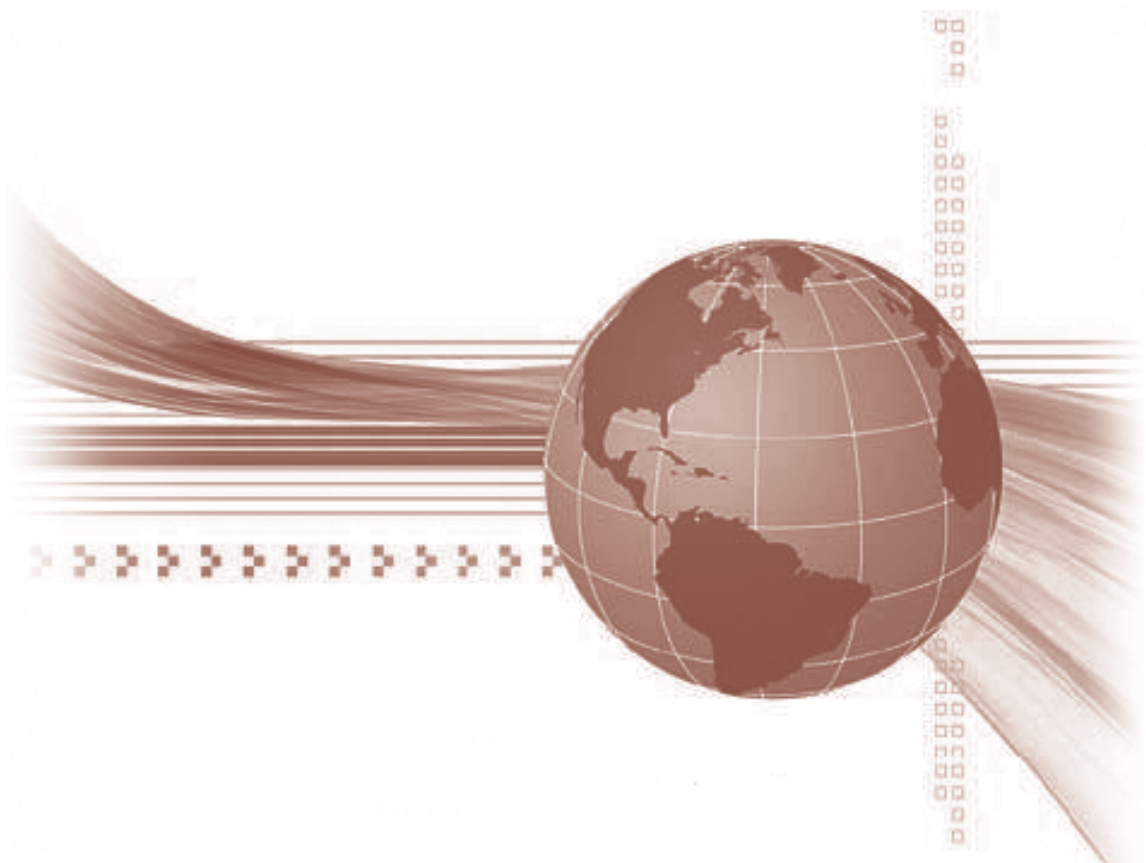
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

3/2011

YEAR **(LVI) 2011**
MONTH **SEPTEMBER**
ISSUE **3**

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

3

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

KNOWLEDGE IN SOFTWARE ENGINEERING

KNOWLEDGE IN DISTRIBUTED COMPUTING

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

A SOFTWARE FRAMEWORK FOR SOLVING COMBINATORIAL OPTIMIZATION TASKS

ISTVAN-GERGELY CZIBULA, GABRIELA CZIBULA AND MARIA-IULIANA BOCICOR

Abstract. Due to the major practical importance of *combinatorial optimization* problems, many approaches for tackling them have been developed. As the problem of intelligent solution generation can be approached with *reinforcement learning* techniques, we aim at presenting in this paper a programming interface for solving combinatorial optimization problems using reinforcement learning techniques. The advantages of the proposed framework are emphasized, highlighting the potential of using reinforcement learning for solving optimization tasks. An experiment for solving the *bidimensional protein folding* problem developed using the designed interface is also presented.

1. Introduction

Combinatorial Optimization (CO) problems have a major practical importance. All these problems are searching for one or more optimal solutions in a well defined discrete problem space. The current real-life combinatorial optimization problems (COPs) are difficult in many ways: the solution space is huge, the parameters are linked, the decomposability is not obvious, the restrictions are hard to test, the local optimal solutions are many and hard to locate, and the uncertainty and the dynamicity of the environment must be taken into account.

Due to the importance of CO problems, many algorithms to tackle them have been developed [3]. Incremental search algorithms use past experience to form feasible solutions. The "goodness" (optimality) of the resultant solution is the only type of feedback available in many cases [8]. Since we expect the system to learn based on its past experience and generate better solutions over

Received by the editors: February 7, 2011.

2010 *Mathematics Subject Classification.* 68N01, 65K10, 68T05.

1998 *CR Categories and Descriptors.* I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*; D.1.5[**Software**]: Programming Techniques – *Object-Oriented Programming*.

Key words and phrases. Software framework, reinforcement learning, protein folding.

time using only this limited information, the problem of intelligent solution generation can be approached with reinforcement learning (RL) [10].

Reinforcement Learning (RL) is an approach to machine intelligence in which an agent can learn to behave in a certain way by receiving punishments or rewards on its chosen actions [10].

The aim of the approach proposed in this paper is to make an abstraction of the issue of solving combinatorial optimization problems using reinforcement learning techniques. In this direction we propose a programming interface and we develop, using the proposed framework, an application for solving the *bidimensional protein folding problem* using reinforcement learning.

The rest of the paper is structured as follows. Our RL based interface proposal is presented in Section 2. An experiment for solving the *bidimensional protein folding* problem developed using the proposed interface is reported in Section 3. Conclusions and further work are outlined in Section 4.

2. The RL based framework

The idea of using RL in optimization problem solving has appeared before [8], and several approaches were developed for solving particular optimization problems [13, 8]. We have previously introduced in [4, 5] a reinforcement learning based model for solving a well known optimization problem within bioinformatics, the *protein folding* problem, which is an *NP*-complete problem [2] that refers to predicting the structure of a protein from its amino acid sequence. Protein structure prediction is one of the most important goals pursued by bioinformatics and theoretical chemistry; it is highly important in medicine (for example, in drug design) and biotechnology (for example, in the design of novel enzymes).

In this section we propose an API that allows to simply develop applications for solving combinatorial optimization problems using reinforcement learning techniques. In the framework that we propose we make an abstraction of the way the optimization problem to be solved is modeled as a reinforcement learning task [10]. This is the major advantage of our RL interface proposal: the reinforcement learning algorithm is defined independent of the way the environment, states and actions are defined.

The interface is realized in JDK 1.6 and has four basic modules: agent, environment, reinforcement learning, and simulation. As in a general agent based system [12], the *agent* is the entity that interacts with the *environment*, that receives perceptions and selects *actions*. The agent learns using *reinforcement learning* to achieve its goal, i.e to find an optimal solution of the corresponding optimization problem. Generally, the inputs of the agent are perceptions about the environment (in our case *states* from the environment),

the outputs are actions, and the environment offers rewards after interacting with it. The interaction between the agent and the environment is controlled by a *simulation* entity.

In the following, we will briefly describe the responsibility of the main elements from the programming interface that we introduce for solving combinatorial optimization problems using reinforcement learning.

Agent. The agent is the entity that interacts with the environment, receives perceptions (states) from it and selects actions. The agent learns by reinforcement and could have or not a model of the environment. It is the basic class for all the agents. The specific agents will implement the *Agent* interface. The main responsibility of the *Agent* class is to select the most appropriate *action* it has to perform in the *environment*.

Environment. The environment basically defines the optimization problem to solve. In our approach, the environment will have an explicit representation as a space of *states*. It is the basic class for all environments. The specific environments will implement the *Environment* interface. The *Environment* has a function that determines the environment to make a transition from a state to another, after executing a specific action. This function also gives the *reward* obtained after the transition. The environment stores an instance of its current *state*.

ReinforcementLearning. Is the class that is responsible with the reinforcement learning process. The framework provides implementations for $Q - learning$, $SARSA$ and $SARSA(\lambda)$. λ refers to the use of an *eligibility trace* [9] for obtaining a more general and efficient learning method. The $EligibilityTrace$ class is responsible with managing eligibility traces.

Simulation. Is the object that manages the interaction between the agent and the environment. An instance of the simulation class is associated with an instance of an agent and an environment at the creation moment. The *simulation* object is responsible with collecting data, managing the learning process and providing the optimal policy that the agent has learned.

Figure 1 shows a simplified UML diagram [6] of the interface, illustrating the core of the RL interface. It is important to mention that all the classes provided by the interface remain unchanged in all applications for solving combinatorial optimization problems using reinforcement learning. What is outside the core are reference implementations.

As a conclusion, we summarize the main advantages of using the framework proposed in this paper:

- Provides an easy way to model optimization problem solving as a reinforcement learning problem.

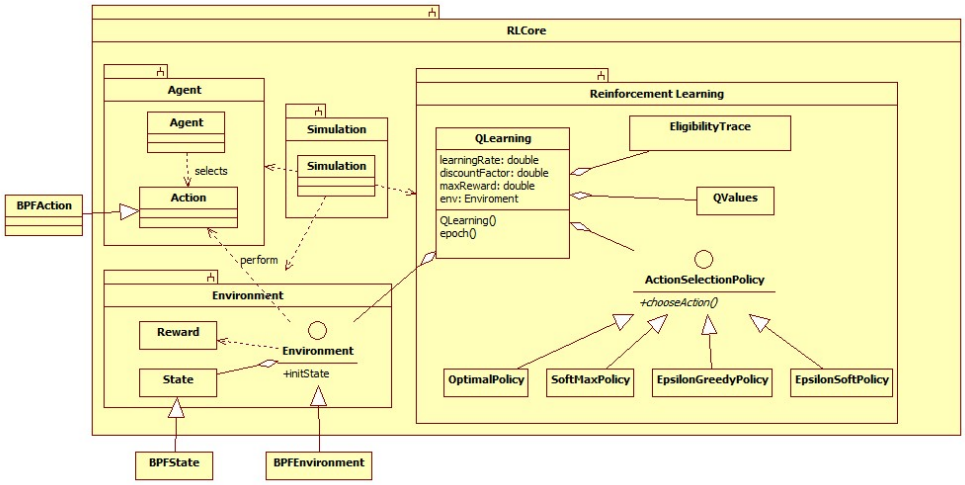


FIGURE 1. The diagram of the programming interface

- The effort for developing an application for solving optimization problems using reinforcement learning is reduced – we need to define only a few classes. The framework offers default implementations for most of the RL related algorithms, the user only needs to define the classes that describe the concrete optimization problem (classes that implement the *State*, *Environment* and *Action* interfaces).
- In a scenario of solving an optimization problem using reinforcement learning, the proposed interface simply allows experimentation with different conceptual models for the states, the actions, different RL algorithms, different reinforcement functions.

3. EXPERIMENT

In order to experimentally evaluate the proposed framework, we are addressing in the following the *Bidimensional Protein Folding Problem* ($BPFP$), more exactly the problem of predicting the bidimensional structure of proteins, a well known optimization problem within bioinformatics. In the proposed experiment, we use the reinforcement learning model for solving $BPFP$ that have been previously introduced in [4].

Let us consider a HP protein sequence $\mathcal{P} = HHPH$, consisting of four amino acids. The aim is to find a configuration of $\mathcal{P}$ whose energy [1] is minimum. The state space of the RL model consists of 85 states, and the action space consists of four actions available to the problem solving agent

and corresponding to the four possible directions $L(Left)$, $U(Up)$, $R(Right)$, $D(Down)$ used to encode a solution [4, 5].

In order to use the interface proposed in Section 2 for solving the above mentioned problem, we have defined specialized classes for which we executed the simulation (*BPFState*, *BPFEnvironment* and *BPFAction*). We have trained the BPF agent using the Q-learning algorithm. As proven in [11], the Q-learning algorithm converges to the real Q-values as long as all state-action pairs are visited an infinite number of times, the learning rate α is small (e.q 0.01) and the policy converges in the limit to the Greedy policy. We remark the following regarding the parameters setting: the learning rate is $\alpha = 0.01$ in order to assure the convergence of the algorithm; the discount factor for the future rewards is $\gamma = 0.9$; the number of training episodes is 64; the ϵ-Greedy action selection mechanism was used.

Using the above defined parameters and under the assumptions that the state action pairs are equally visited during training and that the agent explores its search space (the ϵ parameter is set to 1), the solution reported after the training of the BFP agent was completed is the path $\pi = (s_1 s_2 s_7 s_{28})$ having the associated *configuration* $a_\pi = (LUR)$, determined starting from state s_1, following the *Greedy* policy. The solution learned by the agent has an energy of -1. Consequently, the BPF agent learns the optimal solution of the bidimensional protein folding problem, i.e the bidimensional structure of the protein $\mathcal{P}$ that has a minimum associated energy (-1).

Using the framework proposed in this paper we can simply select other conceptual models for the states space and the actions space within the RL scenario of solving the bidimensional protein folding problem. For example, we can think at the states space as the set of possible solutions of the $BPFP$ and at the action space as the set of possible pull move transformations [7].

4. Conclusions and Further Work

We have introduced in this paper a framework that facilitates research in the direction of solving combinatorial optimization problems using reinforcement learning techniques. We have emphasized the advantages of the proposed framework, highlighting the potential of using reinforcement learning for solving optimization tasks.

Further work will be done in order to apply the framework for other optimization problems, to extend the evaluation of the proposed framework for larger *bidimensional protein folding* problem benchmarks, and to investigate other conceptual models for the states space and the actions space within the RL scenario of solving the $BPFP$.

ACKNOWLEDGEMENT

This work was supported by CNCSIS - UEFISCDI, project number PNII - IDEI 2286/2008.

References

[1] C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.

[2] B. Berger and T. Leighton. Protein folding in HP model is NP-complete. *Journal of Computational Biology*, 5:27–40, 1998.

[3] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, September 2003.

[4] G. Czibula, M. Bocicor, and I. Czibula. A reinforcement learning model for solving the folding problem. *International Journal of Computer Technology and Applications*, 2:171–182, 2011.

[5] G. Czibula, M. Bocicor, and I. Czibula. An Experiment on Protein Structure Prediction using Reinforcement Learning. *Studia Babes-Bolyai Informatica*, LVI(1):25-34, 2011.

[6] http://www.omg.org/technology/documents/formal/uml.htm. UML webpage.

[7] N. Lesh, M. Mitzenmacher, and S. Whitesides. A complete and effective move set for simplified protein folding. In *Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 188–195, New York, NY, USA, 2003. ACM.

[8] V. V. Miagkikh and W. F. Punch III. Global search in combinatorial optimization using reinforcement learning algorithms. In *in Proc. of the 1999 Congress on Evolutionary Computation (CEC99*, pages 189–196. IEEE Press, 1999.

[9] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Mach. Learn.*, 22, January 1996.

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[11] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[12] G. Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[13] W. Zhang and T. G. Dietterich. High-performance job-shop scheduling with a time-delay TD(Lambda) network. In *Advances in Neural Information Processing Systems 8*, pages 1024–1030. MIT Press, 1995.

Babeş-Bolyai University, Department of Computer Science, 1, M. Kogalniceanu Street, 400084, Cluj-Napoca, Romania

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

BUILDING BLOCKS DEV STUDIO. A TOOL FOR COMPONENT BASED DEVELOPMENT

PAUL HORAŢIU STAN

Abstract. This paper presents a technique inspired from hardware engineering that can be applied in software engineering in order to develop flexible and modular systems based on independent components, these components can be written on different programming languages. A graphical user interface tool has been developed during this research. The *BB-DevStudio* combines theoretical results presented in [2] and [1] in order to generate the source code for independent components and to support communication between Java and .NET components. At the end of the article a demo application is presented together with advantages and constraints of the proposed solution.

