CONTENTS

KNOWLEDGE IN

SOFTWARE ENGINEERING

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 181–184

A MULTIAGENT DECISION SUPPORT SYSTEM FOR ASSISTING SOFTWARE MAINTENANCE AND EVOLUTION

GABRIELA CZIBULA(1), ISTVAN GERGELY CZIBULA(1), ADRIANA MIHAELA GURAN(1) , AND GRIGORETA SOFIA COJOCAR(1)

Abstract. The development of tools for building performant, open, scalable and continuously adaptable software systems is a major challenge in software engineering and artificial intelligence researches. The problems related to the maintenance and the evolution of software systems are essential, a major requirement being to develop methodologies for the structural and behavioral optimization of the systems. In this paper we propose an intelligent multiagent decision support system for assisting software developers during the maintenance and evolution of software systems. The proposed system is part of a research project that aims at developing machine learning techniques for the structural and behavioral adaptation of software systems during their maintenance and evolution. The current status of our work is also presented.

1. Introduction

The concept of *decision support system* (DSS) is very broad, because there are many approaches to decision-making, and because of the wide range of domains in which decisions are made. A DSS can take many different forms. In general, we can say that a DSS is a computerized system that facilitates decision making.

The main objective of our research is to use machine learning techniques [1] for structural adaptation of software systems during their maintenance and evolution and for their behavioral self-adaptation, as well. The developed techniques will be incorporated in an intelligent multiagent decision support system for assisting software developers in the maintenance and evolution of software systems.

The aim of this paper is to propose an intelligent multiagent decision support system (*DSSEM - Decision Support System for Software Evolution and Maintenance*) for assisting software developers during the maintenance and evolution of software systems. For the structural adaptation of software systems and for their behavioral self-adaptation we propose to use machine learning [1] techniques.

The rest of the paper is structured as follows. Section 2 presents the motivation of our approach. The architecture of *DSSEM* system is presented in Section 3. Some conclusions and future research directions are given in Section 4.

2000 *Mathematics Subject Classification.* 68N99, 68T99, 68T05.
Key words and phrases. software engineering, decision support system, refactoring, machine learning.

2. Our approach

In the lifecycle of software systems changes appears continuously, because of new functional requirements. These new requirements impose the necessity of adaptive optimization of the systems during their evolution. In order to meet these requirements, both structural and behavioral changes of the systems are needed. The structural changes refer to the improvement of the internal structure of the system in order to adapt itself to the new requirements, and the behavioral changes refer to optimizing the behavior of the system in order to adapt its interaction to the dynamicity of its external environment (users, other systems).

The main objective of our reasearch is the development of machine learning methods and models for structural adaptation and behavioral self-adaptation of software systems during their maintenance and evolution. We focus our researches on two essential problems related to the maintenance and evolution of software systems, *program comprehension* and *software reengineering*. In order to validate the obtained theoretical results, we aim at developing an intelligent multiagent decision support system for assisting software developers in the maintenance and evolution of software systems.

Two subfields of software engineering which deal with activities related to software systems understanding and their structural changes are *program comprehension* and *software reengineering*. Both subfields study activities from the maintenance and evolution of software systems phases. We briefly analyze, in the following, the main aspects related to the activities from the previously mentioned domains which are the focus of our researches.

Program comprehension [2] is the domain that deals with the processes (cognitive or other kind) used by the software developers in order to understand software systems. It is a vital software engineering and maintenance activity, necessary for facilitating reuse, inspection, maintenance, reverse engineering, reengineering, migration, and evolution of existing software systems. Esentially, it focuses on developing metholologies and tools to facilitate the understanding and the management of the increasing number of legacy systems.

Some important activities in program comprehension that we intend to automate are: *design patterns identification*, *concept location* and *aspect mining*.

Software reengineering was defined by Chikofsky and Cross in [3] as the inspection and modification of a software system in order to rebuild it in a new form. Recent researches reveal the importance of software reengineering in the maintenance and evolution of software systems [4]. Software reengineering consists of modifying a software system after it has been reversed engineered to understand it, to add new functionalities or to correct existing errors.

3. DSSEM system

In this section we propose a distributed decision support system for assisting developers in the maintenance and evolution of software systems. As the system is distributed and it requires flexible and autonomous decisions, it has been identified as a multiagent one. The following classes of agents were distinguished at the level of the multiagent system: *Software developers* (human agents); *Personal Assistant Agents* (PA); *Decision Support Agents* (intelligent mobile agents - DSA).

The system that we propose in this paper is a distributed system consisting of some local nodes, where software developers are assisted by *DSSEM* in activities related to software maintenance and evolution (as presented in Section 2), and a central location that supervises the local nodes.

We assume that at each local node, the *human agent* (software developer) use his/her favourite IDE (Integrated Development Environment) for developing a software application $\mathcal{SA}$. At each local node, a *personal assistant agent PA* is also available. Each time a *software developer* needs assistance in a task related to refactoring, design pattern identification, or other tasks supported by *DSSEM* (see Section 2) *PA* agent is activated. *PA* is a software agent that is responsible for interacting with the IDE in which the user develops its application.

On the central location six kinds of *decision support agents* exist: an agent responsible for design patterns identification, one responsible for concept location, one responsible for crosscutting concerns identification, one responsible for introducing design patterns, one responsible for refactoring, and one responsible for the behavioral adaptation of the system. All of them are sleeping until they receive a specific message and they become activated. After a software developer authenticates on a local node and request assistance for an activity, its *personal assistant agent PA* sends an awakening message to the coresponding *decision support agent.*

Once activated, an instance of the corresponding *decision support agent DSA* migrates to the local node, where its interaction with the *personal assistant agent PA* starts (*DSA* agent receives the information that *PA* agent has collected about the software application $\mathcal{SA}$). After the task of *DSA* agent is completed, it sends the results to *PA* agent that will communicate the information to the software developer. After this, *DSA* migrates back to the central location. *DSA* are *intelligent agents* that accomplish their tasks by using machine learning [1] techniques.

We can conclude that the advantages of the *DDSEM* system proposed in this paper are: (1) It benefits from the advantages that the agent based technology offers - especially *autonomy* and *flexibility* [5]. (2) It offers protection of confidential information by using the mobile agent technology. Maintaining security [6] implies that the software developer's intelectual property (source code) might be highly sensitive data that should not leave its location. In such cases, the mobile-agent model [7] offers a solution: a mobile agent is sent to the local node, and at the end of the process the mobile agent is destroyed on the location itself. (3) *DSSEM* system can be simply adapted to any IDE and/or programming languages (such as Eclipse, Visual Studio; Java, C++), only by changing the behavior of *PA*.

We have implemented a prototype of *DSSEM* system, in which the functionality of *refactorings identification* is added. In this direction, *RDSA* agent was developed, i.e., an instance of *DSA* agent that is responsible for identifying refactorings that improve the structure of a software system. For refactorings identification we use a clustering approach that we have previously introduced in [8]. The identified refactorings are suggested to the software developer, that can decide if the refactorings are appropriate.

4. Conclusions and furher work

We have proposed in this paper an intelligent multiagent decision support system (*DSSEM*) that can be used for assisting software developers in some activities related to the maintenance and evolution of software systems. We have presented the architecture of the proposed system, a scenario of using it, and we have also emphasized the system's importance.

Further work may be done in the following directions: to identify solutions for improving the performance of *DSSEM* system using parallel computation; to extend *DSSEM* system by including all the functionalities related to the structural adaptation and behavioral self-adaptation of software systems; to experimentally validate *DSSEM* multiagent system on real software systems.

ACKNOWLEDGEMENT. This work was supported by the research project ID_2268/2009, sponsored by the Romanian National University Research Council.

References

[1] Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
[2] von Mayrhauser, A., Vans, A.M.: Program understanding: Models and experiments. Advances in Computers **40** (1995) 1–38
[3] Cross, J., Chikofsky, E., May, C.: Reverse engineering. Advances in Computers **35** (1992) 199–254
[4] Tilley, S., Smith, D.: Legacy system reengineering. In: Int. Conference on Software Maintenance. (1996) 9–12
[5] Weiss, G.E.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (1999)
[6] Borselius, N.: Mobile agent security. Electronics and Communication Engineering Journal **14** (2002) 211–218
[7] Nwana, H.S.: Software agents: An overview. Knowledge Engineering Review **11** (1995) 205–244
[8] Czibula, I., Serban, G.: Improving systems design using a clustering approach. IJCSNS International Journal of Computer Science and Network Security **6** (2006) 40–49

(1) Department of Computer Science, Babeş-Bolyai University, 1 M. Kogalniceanu Street, Cluj-Napoca, Romania

E-mail address: {gabis, istvanc, adriana, grigo}@cs.ubbcluj.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 185–188

A CLUSTERING APPROACH FOR TRANSFORMING PROCEDURAL INTO OBJECT-ORIENTED SOFTWARE SYSTEMS

ISTVAN GERGELY CZIBULA[1]

Abstract. It is well known that object-oriented programming has mechanisms for constructing highly flexible, adaptable, and extensible systems. Legacy software systems evolution often requires their reengineering to object oriented systems. In this paper we are focusing on the problem of automating the transformation of procedural software systems into object-oriented systems. For this purpose we propose a clustering approach and we develop a partitional clutering algorithm that can be used for assisting software developers in the process of migrating procedural code into object-oriented code. A simple example showing how our approach works is also considered.

1. Introduction

It is well known that object-oriented programming has many advantages over conventional procedural programming languages for constructing highly flexible, adaptable, and extensible systems [1].

Object-oriented concepts are useful concerning the reuse of existing software. Therefore a transformation of procedural programs to object-oriented programs becomes an important process to enhance the reuse of procedural programs. It would also be useful to assist by automatic methods the software developers in transforming procedural code into an equivalent object-oriented one.

Unsupervised classification, or clustering, as it is more often referred as, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects [2].

The main contributions of this paper are:

- To introduce a clustering approach for transforming non object-oriented software systems into object-oriented ones. The proposed approach can be useful during the evolution of non object-oriented software systems.
- To propose a *k-medoids* based clustering algorithm for our goal.

2000 *Mathematics Subject Classification.* 68N30, 62H30.
Key words and phrases. software engineering, object-oriented systems, clustering.

The rest of the paper is structured as follows. A clustering approach for assisting developers in the process of tranforming software systems written in procedural programming languages into object-oriented systems is proposed in Section 2. Section 3 contains some conclusions of the paper and also outlines further research directions.

2. Our approach

Let $S = \{s_1, s_2, ..., s_n\}$ be a non object-oriented software system, where $s_i, 1 \leq i \leq n$ can be a subprogram (function or procedure), a global variable, a user defined type.

In the following we will refer an element $s \in S$ as an *entity*.

In order to transform S into an object-oriented system, we propose an approach consisting of two steps:

- **Data collection** - The existent software system is analyzed in order to extract from it the relevant entities: subprograms, local and global variables, subprograms parameters, subprograms invocations, data types and modules, source files or other structures used for organizing the procedural code.
- **Grouping** - The set of entities extracted at the previous step are grouped in clusters. The goal of this step is to obtain clusters corresponding to the application classes of the software system S.

In the **Grouping** step we propose a *k-medoids* based clustering algorithm, *kOOS* (k-medoids for Transforming Software Sytems into Object-Oriented Software Systems).

In our clustering approach, the objects to be clustered are the entities from the software system S, i.e., $\mathcal{O} = \{s_1, s_2, \ldots, s_n\}$. Our focus is to group similar entities from S in order to obtain groups (clusters) that will represent classes in the equivalent object-oriented version of the software system S.

In order to express the dissimilarity degree between the entities from the software system S, we will use an adapted generic cohesion measure [3]. Consequently, the distance $d(s_i, s_j)$ between two entities s_i and s_j is expressed as in Equation (1).

$$(1) \qquad d(s_i, s_j) = \begin{cases} 1 - \frac{|p(s_i) \cap p(s_j)|}{|p(s_i) \cup p(s_j)|} & if \;\; p(s_i) \cap p(s_j) \neq \emptyset \\ \infty & otherwise \end{cases},$$

where, for a given entity $e \in S$, $p(e)$ defines a set of relevant properties of e, expressed as follows.

- If e is a subprogram (procedure or function) then $p(e)$ consists of: the subprogram itself, the source file or module where e is defined, the parameters types of e, the return type of e if it is a function and all subprograms that invoke e.
- If e is global variable then $p(e)$ consists of: the variable itself, the source files or modules where the variable is defined, all subprograms that use e.
- If e is a user defined type then $p(e)$ consists of: the type itself, all subprograms that use e, all subprograms that have a parameter of type e and all functions that have e as returned type.

We have chosen the distance between two entities as expressed in Equation (1) because it emphasizes the idea of cohesion. As illustrated in [4], "*Cohesion refers to the degree to which module components belong together*". Our distance, as defined in Equation (1), highlights the concept of cohesion. It is very likely that entities with low distances have to be placed in the same application class, and distant entities have to belong to different aplication classes.

kOOS algorithm is a *k-medois* like algorithm that uses a heuristic for choosing the initial medoids.

The entity chosen as the first medoid is the most "distant" entity from the set of all entities (the entity that maximizes the sum of distances from all other entities).

At each step we select from the remaining entities the most distant entity relative to the already chosen medoids. If the selected entity is close enough to the already chosen medoids, then the process is stoped, otherwise the selected entity is added to the medoids list and we try to choose a new medoid from the remaining entities.

We consider an entity close to a set of medoids if the distances between the entity and any of the medoids are less than a given threshold. We have chosen the value 1 for the threshold because distances greater than 1 are obtained only for unrelated entities (Equation (1)).

The idea of the above described heuristic is to choose medoids that will act as seeds for the clusters that will represent application classes in the resulted object-oriented system.

After selecting the initial medoids, *kOOS* behaves like the classical *k-medoids* algorithm.

The main idea of the *kOOS* algorithm that we apply in order to group entities from a software system is the following:

(i) The initial number p of clusters and the initial medoids are determined by the heuristic described above.

(ii) The clusters are recalculated, i.e., each object is assigned to the closest medoid. Empty clusters will be eliminated from the partition.

(iii) Recalculate the medoid i of each cluster k based on the following idea: if h is an object from k such that $\sum\limits_{j \in k}(d(j,h) - d(j,i))$ is negative, then h becomes the new medoid of cluster k.

(iv) Steps (ii)-(iii) are repeatedly performed until there is no change in the partition $\mathcal{K}$.

Each cluster from the resulted partition will represent an application class from the equivalent object-oriented version of the software system S.

We have successfully evaluated our approach on a simple code example written in Borland Pascal.

3. Conclusions and further work

We have presented in this paper a partitional clustering algorithm (*kOOS*) that can be used for assisting software developers in transforming procedural software systems into object-oriented ones. We have also illustrated how our approach is applied for transforming a simple program written in Pascal into an equivalent object-oriented system. Advantages of our approach in comparison with existing similar approaches are also emphasized.

Further work will be done in the following directions:

- To improve the *distance* function used in the clustering process.
- To extend our approach in order to also determine relationships (class hierarchies) between the application classes obtained in the object-oriented system.
- To apply *kOOS* algorithm on large software systems.

ACKNOWLEDGEMENT. This work was supported by the research project TD No.411/2008, sponsored by the Romanian National University Research Council (CNCSIS).

References

[1] Newcomb, P., Kotik, G.: Reengineering procedural into object-oriented systems. In: WCRE '95: Proceedings of the Second Working Conference on Reverse Engineering, Washington, DC, USA, IEEE Computer Society (1995) 237

[2] Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)

[3] Simon, F., Loffler, S., Lewerentz, C.: Distance based cohesion measuring. In: Proceedings of the 2nd European Software Measurement Conference (FESMA), Technologisch Instituut Amsterdam (1999)

[4] Bieman, J.M., Kang, B.K.: Measuring design-level cohesion. Software Engineering **24** (1998) 111–124

(1) Department of Computer Science, Babeş-Bolyai University, 1 M. Kogalniceanu Street, Cluj-Napoca, Romania

E-mail address: `istvanc@cs.ubbcluj.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 189–193

COMDEVALCO FRAMEWORK - PROCEDURAL AND MODULAR ISSUES

B. PÂRV, I. LAZĂR, S. MOTOGNA, I-G. CZIBULA, C-L. LAZĂR[1]

Abstract. This work is part of a series referring ComDeValCo - a framework for Software Component Definition, Validation, and Composition. Its constituents are: a modeling language, a component repository and a set of tools. The current paper describes the status of the project after implementing procedural and modular concepts.

1. Introduction

UML 2.0 introduced the action semantics, aimed to help the construction of *executable models* [3]. The behavior of such models is completely defined down to statement level, making possible their construction and execution by tools able to interpret UML 2.0 semantics.

Unfortunately, this important breakthrough did not increased the use of UML models in the software development process. The main reasons are at least the following: (1) UML is too general (being *unified*) and its semantics contains extension points deferred to the concrete use of the language; (2) the construction of detailed models (down to statement level) with existing tools is too dificult, these tools being too general and (3) there is a lack of (agile) mature development processes based on executable models.

The above premises were the starting point of the ComDeValCo project: a framework for component definition, validation, and composition. Its constituents are: a modeling language, a component repository, and a set of tools.

The software component model is described by an platform-independent *modeling language*. *Component repository* represents the persistent part of the framework, containing the models of all full validated components. The *toolset* is intended to automate many tasks and to assist developers in performing component definition (DEFCOMP) and V & V (VALCOMP, SIMCOMP), maintenance of component repository (REPCOMP), and component assembly (DEFSYS, VALSYS, SYMSYS, GENEXE).

2000 *Mathematics Subject Classification.* 68N30.
Key words and phrases. Software component, Modeling languages and tools, Component repository, Component definition, Component validation, Component assembly.

The original idea of ComDeValCo framework is to use a plaform-independent modeling language able to model software components in a precise way and to employ a set of tools aimed to help developers in component and system definition and validation processes. This paper represents a synthesis of early efforts regarding this framework, most of them being described in separate papers.

The paper is organized as follows: after this introductory section, the next one contains the results belonging to the procedural programming, continued in the third section with modular issues. Last section contains some conclusions and discusses future plans.

2. ComDeValCo: Procedural issues

First phase of the ComDeValCo project had two goals: completing the initial object model and developing first versions of DEFCOMP and VALCOMP tools.

2.1. Initial object model.

The initial object model was started during a feasibility study performed in the first term of 2007. It has a layered architecture, from bottom to top: (1) low-level (syntactical) constructions, (2) execution control statements and (3) program units. Paper [13] contains a detailed discussion.

With this initial object model in mind, an action language PAL (Procedural Action Language) was designed (see [4] for details). It has a concrete (textual) syntax and graphical notations corresponding to statements and component objects.

2.2. Tools for procedural paradigm.

The first version of DEFCOMP allows model construction, execution, and testing, covering also some of the functionality of VALCOMP.

Program units can be expressed in both graphical or textual ways. The two different editing perspectives of DEFCOMP are synchronized, acting on the same model, which uses PAL meta-model.

VALCOMP tool was designed with the agile test-driven development process in mind, allowing developers to build, execute, and test applications in an incremental way, in short development cycles.

DEFCOMP has a debugging perspective also, and the developer can adnotate the model with breakpoints. Besides assertions, used for testing and functional code, PAL includes pre- and post-conditions for procedures/functions and invariants for cycles.

Later on, DEFCOMP and VALCOMP were included in the so-called ComDeValCo *workbench*, and are described in more detail in [1, 2, 7].

3. ComDeValCo: modular issues

The second phase of the ComDeValCo project had had two main goals: implementation of modular concepts and the design of component repository.

3.1. **Modeling language.** The modeling language was improved in two different directions, by (1) including module and execution environment, and (2) extending its type system with structured types.

In order to ease the process of implementing modular concepts, an adaptable infrastructure was created, based on a meta-model defining the concepts of *module* and *execution environment*.

The module is considered in the general case, as a deployment unit, including either (a) several data types, functions, and procedures - as in the traditional modular programming model, or (b) several data types, including components - as in the case of component-based programming. There are dependency relations between modules, which must be specified during module definition phase and must be satisfied before module execution.

Static execution environments load all modules of an application before starting its execution. The proposed model loads modules and starts their execution provided that all dependencies are solved. It also supports dynamic module load/unload facilities. Some of the existing execution environments have these features - like OSGi [11], which assembles Java applications from dynamic modules. Our proposal is described in [5].

The initial modeling language used only primitive data types. Currently, its type system includes a new `ArrayType` class, and PAL grammar was changed to allow the definition of tables (vectors) and structured types, like lists. These achievements are described in detail in [9], which proposes an extensible data type hierarchy.

3.2. **Component repository.** The interactions between component repositories, ComDeValCo workbench and client applications are described in [5, 8].

