata Approach. In J.M. Wing and J. Woodcock and J. Davies, editor, *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems*, volume 1709 of *Lecture Notes in Computer Science*, pages 939–962. Springer-Verlag, 1999.

[CPR01] Christine Choppy, Pascal Poizat, and Jean-Claude Royer. The Korrigan Environment. *Journal of Universal Computer Science*, 7(1):19–36, 2001. Special issue: Tools for System Design and Verification, ISSN: 0948-6968.

[CR97] Gerardo Costa and Gianna Reggio. Specification of abstract dynamic-data types: A temporal logic approach. *Theoretical Computer Science*, 173(2):513–554, 1997.

[Der95] Nachum Dershowitz. Hierarchical Termination. In *Fourth International Workshop on Conditional (and Typed) Rewriting Systems*, volume 968 of *LNCS*, pages 89–105, Amsterdam, 1995. Springer Verlag.

[Eme90] E. Allen Emerson. *Temporal and Model Logic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 997–1072. Elsevier, 1990. J. Van Leeuwen, Editor.

[FJ95] Maribel Fernandez and Jean-Pierre Jouannaud. Modular Termination of Term Rewriting System Revisited. In *Recent Trends in Data Type Specifications*, volume 906 of *Lecture Notes In Computer Science*, pages 252–272. Springer-Verlag, 1995.

[GG88] S. J. Garland and J. V. Guttag. Inductive methods for reasoning about abstract data types. In ACM, editor, *POPL '88. 15^th Proceedings of the conference on Principles of programming languages, January 13–15, 1988, San Diego, CA*, pages 219–228, New York, NY, USA, 1988. ACM Press.

[GH93] John V. Guttag and James J. Horning, editors. *Larch: Languages and Tools for Formal Specification*. Texts and Monographs in Computer Science. Springer Verlag, 1993. With Stephen J. Garland, Kevin D. Jones, Andrés Modet, and Jeannette M. Wing.

[Gra95] Bernhard Gramlich. On Termination and Confluence of Rewriting Systems. In N. Dershowitz and N. Lindenstrauss, editors, *Proc. 4th Int Workshop on Conditional and Typed Rewriting Systems*, volume 968 of *Lecture Notes In Computer Science*, pages 166–185. Springer-Verlag, 1995.

[Har87] David Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[HL95] M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[HO80] Gérard Huet and Derek C. Oppen. *Equations and Rewrite Rules: a Survey*, pages 349 – 405. Formal Language Theory: perspectives and open problems. Academic Press, R. Book, 1980.

[Klo92] J.W. Klop. *Term Rewriting Systems*, chapter 1, pages 1–117. Handbook of Logic in Computer Science. Oxford University Press, Oxford, 1992.

[Kou85] Emmanuel Kounalis. Completeness in data type specifications. In *Proceedings of Eurocal Conference*, volume 204 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1985.

[KT97] Carron Kirkwood and Muffy Thomas. Towards a Symbolic Modal Logic for LOTOS. In *Northern Formal Methods Workshop NFM'96*, eWIC, 1997.

[MC01] C. Shankland M. Calder, S. Maharaj. An Adequate Logic for Full LOTOS. In *Proceedings of the FME'2001 Conference*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[PCR99] Pascal Poizat, Christine Choppy, and Jean-Claude Royer. Concurrency and Data Types: a Specification Method. An Example with LOTOS. In J. Fiadero, editor, *Recent Trends in Algebraic Development Techniques, Selected Papers of the 13th Workshop on Algebraic Development Techniques, WADT'98*, volume 1589 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 1999.

[PRR02] Liang Peng, Annya Romanczuk, and Jean-Claude Royer. A Translation of UML Components into Formal Specifications. In *TOOLS East Europe 2002*, Theo D'hondt Ed., pages 60–75, Kluwer Academic Publishers, 2003.

[Rao95] M. R. K. Krishna Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.