1. Introduction

In hardware development process, an engineer can use many integrated circuits in order to develop a complex system, for instance he/she can use multiplexers, counters, logic AND, logic OR, memory, register etc, these components are interconnected on a main board and become a complex system. The counter or the multiplexer or other components are developed by a third party company. This principle can be applied both on hardware and software development process, for example a software system can have many independent components that should be interconnected on a main board in order to become a complex system. These components can be developed by a third party company. Nowadays there are many third party components that are used in software systems, but the connections between them are made by a developer, on the other hand a person should write a program that will use the third party components, this can introduce many errors; more than this,

Received by the editors: March 30, 2011.

2010 *Mathematics Subject Classification.* 68N15.

1998 *CR Categories and Descriptors.* D.2.11 [**Software**]: Software Engineering – *Software Architectures*; D.2.13 [**Software**]: Software Engineering – *Reusable Software*.

Key words and phrases. Component Based Development, Automatic Source Code Generation, Software Architectures.

changing the code written by a developer is an expensive process that needs time and money [1].

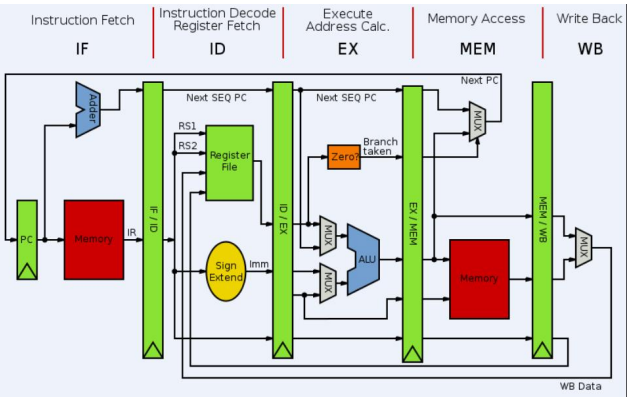


FIGURE 1. MIPS Architecture

2. The problems

The general context of this article is focused on: (1) how to access remote objects developed for different programming languages in a transparent way, like they were written for client programming language and (2) presenting a technique for automatically generate the source code for a software component that contains subcomponents.

Regarding (1) the problem is: how the parameters can be passed to the remote methods without being necessary to serialize them on client platform and restored on remote platform in order to be used there. The main idea presented in this article is that each object should be executed in the address space of the process which has created itself.

Regarding (2) the problem is: based on an XML component specification file a skeleton source code should be generated, after that a developer can add the source code that implements the business logic.

The (1) and (2) should be combined in order to support source code generation for components written on different platforms that can communicate in a transparent way.

This section is composed by three parts, first presents the current interoperability and component based approaches and the second describes the proposed solution. The proposed solution has two parts: (1) the interoperability solution and (2) the component based solution.

3. Proposed approach

3.1. **Proposed Architecture for platform interoperability.** The proposed solution is based on the Proxy design pattern presented in [16].

In the proposed solution for each remote object a proxy object is generated on the client side. When the client platform creates the proxy then the server creates the real object and store it in a hash table based on an automatically generated identifier. When the client side platform erase the proxy object, for instance the garbage collector decides to remove from memory the unreferenced objects then the server side platform will remove the real object from the remote objects hash table. When a client object invokes a method on the proxy object the request is redirected to the remote object.

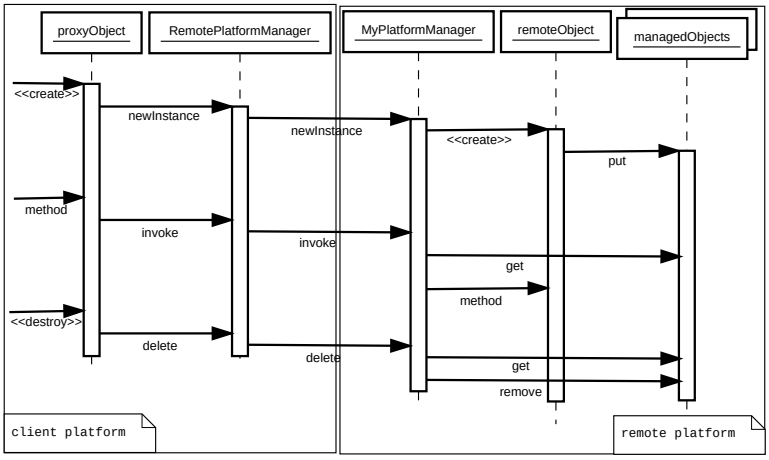


Figure 2. Communication Architecture

3.2. **Component Based Solution Architecture.** Figure 5 presents a print screen with the main architecture of a software system developed using Building Blocks framework, this tool is a result of the proposed research.

Two ports *O1* and *I1* can be connected if they have the same data type *T*. The source code for the *demoIndepComp* will be automatically generated based on a connection specification file, which is an XML file.

An independent component is an instance of a class. The component class should implement a standard interface called *IndependentComponent*. The component communicates with the environment using properties and events.

Figure 3 presents the conceptual model of the proposed solution. A component can have subcomponents, properties, links and subcomponent links. The direction of a property can be: (1) input, (2) output and (3) input/output.

A link is a connection between two properties of two subcomponents and a subcomponent link is a connection between a property of a subcomponent and a property of the parent component.

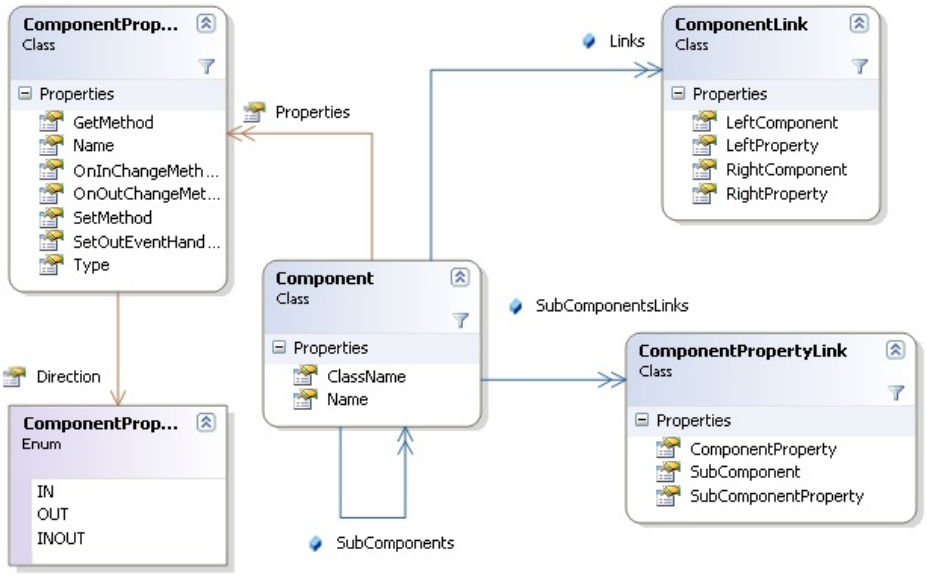


FIGURE 3. Independent Components Model

4. Research Results

4.1. **The developed framework for interoperability.** During this research a framework for interoperability between .NET and Java has been developed. Current version allows Java classes to be used in .NET and vice-versa. The general context is: there is a data type *RemoteType* on the developing platform *RemotePlatform* and it should be used in a class *C* on developing platform *ClientPlatform*. Based on the *RemoteType* an XML file is generated that specifies how the type can be used. Using this XML file the source code for the *ProxyType* is automatically generated. The *ProxyType* is written in the *ClientPlatform* programming language. This process is exposed in the figure 4.

4.2. **The Building Blocks Dev Studio.** Figure 5 presents a BBDevStudion print screen. The picture shows an application model with two subcomponents components: (1) UserInterface and (2) ALU. Both components have two properties: (1) Request and (2) Response. The data type of the Request property

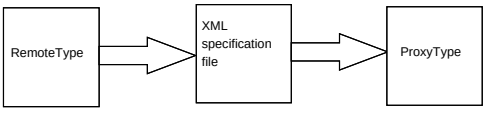


FIGURE 4. Automatically Generating the Proxy Type

is ALURequest and the data type of the Response property is ALUResponse. These two types are declared in a .NET dll file.

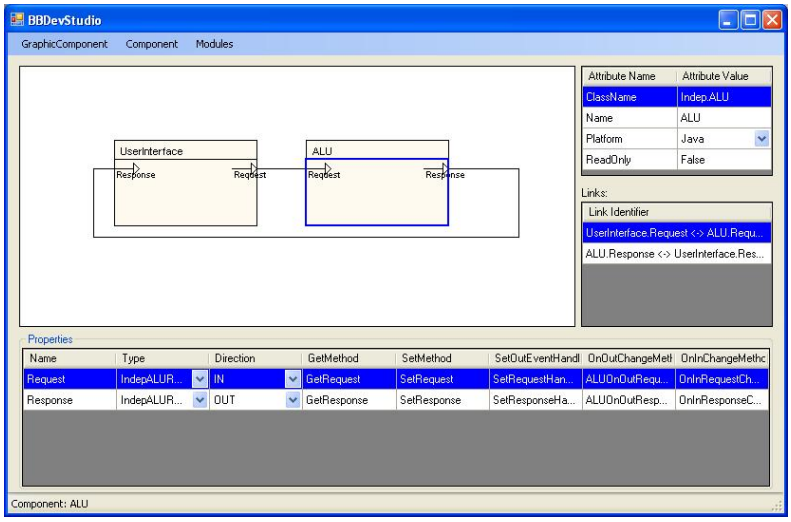


FIGURE 5. BBDevStudio Print Screen

The *UserInterface* component can use easily the *ALURequest* and *ALUResponse* data types because are written in .NET but the *ALU* component cannot use directly these data types, it can only use proxy types written in Java that can communicate with real data types written in .NET.

On the other hand, the parent component that contains the *UserInterface* and *ALU* components, can easily create the instance of *UserInterface* but should use a proxy to *ALU* for working with the real *ALU* component written in Java. The *ALU* component should use a proxy to the parent component in order to notify that a *Response* is ready to deliver to the *UserInterface* component.

ACKNOWLEDGMENT

The author wishes to thank for the financial support provided from programs co-financed by the Sectorial Operational Programme Human Resources

Development, Contract **POSDRU 6/1.5/S/3** "Doctoral studies: through science towards society".

The author would like to thank professor Bazil Pârv for his precious help.

References

[1] Paul Horatiu Stan, *A proposed technique for component based software development*, ZAC 2010, 52-56.
[2] Paul Horatiu Stan, Camelia Serban, *A proposed approach for platform interoperability*, Studia UBB 2010, 87-98.
[3] B Nolan, B Brown, L Balmelli, T Bohn, U Wahli *Model Driven Systems Development with Rational Products* IBM 2008.
[4] Edward L. Lamie *Real-Time Embedded Multithreading using ThreadX and MIPS*, Newnes Pap 2008
[5] David Chappell. *Introducing SCA*, DavidChappell and associates, July 2007, http://www.davidchappell.com/writing/Introducing_SCA.pdf last accessed on June 09, 2011.
[6] *SCA Service Componeent Architecture, Assembly Model Specification* 2007, http://www.osoa.org/display/Main/The+Assembly+Model, last accessed on June 09, 2011.
[7] Dominic Sweetman *See MIPS Run, Second Edition* Morgan Kaufmann; 2 edition 2006.
[8] Ingo Szpuszta, Mario Rammer, *Advanced .NET Remoting 2nd Edition*, APRESS February 2005.
[9] Bill Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, Philippe Vanderheyden *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*, IBM 2004
[10] Stephen A. White. *Introduction to BPMN* 2004 http://www.bptrends.com/publicationfiles/07-04 WP Intro to BPMN - White.pdf, last accessed on June 09, 2011.
[11] Vincent Massol, *JUnit in Action*, Manning Publications November 2003.
[12] Clemens Szyperski, *Beyond Object-Oriented Programming*, second edition, ACM Press New York 2002
[13] Bruce Eckel, *Thinking in Java*, fourth edition, MindView, Inc, 2002.
[14] Ethan Cerami, *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI and WSDL*, O'Reilly Media, Inc. February 2002.
[15] William Grosso, *Java RMI (Java Series)*, O'Reilly Media, Inc. Octomber 2001.
[16] Erich Gamma, Richard Helm, and Ralph Johnson, and John Vlissides, *Design Patterns: 50 specific ways to improve your use of the standard template library.* Pearson Education, Inc 1995.
[17] Model Driven Architecture, http://www.omg.org/mda, last accessed on June 09, 2011.
[18] http://www.w3.org/TR/soap, accessed on June 09, 2011
[19] http://en.wikipedia.org/wiki/Component-based_software_engineering *last accessed on June 09, 2011*

Babeş-Bolyai University, Department of Computer Science, M. Kogălniceanu St., 400084 Cluj-Napoca, Romania
E-mail address: horatiu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

JSET - JAVA SOFTWARE EVOLUTION TRACKER

ARTHUR-JOZSEF MOLNAR

Abstract. This paper introduces the Java Software Evolution Tracker, a visualization and analysis tool that provides practitioners the means to examine the evolution of a software system from a top to bottom perspective, starting with changes in the graphical user interface all the way to source code modifications.

1. Introduction

Software tools occupy an important place in every software practitioner's toolbox. They can assist in virtually all activities undertaken during the life of software starting from requirements analysis to test case design and execution. By studying the evolution of widely used IDE's such as Eclipse [3] one can see that each new version ships with better and more complex tools for aiding professionals in building higher quality software faster. Modern environments include tools for working with UML artifacts, navigating source code and working with a wide variety of file types.

However, modern day software systems fall into many categories, each having unique requirements, artifacts and processes. Our goal is to incorporate this knowledge into tool development efforts and use the latest available results from research in developing new, useful tools for practitioners in software development.

The rest of this paper is structured as follows: the next section will present the work jSET is based on. The third section will describe the tool in detail, while the fourth overviews its current limitations. The last section is reserved for conclusions and future work planned.

Received by the editors: March 30, 2011.

2010 *Mathematics Subject Classification.* 68N15.

1998 *CR Categories and Descriptors.* D.2.2 [**Software**]: Software Engineering – *Design Tools and Techniques*.

Key words and phrases. Software visualization, Software tool, Static analysis.

2. Related work

The development of jSET was made possible by two tools that come from the academic environment. They are presented in the following paragraphs together with earlier efforts of using them for software visualisation.

The first of these tools is called GUIRipper, part of the comprehensive GUITAR toolset [2]. GUIRipper acts on a GUI driven target application [7] that it runs and records all the widgets' properties across all the application's windows. The resulting GUI model is saved in XML format for later use. When developing jSET, additional functionality for recording widget event handlers and capturing screenshots was programmed into GUIRipper. This modified version can be found at the jSET website [9].

The second application is the Soot analysis framework [8]. Soot is a static analysis framework that targets Java bytecode; all its implemented analyses are performed without running the target application. One of the most important artifacts produced by Soot is the application's call graph: a directed graph that describes the calling relations between the application's methods [4]. As it is computed statically, it does not provide information regarding order of calling or execution traces. For use with jSET, a suitable model that persists the computed callgraph was developed and implemented as a wrapper over Soot.

The applications described above laid the groundwork for the development of advanced software tools. Some of these earlier efforts, that served as inspiration for jSET's development are discussed in the following paragraphs.

Possibly the earliest of such tools is JAnalyzer [1], *a visual static analyzer for Java* developed by Bodden et al. JAnalyzer leverages call graph information generated by Soot and graphically displays the calling relations in a program. It also allows viewing the source code of compiled methods, thus creating a link between the application bytecode and its sources.