The starting point in the work of providing a correct taxonomy for components was the establishment of classification criteria. Several concrete approaches were considered, including information domain and functionality. These criteria may be used in searching of components stored in the repositories. All these matters are discussed in great detail in [8].

The root of all objects managed by the component repository is `RegistryObject` (see `Figure` 1), from which all components inherit (thru `ExtrinsicObject` class). In order to provide a great degree of flexibility, the hierarchy contains distinct classes for the two concepts: *classification* and *classification scheme*.

In order to describe the representation of components in the repository, an object model was defined; it includes all component types covered by the modeling language and allows for adding new ones.

Component representation format complies to OASIS RIM (Registry Information Model) standard [10]. To achieve this, the class `ExtrinsicObject` was extended by subclasses specific to component-based applications: `DModule`, `DComponent`, `Capability` - functionality, and `Requirement` - component dependencies .

3.3. **The toolset.** The functionality of DEFCOMP and VALCOMP tools was extended to include module and execution environment concepts. Papers [5, 6] describe

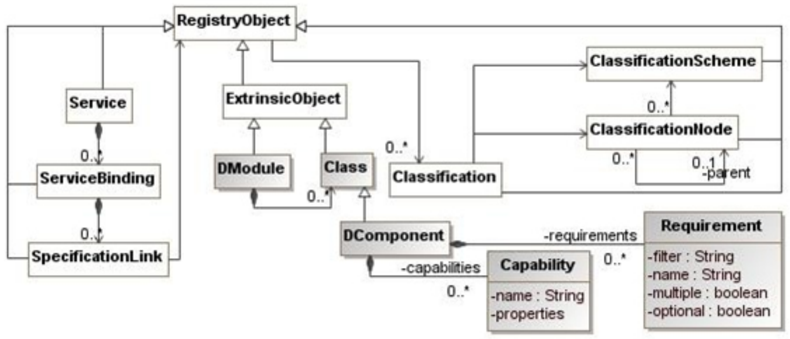


FIGURE 1. Representation of objects in repositories

in great detail UML stereotypes aimed to define these new constructs included in the modeling language.

DEFSYS and VALSYS were initially considered as tools for *developing/verifying and validating software systems* by assembling components taken from component repositories. Later on, by adopting a test-driven development method, these two sub-processes (component definition and system definition) were considered as a whole, and DEFCOMP and VALCOMP tools address all needed functionality. This way, the functionality of ComDeValCo workbench covers both component/software system development/verification and validation activities. These results are described in more detail in [2, 7].

4. Conclusions and Further Work

This paper describes procedural and modular aspects of the ComDeValCo project. Future developments of the framework will include object-oriented and component-based issues, as well as repository management and system-level tools.

These steps are considered within the planned evolution of the ComDeValCo framework, which include activities for improving the modeling language and component repository infrastructure, as well as continuous maintenance of the tools aimed to operate in the component/system definition, validation, evaluation, and simulation.

Acknowledgements

This work was supported by the grant ID_546, sponsored by NURC - Romanian National University Research Council (CNCSIS).

References

[1] Czibula, I-G., Lazăr, C-L, Prv, B., Motogna, S., Lazăr, I, ComDeValCo Tools for Procedural Paradigm, Proc. Int. Conf. on Computers, Communications and Control (ICCC 2008), Felix Spa, Romania, 15-17 May, 2008, (*Int. J. of Computers, Communications and Control* Suppl Issue), pp. 243-247.

[2] Czibula, I-G., ComDeValCo Workbench: Activity Modeling and Execution, Proc. Conf. *Cluj Academic Days, Computer Science Track*, Cluj, Romania, June 4, 2008, pp. 67-74.

[3] Mellor, S.J., Balcer, M.J, *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley, 2002.

[4] Lazăr, I., Pârv, B., Motogna, S., Czibula, I-G., Lazăr, C-L., An Agile MDA Approach for Executable UML Activities, *Studia UBB, Informatica*, LII, No. 2, 2007, pp. 101-114.

[5] Lazăr, I., Pârv, B., Motogna, S., Czibula, I-G., Lazăr, C-L., iComponent: A Platform-independent Component Model for Dynamic Execution Environment, in Proc. of 10th Int. *Symposium on Symbolic and Numeric Algorithms for Scientific Computing* SYNASC 2008, Timişoara, Romania, September 26-29, 2008. To appear in IEEE Press.

[6] Lazăr, I., Pârv, B., Motogna, S., Czibula, I-G., Lazăr, C-L., An Agile MDA Approach for the Development of Service-Oriented Component-Based Applications, in Proc. Int. Conf. *Complexity and Intelligence of the Artificial and Natural Complex Systems* CANS 2008, Târgu Mureş, Romania, November 8-10, 2008. To appear in *Int. J. of Comp., Comm. and Control.*

[7] Lazăr, C-L., Lazăr, I., On Simplifying the Construction of Executable UML Structured Activities, *Studia UBB, Informatica*, LIII, No. 2, 2008, pp. 147-160.

[8] Motogna, S., Lazăr, I., Pârv, B., Czibula, I-G., Lazăr, C-L., Component Classification Criteria for a Platform-independent Component Repository, in Proc. of 5th Int. *Conf. of Applied Mathematics* ICAM 2008, Baia Mare, Romania, September 18-20, 2008. To appear in *Carpathian J. Math.*

[9] Motogna, S., Pârv, B., Lazăr, I., Czibula, I.G., Lazăr, C.L., Extension of an OCL-based Executable UML Components Action Language, *Studia UBB, Informatica*, LIII, No. 2, 2008, pp. 15-26.

[10] OASIS Registry Information Model, http://docs.oasis-open.org/regrep/v3.0/regrep-3.0-os.zip

[11] OSGi Alliance, OSGi Service Platform Core Specification, Release 4, Version 4.1.(2007), http://www.osgi.org/.

[12] Pârv, B., Motogna, S., Lazăr, I., Czibula, I.G., Lazăr, C.L., ComDeValCo - a framework for software component definition, validation, and composition, *Studia UBB, Informatica*, LII, No. 2, 2007, pp. 59-68.

[13] Pârv, B., Motogna, S., Lazăr, ComDeValCo Framework - the modeling language for procedural paradigm, *Int. J. of Comp., Comm. and Control*, III, No. 2, 2008, pp. 183-195.

$^{(1)}$ Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1, M. Kogălniceanu, Cluj-Napoca 400084, Romania

E-mail address: `bparv,motogna,ilazar,czibula@cs.ubbcluj.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 194–197

RAPID PROTOTYPING OF CONVERSATIONAL WEB FLOWS

I. LAZĂR, S. MOTOGNA, AND B. PÂRV[1]

ABSTRACT. In this paper we present a rapid prototyping development approach for conversational web flows based on Model-Driven Architecture (MDA). Prototypes are defined as executable UML models. In order to ensure simple and fast definition of UML models, we apply the DRY ("don't repeat yourself") principle and the concept of "convention over configuration". Update transformations in the small may be applied on model elements in order to rapidly create new elements based on some conventions.

1. Introduction

Prototyping refers to a development approach centered on developing prototypes (executable model of a system that reflects a subset of its properties) early in the development process, to allow early feedback and analysis [4]. Prototyping techniques based on MDA rely on using the UML and profiles. Today, the effort of defining a standard execution semantics for UML enters the final state of adoption through Foundational UML (fUML) which defines a "basic virtual machine for the UML" [7].

The proposed solution is a prototyping approach for conversational web flows based on the following techniques. The model prototypes are captured as executable fUML models [7] and conform to iComponent profile [6]. For simple and fast definition of fUML models we use model transformations [3] for generating new elements based on existing best practices for developing web applications.

After this introductory section, the next one presents iComponent profile, the third section describes our rapid prototyping method, and the last one draws conclusions and future development plans.

2. iComponent Profile

iComponent (injected component) [6] has been designed as a platform-independent component model for service-oriented applications (SOA). Recently, we have extended iComponent for prototyping SOA on OSGi platform [5] by adding new stereotypes for domain classes and Model-View-Controller (MVC) architectures - see Figure 1. A short description of this profile is given below (see [5]).

2000 *Mathematics Subject Classification.* 68U07, 68U20, 68U35.
Key words and phrases. rapid system prototyping, conversational web flow, executable UML, contextual component, dependency injection.

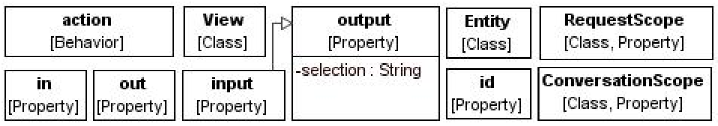


FIGURE 1. iCOMPONENT Profile Extract

Entity and *id* can be used to mark instances from domain that have identity. *View* corresponds to the view part of an MVC architecture and controllers are defined as simple components having behavioral elements marked as *action*s.

For binding data between actions and views we use the *context* abstraction [2], and the *bijection* mechanism [1] which is a generalization of the dependency injection as follows. Whenever a component's action is invoked, values from the contexts are *injected* into the properties of the component being invoked, and also "outjected" from the component's properties back to the contexts.

In this paper we add new stereotypes for modeling two contexts of web applications: the *request context* and the *conversation context* which represent instances required for processing a request, respectively a conversation. A conversation is a unit of work from the point of view of the user and can span across multiple requests. For instance, registering a new customer can be designed as a conversation composed of two separate requests, submitting the account information and then the personal details.

RequestScope and *ConversationScope* are used to associate components and their properties to these contexts. Information flow direction must be specified using *in* and *out* stereotypes.

View properties can be marked as *input* properties - for entering data, or as *output* properties - for displaying data. The *model* part of MVC is represented by all instances associated to the request and conversation contexts. By convention, a view property is either directly mapped to a context property, or indirectly using a *selection* expression written in OCL.

3. DEVELOPMENT APPROACH

Each of the following subsections describes a step of our prototyping approach on a simple case study for registering a new customer.

A. Create a new action. A creation wizard [3] can be used to rapidly create a new action. Figure 2 - (a) presents the result of this creation process for a new *index* action and a new *Home* owner component: a *Home* component type associated to the *RequestScope* is created, an *index* operation is added which is described by an empty *index* activity (not shown in Figure 2); an *Index* view is also created and a simple *out welcome* property is associated to the request context. Now we can run the model prototype, select a controller and then execute an action; a view is selected in order to render the response - by convention, a view with the same name as the action.

B. Create domain classes. Entities and value objects can be created using any UML tool. We establish their properties and then run a generation wizard for

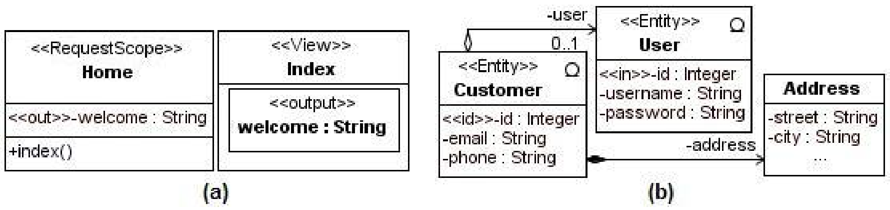


FIGURE 2. (a) Creating a New Action (b) Domain Classes

generating identity properties and basic persistence operations - Figure 2 - (b). See [5] for more details about our Object-Relational Mapping infrastructure.

C. Create a new conversation and design its flow. As for actions, we are prompted to enter the conversation name and to choose its owner component. Figures 3 and 4 present the result of creating a *NewCustomer* conversation and a new *EditCustomer* owner component: an *EditCustomer* class associated to the *ConversationScope* is created and a *NewCustomer* state machine is added in the context of this class. At this stage, the generated artifacts do not contain other elements. For saving space, Figures 3 and 4 show only the final result of the model prototype.

Now we design the states and the conversation flow. Before running the model prototype we change the content of *Index* in order to execute our conversation - see Figure 5. A view is dynamically generated for each state. At the moment, we have obtained an executable conversation, but no data is presented. Next, we are going to enhance our conversation by adding actions and establishing data bindings.

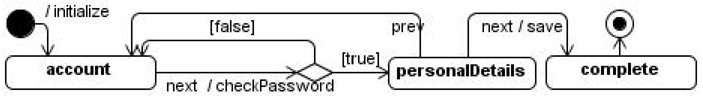


FIGURE 3. *NewCustomer* State Machine

D. Add actions to a conversation. At this step we add behaviors to the state machine in order to produce the data required by the conversation. fUML activities may be added for specifying behaviors to be performed on a transition when the transition fires, or as entry/exit behaviors executed when a state is entered/exited. All activities must belong to the component, in order to easily access the conversational state. Figures 3 and 4 show the result of this process for our case study. *initialize* is added to create a new *customer* instance, *save* is added to store the new customer, and *checkPassword* is an intermediate step.

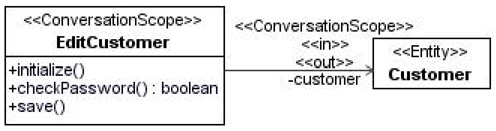


FIGURE 4. *EditCustomer* Controller

E. Generate views. Finally we can generate the views - see Figure 5. The views contain only presentation elements together with their mappings, the navigation part being defined by the state machine. When running the model prototype, actions corresponding to outgoing transitions of a state are dynamically added for each view.

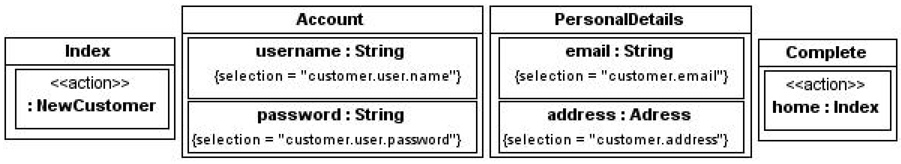


Figure 5. Generated Views

4. Conclusions and Further Work

We have extended iComponent for modeling conversational web flows. The new added stereotypes together with the introduced model transformations allow rapid generation of executable fUML models. We have also extended the runtime infrastructure of our component model in order to obtain a framework for rapid prototyping of web applications. As a future direction, we intend to refine the stereotypes for creating views, and to extend iComponent for bussiness processes.

ACKNOWLEDGEMENTS. This work was supported by the grant ID_546, sponsored by NURC - Romanian National University Research Council (CNCSIS).

References

[1] JBoss. *Seam Contextual Component Model.* JBoss Inc., 2006.
[2] JCP. *JSR-299: Contexts and Dependency Injection for Java.* JCP, 2009.
[3] D.S. Kolovos, R.F. Paige, and F.A.C. Polack. Update transformations in the small with the epsilon wizard language. *Journal of Object Technology*, 6(9):53–69, 2007.
[4] Fabrice Kordon and Luqi. An introduction to rapid system prototyping. *IEEE Transactions on Software Engineering*, 28(9):817–821, 2002.
[5] I. Lazar, S. Motogna, B. Parv, I.-G. Czibula, and C.-L. Lazar. Rapid prototyping of service-oriented applications on osgi platform. In *4th Balkan Conference in Informatics.* Submitted, 2009.
[6] S. Motogna, I. Lazar, B. Parv, and I.-G. Czibula. An Agile MDA Approach for Service Oriented Components. In *6th International Workshop on Formal Engineering Approaches to Software Components and Architectures.* Accepted, 2009.
[7] OMG. *Semantics of a Foundational Subset for Executable UML Models.* OMG, 2008.

(1) Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: {ilazar,motogna,bparv}@cs.ubbcluj.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 198–201

COMPONENT MODELS' SIMULATION IN ContractCML

VLADIELA PETRAŞCU(1), DAN CHIOREAN(1), AND DRAGOŞ PETRAŞCU(1)

Abstract. Through the support offered for simulation, executable modeling provides models with a deeper analytical power. In the software components field, simulation is highly valuable, since it allows performing component interoperability checks. Within this paper, we propose a simulation method for ContractCML component services in the context of the XMF framework.

1. Introduction

In [4], we have introduced ContractCML, a hierarchical component DSML focused on representing components' usage contracts. To date, the language metamodel only supports the first two contract levels (syntactic and semantic), but its highly extensible architecture allows the two remaining ones (synchronization and quality-of-service) to be easily added. The language was thought as the backbone of a framework meant to support a rigorous specification of software components, as well as their semantically correct assembling into component-based applications.

Through the present paper, we intend to further contribute to such a framework, by proposing a simulation method for ContractCML component services. Adding simulation capabilities to our language is highly beneficial, since it allows us to test component assemblies and perform component interoperability checks early in a system's lifecycle. The proposed method relies on our approach for representing components' semantic contracts, which is detailed in [4]. The simulation takes place in the context of the XMF framework [1],[3], being based on a XCore representation of the ContractCML metamodel and the use of XOCL (an executable OCL-like language).

2. Proposed Simulation Method

To facilitate understanding of the proposed simulation method, we first recall the metamodel concepts needed in order to represent syntactic and semantic contracts for software components (diagrams are omitted due to space constraints of the paper, see [4] for details). From a syntactic perspective, a ContractCML component `Interface` consists of a collection of `Services`, each of which owns a sequence of `Arguments`. The Design by Contract - style semantics of such an `interface` can be described by means of its associated `DBCSpec` metaclass instance. To accomplish this, each `dBCSpec`

2000 *Mathematics Subject Classification.* 68N30, 68U20.
Key words and phrases. DSML, XMF, metamodel, software component, simulation.

owns an information model, instance of the `InfoModel` metaclass. An information model can be thought of as a typical class model. However, among the classes in the information model, there is one having a special status and which will be called *the main class of the information model* in the following. This class, instance of the `InterfClass` metaclass, owns operations having signatures identical to those of the interface services. The other classes in the information model are referred to as information-type classes; these are instances of an `InfoTypeClass` metaclass and alltogether they abstract that part of a component's state that may be affected by the execution of interface services. The main class should have at least one composition relationship with an information-type class. The semantics of interface services is expressed by means of pre/post-condition pairs, written as OCL expressions in the context of the main class and in terms of the information-type classes.

As emphasized in [2], [4], the main purpose of an interface information model is to support the specification of level two semantic contracts for software components. However, by employing an executable OCL-like language, such as XOCL [3], it becomes possible to use the information model for simulation purposes as well. Namely, we may simulate a service belonging to the provided interface of a component by defining an XOCL body for its homonymous operation located in the main class of the interface's information model, and then executing it. Simulation of a provided interface service will be thus performed in terms of the interface's information model. Still, this simple technique works only for components that have no requirements on the environment. In order to realistically simulate those having both provisions and requirements, it became necessary to enrich the ContractCML metamodel, enabling it to represent not only usage contracts, but also *realization contracts.*

Unlike usage contracts, which are for clients, realization contracts are for the component realizers/implementors [2]. A realization contract attached to a component type contains information regarding the way in which its provided services should be designed in terms of the required ones. In [2] these interactions are described using UML collaboration diagrams. In ContractCML, we allow expressing such contracts by means of a special type of class, called `RealizationClass`, an instance of which may be attached to a component type (Figure 1). Any `RealizationClass` instance

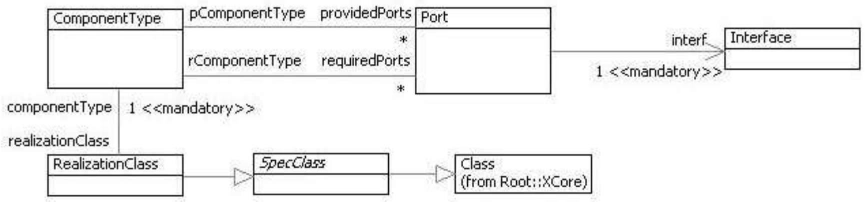


FIGURE 1. ContractCML `ComponentTypes`

attached to a component type has to fulfill two mandatory constraints, both of them being formalized as metamodel WFRs: (1) it has to inherit all main classes from the information models of interfaces exposed through the component type's provided ports, and (2) it has to hold a reference to each main class from the information model of each required interface. The realization contracts themselves are then expressed by overriding the inherited methods (which perform simulation in terms of the interface information models) such that they delegate calls to the owned references. In case a certain component type has no required interfaces (acting as a leaf in a component architecture), then no overriding is needed, the simulation of a provided service being performed at the level of the information model attached to the interface to which the service belongs.