[RL97] Gianna Reggio and Mauro Larosa. A graphic notation for formal specifications of dynamic systems. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, volume 1313 of *Lecture Notes in Computer Science*, pages 40–61, Graz, Austria, September 1997. Springer-Verlag. ISBN 3-540-63533-5.

[Roy01a] Jean-Claude Royer. Formal Specification and Temporal Proof Techniques for Mixed Systems. In *Proceedings of the 15th IPDPS 2001 Symposium, FMPPTA*, San Francisco, USA, 2001. IEEE Computer Society.

[Roy01b] Jean-Claude Royer. The Vending Machine: Mixed Formal Specifications and Proofs. Rapport de recherche IRIN-01.1, IRIN, 2001. http://www.sciences.univ-nantes.fr/irin.

[Roy02] Jean-Claude Royer. Temporal Logic Verifications for UML: the Vending Machine Example. In *Proceedings of the fourth Rigorous Object-Oriented Methods Workshop*, 2002.

[STB97] Carron Shankland, Muffy Thomas, and Ed Brinksma. Symbolic Bisimulation for Full LOTOS. In *Algebraic Methodology and Software Technology AMAST'97*, volume 1349 of *Lecture Notes in Computer Science*, pages 479–493. Springer-Verlag, 1997.

[Tur93] Kenneth J. Turner, editor. *Using Formal Description Techniques, An introduction to Estelle, Lotos and SDL*. Wiley, 1993. ISBN 0-471-93455-0.

[Wir90] Martin Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, chapter 13, pages 675–788. Elsevier, 1990. J. Van Leeuwen, Editor.

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://ai.ijs.si/informatica/
http://orca.st.usm.edu/informatica/

Informatica

An International Journal of Computing and Informatics

Archive of abstracts may be accessed at USA: http://, Europe: http://ai.ijs.si/informatica, Asia: http://www.comp.nus.edu.sg/ liuh/Informatica/index.html.

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 2003 (Volume 27) is
– USD 80 for institutions,
– USD 40 for individuals, and
– USD 20 for students
Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

LaTeX Tech. Support: Borut Žnidar, Kranj, Slovenia.
Lectorship: Fergus F. Smith, AMIDAS d.o.o., Cankarjevo nabrežje 11, Ljubljana, Slovenia.
Printed by Biro M, d.o.o., Žibertova 1, 1000 Ljubljana, Slovenia.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. R. Murn, 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 Education, Science and Sport, Trg OF 13, 1000 Ljubljana, Slovenia.

Volume 27 Number 1 April 2003 **ISSN 0350-5596**

Informatica

An International Journal of Computing and Informatics

Introduction		**1**
A Decentralized Approach to the Integration of Life Science Web Databases	Z.B. Miled, N. Li, M. Baumgartner, Y. Liu	**3**
doMosaic - Analysis of the mosaic-like domain arrangements in proteins	D.T. Gerrard, E. Bornberg-Bauer	**15**
Mining and Validating Gene Expression Patterns: an Integrated Approach and Applications	S.-M. Tseng, C.-P. Kao	**21**
Fault detection and isolation using hybrid parameter estimation and fuzzy logic residual evaluation	B. Athamena, H.A. Abbassi	**29**
Practical Construction for Multicast Re-keying Schemes using R-S Code and A-G Code	C.-Y. Bai, R. Houston, G.-L. Feng	**39**
Building and managing software reuse libraries	Z. Houhamdi	**49**
Deriving self-stabilizing protocols for services specified in LOTOS	M. Kapus-Kolar	**57**
Embedding Complete Binary Trees into Faulty Flexible Hypercubes with Unbounded Expansion	J.-C. Lin S.K.C. Lo	**75**
Supporting the development of time-triggered co-operatively scheduled (TTCS) embedded software using design patterns	M.J. Pont	**81**
The GAT Approach to Specifying Mixed Systems	J.-C. Royer	**89**
An Algorithm for Computing the Optimal Cycle Time of a Printed Circuit Board Assembly Line	D.M. Kodek M. Krisper	**105**