A more advanced undertaking is described in [5] where the author presents a call graph comparison tool that ranks the differences according to their importance. The same paper also introduces a browser application for navigating call graphs, similar to JAnalyzer.

3. jSET - Java Software Evolution Tracker

jSET is a software analysis and visualisation tool created for practitioners and researchers alike. The three main ideas guiding its development are:

(1) Provide an advanced visualization tool for easily accessing static analysis results inside a software project environment without the need for a laborious setup phase.

(2) Integrate the obtained results in the context of GUI driven applications by offering a seamless transition from the GUI's level all the way to the application's source code.
(3) Facilitate identification and analysis of changes across versions of a software system from changes in the GUI down to source code modifications.

In order to use jSET, the first step is to create a project. A jSET project is an XML file that contains information about the locations off all the necessary artifacts. For a valid project, the following data is required:

- The GUI model obtained by running our modified version of GUIRipper on the target application.
- The callgraph obtained by running jSET's Soot wrapper on the target application.
- The target application's bytecode (including used libraries)
- The target application's source code[1].

It is important to note that all the steps of building a project can be easily automated. Both GUIRipper and our Soot wrapper can be executed via command line and manual intervention using configuration files is required only for certain changes in the target application such as specifying special handling for some GUI elements (e.g: exempting components from analysis) or updating the application's libraries. As such jSET is easily integrateable with the target application's build system. More detailed information and project examples are available on the jSET website [9].

The jSET application can be used in two modes: project exploration and project comparison. When starting the application, the user must select one or two projects to load. Selecting one defaults jSET to project exploration mode. The comparison mode can be used to display the differeces between the target application's versions[2]. Figure 1 shows the tool in comparison mode, the target application being an early version of the open-source FreeMind mind mapping software[3].

The tool's user interface is rather similar for both modes but because of differences in the information displayed, the following paragraphs present both in detail, starting with project exploration mode.

3.1. **Project exploration mode.** jSET's user interface consists of several panes displaying information about the loaded project. The left-side pane

[1]Only if viewing the source code is desired

[2]The projects should capture the same application at different versions, however this is not enforced

[3]https://sourceforge.net/projects/freemind/

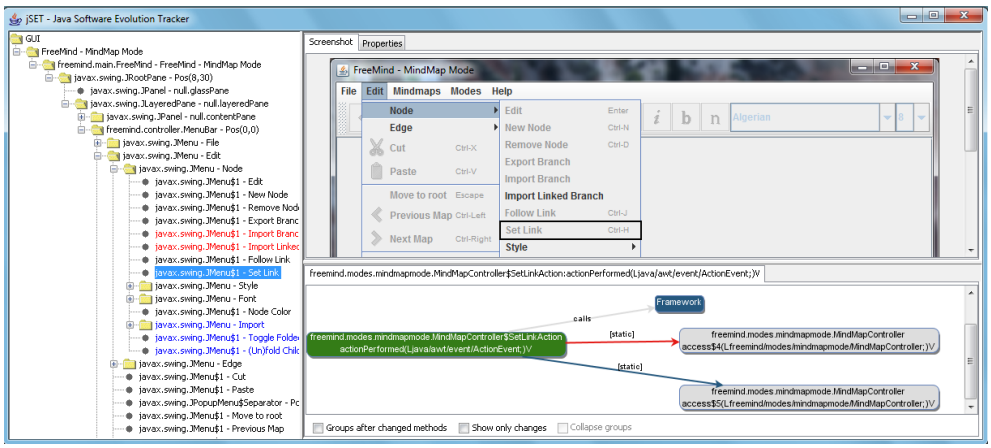


FIGURE 1. jSET in Project Comparison mode

displays the target application's user interface hierarchy. When a widget is selected from it the right hand tabbed pane displays its important properties[4] and a screenshot showing it as it appears when the target application is running, shown in Figure 1.

Among the displayed properties we can find the handlers associated for the widget's events. Some of them are attached by the platform as they control behind the scenes aspects regarding the GUI. Others however are defined by the application itself. One of jSET's original contributions concerns visualising the target application's event handling. When selecting one of the handlers, the relevant part of the application's call graph is displayed in the right lower pane. Here the user can examine what methods might be run when a certain event is fired (e.g: a button is clicked on the GUI). It is important to mention that the displayed graph is only part of the application's call graph, as the entire structure is too complex to be displayed at once for all but the most trivial programs [5].

3.2. **Project comparison mode.** Since the compare view's layout is similar to the project exploration one, this section will discuss only the relevant differences.

The first difference regards the GUI tree shown on the left hand side. While the exploration mode displays the target application's GUI hierarchy, the comparison mode also displays the differences between the projects GUIs. Looking at Figure 1, we can see certain items from the hierarchy are color coded. Red items represent widgets that can no longer be found on the newer

[4]Like swingExplorer does (http://www.swingexplorer.com)

version, while items in blue are widgets not found on the older one. Green items represent widgets affected by underlying changes in their event handler code.

The second way compare mode differs from exploration regards the graph display. For widgets identified in both loaded versions, a new way of displaying call graphs was developed. The astute reader will notice the same color coding used as in the case of GUI widgets, this time customized for method calls. As such, the displayed graph will actually be the reunion of the event handlers' call graphs across versions: red edges represent calls removed from the newer version while blue edges show new method calls. Green is used for methods that underwent code changes between the versions. Right clicking a node brings up a menu that allows comparing the bytecode or source code (if available) of the selected method between versions.

The jSET application is an ongoing effort of providing practitioners with tools based on the latest accomplishments in research. The exploration mode is useful for understanding how the target application works by identifying events that cause code to run and analyzing the calling relations between the application methods. The comparison mode allows tracing the evolution of a software system; this enables evaluating the changes across versions in a top to bottom fashion, from the GUI level to source code statement changes.

4. Limitations

Although much thought went into the design and implementation of jSET, there are some aspects that limit its usability. Some of them stem from inherent limitation of the tools jSET itself is based on. The following list attempts a brief overview of these limitations:

- *Dynamic user interfaces.* Some applications create and dispose of GUI elements dynamically which sometimes makes it impossible to save a complete model of the GUI.
- *Reflection.* Applications using reflection might instantiate classes and run methods that are not captured in the callgraph, leading to an incomplete representation in jSET.
- *Interacting widgets.* In some cases, events fired on widgets might fire new events on other GUI elements. This is not accounted for by the current version of jSET, and in these cases library callbacks might occur that are not captured in the displayed graphs.

5. Conclusions and future work

In this paper we presented jSET, a new software visualisation and analyis tool. jSET introduces a new approach for software visualization that goes from

GUI to source code statement level. The tool implements a new way of displaying changes in the target application's GUI across versions and integrates this information with changes in the source code.

Future work planned includes devising new algorithms for identifying equivalent GUI elements across program versions, improving the state of the tool's limitations and using jSET as the software visualisation tool of a fully automated regression test procedure for GUI based applications [6].

Acknowledgements

The author was supported by programs co-financed by The Sectoral Operational Programme Human Resources Development, Contract POS DRU 6/1.5/S/3 - "Doctoral studies: through science towards society"

References

[1] Eric Bodden, *Janalyzer: A visual static analyzer for java*, (2003), As contribution for the SET Awards 2003, category computing.

[2] Daniel Hackner and Atif M. Memon, *Test case generator for GUITAR*, ICSE '08: Research Demonstration Track: International Conference on Software Engineering (Washington, DC, USA), IEEE Computer Society, 2008.

[3] Daqing Hou and Yuejiao Wang, *An empirical analysis of the evolution of user-visible features in an integrated development environment*, Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research (New York, NY, USA), CASCON '09, ACM, 2009, pp. 122–135.

[4] Ondrej Lhotak, *Spark: A flexible point-to analysis framework for java*, Tech. report, McGill University, Montreal, 2002.

[5] Ondrej Lhotak, *Comparing call graphs*, Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (New York, NY, USA), PASTE '07, ACM, 2007, pp. 37–42.

[6] Atif Memon, Adithya Nagarajan, and Qing Xie, *Automating regression testing for evolving gui software*, Journal of Software Maintenance **17** (2005), 27–64.

[7] Atif Muhammed Memon, *A comprehensive framework for testing graphical user interfaces*, Ph.D. thesis, 2001, AAI3026063.

[8] Vijay Sundaresan, *Practical techniques for virtual call resolution in java*, Tech. report, McGill University, 1999.

[9] Website., https://sourceforge.net/projects/javaset.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Department of Computer Science, 1 M. Kogalniceanu, Cluj-Napoca 400084, Romania

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

STORAGE INDEPENDENCE IN DATA STRUCTURES IMPLEMENTATION

VIRGINIA NICULESCU

Abstract. Design patterns may introduce new perspectives on the traditional subject of data structures. They introduce more flexibility and reusability in data structures implementation and use. We analyze in this paper some design patterns that can be used for data structures implementation, and their advantages. This analysis emphasizes how design patterns could be used in order to obtain implementation of the data structures based on storage independence.

1. Introduction

Data structures [4, 8] represent an old issue in the Computer Science field. By introducing the concept of *abstract data type*, data structures could be defined in a more accurate and formal way. A step forward has been done on this subject with object oriented programming [1]. Object oriented programming allows us to think in a more abstract way about data structures. Based on OOP we may define not only generic data structures by using polymorphism or templates, but also to separate definitions from implementations of data structures by using interfaces.

Design patterns may move the things forward, and introduce more flexibility and reusability for data structures.

2. First Level and Second Level Data Structures

The study of the different data structures emphasizes the fact that we can make the following classification:

- *first level* or fundamental data structures;
- *second level* data structures which are characterized by the fact that their implementations use first level data structures.

Received by the editors: March 30, 2011.

2000 *Mathematics Subject Classification.* 68P05.

1998 *CR Categories and Descriptors.* E.1 [**Data**]: Data Structures; E.2 [**Data**]: Data Storage Representation.

Key words and phrases. data structures, design patterns, genericity, representation.

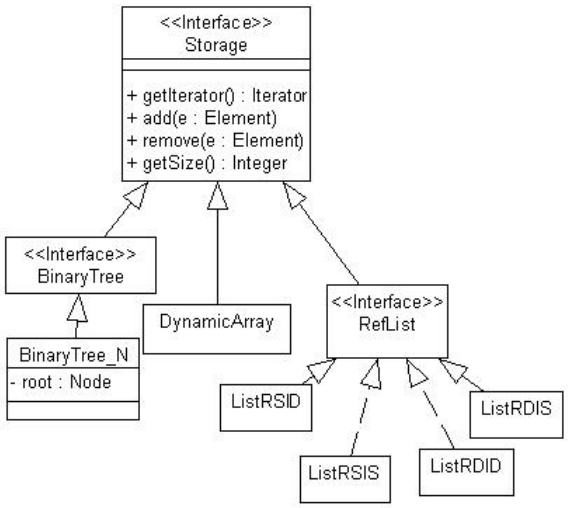


FIGURE 1. Storage interface, and some fundamental data structures - arrays, lists and trees.

The arrays, and linked representations for lists and trees are considered *first level* data structures. In order to implement a set or a map we can use an array, or a linked list or a tree; so sets and maps are examples of second level data structures. Figure 1 presents the UML class diagram for some first level data structures. We have considered dynamic arrays, lists that are implementation of an interface RefList for which the positions of elements in the list are of a reference type (singly or doubly linked, with dynamic or static allocation), and binary trees with a linked representation using nodes. (A reference is considered to be any value that is used in order to obtain another value; examples of references are: memory addresses (pointers), indices in a table, etc.)

2.1. ***Adapter.*** The problem that could arise is when we have an already developed library for fundamental data structures that is not based on this framework. In this case the *Adapter* [3] design pattern can be used.

Adapter design pattern allows the conversion of the interface of a class into another interface clients expect. *Adapter* lets classes work together that could not otherwise because of incompatible interfaces.

The adaptation has to be done in a way that minimize the time-complexities of the implementations of adapted methods. For example, in order to adapt a linked list to be used as a simple storage we may define the method add from

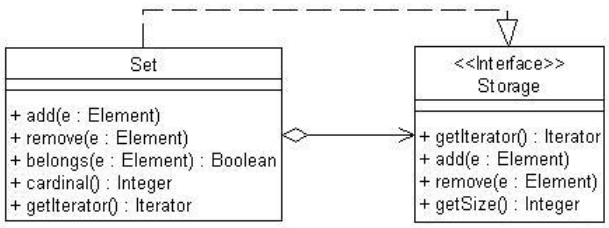


FIGURE 2. Building sets using generic storage for representation.

the Storage interface by using the method addFirst from the list implementation which has a time complexity of $\Theta(1)$.

2.2. ***Bridge.*** In order to implement a second level data structure we have to start from the corresponding abstract data type corresponding to which we may define an interface, and then based on the possible representations to implement concrete classes. The process could be simplified by using the *Bridge* design pattern.

Bridge design pattern decouples an abstraction from its implementation so that the two can vary independently.

Generally, if we have different ways of representation or storage, for a data structure, we may separate the storage from the data structure using *Bridge* design pattern.

We may consider the case of *Set* data structure. The advantages of this separation is that we will have only one class *Set*, and we may specify when we instantiate this class what kind of storage we want, for a particular situation.

This solution for implementing sets uses a reference to a general storage. The diagram of the Figure 2 presents the details of this solution. Set – the new created data structure – could also be seen as a storage that can be used in other contexts, and because of this the class Set implements the interface Storage too.

Since the class *Set* uses a storage in order to store its elements, the constructor of the class Set have to be able to initialize this storage. A direct and simple solution –but not the best – would be to give to this constructor a parameter (of type storage) that could initialize the storage.

The specific operations of the Set data structure are implemented based on the operations of the storage.

Other examples may be considered for multi-sets, maps, dictionaries, a.s.o.

Important restrictions related to the storage are that initially it has to be empty, and also it has to be an unshared storage.

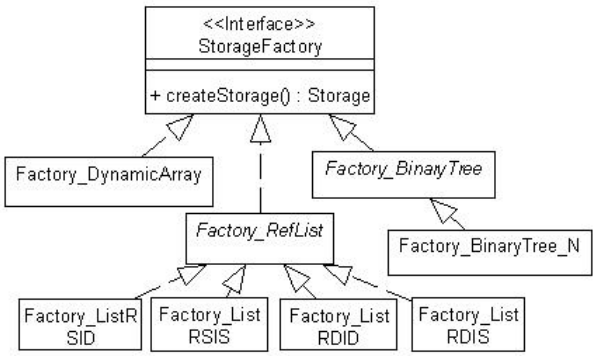


FIGURE 3. Factories for creating arrays, lists, and binary trees.

Generally, the data structures are used in very many different contexts and because of this it becomes important to allow their creation in a flexible and dynamical way.

These requirements could be achieved by using creational design patterns.

More precisely, we will use *Abstract Factory* to create each special storage dynamically.

Singleton design pattern assures the fact we have only one instance created for a certain type, and we have a global access point to it. Since we don't need more than one instance of a specific factory class, *Singleton* design pattern will be used for each.

Abstract Factory design pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes [3].

The concrete products, in which we are interested in, are the fundamental data structures. This means that we define factories for each type of these products; the method createStorage() returns an empty storage of a specialized type. The Figure 3 emphasizes this.

The solution for implementing sets using *Bridge* but also *Abstract Factory* and *Singleton* is presented in the Figure 4.

3. Specialized Storages

An evaluation has to be done relative to the efficiency of the implementation of the operations add, and remove. For a general storage, the post-condition of the operation add specified just the fact that the parameter (of type Element) exists in the storage. In a similar way the remove operation assure the fact that one instance equal to the parameter has been removed from the storage. The implementation of the operation belongs of the class

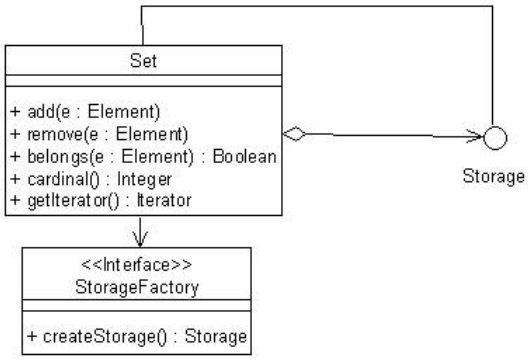


FIGURE 4. Building sets using factories for creating different storage for representation.