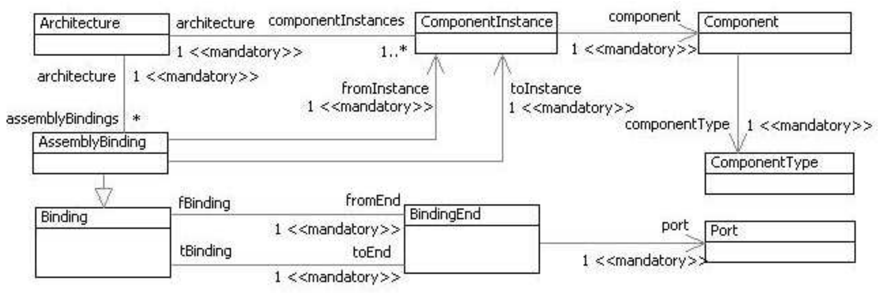


FIGURE 2. ContractCML `Architectures`

Apart from representing realization contracts, we have added a `Simulator` class at the metamodel level. In order to simulate any of the services provided by a particular component instance, it is necessary to create such a `Simulator` object. It is assumed that the component instance is part of an architecture in which its requirements are provided by other component instances and so on. The `Simulator` constructor takes the component instance as an argument, which it uses in order to instantiate what we have called a `simulationBase`. The latter is, in fact, a completely and recursively configured instance of the `RealizationClass` corresponding to the argument's component type. We give in the following the XOCL body of the `Simulator` operation returning the simulation base. All navigations involved in describing the operation can be tracked in figures 2 and 1.

```
@Operation getSimulationBase(inst:ComponentInstance):XCore::Element
  let arch = inst.architecture;
      compType = inst.component.componentType then
      cls = compType.realizationClass then res = cls()
  in @For reqPort in compType.requiredPorts do
       let portName = reqPort.name;
           binding = arch.assemblyBindings->select(b|
             b.fromEnd.port = reqPort and b.fromInstance = inst)->sel then
           instance = binding.toInstance then
           reqInstance = getSimulationBase(instance)
       in res.set(portName,reqInstance)
       end
     end;
     res
  end
end
```

Having such a base configured, simulation of a service provided by the component instance is achieved by calling the `simulate` operation on its corresponding `Simulator` instance. The service itself, as well as the sequence of values for its arguments should be passed as parameters.

```
@Operation simulate(service:Service,arguments:Seq(Element)):XCore::Element
  let cls = self.simulationBase.of() then
      op = cls.allOperations()->select(o|
        o.name.asSymbol() = service.name.asSymbol())->sel
  in op.invoke(self.simulationBase,arguments,null)
  end
end
```

3. Conclusions

Through this paper, we have provided a simulation method for ContractCML components' services within the XMF execution framework. To date, we do not know of similar approaches to simulation in the literature. The feasibility of our proposal has been proved using a variant of the Reservation System case study from [2]. For now, the simulations are executed inside the XMF console, but the development of a graphical concrete syntax for ContractCML and of an associated graphical editor are planned as future work.

4. Acknowledgement

This research has been realized in the framework of the IDEI research project "Frame based on the extensive use of metamodeling for the specification, implementation and validation of languages and applications", code ID_2049, financed by the Romanian National University Research Council (CNCSIS).

References

[1] XMF Web page: http://itcentre.tvu.ac.uk/∼clark/xmf.html.

[2] John Cheesman and John Daniels. *UML Components: A Simple Process for Specifying Component-Based Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[3] Tony Clark, Paul Sammut, and James Willans. *Applied Metamodeling. A Foundation for Language Driven Development.* Ceteva, 2008.

[4] Vladiela Petraşcu, Dan Chiorean, and Dragoş Petraşcu. ContractCML - a Contract Aware Component Modeling Language. *Tenth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'08)*, 2008. (to appear).

(1) Babeş-Bolyai University of Cluj-Napoca, Romania
E-mail address: `{vladi,chiorean,petrascu}@cs.ubbcluj.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 202–205

EFFORT ESTIMATION BY ANALOGY USING A FUZZY CLUSTERING APPROACH

MILITON FRENŢIU [(1)] AND HORIA F. POP[(1)]

Abstract. Estimation of cost and time to develop a software product P is a well known problem. Among the possible ways to do it is analogy with old finished projects. We describe an experiment using different versions of the Fuzzy c-Means algorithm, to find projects similar to P. Then the information about these projects is used to estimate the effort needed to build P.

1. Introduction

Estimation of cost and optimal time to develop a software product is a well known problem. There are various methods to solve this problem:

- expert opinion, i.e. guessing on personal experience;
- analogy with already finished projects;
- decomposition of the system into its smallest components and estimating the effort to build each component;
- models using mathematical formulas to compute the cost.

There are some models for cost estimation. Here we mention COCOMO I [Boe81], Albrecht's Function Points [Alb83], and COCOMO II [Boe85]. The COCOMO I is based on the size of the product, which is unknown at the beginning of the software process. Albrecht's model is based on the requirements and uses the information about the input data, the needed results, the files used in the system and the inquiries made by the user. COCOMO II effort estimation equations are adjusted basic equation using 17 cost drivers and five scale factors. These cost drivers are personnel attributes, process attributes, product attributes, and computer attributes.

To predict by analogy the cost (effort) and time to develop a software system P means to find the most similar finished system S and to use its known characteristics for prediction [She96]. The most similar system is that one which is closest to P in the Euclidean space.

First problem that arise is to define and compute the similarity of two projects S and P. We need some important characteristics of these projects, and the possibility to obtain these characteristics from the requirements of the projects. They appear in Albrecht's model, or in COCOMO2, and may be quantifications for the complexity

2000 *Mathematics Subject Classification.* 68N30.
Key words and phrases. Software metrics, Cost estimation, Fuzzy clustering.

of the system, the required reliability, the number of inputs, the number of outputs, the number of files, the number of screens and so forth. If we use n characteristics to describe a project, it is represented as a point in the n-dimensional Euclidean space. The similarity of two projects may be defined as the distance between the corresponding points.

Estimating the cost of P by analogy is reduced to finding the closest previously completed similar project S. Then the actual data from the project S is extrapolated to estimate the cost of P.

An improvement to the above described method is to select some projects most similar to P and to deduce the cost for the proposed project from data of these most similar completed projects. Here we suggest using both the well-established Fuzzy Hierarchical Clustering procedure and as well a restricted approach thereof, to find the class C of the most similar projects to P.

The cost of P is computed as a weighted average of $Cost(P;S)$ for $S \in C$, i.e.

$$Cost(P) = \sum w(S) \times Cost(P;S), S \in C$$

where the costs $Cost(P;S)$ are estimations of $Cost(P)$ based on information available for project S, the weights $w(S)$ are determined based on the membership degrees of all items in the class and the sum of all $w(S)$ for all $S \in C$ is 1. Alternate estimations of the cost will be studied in a future paper.

2. Restricted Fuzzy Clustering

Let us consider a set of classified objects, $X = \{x^1, \ldots, x^p\} \in \mathbb{R}^s$ and the fuzzy partition $P = \{A_1, \ldots, A_n\}$ corresponding to the cluster substructure of the set X. Let $x^0 \in \mathbb{R}^d$ be an object that needs to be classified with respect to the fuzzy partition P.

The algorithm we are presenting here computes the optimal fuzzy partition $\tilde{P}$ corresponding to the set $\tilde{X} = X \cup \{x^0\}$, by using a mechanism similar to Fuzzy n-means, with the difference that the membership degrees of the objects in X to the classes A_i, $i = 1, \ldots, n$ may not be modified.

In what follows we consider a metric d in the Euclidean space $\mathbb{R}^s$. We will suppose that d is norm induced, so $d(x,y) = (x-y)^T M (x-y), x, y \in \mathbb{R}^s$, where M is a symmetrical and positively defined matrix.

The objective function we have in mind for our problem is similar to that for the Fuzzy n-Means Algorithm:

$$\tilde{J}(\tilde{P}, L) = \sum_{i=1}^{n} \sum_{j=0}^{p} \left(A_i(x^j)\right)^2 d^2\left(x^j, L^i\right),$$

with the mention that $A_i(x^j)$ are kept constant for each i and for $j = 1, \ldots, p$.

The main result with respect to determining the fuzzy partition $\tilde{P}$ and its representation L minimizing the function $\tilde{J}$ is the following

Theorem. (i) The fuzzy partition $\tilde{P} = \{A_1, \ldots, A_n\}$ has the minimum value of the function $J(\cdot, L)$ if and only if

$$A_i(x^0) = \frac{1}{\sum_{k=1}^{n} \frac{d^2(x^0, L^i)}{d^2(x^0, L^k)}}. \tag{1}$$

(ii) The set of prototypes $L = \{L^{,} \ldots, L^n\}$ has the minimum value of the function $J(\tilde{P}, \cdot)$ if and only if

$$L^i = \frac{\sum_{j=0}^{p} \left(A_i(x^j)\right)^2 x^j}{\sum_{j=0}^{p} \left(A_i(x^j)\right)^2}. \tag{2}$$

With this result, the optimal membership degrees for x^0 to the classes A_i will be determined using an iterative method in which $\tilde{J}$ is successively minimized with respect to $\tilde{P}$ and L. The process will start with the initialization of prototypes L^i to the values that correspond to the fuzzy membership degrees of the original fuzzy partition P.

The resulted algorithm, **Restrictive Fuzzy n-Means Clustering Algorithm**, follows:

S1 Let us have X and P as given variables.
S2 Determine the initial positions of the prototypes L^i according to value of P.
S3 Determine the membership degrees $A_i(x^0)$, $i = 1, \ldots, n$, using relation (1).
S4 Determine the new positions of prototypes L^i, $i = 1, \ldots, n$, using relation (2).
S5 If the new positions of the prototypes L^i are close enough to the former positions, then **stop**, else return to step S3.

3. Experiment

We have used the information we had about 19 student software projects to estimate the cost of a new project (numbered 20). This information refers to the complexity, reliability, considered difficulty (all three metrics have values between 1 and 5, denoting very low, low, normal, high and very high), number of inputs, number of outputs, number of files and number of screens). Also, the computed function points and the cost of these projects are known.

The final partition produced using this 20×8 data matrix contains three classes. All the three classes are well separated. The core elements of the classes have quite high fuzzy membership degrees, while only a few of elements per class, not members of the class, have fuzzy membership degrees reasonably large.

We are estimating the cost of project 20 using the projects member of the very same class project 20 is a member of. Very similar results are obtained using the other suggested approach, i.e. Restricted Fuzzy Clustering, where we first find the cluster substructure of projects 1-19 and then embed the 20-th project as an extra data item in the former fuzzy partition.

4. Conclusions and Future Research

We have used fuzzy classification as a way to find the projects similar to our project P. There are usually more than one very similar projects, and these are the projects found in the same class with P. Then, to predict the cost/effort needed to built P, we have used the information about all these projects similar to P.

It is considered that estimating by analogy is a superior technique to estimation via algorithmic model in at least some circumstances [She96]. However, it requires to maintain a database with information about the finished systems. Oftenly, the information about finished projects is incomplete, and we cannot find and add the missing information about these old projects. It would be useful to obtain the classification in the absence of some information, and we intend to experiment such a method in the near future.

Also, factorial analysis [Wat67] may be used to predict the effort to build a new project from the information about finished projects. As a further work, we intend to compare the predictions obtained with these three different methods!

5. Acknowledgement

This material is based upon work supported by the Romanian National University Research Council under award PN-II no. ID_550/2007.

References

[Alb83] Albrecht A. J., Gaffney Jr J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. IEEE Transactions on Software Engineering 9 (6), 639-648.

[Boe81] Boehm B. W. (1981). Software engineering economics, Prentice-Hall: Englewood Cliffs, N.J.

[Boe85] Boehm B. W., Clark B., et al. (1995). Cost models for future software life cycle processes: COCOMO 2.0. Annals of Software Engineering 1, 57-94.

[She96] Shepperd M., Schofield C., Kitchenham B., Effort Estimation Using Analogy, Proceedings of the 18th International Conference on Software Engineering, Berlin, 170-178.

[Pop95] Pop H. F., Supervised fuzzy classifiers. Studia Universitatis Babes-Bolyai, Series Mathematica 40, 3 (1995), 89-100.

[Wat67] Watanabe S., Haven H., Karhunen-Loeve expansion and Factor analysis. Theoretical remarks and applications, in Transactions of the Fourth Prague Conference on Information Theory, Statistical Decision Functions, Random Processes, Prague 1967.

(1) Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania

E-mail address: mfrentiu,hfpop@cs.ubbcluj.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 206–210

SOFTWARE COST ESTIMATION MODEL BASED ON NEURAL NETWORKS

CĂLIN ENĂCHESCU(1) AND DUMITRU RĂDOIU(2)

Abstract. Software engineering is providing estimation models in order to evaluate the costs for important phases of software development such as requirements analysis, high and low level design, development, testing, deployment and maintenance. The paper proposes a software cost estimation model based on neural networks and compares it with traditional cost estimation models. The paper argues that there are at least two considerable advantages for the neural network model: neural networks can learn from previous experience and secondly, neural networks can discover a complete set of relations between dependent variables (software cost, software size, required effort, etc.) and independent variables (complexity of the project, number of inputs and outputs, files, various cost drives etc.).

1. Software cost estimation

A very critical aspect for suppliers and clients in a software project is the accuracy of the software project cost estimation [3]. This accuracy represents the basic argument in contract negotiation, project plan and project controls. Accuracy has a great influence on the following items [13]:

- Prioritization and classification of the software development projects;
- Accurate estimation of the required human resources;
- Evaluation of the financial impact determined by the chance requests;
- Efficient control of the project based on realistic and efficient resource allocation;
- Keeping the project cost in the reasonable limits as agreed with the client.

The estimation of a project cost (measured in currency) is based (in broad lines) on the following estimations [5]:

- Project size (e.g. measured in functional points);
- Project effort (e.g. measured in person/months);
- Project duration (e.g. measured in working days);
- Project schedule (measured in calendar days);
- Other cost drivers (e.g. labor cost, organizational overhead, procurement costs);

2000 *Mathematics Subject Classification.* 68N30, 68T05, 62M45.
Key words and phrases. software development cost estimation, software metrics, neural networks.

- Constraints and priorities.

One approach starts by decomposing the expected deliverable of the project into manageable chunks (product breakdown structure), using it to build a work breakdown structure (WBS) and making a direct estimation of the required effort using expert opinion (experienced professionals who can make estimations based on previous experience). Size of the project and required effort can be converted afterwards in project duration and project cost using empirical formulas. Estimations made for each phase of the project are summed up generating overall size, effort, schedule and cost [17]. For better estimations, the results could be weighted by factors (e.g. complexity of the project, familiarity of the team with the required technology or business knowledge, risks), generating the final project estimations. In the last 3 decades different models for software cost estimation have been proposed, ranging from empirical models [12] to elaborated analytical models [3], [5], [16]. Empirical models use data from previous software development projects in order to make predictions for similar projects. The analytical model uses a formal approach, based on global presumptions, like how well can a team member solve a task and how many issues are possible to appear. Both approaches lead to the conclusion that accurate cost estimation in a software development project remains a very difficult task [11].

2. Estimation process for software development

An estimation process follows a number of steps [3]:

(1) Setting cost estimation objectives;
(2) Developing a Project Plan for each request and each resource;
(3) Analyzing the software development requests;
(4) Working out as much detail about the software development project;
(5) Using an alternative cost estimation technique for cross validation;
(6) Comparison between estimations and through repeated iterations;
(7) Permanent controlling of the project during the development phase.

In order to achieve a precise estimation, no matter what cost estimation model has been chosen, there must be a permanent fine tuning of several parameters during the whole project development life cycle. The fine tuning can have influence on the resource allocation strategy for the estimated software development project without changing the overall cost estimation.

3. Software metrics

The software metrics have a great influence in the estimation of the cost of a software development project. There are several software metrics used to estimate the size of a software development project: ***Lines Of Code (LOC)*** and ***Functional Points (FP)*** metrics being the most popular.

- ***Lines of Code metrics (LOC)***: this software metrics is based on the calculation of the number of source code instructions, excepting comments. ***LOC*** metrics is unfortunately dependent on the technology and programming language used for the development and can be used for the cost estimation only in the final phases of development when the entire software project has been finalized [2].

• ***Functional Points metrics (FP)***: this software metrics takes into account the program functionalities (e.g. number of interrogations, reports), program complexity (e.g. number of logical files) and a large number of "adjustment" factors (e.g. does the system require real time processing?) [2].

• ***Object points metrics (OPM)***: this hybrid software metrics (borrows concepts from Functional Points metrics and object-oriented paradigm although the concept "object" is not used as it is used in object oriented paradigm literature) assesses the complexity of required "objects" like: user screens, reports, 3GL components etc. Each object is estimated, a weight between 1 and 10 being attached to each object, simple objects like user screens are estimated with weight 1 and complex objects (3GL component) are weighted with 10. One of the most largely used cost estimation model is COCOMO II [4] which is prized for its results in making estimations in the initial stages of the software development project.

4. Cost estimation for software development projects based on neural networks

There are two general methods to estimate the cost for software development projects: algorithmic methods and heuristic methods. The heuristic methods are generally based on statistical decisions, on regression models or on complex analytical differential equations [1].

4.1. Preliminaries.

As we have already discussed, cost estimation for software development projects has remained for decades one of the major problems to be solved in software engineering [15].

Algorithm-based computers are programmed, i.e. there must be a set of rules which, a priori, characterize the calculation that is implemented by the computer. Neural computation, based on neural networks, solve problems by learning, they absorb experience, and modify their internal structure in order to accomplish a given task [9].

In order to solve a problem using a neural network we have to decide regarding the architecture of the neural network, learning algorithm and upon several issues that are influencing the learning process, like: data encoding, learning rate, weight initialization, activation functions etc. [7]. We are using in our model for cost estimation for software development projects based on neural networks a Multi Layer Perceptron [6] architecture and a specialized Back Propagation type of learning algorithm [7]. The training set used (COCOMO'81) contains historical data from 63 software development projects. The training samples have each 17 attributed related to the software development projects.

4.2. Learning data encoding.

Learning data encoding is necessary before star-ting the learning process in order to make all attributes comparable in their influence upon the learning process. The learning process will stop when a certain error is reached, based on a type of ***Mean Square Error*** [8], or when a certain number of learning epochs has been performed.

After stopping the learning process we have to measure the quality of the trained neural network. For this step we will use a validation set from the training set that was not presented in the learning phase to the neural network. If, the validation phase is producing an acceptable result, then the neural network can be used in production for cost estimations of real life software development projects.

Because the ***COCOMO*** [18] training data set contains some attributes like person/months that have a high variation we have applied a logarithmic transformation in order to normalize such attributes.

4.3. **Experiments and simulations.** We have implemented a neural network of Multi Layer Perceptron type with 2 hidden layers and we have applied an enhanced Back Propagation learning algorithm. Based on this implementation, after the learning and validation phase, we were able to perform realistic cost estimation for software development projects. During these experiments and simulation we have varied the parameters that are influencing the learning process in order to obtain the most efficient neural network model. The parameters that have been taken into account where: dimensionality of the training set, learning rate, number of neurons in the hidden layers, number of epochs [10].

The simulations where performed using the COCOMO [18] dataset as training set. We have randomly chosen 40 projects to be included in the learning phase and the rest of 23 projects have been used for validation.

5. Conclusions

We have applied a new model based on neural networks for the cost estimation of the software development projects. The obtained results are promising and can be an important tool for project management of software development [14]. Based on the fundamental ability of the neural networks to learn from a historical training data set, we can use the experience contained in previously successful or unsuccessful estimations to make new reliable software cost estimations for software development projects.

References

[1] Al-Sakran H., Software cost estimation model based on integration of multi-agent and case-based reasoning. Journal of Computer Science, pag. 276-278, (2006).

[2] Albrecht A., Gaffney J. Jr., Software function, source lines of code, and development effort prediction: A software science validation. IEEE Trans. Software Eng., vol. 9, pp. 639-648, (1983).

[3] Boehm B., Software engineering economics, Englewood Cliffs, NJ:Prentice-Hall, ISBN 0-13-822122-7, (1981).

[4] Boehm B., et al., Software cost estimation with COCOMO II, Englewood Cliffs, NJ:Prentice-Hall, ISBN 0-13-026692-2, (2000).

[5] Cantone G., Cimitile A., De Carlini U., A comparison of models for software cost estimation and management of software projects. Computer Systems: Performance and Simulation, Elsevier Science Publishers B.V., (1986).