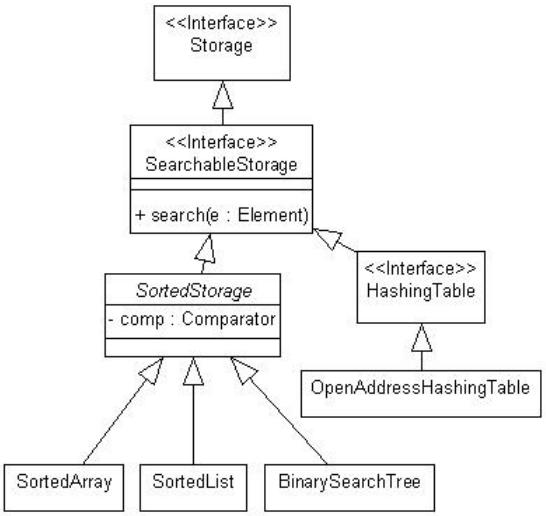


FIGURE 5. Different types of *Storages* and the relations between them.

Set is based on using the iterator – that implies a time-complexity linear in the size of the storage. In order to improve this, a specialized storage has to be defined – SearchableStorage, which add a new method search. A searchable storage is a storage that is able to implement a searching operation with a time-complexity better than for a sequential search. Examples of searchable storages are a hash table, and different types of sorted data structures.

Another useful specialization of a storage is SortedStorage. This kind of storage is useful for the implementation of sorted data structures where elements are compared using an instance of type Comparator. *Comparator* is another design pattern which is very much used in relation with data structure implementation. A SortedStorage is a specialization of a SearchableStorage since for sorted data structures we can define efficient searching operations. The Figure 5 presents the relation between these types of storages. The specification of the interface SortedStorage enforces the postcondition of the method getIterator by imposing the condition that the order in which the elements are iterated is based on comparison criteria specified by the comparator.

4. Conclusions

By separating the concrete representation of a data structure by the behaviour of its type we introduce a new level of indirection and so a new level of abstraction. Using this, we are able to implement data structures based on different fundamental data structures without creating more than one class. So, a new level of genericity is introduced, too.

Storage interfaces also introduce a classification between data structures. We emphasize the fact that each data structure could be used as storage for another data structure or as a generic storage in a generic program.

As it is already known, design patterns are very important mechanism for increasing the level of abstraction in programming. In order to achieve storage independence we have used design patterns as *Abstract Factory, Singleton, Bridge, Comparator* and *Adapter*.

Acknoledgement: This work was supported by CNCSIS - UEFISCDI, project number PNII - IDEI 2286/2008.

References

[1] Bruno R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*, Wiley Computer Publishing, 1999.
[2] H.E. Eriksson, M. Penker, *UML Toolkit*. Wiley Computer Publishing, 1997.
[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995.
[4] E. Horowitz. *Fundamentals of Data Structures in C++*. Computer Science Press, 1995.
[5] D. Nguyen. *Design Patterns for Data Structures*. SIGCSE Bulletin, 30, 1, 1998, 336-340.
[6] V. Niculescu, *Teaching about Creational Design Patterns*, Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts, ECOOP'2003, Germany, July 21-25, 2003.
[7] V. Niculescu. *On Choosing Between Templates and Polymorphic Types. Case-study.*, Proceedings of "Zilele Academice Clujene", Cluj-Napoca, June 2003, pp.71-78.
[8] D.M. Mount. *Data Structures*, University of Maryland, 1993.

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca
E-mail address: vniculescu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

INTEGRATING ALF EDITOR WITH ECLIPSE UML EDITORS

CODRUŢ-LUCIAN LAZĂR

Abstract. In this paper we present an approach for integrating graphical and textual editors for UML models in Eclipse workbench. In our concrete example, we are integrating an Eclipse Xtext based textual editor for OMG Alf language, with the Eclipse UML2 tree editor for UML models. This integration enables the user to view and edit the operation behavior of UML classes from a big UML model using a textual editor with OMG Alf language.

The integration will also benefit the ComDeValCo framework, where we need to model the functionality of the components created with a graphical editor.

1. Introduction

The Executable Foundational UML (fUML [4]) is a computationally complete and compact subset of UML [1], designed to simplify the creation of executable UML models. The semantics of UML operations can be specified as programs written in fUML.

Currently, there is a standardized concrete textual syntax for a fUML based action language. The concrete syntax is standardized by Object Management Group (OMG) and it is called Action Language for Foundational UML (Alf) [3]. A few other action languages exist, some of them being proprietary. None of these action languages are based strictly on fUML, so our research will focus on using the Alf language.

We also introduced an action language based on fUML [7], with a concrete syntax similar to the concrete syntax of the structured programming languages (e.g. Java, C++). As Alf does not restrict the way the concrete textual constructs are mapped to the fUML structures of elements, we will use our

Received by the editors: March 31, 2011.

2010 *Mathematics Subject Classification.* Software, Programming languages.

1998 *CR Categories and Descriptors.* D.2.2 [**Software**]: SOFTWARE ENGINEERING: Design Tools and Techniques – *Computer-aided software engineering (CASE).*

Key words and phrases. Eclipse, EMF, Xtext, fUML, UML, Alf, Action Language.

previous experience in order to define these mappings and implement a textual editor in Eclipse.

In order to be able to use an Alf textual editor in a real environment, it is needed to integrate the textual editor for Alf language with existing UML graphical editors. We have chosen Eclipse as the modeling environment in which to try to develop our modeling tools. The UML graphical editors are implemented using Eclipse tools: tree editors, or diagram editors based on Eclipse Graphical Modeling Framework (GMF) tool. For the Alf language, we can use the Eclipse Xtext utility [5], on top of which to implement our editor. Xtext offers a simple way to define the textual grammar and generates a textual editor that can serve for us as a starting point.

The main problem is that these Eclipse editors work with resources that cannot be nested, and we need to be able to view and edit only parts of the bigger UML model with our Alf editor.

This paper is a technical paper that will describe how we achieved the integration of the graphical UML editor (tree based, in this example) and the textual Alf editor.

The remainder of the paper is organized as follows: section 2 presents the general context and section 3 presents the research problem. Then, section 4 describes the integration approach we propose. Section 5 presents the existing work related to ours and section 6 gives the conclusions of this paper.

2. Background

Creating executable UML models is difficult, because the UML primitives intended for execution (from the UML Action Semantics package) are too low level, making the process of creating reasonable sized executable UML models close to impossible. In order to speed up the process of writing the behavior, different approaches have been used. One approach was to replace the behavior with opaque expressions expressed in the destination programming language (Java, C++) and add them directly in the UML model. Another approach was to fill in the behavior after code generation, in the special marked places. These types of models will not react well to changes in the target programming languages or frameworks. Also, simulating the execution of these types of models and testing them is not easy.

The fUML standard provides a simplified subset of UML Action Semantics package (abstract syntax) for creating executable UML models. It also simplifies the context to which the actions need to apply. For instance, the structure of the model will consist of packages, classes, properties, operations and associations, while the interfaces and association classes are not included.

Creating fUML models is still a hard thing to do, so it is required to use a concrete textual syntax for this.

A concrete textual syntax is in process of being standardized by OMG - Alf language. With this easy-to-use concrete syntax, the creation of executable UML models will become an easy task. A textual syntax enforces a certain way of constructing models, which means that a lot of elements that need to be created explicitly in the graphical UML Activity Diagram can be implicitly derived from the syntax and created automatically. The control flow between the statements is implicit in structured programming languages. Accessing the parameters and variables is done with a simple name reference in text, while the model will contain all the object flow edges and fork nodes needed to represent them. And the examples can continue.

What is required at this point is to create editors based on this Alf language and to integrate them with the existing UML graphical editors and other tools that process UML models.

The action language is also useful for our framework: ComDeValCo (Framework for Software Component Definition, Validation, and Composition) [10, 9, 6]. We want to use it to model the functionality of the components, to simulate the execution of the models and to test the models [8]. The main advantage is to be able to test our models without having to generate code.

3. Research problem

The research problem is to find a way to integrate the textual editor for the Alf language with other graphical editors for UML models in the Eclipse environment.

We have chosen to integrate our textual editor based on Alf language with the tree based editor for UML from Eclipse. This tree based editor will play the role of the graphical editor. We are attempting to make the integration process in such a way that it does not affect the tree based editor, so that we will be able to integrate our textual editor with any other graphical editor for UML models from Eclipse (UML diagram editors based on Eclipse GMF tool, for instance).

4. Integrating Eclipse editors

Similar to any other editor from the Eclipse workbench, the UML tree editor from Eclipse loads the model inside a Resource. Our Xtext based textual editor also loads the model inside a Resource. Each editor will load and save the contents inside the Resource it is working with (by delegating the load and save actions to the Resource).

The general approach is that the contents (model elements) of each Resource will be saved by that Resource. Contents cannot be shared between Resources. A model element can have only one model element owner or no owner (if it is the root model element). All model elements from a contents tree are contained by one and the same Resource.

The Resources cannot be nested, so we cannot have an editor that works on a Resource that represents a partial model from a different Resource. For instance, suppose the UML model is loaded by the UML graphical editor inside an UML Resource. We want to open a textual editor (which needs a Resource to work with) that will edit only a part of the bigger UML model, more specifically, an Activity.

The Xtext editor's parser will re-parse, discard and refresh the whole (or part) of its model at a frequent rate. It is not feasible to have the textual editor work on the whole UML model and simply not show the parts on which we are not interested (packages, classes, ...), inferring this information behind the scene.

Alf standard recommends to save the textual representation of the operation's behavior (which is an Activity) inside a stereotyped Comment attached to the Activity inside the UML model. So, we need to make the Xtext editor load and save the textual representation from that Comment. And the Xtext parser will create the whole UML model elements (actions, nodes, edges, ...) that represent the actual Activity functionality represented by the textual representation from the Comment.

At the same time, the whole UML Activity that is generated by the textual editor based on the Alf code will need to be placed at runtime inside the big UML model, so that it can be validated. The Activity associated with a Classifier's Operation (as its behavior) needs to reside in the same Classifier. The problem that arises is that the Activity will be created by the Xtext editor inside its own Resource and it will be contained by this Resource, which means that the Activity cannot be contained at the same time by the Classifier from the big UML model, which is contained by the UML graphical editor's Resource.

We need to make use of a certain feature from EMF which allows us to use containment reference proxies for certain elements, which means that we need to intervene in our Xtext editor and make it work with proxies for the actual elements that it generates from the textual elements and for the elements that we will place in the big UML model, under the Classifier. We will have one real Activity element and one proxy Activity element. And we need to figure out when to create these elements, where to place each one of them, what to do when the Xtext editor is required to save the Resource (or when the

UML graphical editor wants to save its resource), how often should the Xtext re-generated model elements be replaced in the big UML model.

This means that the Xtext editor's Resource will have a transient in-memory Model element as its root element, and this element will nest the actual elements that are contained by the big UML model, in the other Resource. Logically, the model elements generated inside the Activity are contained by the big UML graphical editor (which is required in order to properly validate the UML model), and the model elements are to be saved inside the Xtext Resource.

We need to use temporary containment proxies in the big UML model, under the Classifier, and keep the actual elements in the Xtext Resource. This will happen only at runtime, while the editing is in process. When a save action occurs, the Xtext Resource will save the textual representation in the stereotyped Comment and will replace the Activity from the Classifier from a proxy Activity to the actual Activity, thus performing the actual change in the big UML model. At this point, the UML model will contain the new elements generated with the textual editor, which may be explored with the UML editors or used with other tools (code generators, for instance).

5. Related Work

Being able to integrate different editors is a natural requirement for every development environment. However, what we present in this paper is strictly dependent on the Eclipse IDE.

There have been Eclipse projects where the textual and graphical editors have been mixed. TEF [2] provides a textual editor integrated as a popup inside the graphical editor, using custom concrete and abstract syntaxes.

The Xtext framework also has two integration examples. In one example the graphical and textual editors both work on the whole model, which is not an acceptable approach, as the UML models may become large models and the Xtext parser is refreshing the whole model quite frequently. And in the other example the textual editor is integrated as a popup inside the graphical editor. The popup is still in an experimental phase and it doesn't work and cannot be used as a fully fledged textual editor.

6. Conclusions and Further Work

Creating reasonable sized executable UML models is hard, because the UML primitives from the UML Action Semantics package are too low level. The fUML standard provides a simplified subset of UML Action Semantics package and it also simplifies the context to which the actions need to be

applied. However, an easy-to-use concrete textual syntax is still needed in order to speed up the process of creating executable models.

In this article, we have presented a way to integrate a textual action language editor with the existing UML graphical editors in the Eclipse IDE. The action language for which we want to create a textual editor is Alf and it is standardized by OMG. Alf is based on fUML and it follows the structured programming principles, with a concrete textual syntax similar to the most popular object-oriented programming languages (e.g. Java, C++).

In the future, we plan to create a fully functional editor based on Alf action language and also to integrate it into our ComDeValCo framework. We already have a prototype textual editor that uses fUML for the abstract syntax and is implemented as an Eclipse plug-in using Xtext.

References

1. *Uml superstructure specification*, 2007.
2. *Textual editing framework (tef)*, 2008.
3. *Concrete syntax for uml action language (action language for foundational uml - alf)*, 2010.
4. *Semantics of a foundational subset for executable uml models (fuml)*, 2011.
5. Eclipse Foundation, *Xtext - language development framework*, 2011.
6. C.-L. Lazăr and I. Lazăr, *On Simplifying the Construction of Executable UML Structured Activities*, Studia Universitatis Babeş-Bolyai, Informatica **LIII** (2008), no. 2, 147–160.
7. C.-L. Lazăr, I. Lazăr, B. Pârv, S. Motogna, and I.-G. Czibula, *Using a fUML Action Language to construct UML models*, 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (2009), 93–101.
8. ———, *Tool Support for fUML Models*, International Journal of Computers, Communications & Control (2010), no. 2, 775–782.
9. I. Lazăr, B. Pârv, S. Motogna, I.-G. Czibula, and C.-L. Lazăr, *An Agile MDA Approach for Executable UML Structured Activities*, Studia Universitatis Babeş-Bolyai, Informatica **LII** (2007), no. 2, 101–114.
10. B. Pârv, S. Motogna, I. Lazăr, I.-G. Czibula, and C.-L. Lazăr, *ComDeValCo - a Framework for Software Component Definition, Validation, and Composition*, Studia Universitatis Babeş-Bolyai, Informatica **LII** (2007), no. 2, 59–68.

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

BAYES-NASH EQUILIBRIUM IN THE PRESENCE OF INFORMATION SOURCES: COMPUTATIONAL ISSUES

BAZIL PÂRV[1] AND ILIE PARPUCEA[2]

Abstract. This paper discusses computational issues of finding Bayes-Nash equilibrium (BNE) in the presence of information sources, which separate game-specific information from environment-based information. A general algorithm is given.

1. Introduction

Bayesian decision theory is concerned with the question of how a decision maker should choose a particular action from a set of possible choices if the outcome of the choice also depends on some unknown state (from the states of the world). In our approach, the decision maker is modeling the information received by the system (i.e. new information) as an *information source* [3]. A decision problem involves one or several information sources. We assume that each person is able to represent his beliefs, as the likelihood of the different n states of the information source, by a subjective discrete probability distribution [6].

The structure of this paper is as follows. After this introductory part, the next section contains background information and notations used throughout the paper. The third section contains a short description of the classical and proposed solution, as well as the steps of an original algorithm based on the externalization of information sources which computes Bayes-Nash equilibrium for a class of games with incomplete information. Finally, the last section draws some conclusions and outlines future research.