[6] Enăchescu C., Rădoiu D., Adjei O., Learning strategies using prior information. IJICIS-International Journal of Intelligent Computing and Information Science, Vol. 5, Nr. 1, pp. 381-393 (2005).
[7] Enăchescu C., Properties of Neural Networks Learning. 5th International Symposium on Automatic Control and Computer Science, SACCS'95, Vol.2, 273-278, Technical University (1995).
[8] Enăchescu C., Neural Computing. Ph.D. Thesis, Babes-Bolyai University, Cluj-Napoca (1997).
[9] Haykin S., Neural Networks. A Comprehensive Foundation. IEEE Press, MacMillian, (1994).
[10] Hertz J., Krogh A., Palmer R.G., Introduction to the Theory of Neural Computation, Addison-Wesley Publishing Co. (1992).
[11] Idri, A., Khoshgoftaar M, Abran A., Can neural networks be easily interpreted in software cost estimation?, Proceedings of the 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02, (2002).
[12] Kelly, M., A methodology for software cost estimation using machine learning techniques, Master Thesis, Report A851372, United States Navy, (1993).
[13] Kemerer C., An Empirical Validation of Software Cost Estimation Models. Comm. ACM, vol. 30, pp. 416-429, May, (1987).
[14] Partridge D., Artificial Intelligence and Software Engineering, Glenlake Publishing Co., ISBN: 0-8144-0441-3, (1998).
[15] Selby R., Porter A., Learning from examples: Generation and evaluation of decision trees for software resource analysis. IEEE Trans. Software Eng., vol. 14, pp. 1743-1757, (1988).
[16] Putnam L. H., A general empirical solution to the macro software sizing and estimating problem. IEEE Transaction on. Software Eng., pp. 345-361, (1998).
[17] Windle D. R., Abreo L. R., Software Requirements Using the Unified Process: A Practical Approach.+ Prentice Hall, ISBN: 0130969729, (2002).
[18] ***, COCOMO, http://en.wikipedia.org/wiki/Cocomo.

(1) "Petru Maior" University of Târgu Mureş, Computer Science Department
E-mail address: `ecalin@upm.ro`

(2) "Petru Maior" University of Târgu Mureş, Computer Science Department
E-mail address: `Dumitru.Radoiu@Sysgenic.com`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 211–214

ONTOLOGY DEVELOPMENT: A SOFTWARE ENGINEERING APPROACH

DUMITRU RĂDOIU(1) AND CĂLIN ENĂCHESCU(2)

Abstract. Indentifying several analogies between ontology development and object oriented analysis and design, the paper reviews the existing literature to assess the practical utility of a software engineering in ontology development as well as its limits.

1. Semantic Web, Ontology and why should we develop them

Semantic Web [4] is the new-generation Web (Web 3.0) that tries to represent information in such a way that it can be used by machines not only by humans (as it is the case with hypertext mark-up languages based Web 1.0); it's about making information stored in *Web Resources* accessible for automatic processing, integration, and reuse across applications [1]. This leads to large-scale interoperation of Web services, to creation of a Web of machine-understandable and interoperable services which intelligent agents can discover, execute, and compose automatically [2].

First step to accomplish this vision is declaring how the information of a domain (e.g. education) is semantically organized: generic entities, their possible properties, what type of relationships could we expect to exist among entities, how different entities could be grouped (taxonomy), what terms (vocabulary) we agree to use when presenting new information from this domain. This is called developing an *ontology* for the domain (an ontology and not the ontology because there is more than one way to develop it).

Second step - when we make a resource available on the Internet - is to declare that a this *Web Resource* adheres to the previously mentioned semantic structure, to the aforementioned *ontology*. Although there's no general agreement with respect to the term *knowledge*, we shall use it with the meaning "organized semantic information" (different from *human knowledge*) and the action of making it available in this form is referred as providing *semantic-based access*.

Web 1.0, based on mark-up languages, was concerned only with how to *display information* in a consistent manner for human consumption. Knowledge was achieved by humans accessing information distributed in billions of Web Resources. Search engines would return answers to queries without having any idea about the meaning of

2000 *Mathematics Subject Classification.* 68N30, 68T05, 62M45.
Key words and phrases. ontology development, semantic web, software engineering.

the returned hits. Most of the hits were useless and extracting structured information was the job of the humans. Developing an ontology for a domain and writing content which adheres to it allows automatic extraction of new knowledge from different web resources either to make it available for further automatic processing or to save humans effort in finding high value structured information. Building an ontology of a domain is not a goal in itself; it is just a means to an end.

The first benefit is that we share a common understanding of the way the information in a domain is structured [3]. The second (with huge implications) is that we can now separate the process of structuring the information of a domain from the process of automatically retrieving new structured information from different web resources.

2. Defining Ontology

"An ontology is an explicit representation of a shared understanding of the important concepts in some domain of interest" [5]. *Human knowledge* is subjective. Ontology is conceived to explicitly describe a cognitive structure upon which individuals "objectivity agree about subjectivity" [6].

An ontology can be expressed at different levels of abstractions:

- in a natural language, as sets of declarative statements
- in a visual modeling language (e.g. UML) as a set of glyphs syntactically and semantically organized
- in a formal language (e.g. OWL) as a set of formal statements

Natural language statements are difficult to process by a computer (program). Visual representations - although very expressive - don't provide enough flexibility. Hence most ontology editors allow visual representations which they convert automatically in a formal language.

So, the aim of any *ontology* is to capture the intrinsic conceptual structure of the domain by providing a "catalog" of the type of "things" assumed to exist in the domain, their properties and how can we "reason" about them. It follows that ontology is concerned with:

- The type of "things" in the domain. "Things" are called *entities* (objects, actors), they may or may not have *properties* (which may take or take not *values*)
- The explicit hierarchical organization/categorization of entities in a domain (named *taxonomy*)
- Possible relations among entities (semantic interconnections). They are described in short sentences (named *triples*)
- What terms (and rules to combine them) we can use when documenting knowledge about the domain (*vocabulary*)
- What inference laws and logic laws can we used when we automatically process a resource which adheres to our ontology

Meeting all of the above requirements qualifies ontology as a *content theory* and its specification calls for elaborate formal languages, namely *ontology representation languages*.

Because there's no "best way" to develop an ontology there may be several ontology (is there a plural ontologies?) about the same domain. Combining them is an active research field.

The high-level list of key application areas for ontology includes: collaboration, interoperation, education, and modeling.

3. Developing Ontology using a Software Engineering Approach

Ontology development is a hard work even with the most advanced ontology development languages, environment and methodology. Here are a few questions requiring answers on which there's a large consensus among the domain experts:

- How do we analyze, design, develop, test, maintain and support ontology to meet knowledge engineer requirements?
- What ontology development methodology is the most suitable to task?
- How do we combine different ontologies describing the same domain?
- Since all parts of the semantic web evolve in time (ontology, ontological knowledge stored in web resources), how do we resolve the problem of knowledge maintenance?

The answer is all but easy. One of the goals is to "partially automate the processes of ontology development and maintenance by developing tools and frameworks to help extract, annotate, and integrate new information with the old one in the ontology" [6]. The paper defends the idea that we can start searching for answers by analogy, looking at the best practices in software engineering. There are several reasons for such an approach:

- an ontology represents a model of a domain like software applications which represent models of the world
- ontology development life cycle and object oriented analysis and design share many characteristics; both start from specific requirements, are completed through an iterative and incremental process
- ontology development like software development uses a ranges of methodologies which specify how to perform a sequence of activities, supported by tools and techniques, to deliver a specific product
- ontologies include classes and subclasses hierarchies similar to object oriented software engineering and similarly uses UML (Unified Modeling Language) for knowledge representation

References

[1] Boley, H., Tabet, S., and Wagner, G., Design rationale of RuleML: a markup language for Semantic Web rules, in: Proc. SWWS'Ol, The first Semantic Web Working Symposium, Cruz, I.E., Decker, S., Euzenat, J., McGuinness, D.L., eds., Stanford University, California, pp. 381-401 (2001);

[2] Mcllraith, S.A., Son, T.C., and Zeng, H., Semantic Web services, IEEE Intelligent Systems 16(2), pp. 46-53, (2001);
[3] Deborah L. McGuinness, *Conceptual Modeling for Distributed Ontology Environments*, Conceptual Structures: Logical, Linguistic, and Computational Issues, pp. 100-112, (2000);
[4] W3C Semantic Web http://www.w3.org/2001/sw/ (retrieved March 27, 2009);
[5] Kalfoglou, Y., *Exploring ontologies*, in: Handbook of Software Engineering and Knowledge Engineering Vol.1 - Fundamentals, Chang, S.K., ed.. World Scientific Publishing Co., Singapore, pp. 863-887, (2001);
[6] Devedzic, V., *Semantic Web and Education*, Series: Integrated Series in Information Systems , Vol.12 , pp 33, (2006)

(1) "Petru Maior" University of Târgu Mureş, Computer Science Department
E-mail address: Dumitru.Radoiu@Sysgenic.com

(2) "Petru Maior" University of Târgu Mureş, Computer Science Department
E-mail address: ecalin@upm.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 215–218

DURATION ESTIMATION OF A WORK PACKAGE

ATTILA VAJDA(1)

Abstract. Estimating the effort and duration of software work packages will be never an exact science. Too many variables are affecting the result. This article explores the relation between the effort and duration in software engineering projects and presents an empirical model to predict work package duration based on estimated effort and team size.

1. Introduction

In the early days of computing, the project total cost was influenced more by the computer-based system costs then the software costs. Nowadays, software is the most expensive element in most computer based systems; any cost estimation deviation in the software development process will influence considerably the project profit and loss. Several studies [1], [2], [3] demonstrated that grand majority of software development projects are finished over budget and 30% of large projects are never finished. No wonder that project cost estimation became crucial, challenging aspects of the software development process and theme of many researches focusing on minimizing the gap between the forecasted and real software project costs. Related to project cost estimations, several researches were done already starting with 1965 (Nelson), then in the 70s the SLIM model was created by Putnam and Myers, in the 80's the PRICE-S model by Park, the SEER by Jensen, the COCOMO by Boehm, in the 90's the Checkpoint by Jones [4]. The estimation models, techniques resulted form the researches are numerous, some are sophisticated mathematical and statistical models, and others are expert system based [5]. Most of them are calibrated for medium and long term projects, only a very few can be adjusted for small projects performed by small teams. Many people work on small projects, which are generally defined as a staff size of a couple of people and a schedule of less than six months. Because the estimates for these small projects/work packages (WP) are highly dependent on the capabilities of the individual(s) performing the work, the best approach is to be estimated directly by those assigned to do the work. In the reality this is not always possible, and usually the project effort estimation is performed by a single person who can be the technical leader or a senior developer. Then the project duration is

2000 *Mathematics Subject Classification.* 68N30.
Key words and phrases. project duration, effort estimation, software engineering.

calculated as the estimated effort divided by the number of resources assigned to the project and eventually multiplied with a risk factor.

Since the WP can be considered as small projects, and any medium and long term project can be divided into multiple WPs, this paper focuses on WP duration estimations and proposes a simple empirical model for forecasting the work package duration based on historical estimation errors of the team members.

2. Work package effort estimation

The aim of this process is to predict the most realistic use of effort to finish a WP. In case of WP estimations, the experience based techniques are more useful, since in most cases there is no quantified, empirical data to use for the estimation process [4]. This technique is based on the knowledge and experience of practitioners seasoned within a domain of interest. It provides the estimates based upon a synthesis of the known outcomes of all the past projects he or she previously participated. Two techniques have been developed which capture expert judgment: Delphi and Work Breakdown Structure [4], [5]. In case of small projects/teams the second technique is more widely used, and it is based on the decomposition technique of "divide and conquer" approach. After decomposing the WP into components, each component individually needs to be estimated to produce the overall estimate. The preferred estimation method for small software projects is the Three Point Estimation (*TPE*) technique, which is based on statistical methods, and in particular, the Normal distribution. With the TPE the planner produces three figures for every component, and the total work package effort will be the sum of the estimated component efforts, as follows:

$$E_{WP} = \sum_c E_c = \sum \left(\frac{(a + 4 \cdot m + b)}{6} \right)_c \tag{1}$$

where E_{WP} is the total effort of the WP, E_C is the effort estimated for a component (task), a is the best-case estimate, m the most likely estimate and b the worst-case estimate of the task.

3. Estimating work package duration

WP duration is the calendar period that it takes from the beginning of the work till the moment it is completed. Many works like [7], [8] proposed to predict the duration of the software development project based on project effort. In case of most effort and duration estimation models the number of resources (P) involved in the project are defined in the last step of the estimation process, while in case of WP, P is known already right after the effort estimation (E), the WP duration (D) is calculated as follows:

$$D = f(E, P) = \frac{E_{WP}}{P} \tag{2}$$

Since the aspiration of every project manager/stakeholder is to have its project completed as soon as possible, the tendency is to minimize D by increasing P. Because usually the persons involved in the projects have different productivity indexes (Pid), the relation (2) needs a correction, which will adjust D with a coefficient resulted from the Pid of each project team member. The Pid can be calculated as a report between the estimated (E_{cest}) and the actual effort (E_{creal}) a team member spent on the c component:

$$Pid_i = \frac{E_{Creal}}{E_{Cest}} \tag{3}$$

Depending on how the original estimations were performed, based on the organization historical data using TPE technique, or based on a single person experience, the Pid will reflect the person's relative productivity compared to the organization's average or compared to that person productivity. If we introduce relation (3) in relation (2), the result will be an adjusted D (D_{adj}):

$$D_{adj} = \frac{E_{WP}}{\sum_i Pid_i} = \frac{E_{WP}}{P} \cdot \left(\sum_i \frac{Pid_i}{P} \right)^{-1} = \frac{E_{WP}}{P \cdot prod} \tag{4}$$

where *prod* is the team's productivity coefficient. The value of *prod* can be influenced by the correctness of the E estimations and the learning curve of the personal. For more accurate *prod* calculations, the Pid_i needs to be calculated based on historical data, or predicted, taking into the trend of its professional evolution. In some situations, like new team members, new project type where no historical data can be used, the estimation will be erroneous. For this situation D_{adj} should be adjusted periodically during the project, by predicting the *prod* of the team based on their evaluation till that moment in the project.

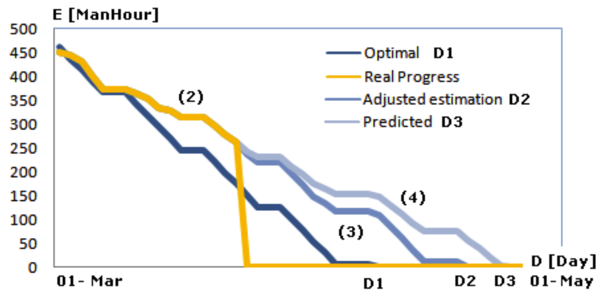


FIGURE 1. Comparing D estimations

Figure 1 shows the different results of the estimations: $D1$ was estimated with relation (2), $D2$ with the adjusted relation (4) and since the real progress is slower then expected, after the *prod* readjustments the $D4$ is obtained, which is more realistic then the previous ones.

4. CONCLUSION

Estimation of WP duration is a difficult and key problem in software engineering. There are several estimation models, but most of them are adjusted for medium and long term projects. Researches like [6] based on historical data of 396 projects of IS-BSG data set, showed that there exist a relation between E and D, which is nonlinear in case of medium and large projects, but can be considered as linear in case of small projects. For small projects, simple duration estimation model is needed, which can be adjusted easily to the team, project type, and gives a correct result based on the estimated effort and team size. By using the model presented in this paper, we reduce the project uncertainty by allowing the manager to obtain more accurate project duration estimations. Based on the Pid value the personal professional evaluation can be observed.

The model can be further improved by taking into account external factors which can influence the team productivity factors like team experience, team motivation, technical knowledge, etc. and to develop a model for more accurate productivity predictions.

References

[1] A. W. Chow, B. D. Goodman, J. W. Rooney, C. D. Wyble, Engaging a corporate community to manage technology and embrace innovation, IBM Systems Journal, Vol. 46, No. 4, 2007
[2] Steve McConnell, Rapid Development, Microsoft Press, Washington, 1996
[3] Gibs.W.W. Software's Chronic Crisis, Scientific American, September, 1994
[4] Boehm B., Abts C. Software Development Cost Estimation Approaches - A Survey, University of Southern California, Los Angeles, March 2000
[5] Temnenco V., Software Estimation, Enterprise-Wide, IBM The Rational Edge, Vol. June 2007
[6] Olign S., Bourque P., Abran A., Fournie B., Exploring the Relation Between Effort and. Duration in Software Engineering Projects, World Computer Congress 2000, Beijing, China
[7] Heemstra F.J, Softwae cost estimation, Information and Software technology 34 (10): 627-639,1992
[8] Kitchenham B.A.,Empirical studies of assumptions that underlie software cost estimation models, Information and Software technology 34 (4): 211-218,1992

(1) Sysgenic Group, Alunis 48, Targu Mures, Romania
E-mail address: `attila.vajda@sysgenic.com`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 219–222

A FORMAL CONCEPT ANALYSIS APPROACH TO ONTOLOGY SEARCH

IOAN ALFRED LETIA$^{(1)}$ AND MIHAI COSTIN$^{(2)}$

Abstract. Finding the most suited ontology for a given domain or problem is not an easy task but through our work here we are trying to ease this endeavor by providing a semiautomated search mechanism. Semiautomated, because we need our search target to be defined by an expert. Once that target defined, by combining the processing power of MAS with FCA, we analyse the search space in order to find the most compatible ontology with the initial target.

1. Motivation and Goal

The main step for solving a certain problem has always been finding or creating the best tool for the job at hand, no matter what the domain was.

In information science, our domain of interest, the tool in question is an ontology capable of mapping the problem domain with success. Finding the right ontology for a certain task has been even from the begining one of the theoretical strong-points of the semantic web but things are not as easy as it seems [8]. The right ontology is never easy to find and creating one is even harder.

This, without any doubt, involves some sort of intelligent ontology search, a guided search that has an array of ontologies as search space. For this paper, we are using a predefined set of existing ontologies as search space. The process and the results themself can later on be applied to a larger scale, all being reduced afterwards to a scalability problem.

We mentioned guided search because our process starts from a well defined domain. This domain must be described by some expert in the field using a set of concepts, concepts that will represent the search target. Our system uses this set of concepts in order to find the best suited ontology by expanding and matching it against the concepts found in the search space.

2. Proposed Solution

Having the initial domain defined by an expert in the field, as to direct our search, the search agents can then start trying to find the concepts that match the initial set.

2000 *Mathematics Subject Classification.* 68T30, 94A15.
Key words and phrases. Ontology, Formal Concept Analisys, Multiagent Systems, Search methods.

This search is conducted by using the properties of the concepts in the initial domain as targets, and the properties of the concepts in the array of available ontologies as search space. The most common, and probably the most efficient, method employed is the direct string search and this means that the search agent will try to find the given attributes (as strings or text) in the processed ontologies.

The multiagent system has to extract new concepts from the search space, matching the initial set and so, expanding it. We are using two different sets, so that we won't loose the initial set. The expanded one is employed when processing an ontology to guide the search. After the current ontology is processed, the expanded set may be kept for later usage, but the search through the next ontology has to start from the initial set again. This assures us that the search in one ontology will not be affected by results from another ontology. However, we can imagine a scenario where the search should be influenced by previous findings from other ontologies - and that is, when building a new ontology from an existing array of ontologies with a guiding set of concepts. But this scenario will probably be taken care of in future works.

The actors of the process are agents, part of a multiagent system, agents that are specialized for specific tasks - for example we have an agent that will take care of the "set of concepts into lattice" transformation and an agent that will extract the search elements from the initial domain. However, we will refer to these specific agents with the generic name agent.

The initial context is defined by the initial domain - given by an expert in the field and considered to contain enough information in order to describe the target of the search - and the array of available ontologies - in our scenario, this resumes to a couple of ontologies much like the ones we showed the snippets from.

Using this setup, we iterate through the available ontologies and we apply our search process, process that consists of a few major steps, to be described.

From the initial domain, the agent extracts the search elements that will be used to guide the search process itself. These search elements consist of properties and attributes from the concepts in the domain, so basically strings. These strings will be the target of the search process later on and will guide the agents into finding the lattice starting points in the search space.

Also from the initial domain, the agent extracts the initial lattice, by considering the ontology concepts to be the "formal objects" and their attributes to be the "formal attributes" of the lattice. This initial, or core lattice, in collaboration with the extracted lattices from the search space will later on provide the generated lattice and the generated lattice is our way of determinate the best ontology to use for the given domain.

As already mentioned, we are using string search in order to find a new concept or set of concepts that have to be added to our lattice. This part of our process might require some help from a word thesaurus and we base our affirmation on the fact that ontologies are mainly composed out of concepts that have a representation as a word in a certain language. This thesaurus, and it can be something similar to WordNet [1],

[1]http://wordnet.princeton.edu/

can help us find the search elements in the search space by using synonyms, meronyms and other word relations that can alone be the subject of an entire paper.

The concepts found in the previous step are, each one of them, the start point of an "extracted lattice". This lattice is to be constructed in a similar manner with the initial lattice but taking into account a concept radius. This can be seen as sort of a threshold for a concept set extracted from the search space. Starting from a found concept, the agent constructs a lattice by including all the concepts related to the found concept while staying withing the boundaries given by the concept radius. The concept radius can be any function defining types of relations between concepts and a simple example of such a function is the ontology relations set itself.

After the lattice has been extracted from the search space around the starting point, the agent starts integrating it in the main lattice. This can be done by means similar to FCA-Mapping[2] but since we started the search with the concepts from the main lattice, we already have the information regarding to what concept to link the extracted lattice to. Each starting point can be linked to a subset of concepts from the initial domain when they are found in the search space, and this process of integration will be done for each one of them.

After obtaining the generated lattice, by combining the initial lattice with the extracted ones, we can measure the degree of similarity between the initial domain and the current ontology by analyzing the generated lattice. There are multiple metrics we can use here, including the number of lattice concepts, the structure of the lattice, how well connected are the concepts and how many clusters we have and so on.

3. Related Work

Formal Concept Analysis has been mixed in the computer science field for a while now and not without good reasons according to its strong supporters for its benefits [11], but not until fairly recently researchers in the ontology field became interested in its potential [13, 2]. At the moment there are applications both theoretical [10, 9] and practical[2] that are using the power of FCA and this shows, without doubt, that FCA has some leverage in the knowledge related fields. However, we have the string belief that FCA has much more potential in this area as an ontology tool for tasks like similarity[3], mapping, validation and enhancement.

References

[1] Timothy J. Everts, Sung Sik Park, and Byeong Ho Kang. Using formal concept analysis with an incremental knowledge acquisition system for web document management.

[2] Liya Fan and Tianyuan Xiao. FCA-Mapping: A method for ontology mapping. In *Proceedings of the European Semantic Web Conference*, 2007.

[3] Anna Formica. Concept similarity in formal concept analysis: An information content approach. In *Knowledge Based Systems*, 2007.

[4] Suk-Hyung Hwang. A triadic approach of hierarchical classes analysis on folksonomy mining. In *IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.8*, August 2007.

[2]http://sourceforge.net/projects/conexp

[5] Robert Jaschke, Andreas Hotho, Christoph Schmitz, Bernhard Ganter, and Gerd Stumme. Discovering shared conceptualizations in folksonomies. In *ScienceDirect*, November 2007.

[6] Konstantinos Kotis, George A. Vouros, and Konstantinos Stergiou. Towards automatic merging of domain ontologies: The hcone-merge approach. In *Web Semantics: Science, Services and Agents on the World Wide Web 4*, 2006.

[7] Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. Structured objects in owl: Representation and reasoning. In *17th International World Wide Web Conference, Beijing, China.*, 2008.

[8] Daniel Oberle, Anupriya Ankolekar, Pascal Hitzler, Philipp Cimiano, Michael Sintek, Malte Kiesel, Babak Mougouie, Stephan Baumann, Shankar Vembu, Massimo Romanelli, Paul Buitelaar, Ralf Engel, Daniel Sonntag, Norbert Reithinger, Berenike Loos, Hans-Peter Zorn, Vanessa Micelli, Robert Porzel, Christian Schmidt, Moritz Weiten, Felix Burkhardt, and Jianshen Zhou. Dolce ergo sumo: On foundational and domain models in the smartweb integrated ontology (swinto). In *Web Semantics: Science, Services and Agents on the World Wide Web 5*, 2007.

[9] Marek Obitko, Vaclav Snel, and Jan Smid. Ontology design with formal concept analysis. In *Proceedings of the CLA 2004 International Workshop on Concept Lattices and their Applications, Czech Republic.*

[10] Uta Priss. Linguistic applications of formal concept analysis. In *Formal Concept Analysis*, pages 149–160, 2005.

[11] Uta Priss. Formal concept analysis in information science. In *Annual Review of Information Science and Technology*, pages 521–543, 2006.

[12] Gerd Stumme and Alexander Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *Proceedings of IJCAI*, pages 225–234, 2001.

[13] Yi Zhao, Xia Wang, and Wolfgang Halang. Ontology mapping based on rough formal concept analysis. In *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, 2006.

(1) Technical University of Cluj-Napoca, Department of Computer Science, Baritiu 28, RO-400391 Cluj-Napoca, Romania
E-mail address: letia@cs-gw.utcluj.ro

(2) Technical University of Cluj-Napoca, Department of Computer Science, Baritiu 28, RO-400391 Cluj-Napoca, Romania
E-mail address: mihai.costin@gmail.com

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 223–226

HIGH COUPLING DETECTION USING FUZZY CLUSTERING ANALYSIS

CAMELIA ŞERBAN(1)

Abstract. Coupling is an important criterion when evaluating a software design because it captures a very desirable characteristic: a change to one part of a system should have a minimal impact on other parts. An excessive coupling plays a negative role on many external quality attributes like reusability, maintainability and testability.

This paper aims at presenting a new approach concerning the identification of those classes with high coupling, from an object oriented system.

1. Introduction

In order to keep the software system easy to maintain, the assessment of its design according to well established principle and heuristics, has to be made through the entire development lifecycle. The principle of low coupling is reflected by all the authors [2, 3, 5] that propose design rules and heuristics for object-oriented programming. They all converge in saying that coupling should be kept low. A change to one part of a system it is important to have a minimal impact on other parts. Those design entities that could propagate a lot of changes in the system, if something has been changed inside them, have to be identified and reviewed.

In this article, we our goal is to identify the design entities that are strongly coupled with the rest of the system. The approach is based on fuzzy clustering analysis. Some metrics are used in order to quantify those aspects consired important with regard to coupling.

The remainder of this paper is organized as follows. Section 2 describes the theoretical background for our approach. Section 3 presents and discusses the experimental results obtained by applying the proposed approach on an open source application, called *log4net* [8]. Section 4 reviews related works in the area of detection design flaws. Finally, Section 5 summarizes the contributions of this work and outlines directions for further research.

2000 *Mathematics Subject Classification.* 68N30, 03E72.
Key words and phrases. coupling, metrics, fuzzy analisys.

2. Theoretical background

Object oriented design quality evaluation implies identification of those design entities that are relevant for the analysis and of the software metrics that best emphasize the aspects (design priciple/heuristics) that we want to quantify, and the interpretation of the measurements results obtained. These elements were described in detail in our previous work [9]. In what follows we will not get into their details, but we will try to list them, addapted to our current problem.

2.1. **A meta-model for object-oriented systems.** In [4] a meta-model for object-oriented systems is defined as a 3-tuple, $S = (E, P, R)$ where, E represents the set of design entities of the software system (classes, methods, attributes, parameters or local variables), P represents the set of properties of the aforementioned design entities (e.g. abstraction, visibility, reusability, reuse, binding) and R represents the set of relations between the entities of set E (e.g methods access).

2.2. **Some relevant coupling metrics.** A significant number of coupling metrics have been defined in the literature [10, 11, 12]. Because, we aim to identify those classes in which a change would significantly affect many other places in the source-code, we select coupling metrics that better emphasize this aspect.
Changing Methods (CM) is defined as the number of distinct methods in the system that would be potentially affected by changes operated in the measured class.
Weighted Changing Methods (WCM) [4]. WCM is computed as the sum of the "weights"(number of distinct members from the server-class that are referenced in a method) of all the methods affected by changes operated in the measured class.
Changing Classes (CC). The CC metric is defined as the number of client-classes where the changes must be operated as the result of a change in the server-class.

2.3. **Fuzzy clustering analisys.** Consider $C = \{c_1, c_2, ..., c_l\}$, $C \subset E$, the set of all classes from the meta-model defined in Section 2.1. Each class, c_i from C set is described by a 3-dimensional vector, $c_i = (cm, wcm, cc)$, where the components of c_i vector are the computed values for the metrics defined before.

Next, our goal is to group similar classes, in order to obtained a list of "suspects" (classes tightly coupled).

Because is hard to establish thresholds for the applied measurements, we used a fuzzy clustering analysis. An object will belong to more than one class with different membership degree. Also, in order to determine the number of clusters, which is a data entry for a fuzzy clustering generic algorithm, we considered the Fuzzy Divisive Hierarchic Clustering (FDHC) algorithm [7]. This algorithm produce not only the optimal number of classes (based on the needed granularity), but also a binary hierarchy that show the existing relationships between the classes.

3. Case study

In order to validate our approach we have used the following case study. The system proposed for evaluation is log4net [3], an open source application. It consists of 214 classes. The elements from the meta-model described in Section 2.1 have been identified. For each class from C set, we compute the values of the metrics defined in section 2.2. The algorithm FDHC, applied on this data, determine a fuzzy partition of C set. Table 1 briefly described this partition.

After the first step of the algorithm (the first binary partition), cluster 1 already contains the first list of five "suspects". Table 2 presents some details regarding this list. We are not interested for a further division for this cluster. The second cluster has been split. The first subcluster, 2.1 contains a list of 16 classes which have a big coupling, but not comparable with that of entities from Table 2. The second subcluster, 2.2 has been split and we obtained a list of 53 classes with normal coupling, subcluster 2.2.2, and a list of 140 classes with zero or low coupling, subcluster 2.2.1.

Due to space restrictions, we include in this paper only some important aspects concerning the results obtained. All other numerical data are available from the authors by request.

Class Partition	1.	2.1.	2.2.1	2.2.2
No. of items	5	16	140	53
Coupling level	strong	high	low	normal

Table 1. Fuzzy Partition

Object(Class)	CM	WCM	CC	Fuzzy Partition
Appender.AppenderSkeleton	40	20	38	1
Util.FormattingInfo	34	33	34	1
Core.Level	65	14	22	1
Util.LogLog	141	55	111	1
Util.SystemInfo	47	30	43	1

Table 2. Tightly Coupled Classes

4. Related work

During the past years, various approaches have been developed to address the problem of detecting and correcting design flaws in an object-oriented software system using metrics.

Marinescu [4] defined a list of metric-based detection strategies for capturing around ten flaws of object-oriented design at method, class and subsystem levels as well as patterns. Tahvildari et al. [1] proposes a framework for detected object oriented design flaws using a generic OO design knowledge-base.

However, how to choose proper threshold values for metrics is not addressed in these research.

5. Conclusions and Future Work

We have presented in this paper a new approach, based on metrics and fuzzy clustering analysis, that address the issue of coupling detection in an object-oriented system. The main advantage of our approach is that of avoiding the problem of setting up the thresholds for metrics that we used.

In the future work we aim to apply this approach for more case studies and for other design principle.

6. Acknowledgement

This material is based upon work supported by the Romanian National University Research Council under award PN-II no. ID_550/2007.

References

[1] Mazeiar, S., Shimin, Li. and Ladan, T. : A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws. Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC06), (2006).

[2] Fowler, M. Beck, K. Brant, J. Opdyke, W. and Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, (1999).

[3] Johnson, R. and Foote, B.: Designing reuseable classes. Journal of Object-Oriented Programming, 1(2):22–35, June (1988).

[4] R. Marinescu, Measurement and quality in object-oriented design. Ph.D. thesis in the Faculty of Automatics and Computer Science of the Politehnica University of Timisoara, (2003).

[5] Riel, A.J.: Object-Oriented Design Heuristics. Addison-Wesley, (1996).

[6] Coad, P. and Yourdon, E. : Object-Oriented Design. Prentice Hall, London, 2 edition, (1991).

[7] Dumitrescu, D.: Hierarchical pattern classification, Fuzzy Sets and Systems 28, 145–162, (1988).

[8] Project log4net.: http://logging.apache.org/log4net.

[9] Serban, C. and Pop, H.F.: Software Quality Assessment Using a Fuzzy Clustering Approach. Studia Univ. Babes-Bolyai, Series Informatica, 2 (2008), 27–38.

[10] Chidamber, S. and Kemerer, C.: A metric suite for object- oriented design. IEEE Transactions on Software Engineering, 20(6):476–493, June (1994).

[11] W. Li and S. Henry. Maintenance Metrics for the Object Oriented Paradigm. IEEE Proc. First International Software Metrics Symp., pages 5260, may 1993.

[12] Lorenz, M. and Kidd, J.: Object-Oriented Software Metrics. Prentice-Hall Object-Oriented Series, Englewood Cliffs, NY, (1994).

[1] Babeş-Bolyai University, Department of Computer Science, 1 Kogalniceanu St., 400084, Cluj-Napoca, Romania

E-mail address: `camelia@cs.ubbcluj.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 227–230

EFFICIENT RECURSIVE PARALLEL PROGRAMS FOR POLYNOMIAL INTERPOLATION

VIRGINIA NICULESCU[1]

ABSTRACT. *PowerList* and *PowerArray* theories are well suited to express recursive, data-parallel algorithms. Their abstractness is very high and assures simple and correct design of parallel programs. This high level of abstraction could be reconciled with performance by introducing data-distributions into these theories. We present in this paper the derivation of a correct recursive program for Lagrange interpolation. The associated costs are analysed and based on this an improvement of the program is presented.

1. INTRODUCTION

In this paper we extend the work presented in [5] by using data distributions for parallel programs defined using *PowerArray* structures. Also, by using these distributions we define also a possibility to define set-distributions.

2. DISTRIBUTIONS

The ideal method to implement parallel programs described with *PowerLists* is to consider that any application of the operators *tie* or *zip* as deconstructors, leads to two new processes running in parallel, or, at least, to assume that for each element of the list there is a corresponding process. A more practical approach is to consider a bounded number of processes n_p. In this case we have to transform the input list, such that no more than n_p processes are created. This transformation of the input list corresponds to a data distribution.

2.1. *PowerList* Distributions. Distributions were introduced in [5], where the advantages are presented of using them, too. Functions defined on *PowerList*s can be easily transformed to accept distributions. By formally introducing the distributions on PowerLists, we can evaluate costs, depending on the number of available processors - as a parameter.

2000 *Mathematics Subject Classification.* 68Q85, 65Y20.
Key words and phrases. parallel computation, abstraction, design, distribution, data-structures.

2.2. ***PowerArray* Distributions.** Distributions could be also introduced for *PowerArray* data structures. The operators *tie* or *zip* may be used, on each dimension. Depending on the possible combinations in the bidimensional case, we arrive to 4 types of data-distributions: *linear-linear, linear-cyclic, cyclic-linear, cyclic-cyclic.*

2.2.1. *Function Transformation.*

Theorem 1. *For a PowerArray*$X.n_0.n_1$ *function* f*, and a corresponding distribution* $distr.p_0.p_1$*, the function* f^p*, defined by:*

$$(1)\qquad \begin{aligned} &f^p.(u \diamond_i v) = \Phi_i\left(f.x_0, f.x_1, \ldots, f.x_m, u, v\right), i = 0, 1\\ &f^p.[l] = [f^s.l]\\ &f^s.u = f.u \end{aligned}$$

has the following property:

$$(2)\qquad f = flat \circ f^p. \circ distr.p_0.p_1$$

where, the function $flat$ *has to be defined based on the same operators as the function* f*.*

The proof contains a demonstration for each dimension, which is similar to the proof of Theorem of *PowerList* functions transformations, given in [5].

2.3. **Set-Distributions.** By using set-distribution, a data element is distributed to more than one process [4].

- One possibility to introduce set-distributions on these special types of data structures is to increase the dimension of the data structure by replication, and then apply a distribution on the obtained list.
- Another possibility to introduce set-distribution is to apply first a distribution, and then use a replication for the distributed data structure.

For *PowerList*s, a simple 2^p times replication on the second dimension, $(p \geq 0)$, is defined by:

$$(3)\qquad \begin{aligned} &rep_1.p.X = rep_1.(p-1).X \mid_1 rep_1.(p-1).X\\ &rep_1.0.X = X \end{aligned}$$

Replication could be also combined with different permutation functions. For example, replication with right rotation.

2.4. **Distribution Costs.** Data-distribution implies some specific costs which depends on the maximum number of distinct elements (θ_d) which are assigned to a processing element. We denote this cost by T_d, and its complexity order depends on θ_d.

In *PowerList* case, this cost is equal to the length of the sublists created after the execution of the distribution function *distr.p.* If the input list has 2^n elements and we apply the distribution function with the first argument equal to $p, p \leq n$, then the distribution cost has the complexity order equal to $O(2^{n-p})$.

In the set-distribution case, where data replication is used, this kind of costs depends on more parameters. If we use a simple replication on a new dimension, and then apply a Cartesian distribution, T_d is not influenced by the distribution on the new dimension. But, if a more complicated replication is used, T_d depends on the particular replication function.

3. Lagrange Interpolation

The evaluation of the Lagrange polynomial in a given point x is an example of application where the set-distributions could be used with success. The input lists are: $X = [x_0, \ldots, x_{m-1}]$ distinct points; $F = [f.x_0, \ldots, f.x_{m-1}]$ function values. Because of the lack of space we will not present the entire derivation of the program.

$$\begin{aligned}&lagrange.X.F.x = red(+).(L.X.x * F)\\&L.X.x = U1.X.x/U2.X\\&U1.X.x =< red(*).(< x- > .X)\ /\ \ > .(< x- > .X)\end{aligned}$$

Function $U2$ computes the list of products formed by $(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_{m-1})$, for any $i : 0 \leq i < m$. Since in these calculations an element appears in computations more than once, the use of replication is appropriate.

First, we consider a replication that is combined with a rotation function $repR_1.n.X$. In this way, each column contains different elements, which means that each column contains the elements of list X in a different order.

$$\begin{aligned}&U2.X = UA.(repR_1.n.X)\\&UA.A = red_1(*).(dif.(first.A).A)\end{aligned}$$

The function UA is the most costing one, and we will refer to it in more detail.
Distribution We define and use a Cartesian distribution $distr^{ll}.p_0.p_1$, where $p_0, p_1 < n$ and $p_0 + p_1 = p$. This means that $2^{p_0} * 2^{p_1} = 2^p$ processors are available for working in parallel. In order to accept the distribution, the function UA is transformed:

$$\begin{aligned}&U2.X = UA^p.(distr^{ll}.p_0.p_1.(repR_1.n.X))\\&UA^p.A = red_1^p(*).(dif^p.(first.A).A)\end{aligned}$$

Time-complexity is $O((p_1 * 2^{n-p_0})\alpha + 2^{2n-p})$
Distribution costs In order to reduce the cost of data distribution, the number of distinct elements on each processor has to be as small as possible;
$\theta_d(distr^{ll}.p_0.p_1.A) = min\{2^n, 2^{n-p_0}+2^{m-p_1}-1\}$ But, this cost is very high, so we have to change the replication method. We can start from a distribution (one-dimensional $distr.p_0.X$) of the list X, then to replicate this distributed list based on $repR_1.p_0.\cdot$, and then to use replication based on $repR_1.(n-p_0).\cdot$ for each resulted sublist. After we apply the function $flat$ we obtain a matrix B that has the property that each column is a different permutation of the list X, and we may apply the same function UA to compute the products $u.x_i$.

If we apply a Cartesian distribution $distr^{ll}.p_1.p_0$, each resulted submatrix will have only 2^{n-p_0} distinct elements, if $p_1 \geq p_0$, and $2^{n-p_0} \times 2^{p_0-p_1}$ distinct elements if $p_1 < p_0$. The analysis of the time-complexity and the distribution cost leads to the conclusion that the best decomposition of p is $p = p_0 + p_1 = 2p_0, p_0 = p_1$, provided that p is even.

4. Conclusions

One-dimensional, multidimensional, and set-distributions could be defined for *PowerList* and *PowerArray* data structures, and functions defined on these special data structures are transformed to accept distributed input data, and then costs analysis could be done.

We have several advantages of formally introducing the distributions; the first is that it allows us to evaluate costs, depending on the number of available processors - as a parameter. In the *PowerArray* case, this is especially important, since we may choose the most advantageous factorization of the number of processors on each dimension. The analysis of the possible distributions for a certain function may lead to an improvement in the design decisions, too. Another advantage is that we may control the parallel decomposition until a certain level of tree decomposition is achieved; otherwise parallel decomposition could be done, for example, in a 'deep-first' manner, which could be disadvantageous. Also, after the introduction of the distributions functions, mapping on real architectures with limited number of processing elements (e.g. hypercubes) could be analyzed.

ACKNOWLEDGEMENT

This work was supported by the research project ID 2268/2009, sponsored by the Romanian National University Research Council (CNCSIS).

References

[1] Gorlatch, S.,: *Abstraction and Performance in the Design of Parallel Programs*, CMPP'98 First International Workshop on Constructive Methods for Parallel Programming, 1998.

[2] Kornerup, J.: *Data Structures for Parallel Recursion.* PhD Thesis, Univ. of Texas, 1997.

[3] Misra, J.: *PowerList: A structure for parallel recursion. ACM Transactions on Programming Languages and Systems*, Vol. 16 No.6 (1994) 1737-1767.