Received by the editors: April 8, 2011.

2000 *Mathematics Subject Classification.* 62C10, 91B24.

1998 *CR Categories and Descriptors.* I.6.5 [**Simulation and modeling**]: Model Development – *Modeling methodologies*; H.4.2 [**Information systems applications**]: Types of systems – *Decision support*.

Key words and phrases. game with incomplete information, Bayes-Nash equilibrium, information source.

2. Background material

Consider a *game with incomplete information* (as in [2]), denoted by:
$\Gamma_t = (I, (F_i)_{i\in I}, (\Pi_t^i(f,\theta))_{i\in I}, (\Theta_i)_{i\in I}, \mu_t)$,
where:

- I is the set of players, $|I| = m$,
- F_i is the strategy set for player $i, i = \overline{1,m}$, and $F = F_1 \times F_2 \times \cdots \times F_m$ is the the set of all possible strategy profiles;
- $f = (f_1, f_2, \cdots, f_m) \in F$ is a joint strategy or strategy profile;
- Θ_i is the set of types for the player i, and $\Theta = \Theta_1 \times \Theta_2 \times \cdots \times \Theta_m$ is the joint type space;
- $\theta = (\theta_1, \theta_2, \cdots, \theta_m) \in \Theta$ is the joint type of all players;
- $\Pi_t^i(f,\theta)$ is the payoff function for player i at the moment t if the strategy f and the type combination θ are chosen. Note that the payoff for the player i may depend not only on its type θ_i, but also on the other players' type, denoted by θ_{-i}.
- μ_t - the probability distribution on the set Θ at the moment t. This is the uncertainty-dominated component of the game, which draws our attention in this paper.

In our exposition, we assume that type sets Θ_i are finite; consequently, Θ is a finite set also. $\mu_t(\theta), \theta \in \Theta$ denotes the probability of chosing type combination θ at the moment t. As in [5], we assume, without loss of generality, that players have incomplete information about their opponents' payoffs but have complete information about the strategies of all other players.

A strategy profile $f^*(\theta) = (f_1^*(\theta_1), f_2^*(\theta_2), \cdots, f_m^*(\theta_m))$ constitutes a *Bayes-Nash equilibrium* (see [2] and [4]) of a game Γ_t with incomplete information if the following inequality:

$$\sum_{\theta_{-i}\in\Theta_{-i}} \Pi_t^i(f_i^*(\theta_i), f_{-i}^*(\theta_{-i}), \theta_i, \theta_{-i})\mu_t(\theta_{-i}|\theta_i) \geq \sum_{\theta_{-i}\in\Theta_{-i}} \Pi_t^i(f_i(\theta_i), f_{-i}^*(\theta_{-i}), \theta_i, \theta_{-i}) \cdot \mu_t(\theta_{-i}|\theta_i)$$

holds for all possible players $i \in I$ and all types $\theta_i \in \Theta_i$ and all strategies $f_i \in F_i$.

The information source S_{t+1} is a possible probability distribution of states at the moment $t+1$. Γ_{t+1}, the game with incomplete information at the moment $t+1$, based on S_{t+1}, can be defined recursively as follows:

$\Gamma_{t+1} = \Gamma_t(S_{t+1})$, where $\Gamma_{t+1} = (I, (F_i)_{i\in I}, (\Pi_t^i(f,\theta))_{i\in I}, (\Theta_i)_{i\in I}, \mu c_t)$.

The probability distribution μ_t conditioned by the information source S_{t+1}, denoted by μc_t, is the probability distribution μ_t updated by the information

source S_{t+1}. The game Γ_{t+1} is the updated game Γ_t based upon S_{t+1}. This information source updates probability distribution μ_t on Θ and thus the equilibrium of Γ_t is modified.

The above notations allow us to define the *Bayes - Nash equilibrium* of the game Γ_{t+1} as a list of decision functions $(f_1^*(.), \cdots, f_m^*(.))_{t+1}$, such that for all possible players $i \in I$ and all types $\theta_i \in \Theta_i$, the inequality:

$$\sum_{\theta_{-i} \in \Theta_{-i}} \Pi_t^i \left(f_i^*(\theta_i), f_{-i}^*(\theta_{-i}), \theta_i, \theta_{-i}\right) \cdot \mu c_t(\theta_{-i}|\theta_i) \geq$$
$$\sum_{\theta_{-i} \in \Theta_{-i}} \Pi_t^i \left(f_i, f_{-i}^*(\theta_{-i}), \theta_i, \theta_{-i}\right) \cdot \mu c_t^i(\theta_{-i}|\theta_i)$$

holds for all strategies $f_i \in F_t$.

The equilibrium of Γ_{t+1} can differ from the equilibrium of Γ_t due to the information in S_{t+1}. For a given player $i \in I$ the updated equilibrium of Γ_{t+1} is:

$f_i^*(.) = \sum_j f_{ij}^*(.) \cdot p^j(t)$,

where $f_{ij}^*(.)$ is the equilibrium of player i for the state s^j of the information source S_{t+1} and $p^j(t)$ is the probability that $S_{t+1} = s^j$.

3. Computing the equilibrium

3.1. **Classical approach.** According to [7], the computing of equilibrium for a game with incomplete information involves the following steps:

(1) Specify a computational mechanism;
(2) Generate candidate strategies;
(3) Estimate the empirical game;
(4) Solve the empirical game;
(5) Analyze the results.

One of the drawbacks of this approach is that it does not emphasize the dynamics of uncertainty, i.e. the transformation $\mu_t \to \mu_{t+1}$ is hidden. Once the equilibrium is computed at moment t, it is hard to decide when to compute it again (i.e. to decide when the moment $t+1$ is arrived).

The transition $t \to t+1$ is purely formal: the re-computing of equilibria is performed each time it is needed; this is not discussed explicitely in the literature. Our proposal, i.e. the externalization of information sources offers a more precise trigger to re-compute equilibria.

3.2. **Externalization of information.** The proposed approach models the information environment with the help of *n information sources*. Input data considered are the same as in classical approach: historical data, player data,

including payoff functions and strategies, probability distribution $\mu_t = \mu(t)$ and the computed equilibrium at the moment t. Our approach considers the process of updating probability distribution at the $t+1$ moment, $\mu_{t+1} = \mu(t+1)$, as a separate step; this update process uses estimates of information sources for the $t+1$ moment.

The most important thing here, described using a simple example in [8], is the way in which the uncertain component of the game, μ_t is updated (or estimated). The intrinsic uncertainty of this component is due to the fact that information which influences its probability distribution is not entirely known. The solution of the game at the moment $t+1$ is given by the computing of the component μ_{t+1}, which is (seems to be) an update of the distribution μ_t based on the predictions regarding information sources for the moment $t+1$.

This way, the game at the moment $t+1$ is the game at the moment t with the uncertain component μ updated: $\Gamma_{t+1} = \Gamma_t(\mu_t \to \mu_{t+1})$.

The transition $t \to t+1$ is purely formal and has the following semantics: when the state of an information source (i.e. the probability distribution of the random variable associated to it) is changed, the equilibrium needs to be re-computed.

Algorithm 1: Computing Bayes-Nash equilibrium in the presence of information sources

---

Data: $m \in \mathbf{N}$ number of players; $n \in \mathbf{N}$ number of information sources; $F_1, F_2, \cdots, F_m$ strategies of the players; $\Theta_1, \Theta_2, \cdots, \Theta_m$ types of the players; $(\mu c_0^1, \mu c_0^2, \cdots, \mu c_0^n)$ probability distributions of states of information sources; $(\overline{p_1}, \overline{p_2}, \cdots, \overline{p_m})_{t=0}$ average prices; $\Pi_i, i = \overline{1,m}$ payoff functions; T prognosis (planning) horizon

Result: $Sol = \{(p_1^*, p_2^*, \cdots, p_m^*)_t, t = \overline{1,T}\}$.

begin

  $t \longleftarrow 0$

  while $t < T$ **do**

    { *Step* 1: estimate $\mu c_{t+1}^j, j = \overline{1,n}$ and $(\overline{p_1}, \overline{p_2}, \cdots, \overline{p_m})_{t+1}$}

    { *Step* 2: build the complex information source SC_{t+1}}

    { *Step* 3: update the probability distributions $\mu c_{t+1}^j, j = \overline{1,n}$}

    { *Step* 4: rebuild the payoff functions $\Pi_i, i = \overline{1,m}$}

    { *Step* 5: compute the equilibrium prices $(p_1^*, p_2^*, \cdots, p_m^*)_{t+1}$}

    $t \longleftarrow t+1$

Remarks

(1) Each player produces a single product, slightly different from the products provided by the other players. The strategies of the players are represented by the equilibrium prices $(p_1^*, p_2^*, \cdots, p_m^*)_t$ for their products at each moment t in the planning horizon, $t = \overline{1,T}$.
(2) In *Step* 3, probability distributions μc_{t+1}^j are conditioned by the state of the complex information source SC_{t+1}.
(3) In *Step* 4, if payoff functions are given in analytical form, they need to be rebuilt, as shown in [8].
(4) The estimation of the equilibrium in *Step* 5 depends on the form of payoff functions. If they are continuously differentiable with respect to the prices and an analytical solution can be found, as shown in [8], then the solution is straightforward. If payoff functions are given in table form, or an analytical solution is not known, the general approach is to reduce the game Γ_{t+1} to a game with complete information (as discussed in [1]) and to compute its Nash equilibrium using known algorithms. This single step is subject to recent research for particular games.
(5) Another key issue in implementing this algorithm is the representation of the components of the game. Simple examples involving two players and several information sources are easy to manage, but the space required is growing at least polynomial in the number of players and information sources considered.

4. Conclusions and further work

The essential difference between the classical approach and the one proposed in this paper is given by the separation of the information external to the game from the game-specific information. This separation follows the *separation of responsibilities* principle. This way, both external and internal elements of the game are easier to model and understand.

The classical approach does not make any distinction between these two categories of information; more precisely, the influence of external information on the uncertainty that dominates the game is not taken into account or quantified. By splitting the game information into external and internal, the former being modeled by information sources, the influence of external environment on the variation of the solution is better captured and quantified. This provides a better evaluation of the contribution of individual factors to the predicted equilibrium.

Another advantage of this separation is that it allows a better, easier calibration of the model, by comparing the computed equilibrium with real solution, taken from historical data.

Further work will include the detailed study of concrete problems involving games with incomplete information, and comparing the results obtained using classical and proposed approaches. The general algorithm presented here will be implemented in several game-specific applications.

Acknowledgements

This work was supported by the grant ID_2586, sponsored by NURC - Romanian National University Research Council (CNCSIS).

References

[1] Ceppi, S., Gatti, N., Basilico, N. Computing Bayes-Nash Equilibria through Support Enumeration Methods in Bayesian Two-Player Strategic-Form Games, *Proc. of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops*, IEEE Press, pp. 541-548.

[2] Eichberger, I., *Game Theory for Economists*, Academic Press, 1993.

[3] Florea, I., Parpucea, I., *Economic Cybernetics* (Romanian), Babeş-Bolyai University, 1993.

[4] Fudenberg, D., Tirole, J., *Game Theory*, MIT Press, 1991.

[5] Harsanyi, J.C., Games with Incomplete Information Played by Bayesian Players, Parts I, II, III, *Management Science*, 14, 1967, pp. 159-182, 320-334, 486-502.

[6] Hirshleifer ,J., Riley J. G., *The Analytics of Uncertainty and Information*, Cambridge University Press, 1995.

[7] Mackie-Mason, J.K., Wellman, M.P., Automated Markets and Trading Agents, chapter 28 in Tesfatsion, L. and Judd, K.L. (eds) *Handbook of Computational Economics*, Vol. 2, Elsevier, 2006.

[8] Parpucea, I., Pârv, B., Socaciu, T. Modeling Uncertainty in a Decision Problem by Externalizing Information, *Int. J. of Computers, Communication, and Control*, Vol. 6 (2011), No. 2, pp. 328-336.

(1) Babes-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu str., Cluj-Napoca 400084, Romania

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

(2) Babeş-Bolyai University, Faculty of Economic Sciences and Business Administration, 58-60 Teodor Mihali str., Cluj-Napoca 400591, Romania

E-mail address: ilie.parpucea@econ.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

WHY MUST WE TEACH VERIFICATION AND VALIDATION TO UNDERGRADUATES?

MILITON FRENŢIU AND FLORIN CRĂCIUN

Abstract. The way program verification is taught is firstly described. The verification methods are shortly presented and the objectives of the Verification and Validation (V&V) course are discussed. The main gain on the students education is finally underlined.

1. Introduction

The most important property of a program is whether it accomplishes the intentions of its user, i.e. if it is correct. We have all observed the permanent and rapid growth of computer usage in all fields of human activity. The need for new programs is immense and their complexity is continuously growing. There are known programs with millions of lines written by hundreds of people. These more complex programs cannot be developed like the old small ones. It has become necessary to analyse software costs over the entire life cycle of the system. It is known today that the major errors are due to poor design of the system not to bad coding. After all, the fraction of time needed for programming is about 20% of the entire development process.

The need for more reliable software [20, 25, 14], and the role of Verification and Validation [3, 15, 35] are well known. Investigations have shown that formal verification procedures would have revealed the exposed defect in, e. g., the Ariane-5 missile, Mars Pathfinder, Intels Pentium II processor, or the Therac-25 therapy radiation machine [5].

Today software systems have become essential parts of our daily life. Computers are used in all fields of human activities. More and more programs are needed, and their complexity increases continuously. Software critical systems [1] require error-free programming. To obtain programs without errors we

Received by the editors: April 8, 2011.

2010 *Mathematics Subject Classification.* 68Q60, 68N30.

1998 *CR Categories and Descriptors.* D.2.4 [**Software**]: Software engineering – *Software/Program Verification*; D.2.5 [**Software**]: Software engineering – *Testing and Debugging*.

Key words and phrases. verification and validation, education.

need a new way of writing programs, new people, much better educated, able to do this.

There is a contradiction between the desire to obtain a system as quickly as possible, and to have a correct system. The experience shows that more than 75% of finished software products have errors during maintenance, and deadlines are missed and the cost overrunning is a rule, not an exception. It was estimated [34] that more than 50% of the development effort is spent on testing and debugging. Nevertheless, some errors are not detected by testing, and some of them are never detected. More, there are projects that have never been finished [8]. And it is not an exception; it is estimated that from each six large projects two of them are never finished.

Almost all students have learned some programming at school. They think they know what programming is all about. They express resentment when they are forced to document their activity, to design the program, or to think to the correctness of their programs. They go directly to computer and introduce their programs, and then run them. If the first execution seems OK they are satisfied. They don't like to think first to the algorithm and to describe it on the paper. Also, they are not used to test seriously their programs. We need to fight with these people, to change their habit, to educate them in a different way. That is why we have to stress from the very beginning that "programming is a high intellectual activity" [19], that "they must think first, program later" [23], that the correctness is the most important property of good programs.

2. The contents of Verification and Validation Course (V&V)

The validation and verification of software systems is a major issue in Software Engineering. The objective of this course is to train future computer scientists and engineers in the fundamental concepts on which verification and validation of software systems are based. Though the orientation of the course is practical in nature, its goal is to teach the fundamental principles that are needed to build reliable systems needed in various fields.

V&V is one of the important courses that contribute to obtain well-educated practitioners. We teach such a course to the third year undergraduates [15]. The theoretical basis for building reliable software products is given here. The course consists of three main parts:

- the theory of program correctness;
- the methods of verification and validation;
- the consequences on software engineering practice.

The entire curricula of this course, and also, the undergraduate study program may be seen at [31].