[4] Niculescu, V.: *On Data Distributions in the Construction of Parallel Programs*, The Journal of Supercomputing, Kluwer Academic Publishers, 29(1): 5-25, 2004.

[5] Niculescu, V.: "Data Distributions in *PowerList* Theory", *Lecture Notes of Computer Science Vol. 3722: Theoretical Aspects of Computing*, Proceedings of ICTAC 2007, Springer-Verlag, 2007: 396-409.

[1] Department of Computer-Science, Babes-Bolyai University, 1 M. Kogalniceanu, Cluj-Napoca, Romania

E-mail address: `vniculescu@cs.ubbcluj.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 231–234

SKEPTICAL REASONING IN CONSTRAINED DEFAULT LOGIC USING SEQUENT CALCULUS

MIHAIELA LUPEA(1)

ABSTRACT. Constrained default logic is a version of Reiter's classical default logic satisfying desirable formal properties as supraclassicality, semi-monotonicity, commitment to assumptions. In this paper we propose an axiomatic system called *skeptical constrained default sequent calculus*, based on sequent and anti-sequent calculi from classical logics. This system is used to formalize and study from theoretical point of view the skeptical nonmonotonic reasoning process modelled by constrained default logic.

1. INTRODUCTION

Constrained default logic [7], as a version of Reiter's default logic [6], models the nonmonotonic reasoning using special inference rules called *defaults*, to overcome the lack of information.

The defaults extend a given set of facts obtaining one or more sets called *extensions* which contain the *nonmonotonic consequences*(beliefs). The extensions represent possible belief sets of an agent reasoning about the initial theory. A *credulous reasoning perspective* means that an agents' beliefs belong to at least one extension. *Skeptical consequences* are more robust beliefs because they belong to all extensions of a theory. In the papers [2, 8] the properties of the nonmonotonic credulous/skeptical inference relations in default logics are studied.

Theoretical studies in the direction of the axiomatization of the credulous/ skeptical nonmonotonic inference process modelled by different versions of default logic are presented in the papers [1, 3, 5].

In this paper we propose an axiomatic system called *skeptical constrained default sequent calculus.* This system is used to formalize and study the skeptical nonmonotonic inference modelled by constrained default logic.

2. CONSTRAINED DEFAULT LOGIC

Definition 2.1. A *default theory* $\Delta = (D, W)$ contains a set W of consistent formulas (*facts*) of first order logic and a set D of *default rules.* A *default* has the

2000 *Mathematics Subject Classification.* 03B79, 68T15, 68T27, 68T20.
Key words and phrases. constrained default logic, skeptical default inference, sequent and anti-sequent calculi.

form $d = \frac{\alpha:\beta}{\gamma}$, where: α is called *prerequisite*, β is called *justification* and γ is called *consequent.* A default $d = \frac{\alpha:\beta}{\gamma}$ can be applied and thus derive γ if α is believed and "it is consistent to assumed β"(meaning that $\neg\beta$ is not believed).

From all versions of default logics, constrained default logic has the most desirable formal properties: supraclassicality, semi-monotonicity, commitment to assumptions, due to the global consistency condition for the applied defaults.

Definition 2.2.[4] Let D be a set of defaults. The set of *residues* and the set of *justifications* of D with respect to a set C of formulas are defined as follows: $Res_D^C = \left\{\frac{\alpha}{\gamma} | \frac{\alpha:\beta}{\gamma} \in D, C \cup \{\beta, \gamma\} \not\Rightarrow false\right\}$,
$Justif_D^C = \left\{\beta | \frac{\alpha:\beta}{\gamma} \in D, C \cup \{\beta, \gamma\} \not\Rightarrow false\right\}$.

The residues corresponding to the applied defaults are monotonic rules and are used to reduce the nonmonotonic reasoning process modelled by constrained default logic into a monotonic one according to the following theorem based on a theorem from [4].

Theorem 2.1. Let $\Delta = (D, W)$ be a default theory. (E, C) is a constrained extension of Δ if $E = Th^{res}(W, Res_D^C)$, $C = Th(Th^{res}(W, Res_D^C) \cup Justif_D^C)$, where $Th(\cdot)$ is the classical consequence operator and $Th^{res}(\cdot, R)$ is the consequence operator of the predicate formal system enhanced with the set R of residues.

E is the *actual extension* embedded in the *reasoning context* C. We remark that the nonmonotonic reasoning process modelled by constrained default logic is guided by a maximal consistent reasoning context.

3. Sequent and anti-sequent calculi for residues

The two complementary systems: sequent and anti-sequent calculi [1] for classical logics are enhanced with specific rules for residues [1, 5] preserving the soundness and completness of the new axiomatic systems. We use the same metasymbols $\Rightarrow$ and $\not\Rightarrow$ to express the inference relations in classical sequent/anti-sequent calculi and in the corresponding systems, enhanced with residues.

<u>*Sequent rules for residues:*</u> <u>*Anti-sequent rules for residues:*</u>

$(Re_1)\frac{\Gamma \Rightarrow \Psi}{\Gamma, \frac{\alpha}{\gamma} \Rightarrow \Psi}$ $(Re_2)\frac{\Gamma \Rightarrow \alpha \quad \Gamma, \gamma \Rightarrow \Psi}{\Gamma, \frac{\alpha}{\gamma} \Rightarrow \Psi}$ $(Re_3)\frac{\Gamma \not\Rightarrow \alpha \quad \Gamma \not\Rightarrow \Psi}{\Gamma, \frac{\alpha}{\gamma} \not\Rightarrow \Psi}$ $(Re_4)\frac{\Gamma, \gamma \not\Rightarrow \Psi}{\Gamma, \frac{\alpha}{\gamma} \not\Rightarrow \Psi}$

4. Axiomatization of skeptical reasoning in constrained logic

Definition 4.1. Let $\Delta = (D, W)$ be a default theory. A *skeptical constrained default sequent* has the *syntax*: $Constr; (W, D); Res \longmapsto U$. The set U of formulas is called *succedent.* The *antecedent* contains $Constr$(a set of constraints expressed using the modalities: M and L), the default theory (W, D) and Res (the set of residues corresponding to the applied defaults).

Definition 4.2. (semantics of a skeptical constrained default sequent)
The *skeptical constrained default sequent*: $Constr; (W, D); Res \longmapsto U$ *is true* if $\vee U$ belongs to all constrained default extensions of the theory (W, D), that satisfy the constraints $Constr$ and Res contains the residues of the applied defaults.

Definition 4.3. For a default theory Δ the *skeptical constrained default axiomatic system* is: $Sk_{\Delta}^{cons} = (\Sigma_{Sk_{\Delta}}^{cons}, F_{Sk_{\Delta}}^{cons}, A_{Sk_{\Delta}}^{cons}, R_{Sk_{\Delta}}^{cons})$.
$\Sigma_{Sk_{\Delta}}^{cons}$ - the alphabet, $F_{Sk_{\Delta}}^{cons}$= all classical sequents/anti-sequents enhanced with residues and all skeptical constrained default sequents.
$A_{Sk_{\Delta}}^{cons}$ - the axioms (all classical basic sequents and basic anti-sequents).
$R_{Sk_{\Delta}}^{cons} = \{S_1, S_2, S_3\} \cup \{Re_1, Re_2, Re_3, Re_4\} \cup \{$classical sequent/anti-sequent rules$\}$

Based on Theorem 2.1 we propose the *sequent rules for constrained default logic:*

$(S_1)\dfrac{Constr^M \cup W \not\Rightarrow false \qquad W \cup Res \Rightarrow U}{Constr; (W,D); Res \longmapsto U}$, where $Constr^M = \{\alpha | M\alpha \in Constr\}$

$$(S_2)\frac{Constr \cup \{M(\beta\wedge\gamma)\}; (W,D); Res \cup \left\{\frac{\alpha}{\gamma}\right\} \longmapsto U \qquad Constr \cup \{L\neg(\beta\wedge\gamma)\}; (W,D); Res \longmapsto U}{Constr; (W, D \cup \left\{\frac{\alpha:\beta}{\gamma}\right\}); Res \longmapsto U}$$

$$(S_3)\frac{W \cup \left\{\beta\wedge\gamma | \frac{\alpha:\beta}{\gamma} \in D\right\} \not\Rightarrow \delta}{Constr \cup \{L\delta\}; (W,D); Res \longmapsto U}$$

S_1 verifies if the reasoning context (obtained using the set of applied defaults) is consistent (left premise) and if $\vee U$ is derivable from the set of facts and the corresponding residues (right premise). Using S_2 we search exhaustively if $\vee U$ is a skeptical consequence of the theory Δ considering that $\frac{\alpha:\beta}{\gamma}$ is applied (left premise) and that the default $\frac{\alpha:\beta}{\gamma}$ is not applied (right premise). The rule S_3 checks if there are constrained extensions that satisfy the constraint $L\delta$.

Theorem 4.1. The skeptical constrained default sequent calculus is *sound and complete*: a skeptical constrained sequent is true if and only if it can be reduced to classical basic sequents and basic anti-sequents using $R_{Sk_{\Delta}}^{cons}$.

Consequence: A formula X is a skeptical constrained default consequence of the default theory (D, W) if and only if the skeptical constrained default sequent $\emptyset; (W, D); \emptyset \longmapsto X$ is true.

Example 4.1. The default theory $\Delta = (D, W)$, $D = \left\{\frac{Q:P}{S}, \frac{S:T}{T}, \frac{R:\neg P}{H}, \frac{H:T}{T}\right\}$, $W = \{Q, R\}$ has two constrained default extensions: $(E1, C1) = (Th(\{Q, R, S, T\}), Th(\{Q, R, S, T, P\}))$ and $(E2, C2) = (Th(\{Q, R, H, T\}), Th(\{Q, R, H, T, \neg P\}))$.

We check if T is a skeptical constrained default consequence of the theory Δ:

$$\frac{\Pi_1 : \{M(P\wedge S)\}; (\{Q,R\}, \{d2,d3,d4\}); \left\{\frac{Q}{S}\right\} \longmapsto T \qquad \Pi_2 : \{L\neg(P\wedge S)\}; (\{Q,R\}, \{d2,d3,d4\}); \emptyset \longmapsto T}{\emptyset; (\{Q,R\}, \left\{d1 = \frac{Q:P}{S}, d2 = \frac{S:T}{T}, d3 = \frac{R:\neg P}{H}, d4 = \frac{H:T}{T}\right\}); \emptyset \longmapsto T}(S_2)$$

By applying the rule (S_2), the default sequent Π_1 is reduced to Π_{11} and Π_{12}:

$$\frac{\Pi_{11} : \{M(P\wedge S), M\,T\}; (\{Q,R\}, \{d3,d4\}); \left\{\frac{Q}{S}, \frac{S}{T}\right\} \longmapsto T \quad \Pi_{12} : \{M(P\wedge S), L\neg T\}; (\{Q,R\}, \{d3,d4\}); \left\{\frac{Q}{S}\right\} \longmapsto T}{\Pi_1 : \{M(P\wedge S)\}; (\{Q,R\}, \left\{d2 = \frac{S:T}{T}, d3, d4\right\}); \left\{\frac{Q}{S}\right\} \longmapsto T}$$

The default sequent Π_{11} is a true default sequent as follows:

$$\frac{\dfrac{}{\{Q,R,P\wedge S,T\} \not\Rightarrow false} \qquad \dfrac{\vdots}{\left\{Q, R, \frac{Q}{S}, \frac{S}{T}\right\} \Rightarrow T}(Re2)}{\Pi_{11} : \{M(P\wedge S), M\,T\}; (\{Q,R\}, \{d3,d4\}); \left\{\frac{Q}{S}, \frac{S}{T}\right\} \longmapsto T}(S_1)$$

For reducing Π_{12} we apply the rule (S_3). No extension of the initial theory satisfies the constraint $L\neg T$, so the default sequent Π_{12} is reduced to a basic anti-sequent and therefore is a true sequent according to the following:

$$\frac{\overline{Q,R,T,H,\neg P \not\Rightarrow \neg T}}{\Pi_{12}:\{M(P\wedge S),L\neg T\};(\{Q,R\},\left\{d3=\frac{R:\neg P}{H},d4=\frac{H:T}{T}\right\});\left\{\frac{Q}{S}\right\}\longmapsto T}(S_3)$$

Π_2 can also be reduced to basic sequent/anti-sequents, therefore the initial sequent: $\emptyset;(\{Q,R\},\{d1,d2,d3,d4\});\emptyset \longmapsto T$ is true and thus T is a skeptical constrained default consequence of Δ.

5. Conclusions

In this paper we have introduced an axiomatic system which formalizes the skeptical nonmonotonic inference in constrained default logic. This system is based on sequent and anti-sequent calculi for classical logics and uses specific rules for the defaults. Using the monotonic rules called residues and constraints for the applied defaults, the default reasoning process becomes a monotonic one.

References

[1] Bonatti, P.A., Olivetti, N., "Sequent Calculi for Propositional Nonmonotonic Logics", ACM Trans. Comput. Log., 2002, pp. 226–278.

[2] Lupea, M., "Nonmonotonic inference operations for default logics", ECIT - Symposium on Knowledge-based Systems and Expert Systems, Iasi, Romania, 2002, pp. 1–12.

[3] Lupea, M., "Axiomatization of credulous reasoning in default logics using sequent calculus", 10-th International Symposium SYNASC 2008, IEEE Computer Society, pp. 49–55.

[4] Mikitiuk, A., Truszczynsky, M.,"Constrained and rational default logics", Proceedings of IJCAI-95, Morgan Kaufman, 1995, pp. 1509–1515.

[5] Milnikel, R.S., "Sequent calculi for skeptical reasoning in predicate default logic and other nonmonotonic logics", pp. 1-40, Kluwer, 2004.

[6] Reiter, R., "A Logic for Default reasoning", Artificial Intelligence **13**, 1980, pp. 81–132.

[7] Schaub, T.H., "Considerations on default logics", Ph.D. Thesis, Technischen Hochschule Darmstadt, Germany, 1992.

[8] Stalnaker, R.C., "What is a nonmonotonic consequence relation", Fundamenta Informaticae, 21, 1995, pp. 7–21.

[1] Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania

E-mail address: `lupea@cs.ubbcluj.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 235–238

ALGEBRAIC MODEL FOR THE SYNCHRONOUS SR-FLIP-FLOP BEHAVIOUR

ANCA VASILESCU[(1)]

Abstract. Considering the digital logic level of the computer architecture description, the agent-based approach is applied here to cover both the digital logic circuits specification and verification. We consider the specific structure for both the asynchronous and synchronous *SR* flip-flops. We define the appropriate specification and implementation agents for algebraic modelling the given circuits behaviour and we formally and automatically prove the corresponding bisimilarities between the target agents.

1. Introduction

This paper results represent the authors research area consisting in applying formal methods for modelling the hardware components behaviour and in obtaining an algebraic-based model for the entire computer system behaviour.

Having an algebraic-based model for a multi-agent system has the main advantage of increasing the level of confidence and reliability on the model following the formal methods based verification. For the specific case of a computer system, we have also the advantage of the opportunity of removing all the possible design errors before proceeding to the expensive component manufacturing and assembly.

In modern computer science, the process algebras are extensively used together with the appropriate verification tools such that the combination assures both the modelling of the computer components behaviour and the formal and/or automated verification of the proposed models. The most used such combinations are: SCCS/CWB, CSP/FDR2 and LOTOS/CADP [1, 8]. The SCCS as process algebra and CWB-NC [9] as verification workbench are used here in order to study the concurrent communicating processes involved in the computer hardware components behaviour and to model and verify the target hardware components behaviour.

2. Preliminaries

2.1. **Flip-flops. Computer memory organization.**

A *flip-flop* is a sequential circuit, a binary cell capable of storing one bit of information. It has two outputs, one

2000 *Mathematics Subject Classification.* 68Q85, 68N30, 68Q60, 93C62.
Key words and phrases. bisimulation equivalence, concurrency workbench, digital circuits, flip-flop, SCCS process algebra, specification, verification.

for the normal value and one for the complement value of the bit stored in it. A flip-flop maintains a binary state until it is directed by a clock pulse to change that state. At the different levels of detailing, the theoretical flip-flop might be asynchronous, but the synchronous models are widely used in practice.

2.2. **Process algebra SCCS.** The process algebra SCCS, namely *Synchronous Calculus of Communicating Systems* is derived from CCS [3, 4] especially for achieving the synchronous interaction in the framework of modelling the concurrent communicating processes. The operational semantics for SCCS is given via inference rules that define the transition available to SCSS processes. Combining the product and the restriction, SCCS calculus defines the synchronous interaction as a multi-way synchronization among processes.

A formal approach such as the process algebra SCCS supports a way to relate two different specifications in order to show that those specifications actually describe *equivalent* concurrent systems, for some specific meaning of *equivalence.* For each of the circuits we are interested in, we define a specification Spec which is based on the definition of the circuit and a specification Impl which is based on the behaviour and/or the properties of that given circuit. As demonstration technique, we start with these two specifications Spec and Impl and then we apply a set of SCCS-based algebraic laws in order to formally prove that the low-level specification, Impl, is correct with respect to the higher-level one, Spec. This correctness proof is based on the *bisimulation congruence*, the appropriate equivalence in the theory of concurrent communicating processes.

3. Algebraic model for the synchronous SR flip-flop behaviour

In this section we define both specification and implementation agents for asynchronous SR flip-flops and synchronous SR flip-flops. The core of our results consists in proving, both formally and automatically, the involved agents bisimilarities.

3.1. **The case of asynchronous SR flip-flop.** The specification Spec for the SR flip-flop behaviour might be [3]

$$\text{SpecSR}(m,n) = \sum_{i,j\in\{0,1\}} (\sigma_i\rho_j\overline{\gamma}_m\overline{\delta}_n : \text{SpecSR}(k,l)) \tag{1}$$

where the values k and l are defined by $k = i$ NOR n and $l = j$ NOR m.

In order to achieve the composition and to assure the fork of the output signal, we have to define two morphisms to make two appropriately relabelled copies of the NOR gate. We also define the set E of external actions, $E = \{\sigma_i, \rho_i, \gamma_i, \delta_i | i \in \{0,1\}\}$, in order to form the SCCS product between the two communicating NOR gate-agents.

The SCCS implementation for the SR flip-flop behaviour might be

$$\text{ImpSR}(m,n) = (\text{NOR}(m)[\Phi] \text{ x } \text{NOR}(n)[\Psi]) \upharpoonright E \tag{2}$$

The relation $\text{SpecSR}(m,n) \sim \text{ImpSR}(m,n)$ is established.

3.2. **The case of synchronous SR flip-flop.** From the structural point of view, in order to obtain the synchronous SR flip-flop we consider the asynchronous circuit and we add an extra level of AND gates for involving the clock signal.

We consider two levels for specifying the synchronous SR flip-flop behaviour, a specification and an implementation, and we conclude with the equivalence result for these models.

We propose the higher-level specification for the synchronous SR flip-flop behaviour based on the agents:

$$\text{(3)} \qquad \text{SpecSRs}(m,n,c) = (\text{SpecInput}(c) \times \text{SpecSR}(m,n)) \upharpoonright E\_SRmn(c)$$

where the set of external actions is $E\_SRmn(c) = \{CLK_c, \sigma, \rho, \gamma, \delta\}$ for a specific value of $c \in \{0,1\}$. Note that the parameters m and n have complemented values, by definition of the flip-flop.

We also propose the lower-level specification for the synchronous SR flip-flop behaviour based on the agents:

$$\text{(4)} \qquad \text{ImpSRs}(m,n,c) = (\text{ImpInput}(c) \times \text{ImpSR}(m,n)) \upharpoonright E\_SRmn(c)$$

It is for the interest of this paper results to prove the next equivalence.

Proposition 1. *The previous agents SpecSRs(m,n,c) and ImpSRs(m,n,c) for $(m,n) \in \{(0,1),(1,0)\}$ and $c \in \{0,1\}$ are bisimulation equivalent.*

The previous agents SpecSRs(m,n,c) and ImpSRs(m,n,c) are defined for all of the binary combinations on the input lines S and R of the flip-flop. In order to extend the algebraic model of the flip-flops behaviour to the algebraic model of the more complex sequential logic circuits based on flip-flops, it is useful to have the projection of such an agent on only one input binary combination (S,R). For achieving this target, we have to project each of the generic agents SpecSRs(m,n,c) and ImpSRs(m,n,c) on each of the binary combinations $(S,R) \in \{(0,0),(0,1),(1,0),(1,1)\}$. The final agents are SpecCBBSRs(m,n,S,R,c) and ImpCBBSRs(m,n,S,R,c) for each current state (m,n), input binary combination (S,R) and clock signal c. These agents are also bisimulation equivalent.

3.3. **Automatic verification.** In order to have an automatic verification for the bisimilarities between the corresponding specification and implementation agents already defined in the previous subsections, we have developed the SCCS files with respect to the CWB-NC platform syntax. For each of the Spec-Impl pair of agents, we have verified the appropriate bisimilarity. The corresponding CWB-NC answer is TRUE for all of these tests. We have also obtained successfull CWB-NC automatic verification for the corresponding implementation agents and for the bisimilarities tests.

4. Conclusions

Based on ideas from [3, 5], in this paper we have considered the internal structure of specific memory cells, namely SR flip-flops. As original contribution we have defined specific agents for modelling the concrete flip-flops behaviour for both the asynchronous and synchronous circuit organization and we have proved (formally and automatically) the appropriate agents bisimilarities.

The results of this paper are algebraic models for some important hardware components ready to be integrated in more complex, scalable structures. The components we focused on here are some of the most important hardware components for modern computer organization and design. After the modelling achievements from [6, 7], these modelling attempts represent a further step in the direction of having a global algebraic model of the entire computer system behaviour.

Implicitly, this result of bisimilarity is a guarantee of using these models in other complex circuits. An immediate extension of these results will consist in using these final agents into a more complex digital logic circuit like a standard read/write memory hardware component with input/output controller unit integrated.

References

[1] He J., *Formal Specification and Analysis of Digital Hardware Circuits in LOTOS*, TR CSM-158, August 2000.

[2] Mano M.M., *Digital Logic and Computer Design*, Prentice-Hall Of India Pvt Ltd, 2007.

[3] Milner R., *Calculi for synchrony and asynchrony*, Theoretical Computer Science, 25:267-310, 1983.

[4] Milner R., *Communication and concurrency*, Prentice Hall, 1989.

[5] Vasilescu A., *Formal models for non-sequential processes*, PhD Thesis in Computer Science, Babes-Bolyai University, Cluj-Napoca, 2004.

[6] Vasilescu A., Georgescu O., *Algebraic Model for the Counter Register Behaviour*, IJCCC - Supplem. Issue as Proc. of ICCCC2006, Oradea, Romania, Vol.I:459-464, 2006.

[7] Vasilescu A., *Algebraic model for the intercommunicating hardware components behaviour*, Proc. of the 12th Intl. Conf. on COMPUTERS, Heraklion, Greece, July 23-25, pp.241-246, 2008.

[8] WANG X. et al., *Opportunities and Challenges in Process-algebraic Verification of Asynchronous Circuit Designs*, ENTCS, Vol. 146(2):189-206, 26 Jan 2006, Proceedings of FMGALS 2005.

[9] *** The CWB-NC homepage on http://www.cs.sunysb.edu/~cwb.

(1) *Transilvania* University of Braşov, Faculty of Mathematics and Computer Science, Department of Theoretical Computer Science, Str. Iuliu Maniu nr. 50, 500091, Braşov, Romania

E-mail address: vasilex@unitbv.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 239–243

REVERSE ENGINEERING AND SIMULATION OF ACTIVE OBJECTS BEHAVIOR

DAN MIRCEA SUCIU$^{(1)}$

Abstract. Testing, debugging and understanding concurrent object-oriented programs are difficult tasks. Behavior models, especially statecharts, are appropriate tools for helping the development and maintenance of active objects. The paper presents a process of generating statecharts at runtime and a tool that assists and supports the main steps of this process.

1. Introduction

Today CASE tools include reverse engineering modules which are able to generate static diagrams from structures defined in application's source code. Dynamic diagrams cannot be generated just from the source code. That is because object-oriented languages does not have built-in elements that appear in dynamic diagrams and because the objects behavior could be much clearly observed during runtime. These are the main reasons why we have searched solutions on dynamically generating statecharts. We are proposing a process of extracting statecharts that model the behavior of active objects, based on the formalism described in [4].

2. Process of reversely generating statecharts

Figure 1 illustrates our proposed process of automatically generating statecharts at runtime. We will describe each step of the process in detail.

Step 0. Code generation and/or programming. This is an optional step, because we will take in consideration generating statecharts from any source code (no matter if it was manually implemented or automatically generated). Nevertheless, in absence of complex formal adnotations of dynamic models, the source code cannot be entirely generated automatically.

2000 *Mathematics Subject Classification.* 68N30.
Key words and phrases. statecharts, dynamic modeling, reverse engineering.

Step 1. Source code instrumentation. Source code instrumentation is the process of adding new program logic into the source code of a given program. This new logic is functionally neutral and it is used for some analytic purposes. In our process, code instrumentation is used to log information about the object identity and its attributes values before exiting a constructor or a modifier method.

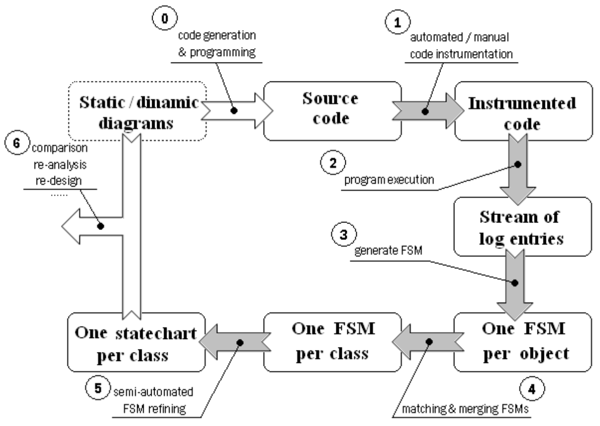


FIGURE 1. Steps of generating statecharts using ActiveCASE components

Step 2. Program execution.

During the program execution, information about current state of active objects are saved on disk, in specific log files, or are directly sent as a stream to the statechart generator component.

Step 3. Generate Finite State Machines (FSMs). During this step, FSMs are generated based on information received from active objects. The method of generating states is straightforward: for each new distinct tuple of attribute values, a new state is generated. If it doesn't exists yet, a new transition is automatically created between the states corresponding to previous and current value tuples.

Step 4. Matching and merging FSMs. At the end of step 3 we obtain a set of FSMs generated for the same class. Usually, for active objects having a complex behavior, the result consists of partial FSMs. In such situations it is necessary to create a consolidated FSM, containing all the matching states and transitions, together with all discovered "exceptions" in the generated FSMs.

Step 5. Semi-automated FSM refining. During this step the FSMs could be refined and transformed in statecharts [3] respecting the states aggregation and generalization. The obtained result is a more compact and/or readable statechart.

Step 6. Comparison, re-analysis, re-design... This step is important for verification and testing of software conformance to specification, analysis and design models.

3. A case study

ActiveCASE is a tool developed for designing, implementing and simulating active objects behavior in concurrent object oriented applications [5]. We have extended it to offer support for first 4 steps of the process of generating statecharts.

In order to illustrate how FSMs are generated in ActiveCASE we will use the same motivating example used in [5]: traffic simulation in a matrix of tracks. The goal is to generate the corresponding FSMs for *Car* class using ActiveCASE components. According to step 0, a statechart is defined for *Car* class, as described in [5].

The code instrumentation (step 1) has been done manually. First of all, the *ActiveObject* class was enriched with two specific methods: *GetCurrentStateToString()* and *GenerateCurrentState()*. First method returns a string with information regarding the current state, whith the following configuration: [*Class:* $< ClassName >$ $; Object :< ObjectIdentity >; Method :< MethodName >$; $attr_1$; ... $attr_n$], where $attr_1; ...attr_n$ is a list of actual attribute values. Each subclass of *ActiveObject* class should rewrite this function in order to add its own specific attribute values. The second method,*GenerateCurrentString()*, saves the current state in a log file on disk or, if the simulation is activated, sends the state strings to the simulator component.

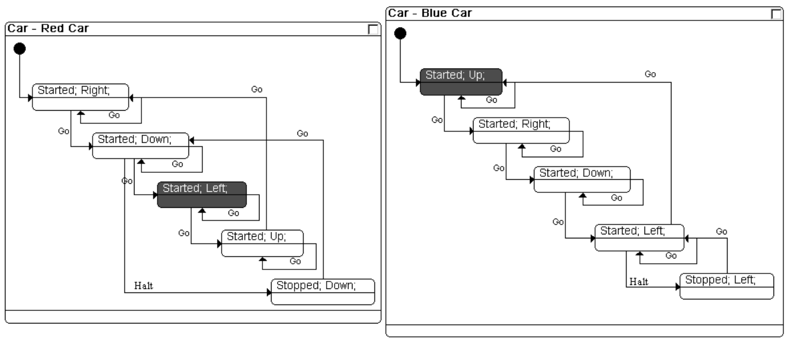


FIGURE 2. FSMs generated for "Red Car" and "Blue Car" objects

Figure 2 shows two FSMs generated for two *Car* objects (called "Blue Car" and "Red Car"). For generating these FSMs, only three *Car* methods were instrumented: the constructor, *Halt* and *Go* methods.

The analysis of the result reveals the fact that the FSMs are not complete, each one having a state that was not generated for the other (state "Stopped; Down;" in "Red Car" FSM, and state "Stopped; Left;" generated for "Blue Car" FSM).

This observation leads us to the idea that merging several FSMs we can obtain a more complete view of the behavior of a particular class of objects.

4. Related work

There are papers that refer to automated generation of statecharts based on scenarios description and their sequence diagram [1], [8], [9], [10]. In [7] the sequence diagrams are extracted from event trace information, generated as a result of running the target software under a debugger. Sometimes, the sequence diagrams are generated from source code, using code instrumentation [6]. Code instrumentation for generating statecharts is also used in [2]. Here the instrumentation is more complex and requires a good understanding of the analyzed source code.

5. Future work and conclusions

Future work consists in continuing the extension of ActiveCASE tool to support steps 4 and 5 of the statechart generator module. Besides it, the automatization of the previous steps of the process, mainly the source code instrumentation task, could improve the usability of the simulation component.

References

[1] H. Chu, Q. Li, S. Hu, P. Chen, "An Approach for Reversely Generating Hierarchical UML Statechart Diagrams", Lecture Notes in Computer Science : Fuzzy Systems and Knowledge Discovery, pp. 434-437, 2006

[2] A. Gupta, "Automated Object's Statechart Generation and Testing from Class-Method Contracts", In 3rd IntlWorkshop on Model Development, Validation, and Verification (MoDeV2a-06) co-located with 9th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp 30-45, Genova, Italy, 2006.

[3] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, vol.8, no. 3, pp. 231-274, June 1987

[4] D. M. Suciu, "Using Scalable Statecharts for Active Objects Internal Concurrency Modeling", Studia Universitatis "Babes-Bolyai" Cluj-Napoca, Series Informatica, Vol. XLV, Nr. 2, 2000, pp. 67-76

[5] D. M. Suciu, "ActiveCASE - Tool for Design and Simulation of Concurrent Object-Oriented Applications", Studia Universitatis "Babes-Bolyai", Series Informatica, Vol. XLVI, Nr. 2, pp. 73-80, 2001

[6] T. Syst, K. Koskimies, "Extracting State Diagrams from Legacy Systems", Lecture Notes In Computer Science; Vol. 1357, Proceedings of the Workshops on Object-Oriented Technology, pp 262-273, 1997

[7] T. Systa, "Dynamic reverse engineering of Java software", Proceedings of the Workshop on Object-Oriented Technology, Lecture Notes In Computer Science Vol. 1743, pp 174 - 175, 1999

[8] S. Vasilache, J. Tanaka "Synthesis of State Machines from Multiple Interrelated Scenarios Using Dependency Diagrams", 2004

[9] J. Whittle, J. Schumann, "Generating Statechart Designs from Scenarios", International Conference on Software Engineering, Proceedings of the 22nd international conference on Software engineering, Ireland, pp 314 - 323, 2000

[10] T. Ziadi, L. Helouet, J.M. Jezequel, "Revisiting Statechart Synthesis with an Algebraic Approach",International Conference on Software Engineering, Proceedings of the 26th International Conference on Software Engineering, pp 242 - 251, 2004

(1) Department of Computer Science, "Babeş-Bolyai" University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

E-mail address: tzutzu@cs.ubbcluj.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 244–248

A PARALLELIZATION SCHEME OF SOME ALGORITHMS

ERNEST SCHEIBER (1)

Abstract. This note presents a parallelizing scheme of an algorithm with an unknown number of tasks or dynamicaly varing number of tasks but using a fixed number of workers or workstations, based on the dispatcher-worker paradigm. A consequence of this approach is that the same scheme is used to implement, as an example, the backtracking algorithm and the quicksort method.

1. Introduction

The purpose of this note is to present a parallelizing scheme for an algorithm involving an a priori unknown number of tasks or dynamically varying number of tasks but using a fixed number of workers or workstations.

Backtracking, quicksort are examples of such algorithms. There were many attempts to develop parallel versions for the backtracking algorithm [4] and for the quicksort method [3, 5].

Our scheme is based on the dispatcher-worker paradigm and it focuses on the data management and the coordination of the worker processes. A consequence of this approach is that the same scheme is used to implement, as an example, the backtracking algorithm and the quicksort method. The specific parts of the solving methods are embedded in the dispatcher and in the workers activities.

2. The parallelizing scheme

As we mention, the scheme is based on the dispatcher - workers model. The dispatcher manages a repository for the data that will be sent to the workers when they will be available to proceed their specific job. A stack may be used as the repository.

The dispatcher and the workers communicate using messages. There are two kind of messages received by a worker:

2000 *Mathematics Subject Classification.* 68W15.
Key words and phrases. parallel algorithms, dispatcher-worker, parallel backtracking.

- *ordinary message* containing the data for the worker to solve a specific task;
- *ending type message* - command to the worker to finish its activity.

The worker actions are given in Algorithm 1.

Algorithm 1 The workers procedure.

```
procedure WORKER
    ending-flag ← false
    repeat
        Receive a message
        if ending-type message then
            ending-flag ← true
        else
            Execute the specific actions of the worker
            Send the response to the dispatcher
        end if
    until ending-flag = true
end procedure
```

After receiving a response, if further processes of the response are required, the dispatcher will push the corresponding data for future tasks in the repository. While there are available workers, the dispatcher extracts records from the repository, creates corresponding ordinary messages and sends them to the workers. To each worker there is associated a flag indicating the state of the worker (occupied or free). The dispatcher counts the number of sent and received messages to the workers. The dispatcher actions are represented in the Algorithm 2.

The available number of workers is denoted by *size*. The receive command is supposed to be blocking. The parallelization occurs in moment when the dispacher send messages to the unoccupied workers - lines 19-24.

Message Passing Interface (MPI) [1, 2, 6] may be used to implement the scheme, but other frameworks can be used, too according [7].

3. Examples

Using the above presented parallelizing scheme, we give details of the implementation of the n-queen problem, the sixteen grid problem via backtracking and the quicksort method.

The n queen problem. A message is an instance of a class containing the rank of the source, a tag and two sequences. The first sequence contains the data sent by the dispatcher to the worker(the j-th element of the sequence represents the column occupied by a queen in the j-th raw), while the second sequence is the result computed by the worker.

Algorithm 2 The dispatcher procedure.

```
procedure DISPATCHER
    Data structures: $Stack\ repository$;
    boolean[ ]$freeWorker_{1\le i\le size}$
    $send\_messages\_number \leftarrow 0$
    $received\_messages\_number \leftarrow 0$
    for $i = 1 : size$ do
        $freeWorker_i \leftarrow$ true
    end for
    Specific initializations of the application
    Send an ordinary-message to the $Worker_1$
    $freeWorker_1 \leftarrow$ false
    $send\_messages\_number \leftarrow send\_messages\_number + 1$
    while $repository$ is not empty or
      $send\_messages\_number \neq received\_messages\_number$ do
        if $send\_messages\_number \neq received\_messages\_number$ then
            Receive a response message
            $received\_messages\_number \leftarrow received\_messages\_number + 1$
            $freeWorker_{sender-rank} \leftarrow$ true
            Process the received message
        end if
        while exists free Workers and the $repository$ is not empty do
            Extracts an object from the $repository$
            Sends an ordinary-message to a free Worker
            $freeWorker_{receiver-rank} \leftarrow$ false
            $send\_messages\_number \leftarrow send\_messages\_number + 1$
        end while
    end while
    for $i = 1 : size$ do
        Sends Ending-type-message to the $Worker_i$
    end for
end procedure
```

If the worker receives the sequence $[x_0, x_1, \dots, x_{k-1}]$ $(k-1 < n)$ with $x_p \in \{0, 1, \dots, n-1\}$ and

$$x_p \neq x_q \qquad \text{and} \qquad |x_p - x_q| \neq |p - q| \qquad \forall p, q \in \{0, 1, \dots, k-1\}, \tag{1}$$

then it executes the following operations:

```
for any $i \in \{0, 1, \dots, n-1\}$ do
    It verifies the validity conditions (1) of the sequence $[x_0, x_1, \dots, x_{k-1}, i]$.
```

```
3:     If the validity conditions are satisfied then $i$ is appended to the sequence of
       results.
4:     Builds a message with the results and sends it to the dispatcher.
5: end for
```

Receiving a responce message, the dispatcher verifies whether some solutions are obtained (i.e. the length of a generated sequence is n). If the generated sequences are not solutions then the dispatcher push them into the repository.

As initialization, the dispatcher sends a message with the sequence [0] to the first worker, while the sequences $[i],\ i \in \{1, \ldots, n-1\}$ are push into the repository.

The quicksort method. The message class contains the rank of the source, a tag, an index, a splitting index and a sequence.

Let us suppose that at a stage of the quicksort method, the sequence to be sorted is $[x_0, x_1, \ldots, x_{n-1}]$. The index of a subsequence $[x_p, x_{p+1}, \ldots, x_q]$ is p. Let us suppose that the dispatcher sends this subsequence to a worker. The worker split the received sequence as it is done in the standard quicksort method. If $[y_0, y_1, \ldots, y_{q-p}]$ is the modified sequence and r is the splitting index, then

$$y_i < y_r, \quad \forall i < r \qquad \text{and} \quad y_j \geq y_r \quad \forall j \geq r.$$

This sequence will be returned by the worker.

Receiving a response message, the dispatcher update the sequence to be sorted and if the length of a splited subsequence is greather than 2, then the parameters of subsequence (the starting index and the length) are pushed into the repository.

References

[1] Z. BAOYIN, 2005, "Jcluster A Java Parallel Environment." Docs distributed with the software, version 1.0.5.

[2] R. BISSELING, 2004, *Parallel Scientific Computation. A structured approach using BSP and MPI,* Oxford Univ. Press.

[3] CHENGYAN ZH., 1996, *Parallel Quicksort algorithm with PVM optimization.* Project report CS62025. Univ. NewBrunswick, Fac. Computer Science, Fredericton.

[4] HAMADI Y., BESSIÈRE CH., QUINQUETON J., 1998, *Backtracking in Distributed Networks.* http://research.microsoft.com/~youssefh/Papers/ecai98.pdf.

[5] TSIGAS PH., ZHANG Y., 2003, *A Simple Fast Parallel Implementation of Quicksort and its Performance Evaluation on SUN Enterprise 10000.* http://www.es.chalmers.se/~tsigas/papers/Pquick.pdf.

[6] SNIR M., OTTO D., HUSS-LEDERMAN S., WALKER D., DONGARRA J., 1996, *MPI: The Complete Reference.* MIT Press, Cambridge, MA.

[7] SCHEIBER E., 2007, *A TSpaces Based Framework for Parallel - Distributed Applications.* Knowledge Engineering Principles and Techniques 1 (2007), Cluj University Press, 341-345.

[(1)] Faculty of Mathematics and Computer Science, "Transilvania" University of Braşov, 50 Iuliu Maniu St., Braşov, Romania

E-mail address: `scheiber@unitbv.ro`

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 249–253

THE MULTI-OBJECTIVE REFACTORING SELECTION PROBLEM

CAMELIA CHISĂLIŢĂ–CREŢU[(1)] AND ANDREEA VESCAN[(1)]

ABSTRACT. The paper defines the Optimal Refactoring Selection Problem (ORSP) and generalizes it as a Multi-Objective ORSP (MOORSP) by treating the *cost* constraint as an objective and combining it with the *effect* objective. It considers the refactoring selection as a multi-objective optimization in the Search-Based Software Engineering (SBSE) field. The results of the proposed weighted objective genetic algorithm on a experimental didactic case study are presented and compared with other recommended solutions for similar problems.

1. Introduction

Software systems continually change as they evolve to reflect new requirements, but their internal structure tends to decay. Refactoring is a commonly accepted technique to improve the structure of object oriented software [3]. The Optimal Refactoring Selection Problem (ORSP) is the identification problem of the optimal refactorings that may be applied on software entities, such that several objectives are kept or improved.