One cannot understand V&V if she/he does not know the concept of program correctness. We are confident that we cannot prove the correctness of a large program with one million lines. But, as Dijkstra [4] underlined, it is necessary to use perfect (correct) simple algorithms for building reliable large programs (but it is not sufficient). The first part of the course presents this concept and gives methods to prove correctness. The Floyd and Hoare methods [10, 18] to prove correctness are given. Then, more important, the accent is put on the methods to achieve this correctness by refinement from specifications. Also, the roles of Dijkstra [6], Hoare [18], Gries [17], Droomey [7], and Morgan [27] are mentioned.

The verification methods discussed in the second part are: proving correctness (already mentioned), testing, inspection, symbolic execution, and model checking. Testing is the oldest verification method and often the only one used. Symbolic execution [22] although similar to testing, differs from testing by the values given to the input variables, which are symbols not concrete values.

Inspection, introduced by Fagan [9], with all later variants (peer reviews [33], walkthroughs [30], active design reviews [29]) is a newer V&V method, as well as Model Checking [2, 21]. Inspections can find up to 80% of defects, much earlier than testing. The formal process developed by Fagan was improved by Gilb [16], who considers that the main advantage of inspection over testing is its possibility of process improvement. The feedback obtained from inspections is useful to eliminate defects and bad habits from the development processes. Also, the cost of eliminating an error by testing is 14.5 times higher than the cost of eliminating it by inspection [32].

At the undergraduate level we cannot afford to teach the mathematical basis of model checking, the theory that lies at the basis of model checking. Instead, the main theoretical aspects are presented in a two hours lecture, and the existence of tools and examples of such tools are shortly presented [15].

It is underlined that all of these methods must be practiced during the software process [15]. Their usage may be informal, or more formal, complete or only some of them, depending on the type of the system which is built. For safety-critical systems all of the above mentioned methods must be used. We consider that all future software engineers should be aware of all verification methods.

The third part of the course presents the Cleanroom methodology [26, 24], the role of V&V for Software Quality Assurance and Software Process Improvement, and the consequences of correctness theory on software engineering practice [11, 14, 15].

3. Consequences of teaching Verification on Software Engineer

We think that we educate our students for their future profession, that last for several decades. Also, they must be prepared for changes in software engineering. They need to acquire now the knowledge needed to build more reliable systems. Certainly, proving correctness of algorithms is not enough to obtain more reliable systems, but it is necessary. For years we ask the students to understand and to respect some important rules of programming [11, 13, 14, 23]. Many of them are simple consequences of the theory of program correctness [11].

The fact that program verification is a very important activity which extends from the first statement of the problem, until the end of the project is repeated many times. And a special attention was given to the inspection of all documents [9, 28]. The students must hear from the beginning that testing is not enough, that inspection of all phases may be more useful.

The future software tester must be well prepared to test software products, the member of the verification team must be an expert in verification activities. Not all students will work in a verification team, but all of them must contribute to build reliable systems.

So, why must we teach Verification and Validation? The knowledge acquired teaching a V&V course have serious consequences on the entire activity of a software engineer. These consequences may be extracted as important rules that must be obeyed by all participants in the software process. We mentioned here only few of them:

- Prevention is better than cure!
- Think first, program later!
- Do it right the first time!
- Prove the correctness of your algorithms!
- Write documentation for all software activities!
- Inspection is superior to testing!
- Discover and Remove the errors as early as possible!
- Use comments to document your code. Use assertions.
- Pay attention to the clarity of your software documents!
- Respect the standards!

4. Conclusions

We need confidence in the quality of our software products. We need to educate the future software developers in the spirit of producing correct, reliable systems. For this we must teach students to develop correct programs. We are aware that usually programmers do not prove the correctness of their programs. There always has to be a balance between cost and the importance

of reliability of the programs. But just when the well educated people do not prove the correctness, their products are more reliable than the products of those "programmers" who never studied program correctness. Therefore, we consider that the students must hear, and must pay attention to the correctness of their products.

From first year, students are taught about program specification and design. The entire life-cycle is presented, the importance of documentation for all steps is underlined. Top-down and the other programming methods are taught, and the students hear that the design is more important than coding. Each part of the design must be specified, the code must be explained by comments, the comments should be neither more nor less than needed. Since we also observed that students do not like to write comments [12] we tried to explain their necessity, and to force them to document the code [13].

References

1. R. W. Butler and S. C. Johnson, *Formal Methods For Life-Critical Software*, Computing in Aerospace 9 Conference (San Diego, California), 1993, pp. 319–329.
2. E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 1999.
3. E. M. Clarke and J. M. Wing, *Formal Methods: State of the Art and Future Directions*, ACM Computing Surveys **28** (1996), no. 4, 626–643.
4. O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*, American Press, 1972.
5. N. Dershowitz, *Software Horror Stories*, www.cs.tau.ac.il/∼nachumd/horror.html.
6. E. W. Dijkstra, *Guarded Commands, Nondeterminacy and Formal Derivation of Programs*, Communications of the ACM **18** (1975), no. 8, 453–457.
7. G. Dromey, *Program Derivation. The Development of Programs from Specifications*, Addison Wesley, 1995.
8. Oz Effy, *When Professional Standards are LAX. The CONFIRM Failure and Its Lessons*, Communications of the ACM **37** (1994), no. 10, 29–36.
9. M. Fagan, *Design and Code Inspections to Reduce Errors in Program Development*, IBM Systems Journal **15** (1976), no. 3, 182–211.
10. R. W. Floyd, *Assigning Meanings to Programs*, Mathematical aspects to computer science. Proc.Symposium in Applied Mathematics (J. T. Schwartz, ed.), no. 19, American Math. Soc., 1967, pp. 19–32.
11. M. Frentiu, *On Program Correctness and Teaching Programming*, Computer Science Journal of Moldova **5** (1997), no. 3, 250–260.
12. ———, *The Impact of Style on Program Comprehensibility*, Proceedings of the Symposium Zilele Academice Clujene (2002), 7–12.
13. ———, *On programming style*, www.cs.ubbcluj.ro/∼mfrentiu/articole/ style.html, 2003.
14. ———, *Correctness, A Very Important Quality Factor in Programming*, Studia Univ. Babeş-Bolyai, Seria Informatica **L** (2005), no. 1, 12–21.
15. ———, *Verificarea si Validarea Sistemelor Soft*, Ed. Presa Universitara Clujeana, Cluj-Napoca, 2010.
16. T. Gilb and D. Graham, *Software Inspection*, Addison-Wesley, 1993.
17. D. Gries, *The Science of Programming*, Springer Verlag, 1981.

18. C. A. R. Hoare, *An Axiomatic Basis for Computer Programming*, Communications of the ACM **12** (1969), no. 10, 576–580.
19. ———, *The mathematics of programming*, Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, 1985, pp. 1–18.
20. C. M. Holloway, *Why Engineers Should Consider Formal Methods*, 16th AIAA/IEEE Digital Avionics Systems Conference, vol. 1, 1997, pp. 1.3–16–1.3–22.
21. J. P. Katoen, *Principles of Model Checking*, MIT Press, 2008.
22. J. C. King, *Symbolic Execution and Program Testing*, Communications of the ACM **19** (1976), no. 7, 385–394.
23. H. F. Ledgard, *Programming Proverbs for Fortran Programmers*, Hayden Book Company Inc., New Jersey, 1975.
24. R. C. Linger, *Cleanroom Software Engineering for Zero-Defect Software*, 5th Intern. Software Engineering Conference (Baltimore), IEEE Comp. Soc. Press, 1993, pp. 1–13.
25. B. Meyer, *Software Engineering in the Academy*, IEEE Computer **34** (2001), 28–35.
26. H. Mills, M. Dyer, and R. Linger, *Cleanroom Software Engineering*, IEEE Software **4** (1987), no. 5, 19–25.
27. C. Morgan, *Programming from Specifications*, Springer, 1990.
28. A. Myers, *A Controlled Experiment in Program Testing and Code Walkthroughs Inspection*, Communications of the ACM **21** (1978), no. 9, 760–768.
29. D. L. Parnas and D. M. Weiss, *Active Design Reviews: Principles and Practices*, 8th Intern. Conf. on Software Engineering, 1985, pp. 215–222.
30. S. R. Schach, *Software Engineering*, IRWIN, Boston, 1990.
31. UBB, *Undergraduate Study Program, Babeş-Bolyai University*, http://www.cs.ubbcluj.ro, 2011.
32. K. E. Wiegers, *Improving Quality Through Software Inspections*, Software Development **3** (1995), no. 4, 55–64.
33. ———, *Seven Truths About Peer Reviews*, Cutler IT Journal (2002).
34. E. Yourdon, *Modern Software Analysis*, Yourdon Press, Prentice Hall Building, New Jersey, 1989.
35. M. V. Zelkowitz, *Role of Verification in the Software Specification Process*, Advances in Computers, no. 36, Academic Press, 1993, pp. 43–109.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania

E-mail address: mfrentiu@cs.ubbcluj.ro, craciunf@gmail.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

DP AUTOMATA AND PETRI NETS

MONICA KERE

Abstract. At the first view there are no similarities between P systems and Petri nets, but this paper describes a connection between them. Petri nets can describe a very large domain of systems. P systems represent a computational model that describe the interaction between chemical processes and their membranes. The dP automata are a class of membrane systems that tries to combine a number of P systems and makes them to communicate. The behavior of such an automaton is made in an non-deterministically maximally parallel way. The objects are moving inside each P system and also between the P systems from the automaton.

This paper will use on the modeling process a place transitions network (PTN). Graphical symbols from PTN will be used to describe the behavior of a dP automaton(dPA). The properties from the PTN will be define for the dP automata providing that an automaton is having the same expressing power as a Petri net.

1. Introduction

The domain of P systems and membrane computing was very much investigated from the beginning. A class of P systems, called dP automata, was introduced in the recent paper [6]. This automaton tries to combine n P systems and makes them to communicate. The P systems from inside a dPA are called components and the rules through which they communicate are called communicating rules. Each P system behaves according to its rules. Those rules are symport/antiport rules. All the P systems from the dPA are interchanging objects from their skin membranes, using the communicating rules, that are also symport/antiport rules. So a dPA will have the rules that are applied for each P system and the rules applied for all the P systems. All the rules are applied in a non-deterministic parallel manner. The environment

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68Q45, 68Q60.

1998 *CR Categories and Descriptors.* D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Petri nets*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation – *Alternation and nondeterminism.*

Key words and phrases. dP automata, Petri nets, P systems.

is common for all the P systems and it contains objects available in arbitrarily many copies.

The formalization of the operations of the symport/antiport rules, as in [5], can be realized as:

- *(ab, in)* or *(ab, out)* are symport rules, which means that a and b pass together through a membrane entering in the former case and exiting in the latter. This rule doesn't allow a or b to pass separately in/out the membrane. For this case it is necessarily another rule like *(a, in)* or *(a,out)* named uniport.

- *(a, in; b, out)* represents an antiport rule, which means that a enters and at the same time b exists the membrane.

Each component can take an input, work on it, communicate with other components, and provide the answer to the problem in the end of a halting computation. For a dPA a configuration is represented through a vector with n components $C = \{L_1, ..., L_n\}$, the component i is the multiset of objects from the P system $i, i = 1, .., n$. A halting computation will accept the string $x_1...x_n$ over O and starting from the initial configuration and applying the internal rules, but also the communicating rules in a non-deterministically maximally parallel manner, take from the environment the substrings $x_1, ..., x_n$, respectively, and eventually halts.

2. dP Automata Properties

To provide some connections between Petri nets(PNs) and dP automata we represent such an automaton using a Place Transition Petri net. The objects will be represented as places with tokens corresponding to the number of copies and the rules will be drawn as transitions. The places are drawn as cycles, the transitions as squares and the connections between them by arrows. The example below shows how a dPA can be modeled using a PTN.

Below we analyze the behavioral properties of a dPA modeled by a Petri net. Important properties for the PTN are described in [4]. These properties can be investigated also for the dP automata. The description of the behavioral properties like *terminating, deadlock-free, boundedness* and *liveness* for a P system is done in [7]. Because a dPA is also a P system, and because a P system modeled with a PTN is still a Petri net, we can introduce these properties as follows below.

Definition 1: For a given dP automaton, we define the following properties:

i) A dPA is *terminating* if the sequence of transitions between configurations is finite.

ii) A configuration C_n in a dPA is said to be *reachable* if there exists a sequence of transactions that transforms C_0 in C_n.

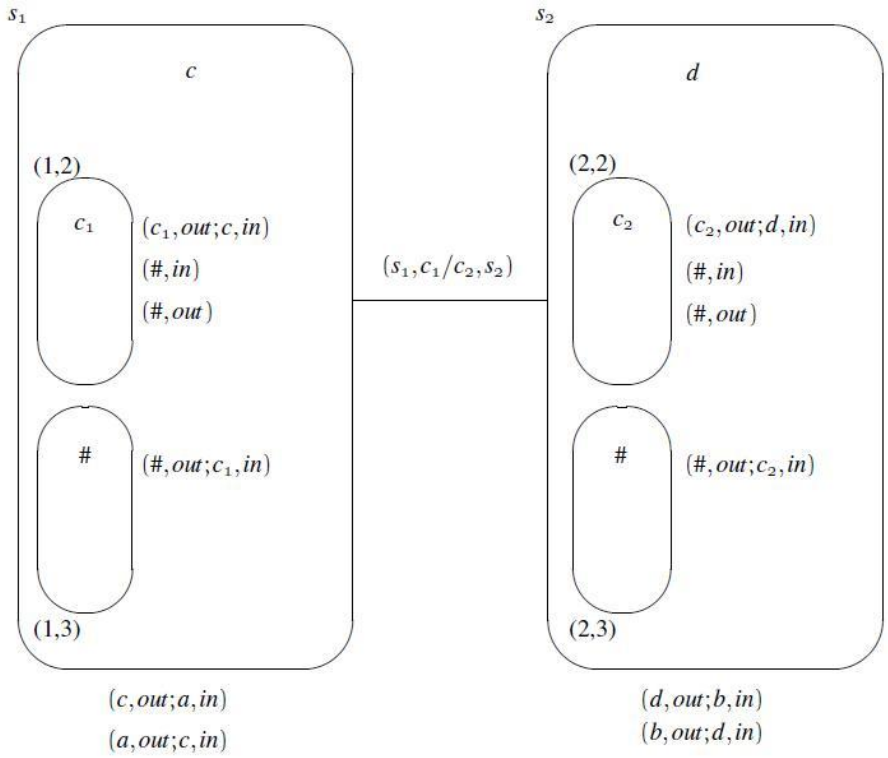


FIGURE 1. The dP automaton [1]

iii) *Boundedness* can be seen in two ways: local and global. *Local boundedness* suppose that the number of copies of each object in each region to be less than a given integer k. For a given integer k, the dPA is said to be *global bounded* if the number of copies of each object in each component is less than k for every configuration reachable from C_0. A dPA is said to be safe if $k = 1$.

iv) A rule is said to be dead in the configuration C if it cannot be executed in any configuration reachable from C. A rule is said to be live if it is not dead in any configuration reachable from C_0. A configuration C is dead if there is no rule which can be executed in C. A dPA is said to be *deadlock-free* if there are no reachable dead configuration. A dPA is *living* if each rule is living.

v) A dPA is said to be *reversible* if for each configuration reachable from C_0 we can reach again C_0.

vi) Giving a configuration C minimum needed to apply a rule r, then r is potentially realized (r can be applied at least one time in some sequences of executions) if and only if C is *coverable*.

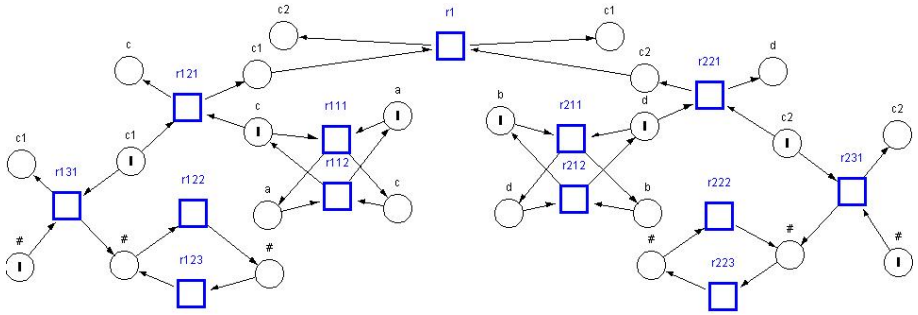


FIGURE 2. The representation using Petri nets for the dP automaton from Fig. 1

vii) A dPA is said to be *persistent* if, for any two rules that can be realized, the execution of one will not disable the other one.

Because a dPA modeled with a PTN is also a PTN and because those properties are given for the PTN and using the definitions from Def. 1 we get the following theorems:

Theorem 1 If the PTN for a given dPA terminates, then the dPA terminates.

Proof: If the dPA doesn't terminate, then there exists an infinite sequence. When the dPA is modeled using the PN, then there also exists an infinite sequence. Every sequence consists of some transaction-steps and send transactions and each of this is a one-to-one mapped to a transition in the PN. So the sequence of transitions in the PN is not finite. Thus the PN doesn't terminate.

Theorem 2 If the configurations in the PTN for a given dPA are reachable, then the configurations in the dPA are reachable.

Theorem 3 If the PTN for a given dPA is bounded, then the dPA is bounded.

Theorem 4 If the PTN for a given dPA has liveness, then the dPA has liveness.

Theorem 5 If the PTN for a given dPA is reversible, then the dPA is reversible.

Theorem 6 If the configurations in the PTN for a given dPA are coverable, then the configurations in the dPA are coverable.

Theorem 7 If the PTN for a given dPA is persistent, then the dPA is persistent.

The proofs for the Theorem 2, 3, 4, 5, 6, 7 are the same like for the Theorem 1.

To define some properties, we classify the dP automata as described below.

Definition 2: Subclasses of dP automata:

1) A dPA is said to be a *state machine* (SM) if in each rule there is exactly one object going out and exactly one object coming inside a region.

2) A dPA is said to be a *marked graph* (MG) if each object in each region is introduced by only one rule and is taken out through exactly one rule.

3) A dPA is said to be a *free-choice* (FC) if every object can either go out or can come inside the region through a unique rule.

4) A dPA is said to be an *extended free-choice* if two objects that are going out from a region through some common rules then all their rules for going out are common.

5) A dPA is said to be an *asymmetric choice* (AC) if two objects that have some rules for going out in common, then one of them has all the rules for going out of the other (and possibly more).

A dPA is said to be *ordinary* if the number of copies of any object in any rule is 1.

A nonempty subset of objects S in an ordinary dPA N is called a *siphon* if every rule that is having an object from S going out from a region is having an object from S entering in a region. If S is not contained under some configuration (doesn't has any copy of its objects), it remains like that under its successors.

A nonempty subset of objects Q in an ordinary dPA N is called a *trap* if every rule that is having an object from Q going inside a region is having an object from Q coming out a region. If Q is contained under some configuration (it has copies of its objects), it remains contained under its successors.

Definition 3: For a SM we define the following properties:

1) A SM is *living* iff is strongly connected and in the initial configuration contains at least one copy of an object.

2) A SM is *safe* iff the initial configuration has at most one copy of an object.

Definition 4: For a MG we define the following properties:

1) A MG is *living* iff in the initial configuration there is at least one copy of an object for each circuit.

2) A live MG is *safe* iff in the initial configuration there is exactly one copy of at least one object from each circuit.

Those properties have been defined for dP automata and they inherit them from Petri nets which describe the automata. Petri nets were more investigated than dP automata so we tried here to make the connection between Petri

nets and P systems stronger and to add more features for the new domain of automata.

3. Conclusions

P systems and Petri nets look like having nothing in common, but there are many things that make a connection between them. Some other authors in their papers, [2], [3], [7] emphasized this connection before. This paper is trying to make stronger the connection between P systems and Petri nets. A class of P systems, called dP automata, is used to be modeled using a Place Transition Network. Petri nets is a tool that can describe graphically distributed, concurrent systems. Graphical symbols from Petri nets are used to describe the dPA. Then some properties from PN are described for dPA providing this connection.

There is a lot of work to be done in this domain. Some other properties can be defined such as rechability criteria, coverability criteria. The analyze can be done using the coverability tree so that the properties can be checked just looking at the loops and dead-ends of this tree. Model Checking can be used to verify the properties defined here for a dPA. So a tool can be build for checking if an automaton can provide a property or not.

References

[1] Freund, R., Kogler, M., Păun, G., Pérez-Jimenéz, M.: *On the power of P and dP Automata*, Annals of Bucharest Univ. Mathematical-Informatics Series, 2010, in press.

[2] Frisco, F.: *P Systems, Petri Nets, and Program Machines*. In: LNCS 3850, pp. 209-223, 2006.

[3] Kleijn, J. and Koutny, M.: *Synchrony and Asynchrony in Membrane Systems*. In: LNCS 4361, pp. 66-85, 2006.

[4] Murata, T.: *Petri Nets: Properties, Analysis and Applications*, vol 77, pp. 541 - 580, IEEE Computer Society (1989)

[5] Păun, A., Păun, Gh.: *The power of communication: P systems with symport/ antiport*. New Generation Computing, 20 (2002), 295-305

[6] Păun, Gh., Pérez-Jimnéz, M.J.: *Solving problems in a distributed way in membrane computing: dP systems*. Int. J. of Computers, Communication and Control, 5, 2 (2010), 238-252.

[7] Zhengwei Qi, Jinyuan You, Hongyan Mao: *P Systems and Petri Nets*. In: LNCS, vol. 2933, pp. 819-847. Springer, Heidelberg (2004)

University of Piteşti, Faculty of Mathematics and Computer Science, Department of Computer Science,, 110040 Piteşti, Str. Târgu din Vale, No.1, Argeş, Romania

E-mail address: monicakere@yahoo.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

GEODESIC DISTANCE-BASED KERNEL CONSTRUCTION FOR GAUSSIAN PROCESS VALUE FUNCTION APPROXIMATION

HUNOR JAKAB

Abstract. Finding accurate approximations to state and action value functions is essential in Reinforcement learning tasks on continuous Markov Decision Processes. Using Gaussian processes as function approximators we can simultaneously represent model confidence and generalize to unvisited states. To improve the accuracy of the value function approximation in this article I present a new method of constructing geodesic distance based kernel functions from the Markov Decision process induced graph structure. Using sparse on-line Gaussian process regression the nodes and edges of the graph structure are allocated during on-line learning parallel with the inclusion of new measurements to the basis vector set. This results in a more compact and efficient graph structure and more accurate value function estimates. The approximation accuracy is tested on a simulated robotic control task.

1. Introduction

The majority of real-life control problems including robotic locomotion requires the efficient handling of continuous and high dimensional state and action spaces, therefore function approximation needs to be employed. In real-life control tasks the ability to handle uncertainties arising from noisy measurements is a deciding factor in terms of performance and efficiency. Gaussian processes (GP) can be used efficiently for the approximation of value functions on continuous state spaces. The nonparametric nature of GP's provides increased flexibility and the resulting fully probabilistic model can be used for appropriate uncertainty treatment. One of the major drawbacks of using GP action-value function approximation with Euclidean distance-based kernel functions is the fact that they cannot accurately represent discontinuities. There are many reinforcement learning (RL) tasks where the state or

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68T05, 68T40,60J25.

1998 *CR Categories and Descriptors.* 1.2.6 [**Computing Methodologies**]: Artificial Intelligence – *Learning*; 1.2.9 [**Computing Methodologies**]: Artificial Intelligence – *Robotics*.

Key words and phrases. Reinforcement learning, Gaussian processes.

state-action value function corresponding to the actual policy is discontinuous in some regions of the space. This discontinuity has great influence on the algorithms performance. In this paper I describe a modality to increase the accuracy of our GP action-value function estimates by introducing a new modality of constructing kernel functions that operate on a graph structure induced by the Markov decision process (MDP) underlying the RL problem. Unlike in [7] the nodes of the MDP induced graph structure are allocated dynamically parallel to the inclusion of new basis vectors in the GP estimator. The resulting graph gives a compact representation of the most important points of the state space. Replacing Euclidean distance in the kernel function with distances defined on the paths between data-points from the MDP induced graph leads to more accurate value function approximations.

2. Notation and background

A formal representation of a reinforcement learning problem is given using Markov decision processes [4]. An MDP is a quadruple $M\left(S, A, P, R\right)$ with the following elements: S is the set of states; A the set of actions; $P(s'|s,a) : S \times S \times A \to [0,1]$ the transition probabilities, and $R : S \times A \to \mathbb{R}$, $R(s,a)$ the reward function. Calculating value functions for a policy π is essential for all value-based reinforcement learning algorithms. Value functions measure the long-term usefulness of a given state or state-action pair based on the cumulative reward received when starting out in that state and following a given policy[1] [8]: $V^{\pi}(s) = E_{\pi}\left(\sum_{t=0}^{\infty} \gamma^t r_t | s_t = s\right)$

Different approaches of using Gaussian processes for estimating value functions have been proposed[2, 5, 3] . In our approach we use as training data the states visited during different episodes of the experiment, and the corresponding – possibly – discounted cumulative rewards $\overline{V}(s_t) = \sum_{i=0}^{H-t} \gamma^i r_{t+i}$ as noisy targets.[2] The model is a GP, completely specified by its mean and covariance function, and it can be used to perform regression directly in a function space, with the resulting $\hat{V}(s)$ being the approximation to the state value function. The elements of the kernel matrix $\boldsymbol{K}$ are $\boldsymbol{K}(i,j) = k(s_i, s_j)$, where k is the kernel function operating on state variables. Having processed n points we have a GP built on the data set $\mathcal{D} = [(s_i, \overline{V}_i)]_{i=1,n}$ which is also called the set of basis vectors (BV). To estimate the value of a new state, s_{n+1}, we compute the predictive mean (1) and variance (2) functions conditioned on the data,

[1]For ease of visualization throughout the article I use state-value functions, but the presented methods naturally apply also to state-action value functions.

[2]We assume that the targets have Gaussian noise with equal variance; one can easily use different *known* noise variance.

given by the posterior GP [6]:

$$\hat{V}_{n+1}|\mathcal{D} \sim \mathcal{N}\left(\mu_{n+1}, \mathrm{cov}(s_{n+1})\right)$$

$$\mu_{n+1} = \boldsymbol{k}_{n+1}\boldsymbol{\alpha}_n \tag{1}$$

$$\mathrm{cov}(\hat{V}_{n+1}, q_{n+1}) = k\left(s_{n+1}, s_{n+1}\right) - \boldsymbol{k}_{n+1}\boldsymbol{C}_n\boldsymbol{k}_{n+1}^T, \tag{2}$$

where $\boldsymbol{\alpha}_n$ and $\boldsymbol{C}_n$ are the parameters of the posterior GP:

$$\boldsymbol{\alpha}_n = [\boldsymbol{K}_q^n + \boldsymbol{\Sigma}_n]^{-1}\overline{V}, \quad \boldsymbol{C}_n = [\boldsymbol{K}_q^n + \boldsymbol{\Sigma}_n]^{-1}. \tag{3}$$

with $\boldsymbol{\Sigma}_n = \boldsymbol{\sigma} I_n$ the covariance of the observation noise and $\boldsymbol{k}_{n+1}$ a vector containing the covariances between the new point and the training points:

$$\boldsymbol{k}_{n+1} = [k(s_1, s_{n+1}), \ldots, k(s_n, s_{n+1})]. \tag{4}$$

The above described method leads to good value function estimates when there are no significant discontinuities in the true value function. Figure 1 shows a comparison between TD and GP approximated value functions for the inverted pendulum (sec.4) balancing task in case of a fixed policy.

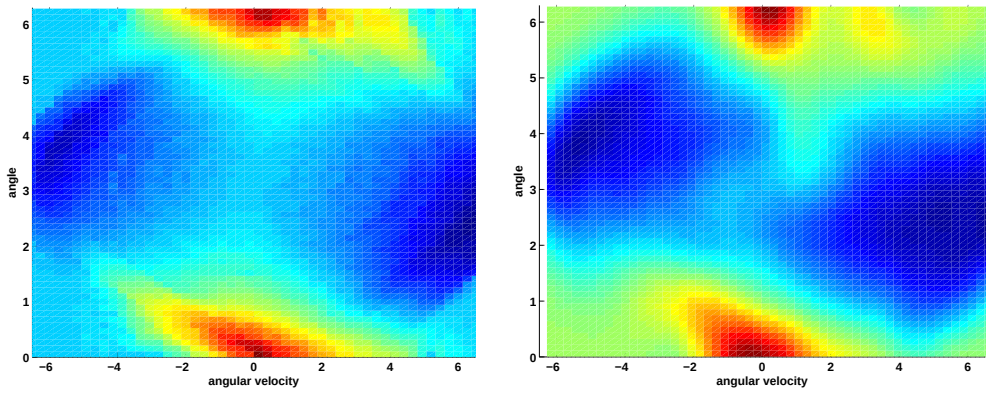


FIGURE 1. Color map of estimated value functions in case of (a) TD approximation , (b) GP approximation

3. GEODESIC DISTANCE BASED ON-LINE KERNEL CONSTRUCTION

To improve the ability of our value function approximator to represent discontinuities a new distance measure for the kernel function k is needed. Let G denote a sparse graph structure induced by the MDP which we will define as follows: G is a graph that has n nodes where $n = |BV|$ is the number of basis vectors present in the GP value function approximator. The connection between these nodes are initialized parallel to the addition of each basis vector to the BV set of the GP. For this procedure sparse on-line GP value function approximation is being used. In this setting only those data-points are added to the BV set which provide significant information gain during the learning

process, thereby reducing the number of the GP parameters drastically. For details of sparse on-line updates consult [1].

Using the GP basis vectors as nodes in our graph construction makes sure that the graph structure remains sparse and the nodes are placed in important regions of the state space. The construction of the MDP induced graph structure during the learning process proceeds as follows: If at time-step t we perform a full update of the GP parameters, adding the data-point s_t to the set of basis vectors, we establish a new node in our graph structure and connect it to the existing graph according to the following rule:

(5)
$$d_{s_t,s_i} = \begin{cases} ED(s_i, s_t) & if s_i = \underset{s_j}{\operatorname{argmin}} \left(\exp\left[-\frac{\|s_{t-j} - s_t\|}{2\sigma_{GP}(s_t)} \right] \right), \\ 0 & \text{otherwise} \end{cases} \quad j = 1 \dots t-1$$

Here d_{s_t,s_i} denotes the connection weight between nodes t and i. I used the notation $ED(\cdot,\cdot)$ to denote the Euclidean distance between two points from the state-action space, and $\sigma_{GP}(s_t)$ to denote the predictive variance of the GP value function approximator at point s_t.