ORSP is an example of a Feature Subset Selection (FSS) search problem in SBSE field. The paper introduces a first formal version definition of the MOORSP and performs a proposed weighted objective genetic algorithm on a experimental didactic case study. Obtained results for our case study are presented and compared with other recommended solutions for similar problems [5].

The rest of the paper is organized as follows: Section 2 presents the formal definition of the studied problem. The proposed approach and the genetic algorithm with several details related to the genetic operators are described in the Section 3. Local Area Network simulation source code was used in order to validate our approach. The paper ends with conclusions and future work.

2. Optimal Refactoring Selection Problem

In order to state the ORSP some notion and characteristics have to be defined. Let $SE = \{e_1, \ldots, e_m\}$ be a set of software entities, i.e., a class, an attribute from a class, a method from a class, a formal parameter from a method or a local variable declared in the implementation of a method. The weight associated with each software entity $e_i, 1 \leq i \leq m$ is kept by the set $Weight = \{w_1, \ldots, w_m\}$, where $w_i \in [0, 1]$ and $\sum_{i=1}^{m} w_i = 1$. A software system SS consists of a software entity set SE together with different types of dependencies between the contained items.

2000 *Mathematics Subject Classification.* 68T20, 68N99.
Key words and phrases. searched-based software engineering, multi-objective optimization, refactoring.

A set of possible relevant chosen refactorings [3] that may be applied to different types of software entities of SE is gathered up through $SR = \{r_1, \ldots, r_t\}$. There are various dependencies between such transformations when they are applied to the same software entity, a mapping emphasizing them being defined by:

$$rd : SR \times SR \times SE \to \{\mathbf{B}efore, \mathbf{A}fter, \mathbf{A}lways\mathbf{B}efore, \mathbf{A}lways\mathbf{A}fter, \mathbf{N}ever, \mathbf{W}henever\},$$

$$rd(r_h, r_l, e_i) = \begin{cases} B, & \text{if } r_h \text{ may be applied to } e_i \text{ only before } r_l, \text{ i.e., } r_h < r_l \\ A, & \text{if } r_h \text{ may be applied to } e_i \text{ only after } r_l, \text{ i.e., } r_h > r_l \\ AB, & \text{if } r_h \text{ and } r_l \text{ are both applied to } e_i \text{ then } r_h < r_l \\ AA, & \text{if } r_h \text{ and } r_l \text{ are both applied to } e_i \text{ then } r_h > r_l \\ N, & \text{if } r_h \text{ and } r_l \text{ cannot be both applied to } e_i \\ W, & \text{otherwise, i.e., } r_h \text{ and } r_l \text{ may be both applied to } e_i \text{ whenever} \end{cases},$$

where $1 \leq h, l \leq t,\ 1 \leq i \leq m$.

The effort involved by each transformation is converted to cost, described by the following function: $rc : SR \times SE \to Z$,

$$rc(r_l, e_i) = \begin{cases} > 0, & \text{if } r_l \text{ may be applied to } e_i \\ = 0, & \text{otherwise} \end{cases},$$

where $1 \leq l \leq t$, $1 \leq i \leq m$.

Changes made to each software entity $e_i, i = \overline{1,m}$ by applying the refactoring $r_l, 1 \leq l \leq t$ are stated and a mapping is defined: $effect : SR \times SE \to Z$,

$$effect(r_l, e_i) = \begin{cases} > 0, & \text{if } r_l \text{ is applied to } e_i \text{ and has the requested effect on it} \\ < 0, & \text{if } r_l \text{ is applied to } e_i \text{ and has } \textit{not} \text{ the requested effect on it} \\ = 0, & \text{otherwise} \end{cases},$$

where $1 \leq l \leq t$, $1 \leq i \leq m$.

The overall effect of applying a refactoring $r_l, 1 \leq l \leq t$ to each software entity $e_i, i = \overline{1,m}$ is defined as: $res : SR \to Z$, $res(r_l) = \sum_{i=1}^{m} w_i * effect(e_i, r_l)$, where $1 \leq l \leq t$.

Each refactoring $r_l, l = \overline{1,t}$ may be applied to a subset of software entities, defined as a function:

$$re : SR \to P(SE),\ re(r_l) = \{\ e_{l_1}, \ldots, e_{l_q} \mid \text{if } r_l \text{ is applicable to } e_{l_u},\ 1 \leq u \leq q,\ 1 \leq q \leq m\ \},$$

where $re(r_l) = SE_{r_l}, SE_{r_l} \subseteq SE - \phi$, $1 \leq l \leq t$.

The goal is to find a subset of refactorings $RSet$ such that for each entity $e_i, i = \overline{1,m}$ there is at least a refactoring $r \in RSet$ that may be applied to it, i.e., $e_i \in SE_r$. Thus, ORSP is the identification problem of the optimal refactorings that may be applied to software entities such that several objectives are kept or improved, like the minimum total cost and the maximum overall effect on the affected software entities.

There are several ways to deal with a multi-objective optimization problem. In this paper the weighted sum method [4] is used.

3. Proposed approach description

The approach presented in this paper uses principles of evolutionary computation and multi-objective optimization. First, the problem is formulated as a multiple objective optimization problem having two objectives: the total cost of applying the refactorings (i.e., rc function) and the overall effect of applying the refactorings (i.e., res function). Because the cost function (f_1) should be minimized and the res function (f_2) should be maximized, we have modified the value of the cost into

MAX minus the real cost, where MAX is the biggest possible cost and the real cost values being less than MAX . Thus, the new function obtained by aggregating the two objectives can be written as: $F(x) = \alpha \cdot f_1(x) + (1 - \alpha) \cdot f_2(x), \alpha \in [0, 1]$.

The goal is to identify those solutions that compromise the refactorings costs and the overall impact on transformed entities. The decision vector $\overrightarrow{r} = (r_1, \ldots, r_m)$, $r_i \in SR, 1 \leq i \leq m$ determines the refactorings that may by applied in order to transform the considered set of software entities SE . The item r_i on the i-th position of the solution vector represents the refactoring that may be applied to the i-th software entity from SE , where $e_i \in SE_{r_i}, 1 \leq i \leq m$.

The algorithm proposed was applied on a simplified version of the Local Area Network (LAN) simulation source code that was presented in [1]. It contains 5 classes with 5 attributes and 13 methods, constructors included. The current version of the source code lacks of hiding information for attributes since they are directly accessed by clients. The abstraction level and clarity may be increased by creating a new superclass for `PrintServer` and `FileServer` classes, and populate it by moving up methods in the class hierarchy.

Thus, for the studied problem the software entity set is defined as: $SE = \{c_1, ..., c_5, a_1, ..., a_5, m_1, ..., m_{13}\}$. The chosen refactorings that may be applied are: *renameMethod, extractSuperClass, pullUpMethod, moveMethod, encapsulateField, addParameter*, denoted by the set $SR = \{r_1, \ldots, r_6\}$ in the following. The dependency relationship between refactorings is defined as $\{(r_1, r_3) = B,\ (r_1, r_6) = AA,\ (r_2, r_3) = B,\ (r_3, r_1) = A,\ (r_6, r_1) = AB,\ (r_3, r_2) = A,\ (r_1, r_1) = N,\ (r_2, r_2) = N,\ (r_3, r_3) = N,\ (r_4, r_4) = N,\ (r_5, r_5) = N,\ (r_6, r_6) = N\}$.

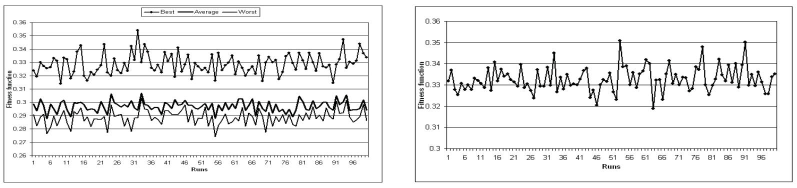
The values of the final effect were computed for each refactoring, but using the weight for each existing and possible affected software entity, as it was defined in Section 2. Therefore, the values of the res function for each refactoring are: 0.4, 0.49, 0.63, 0.56, 0.8, 0.2.

Here, the cost mapping rc is computed as the number of the needed transformations, so related entities may have different costs for the same refactoring. Each software entity has a weight within the entire system, but $\sum_{i=1}^{23} w_i = 1$. For $effect$ mapping, values were considered to be numerical data, denoting estimated impact of refactoring applying. Due to the space limitation, intermediate data for these mappings was not included. An acceptable solution denotes lower costs and higher effects on transformed entities both objectives being satisfied.

The parameters used by the evolutionary approach are as follows: mutation probability 0.7 and crossover probability 0.7. Different number of generations and of individuals are used: number of generations 10, 50, 100, and 200, and number of individuals 20, 100, 200. Current paper studies the aggregated fitness function where objectives have equal weights, i.e., $\alpha = 0.5$.

3.1. **Results obtained by the Evolutionary approach.** In order to compare data having different domain values the normalization is applied firstly. We have used two methods to normalize the data: decimal scaling for the rc function and min-max normalization for the value of the res function. The algorithm was run 100 times and the best, worse and average fitness values were recorded. Figure 1(a) presents the evolution of the fitness function (best, worse and average) recorded for each run within 20 chromosomes populations and 10 generations.

The evolution of fitness function (recorded for the best individual in each run) for 200 *generations* and 100 *individuals* is depicted in Figure 1(b).



(a) Experiment with 10 generations and 20 individuals with eleven mutated genes

(b) Experiment with 200 generations and 100 individuals

FIGURE 1. The evolution of fitness function (best, worse and average) and the best individual evolution

While compared with the previous experiments we noted that we are getting a lower number of different solutions while cumulating the results obtained in all the 100 runs, but the quality (and the number) of these solutions is improving much more. In a first experiment the greatest value of the fitness function is 0.3508 (with 69 individuals with the fitness > 0.33) while in a second experiment this is not more than 0.3536 (55 individuals with the fitness > 0.33). But the best chromosome was found in the experiment with 200 *generations* and 20 *individuals* having the value 0.3562.

The best individual obtained allows to improve the structure of the class hierarchy. Therefore, a new class is added as base class for `PrintServer` and `FileServer` and the `print` method is renamed, its signature is changed and it is moved to the new `Server` class. The correct access to the class fields by encapsulating them within their classes is enabled.

3.2. **Results obtained by others.** Fatiregun et al. [2] applied genetic algorithms to identify transformation sequences for a simple source code, with 5 transformation array, whilst we have applied 6 distinct refactorings to 23 entities. Seng et al. [6] apply a weighted multi-objective search, in which metrics are combined into a single objective function. An heterogeneous weighed approach is applied here, since the weight of software entities in the overall system and refactorings cost are studied. Mens et al. [5] propose the techniques to detect the implicit dependencies between refactorings. Their analysis helped to identify which refactorings are most suitable to LAN simulation case study. Our approach considers all relevant applying of the studied refactorings to all entities.

4. Conclusions and Future work

The paper defines the MOORSP by treating the *cost* constraint as an objective and combining it with the *effect* objective. The results of a proposed weighted objective genetic algorithm on a experimental didactic case study are presented and compared with other recommended solutions for the similar problems.

Current paper discusses the weighted multi-objective optimization, but the Pareto approach is a further step in current research since it proves to be more suitable when it is difficult to combine several objectives into a single aggregated fitness function. More, the cost may be interpreted as a constraint, with the further consequences.

Acknowledgement. This material is based upon work supported by the Romanian National University Research Council under award PN-II no. ID_550/2007.

References

[1] S. Demeyer, D. Janssens, T. Mens, *Simulation of a LAN*, Electronic Notes in Theoretical Computer Science, 72 (2002), pp. 34-56.

[2] D. Fatiregun, M. Harman, R. Hierons, *Evolving transformation sequences using genetic algorithms*, in 4th International Workshop on Source Code Analysis and Manipulation (SCAM 04), Los Alamitos, California, USA, IEEE Computer Society Press, 2004, pp. 65-74.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Software*. Addison Wesley, 1999.

[4] Y. Kim, O.L. deWeck, *Adaptive weighted-sum method for bi-objective optimization: Pareto front generation*, in Structural and Multidisciplinary Optimization, MIT Strategic Engineering Publications, 29(2), 2005, pp. 149-158.

[5] T. Mens, G. Taentzer, O. Runge, *Analysing refactoring dependencies using graph transformation*, Software and System Modeling, 6(3), 2007, pp. 269-285.

[6] O. Seng, J. Stammel, D. Burkhart, *Search-based determination of refactorings for improving the class structure of object-oriented systems*, in Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, M. Keijzer, M. Cattolico, eds., vol. 2, ACM Press, Seattle, Washington, USA, 2006, pp. 1909-1916.

(1) Computer Science Department, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: cretu@cs.ubbcluj.ro, avescan@cs.ubbcluj.ro

KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 254–257

VIRTUAL REALITY REHABILITATION ENVIRONMENT FOR OBSESSIVE–COMPULSIVE DISORDER

SORIN JIBOTEAN[(1)] AND RARES FLORIN BOIAN[(2)]

ABSTRACT. Medical rehabilitation often employs virtual environments due to their flexibility, safety and low costs. The applications of virtual reality in rehabilitation ranges from post-surgery or stroke therapy to ADHD or phobias. The paper presents a virtual environment developed to train obsessive-compulsive disorder (OCD) patients to focus on the task at hand without trying to first arrange everything to a perceived perfect pattern. The application is based on reports of high school results and exam behavior of otherwise good students who suffer from OCD.

1. Introduction and Approach

Medical rehabilitation often employs virtual environments due to their flexibility, safety and low costs. The applications of virtual reality in rehabilitation ranges from post-surgery or stroke therapy [1] to attention deficit and hyperactivity disorder (ADHD) [2, 3] or phobias treatments.

Patient suffering of obsessive-compulsive disorder display symptoms such as: fear of contamination or dirt, having things orderly and symmetrical, or recurrent and persistent thoughts and impulses [4, 5]. Based on reports from the Pediatric Psychiatric Hospital in Cluj–Napoca, high school students suffering on OCD, often fail exams due their focus on arranging things in a perfect pattern instead of focusing on the test or exam subjects.

The application develop is aimed to training such students to focus on the exam subjects while ignoring the lack of pattern of the environment. Following the exposure therapy approach, the virtual reality application, requires the patient to take an exam in a virtual classroom.

2. Related Work

The applications of virtual reality in the medical field is a very active and extensively investigates subject. The closest work to that present in this paper has been done by Rizzo et al. [2] with the Virtual Classroom project for studying ADHD in children. Their virtual environment places the patient in a classroom and presents

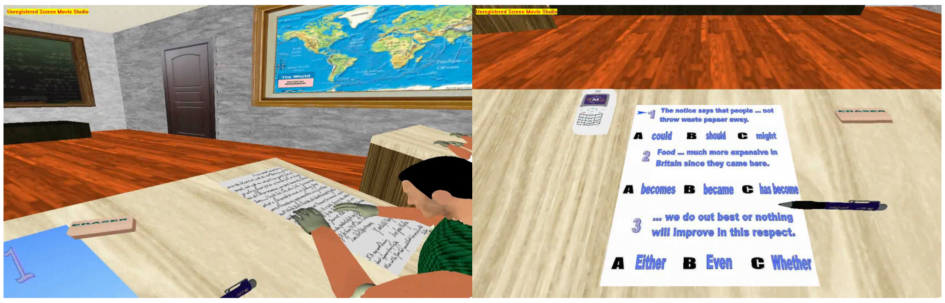
2000 *Mathematics Subject Classification.* 68U05, 68U35, 68U01.
Key words and phrases. virtual reality, virtual rehabilitation, exposure therapy.

him/her with a lesson during which several distractions are played out to test the child's reactions.

Our work differs from the Virtual Classroom in that the patient is required to complete a real test in the virtual environment. Also, the distractions in our application differ in nature from those of Rizzo's virtual classroom

3. Virtual Environment

The virtual reality application developed for OCD treatment was implemented using JOGL, a Java wrapper around the OpenGL library [6, 7]. The environment consists of a virtual classroom in which the patient is positioned sitting at a desk between two other students (see Figure 1a). The patient has a test in front of him and is required to answer the questions in a certain time limit. The patient can interact with the environment using a joystick. He/she can change the view direction or grab objects on the table and reposition them as desired, or answer the test questions.


Unregistered Screen Movie Studio
Unregistered Screen Movie Studio
1 The notice says that people ... not throw waste paper away.
A could B should C might
2 Food ... much more expensive in Britain since they came here.
A becomes B became C has become
3 ... we do out best or nothing will improve in this respect.
A Either B Even C Whether

(A) (B)

Figure 1. Virtual environment. (a) Classroom; (b) Quiz

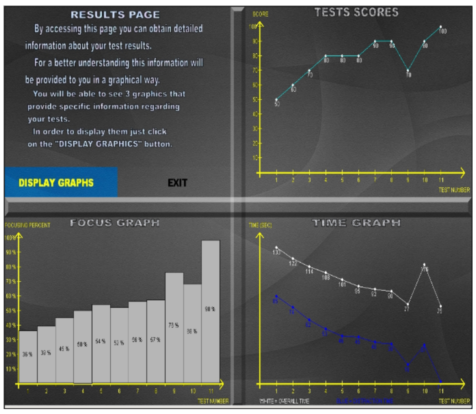
The patient's virtual belongings (pencil, eraser, and cell phone) are placed randomly around the test paper, so they would bother the patient and trigger the need for a pattern. While the patient can grab and move or rotate the objects as desired, they application intelligently replaces them after a certain time, thus breaking the pattern created by the patient. The replacement is done with a slow progressive motion. The level at which the application changes the positions of the objects is controllable, thus allowing the therapist to choose the desired level of distraction for each patient independently.

To add realism to the scene, the neighboring students and the professor at the desk are controlled by the application to move their bodies (mainly the head) in a pattern suggesting reading.

4. Measurements and Visualization

To provide the therapist and the patient to perform a quantitative evaluation of the therapy session, the virtual environment stores in an Microsoft Access database the score of the test, the duration it took the patient to complete the test, and percentage of focus on the test out of the entire time spent in the virtual environment (see Figure 2).

The percentage of focus is calculated using the position of the interaction point (displayed as a cross-hair). The time when the patient focuses on the test is defined as the time the interaction point is above the test, either interacting with the questions or not interacting with anything else. The rest of the time spent arranging the objects on the desk or looking around is considered out–of–focus time. The graph displays the ratio between the focus time and the total test time as a percentage.


RESULTS PAGE

By accessing this page you can obtain detailed information about your test results.

For a better understanding this information will be provided to you in a graphical way.

You will be able to see 3 graphics that provide specific information regarding your tests.

In order to display them just click on the "DISPLAY GRAPHICS" button.

Figure 2. Visualization of recorded measurements

The test results can be view before or after the therapy session. The graphs present a historical view of the therapy records, allowing easy visual evaluation of performance and progress.

5. Future Work

The development of the virtual environment presented here has been completed as the application reached a stable version. The next step is to validate it with specialists in psychology and psychiatry. Finally this work will be validated and its effectively and results evaluated in a pilot study followed by patient trials.

References

[1] R. Boian, Bouzit, and G. M; Burdea, "Dual stewart platform mobility simulator," in *IEEE 9th International Conference on Rehabilitation Robotics*, Chicago IL, 2005, pp. 550–555.

[2] A. Rizzo, D. Klimchuk, R. Mitura, T. Bowerly, J. Buckwalter, and T. Parsons, "A virtual reality scenario for all seasons: The virtual classroom," *CNS Spectrums*, vol. 11, no. 1, pp. 35–44, 2006.

[3] T. Parsons, T. Bowerly, J. Buckwalter, and A. Rizzo, "A controlled clinical comparison of attention performance in children with adhd in a virtual reality classroom compared to standard neuropsychological methods," *Child Neuropsychology*.

[4] *Diagnostic and statistical manual of mental disorders, fourth edition.* Washington, DC: American Psychiatric Association, 1994.

[5] M. C. staff, "Obsessive-compulsive disorder (ocd): Symptoms." [Online]. Available: http://www.mayoclinic.com/health/obsessive-compulsive-disorder/ds00189/dsection=symptoms

[6] Opengl, D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, August 2005.

[7] E. Lengyel, *The OpenGL Extensions Guide*. Charles River Media, 2003.

(1) Babes-Bolyai University, Department of Computer Science, Str. Mihail Kogalniceanu Nr 1, RO-400084 Cluj-Napoca

E-mail address: `jibotean_sorin@yahoo.com`

(2) Babes-Bolyai University, Department of Computer Science, Str. Mihail Kogalniceanu Nr 1, RO-400084 Cluj-Napoca

E-mail address: `rares@cs.ubbcluj.ro`