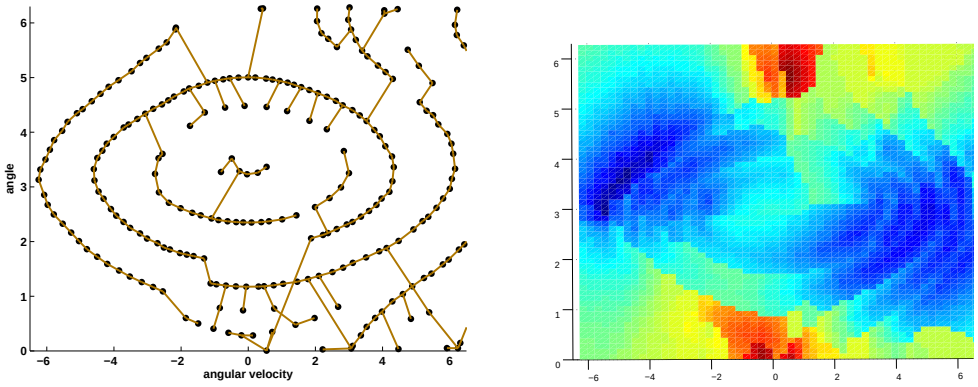


FIGURE 2. (a) Graph structure, (b) GP approx. with geodesic kernel

The small number of graph nodes makes it possible even in case of continuous state-action spaces to represent optimal distances between each node in a lookup table which leads to lesser computational costs. Figure 2.a presents the graph structure obtained after training the GP for a number of episodes on the inverted pendulum problem (sec.4), with a Gaussian policy and a fixed neural-network based controller. Shortest paths between nodes of the graph can be calculated efficiently using Dijkstra's algorithm. Based on this graph structure a new type of kernel function can be built which uses as a distance measure the shortest path between two data-points.

(6)
$$k(s, s') = \exp\left(\frac{SP(s, s')^2}{2\sigma^2} \right)$$

The definition of the shortest path exists only between data-points that are present in the GP basis vector set. In a continuous state-action space visiting the same state-action pair twice has very low probability, therefore we have to define our shortest path measure between two points as the distance between the two basis-vectors that are the closest to the data-points plus the distance of the data-points from these Basis vectors.

$$\begin{aligned} SP(s,s') &= ED(s,s_i) + SP(s_i,s_j) + ED(s_j,s') \\ s_i &= \underset{s_i}{\text{argmax}}(ED(s,s_i)) \quad i = 1 \dots n \\ s_j &= \underset{s_j}{\text{argmax}}(ED(s',s_j)) \quad j = 1 \dots n \end{aligned}$$

The expression for the predictive mean in eq.(1) can be regarded as a weighted linear combination of the value measurements in each basis vector. The weights given by $\boldsymbol{k}_{n+1}$ from eq.(4) using the newly defined covariance function from eq.(6) represent how far each basis vector is located on a sequence of states from the test-point.[3] This can also be regarded as a modified eligibility trace.

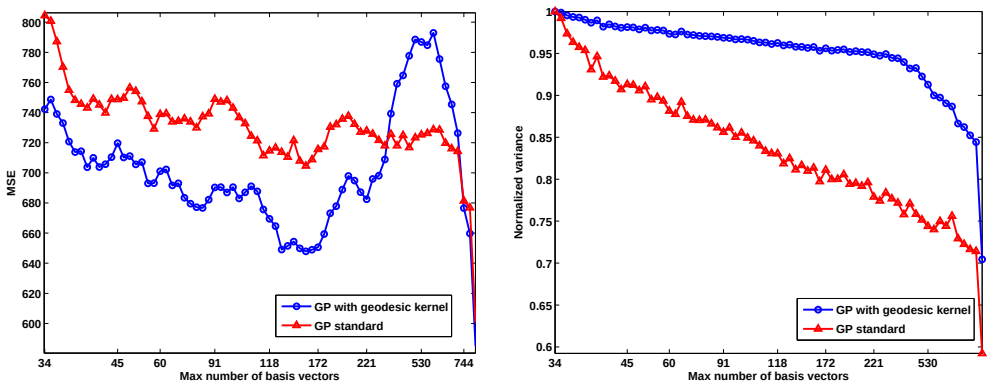


FIGURE 3. (a) Mean squared error, (b) Normalized Variance

4. EXPERIMENTS AND RESULTS

The above presented method was tested on a simulated control problem, the classical pendulum balancing task where both the state and the action spaces are continuous. A state variable $s = (\theta, \omega)$ consists of the angle and angular velocity of the pendulum, actions are the torques that we can apply to

[3]Note that from the definition of the connections between the graph nodes follows that the sequence of states upon which the distance is measured is always executable under the current policy π.

the system, and are limited to a $[-5, 5]$ interval. The performance of the proposed value-function approximation scheme is tested under a fixed Gaussian policy which consists of a deterministic controller and added Gaussian noise with fixed variance $\pi(s, a) = \mathcal{N}(0, \sigma^2) + f_\theta(s)$. As a baseline I used the TD approximation of the corresponding value function, based on 800 episodes of length 150. The state space was discretized to contain 3600 states. Figure 3 shows the approximation accuracy of both standard GP and Geodesic distance based GP value function approximation where the horizontal axis represents the maximum number of allowed basis vectors and the vertical axis measures the mean squared approximation error. In terms of approximation error GP with geodesic Gaussian kernel performs significantly better by low number of basis vectors, and achieves the same performance as standard GP after the number of BV's exceeds a threshold. However the variance of the value function estimates decreases slower when geodesic kernel is being used. There is also a performance decrease in a certain region of max BV numbers where the performance gets worse than standard GP.

5. conclusion

In this article I have presented a modality of dynamically constructing geodesic distance based kernel functions during on-line Gaussian Process value function approximation. Unlike in previous work where fixed resolution graph structures have been used I presented a way to construct the MDP induced graph only between data-points which are important from the information-gain point of view. Experimental results prove the viability of this method.

References

[1] Lehel Csató and Manfred Opper. Sparse on-line Gaussian Processes. *Neural Computation*, 14(3):641–669, 2002.
[2] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.
[3] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with Gaussian processes. In *ICML '05* , pages 201–208, New York, NY, USA, 2005. ACM.
[4] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.
[5] C. E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In L. K. Saul Thrun, S. and B. Schlkopf, editors, *NIPS 2003*, pages 751–759. MIT Press, 2004.
[6] Carl Edward Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
[7] Masashi Sugiyama, Hirotaka Hachiya, Christopher Towell, and Sethu Vijayakumar. Geodesic gaussian kernels for value function approximation. *Auton. Robots*, 25:287–304.
[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Department of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu St., 400084 Cluj-Napoca, Romania
E-mail address: jakabh@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

USING IMPACT ANALYSIS BASED KNOWLEDGE FOR VALIDATING REFACTORING STEPS

ISTVÁN BOZÓ, MELINDA TÓTH, MÁTÉ TEJFEL, DÁNIEL HORPÁCSI, RÓBERT KITLEI, JUDIT KŐSZEGI, AND ZOLTÁN HORVÁTH

Abstract. In the life cycle of a large software product the source code often has to be changed to fit to new requirements, which can be aided by refactorings. In order to minimise the possibility of breaking the functionality of the code, various test suites are used. In this paper, we present a method for examining which test cases are affected by doing a refactoring on the code. If we find that the outcome of a particular test case is not changed by the refactoring, the test case does not need to be run, which might make testing more cost effective. Also, we validate the refactoring by checking that the affected functions behave identically pre and post the transformation. This system is based on a language specific model that is designed to capture knowledge about the software product acquired using impact analysis.

1. Introduction

The refactoring [5] tools try to preserve the properties of the transformed programs using language specific semantic knowledge about the source code and to ensure the safety of statically performed transformations. In case of industrial sized software and some programming languages the usage of complex static analysis sometimes is not enough to decide whether the transformation is safe, and some rarely used language elements can make it impossible. Therefore it is crucial to retest the system after a major change.

In case of the industrial sized software components the developers prefer to perform the necessary refactorings manually against applying the automatic refactoring, even if the manual refactoring is more error prone. The main reason for this situation is based on the fact that the developer can not follow the changes performed on the source code by the refactoring tool. In order to

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68Q99.

1998 *CR Categories and Descriptors.* [**F.3.2**]: Theory of Computation, Logics and Meanings of Programs, Semantics of Programming Languages – *Program analysis*.

Key words and phrases. Erlang, regression test, refactoring validation, property generation.

convince the developer about the reliability of refactorings we have to analyse the impact of the transformations while reducing the cost of the testing process. By using impact analysis we can select the code parts that are affected by a transformation and based on this knowledge we can minimise the cost of regression test.

Our research focuses on Erlang [4], a dynamically typed language. Many of its semantic rules are also dynamic, that makes the static semantic knowledge based analysis hard. In this paper, we examine the various connections between refactorings of the code and test cases. We explore the possibility of leaving some test cases out if we can argue that the refactoring does not change the outcome of the test. We also investigate how we can give further guarantees that the refactoring has successfully transformed the source code using random parameter space testing and metrics based validation.

The paper is structured as follows. In Section 2, we show a motivating example. In Section 3 we present the tool that will store the knowledge we gather about Erlang source code; Section 4 describes change impact analysis based property selection; Section 5 uses the same analysis to find functions that are affected by a code transformation and introduce a validation property. Section 6 discusses related work and Section 7 concludes the paper.

2. Motivating example

```
1  -module(exampl).
2  -export([calc/2, prop1/0, prop2/0]).
3
4  calc(X,Y)->
5      Mul = X*Y,
6      Sum = X+Y,
7      {Mul, Sum}.
8
9  prop1()->
10     ?FORALL({A, B},
11             {int(), int()},
12             element(1, calc(A, B)) =:= A*B).
13 prop2()->
14     ?FORALL({A, B},
15             {int(), int()},
16                 element(2, calc(A, B)) =:= A+B).
```

Figure 1. A module with two QuickCheck properties

```
1  calc(X,Y)->
2      Mul = mul(X,Y),
3      Sum = X+Y,
4      {Mul, Sum}.
5
6  mul(X,Y) ->
7      X*Y.
```

FIGURE 2. A part of **calc** is extracted into **mul**

In Figure 1, we see an Erlang module with a function `calc` that takes two parameters. There are two statements `prop1` and `prop2`, formalised as QuickCheck properties [10]. The function itself calculates the sum and the product of its arguments, and returns the pair of the calculated values.

The first property states that the first element of the returned tuple contains the product of the two arguments. For this, it uses a pair of two generators (`int()`) and binds the generated values to `A` and `B` respectively. If QuickCheck is installed, the module test case can be invoked by calling `eqc:quickcheck(exampl:prop1())`, which will use the generators to produce 100 random pairs of integers, and tests the property. The second property, quite similarly, tests the second component of the return value of the function `calc`.

There are several refactorings that can be applied to the function `calc`. As `calc` is not a good name, one idea is to rename the function. Another possibility is to extract a portion of the function `calc` into another function. For example, the production on line 5 may be extracted to a new function `mul` as seen in Figure 2. The use of this refactoring could change the first element of the return value of function `calc` and does not have any affect on the second element. Since property `prop2` uses only the second element of the return value, it does not depend on the changed source code and it is unnecessary to retest it.

3. Layers of knowledge for impact analysis

3.1. **Source code representation.** RefactorErl [7] is a source code analyser and transformer tool for Erlang [4]. It builds an abstract syntax tree over the source code, then uses several semantic analysers that enrich it into a Semantic Program Graph (SPG) by adding various other levels of knowledge. Information gathered by the analysers include connections between function applications and definitions (revealing function calls), between definitions and uses of variables etc.

The Semantic Program Graph of RefactorErl is capable of storing and efficiently retrieving information from the represented Erlang source code. While it is possible to directly use the SPG to retrieve the information necessary for impact analysis, directly accessing the SPG is too costly. It is more efficient to build a more compact and focused intermediate representation of the source code first. Therefore, we use the SPG to calculate advanced representations such as control, data and behaviour dependency graphs, and we calculate the impact of a refactoring change on dependency graphs.

Changing the source code (by tool assisted semi-automatic refactoring, or manually editing it) affects those program parts that depend on the changed expressions. We grasp knowledge about such connections by building a *Dependency Graph* where

- a **node** represents an Erlang expression and
- an **edge** represents a possible dependency between two expressions.

A change spreads in the source code with different kinds of dependencies: control, data, behaviour etc. We build several flow graphs to calculate these dependencies: Data-flow Graph [9, 13], Control-flow Graph [14] and Behaviour Dependency Graph [15].

3.2. **Dependency Graph - DG.** Working with flow graphs, in order to determine real dependencies is not efficient, as it requires a number of traversals in the CFG for every expression. Thus we use the well known approach used at compilers, we build a dependency graph that eliminates the unnecessary sequencing and includes only direct dependencies.

In building the DG we follow a compositional approach [12]. First we determine the affected functions by performing a transitive closure on the function call graph, starting from the selected functions. We build the CFG for every function from this set separately. These CFGs are intrafunctional, as these do not follow the function applications and message passing. Then from the obtained CFGs we build postdominator trees (PDT) and control dependency graphs (CDG). The next step is the composition stage of the obtained CDGs. In this stage the function application and message passing edges are resolved. With these steps we obtain the composed control dependency graph.

This graph contains only the control dependency edges, some other useful information is necessary to determine real dependencies among the expressions. We extend this composed CDG with data and behaviour dependency edges, calculated from the former introduced graphs. This compound graph is the DG of selected functions and contains necessary information about data, control and behaviour dependencies of the expressions of the functions [14].

4. Change Impact Analysis

In this paper, we focus on impact analysis of refactorings. To find the functions that are affected by a refactoring, we use dependency graph based program slicing [16, 8].

The defined Dependency Graph represents the Erlang expressions as nodes and the dependencies among expressions as edges. A change in an expression in the source code that is changed by a refactoring may affect those expressions that depend on the changed expression. In order to track these changes, we have to traverse the Dependency Graph and gather those expressions that are reachable from the changed expressions. Traversing the Dependency Graph produces a static forward slice of the program. The program slice will contain all expressions that are affected by the refactoring. A function is affected by a refactoring if at least one expression from its body is contained in the slice. We collect all functions that are affected by a refactoring.

Since QuickCheck properties are formalised as Erlang functions, the program slice will contain the affected properties by a refactoring and we can suggest to the user to retest them. In case there is no affected property for an affected function we can check the behaviour equivalency within our refactoring tool.

5. Transforming and deriving QuickCheck properties

As mentioned already, change impact analysis exactly identifies program segments that may be affected by an arbitrary change (such as an effect of a refactoring step) in the code. The result of such an analysis process may include code of the application itself as well as parts of test modules. Due to the nature of impact analysis, the former ones basically cover code pieces whose meaning might have been modified by the previously mentioned change, while the latter sort of code fragments include test routines that may have to be changed accordingly and also they are due to be re-checked.

Those QuickCheck [10] properties that are available at the time of the execution of a refactoring step are transformed similarly, according to the impact of the specific refactoring step. More accurately, when a refactoring step altering the public interface of a function referred within a test module gets performed, calls within the test module are also transformed. For instance, when the “reorder function arguments” transformation changes the order of the arguments in a function definition, applications of that function are transformed accordingly, even those that are located within test modules.

Refactoring steps should preserve the semantics and the behaviour of programs. A properly performed refactoring step consequently results in a program that is semantically equivalent to the original one. In this paper, programs are said to be equivalent if they have the same observable behaviour, that is, they return the same values on given input data and have the same side-effects (such as I/O and exceptions). The validation of refactoring steps obviously includes a phase that checks whether user-defined test cases satisfied before the transformation remain satisfied. However, we can involve some additional techniques for checking behaviour preservation, namely, the most vital property to be checked may be that the original and the refactored program are equivalent.

To establish the fact of equivalence, we should execute each function of both the original and the transformed programs on every possible input value and check whether they all produce the same values and side-effects respectively. Obviously, this method is inapplicable in case of complex programs involving many functions operating on a large domains. Fortunately, we can omit the check of all the function that certainly are not affected by the performed code change. As seen before, impact analysis identifies the functions that might have been changed, so thus we only have to check those.

The problem of large input domains is resolved by using random generation. That is, change-affected functions are checked for semantics preservation by a large number of randomly generated values. As Erlang is dynamically typed, we can supply values of any type as arguments to functions, and at the worst case we get runtime exceptions. Arguably, randomly generating lots of improperly typed tests do not help with catching bugs. To avoid test cases failing for reasons of data being ill-typed, types of functions are inferred and then only well-typed input parameters are applied.

The property evaluates both the old and the new versions of affected functions, compares the results and analyses I/O activity and thrown exceptions. In the case they produce the same results, they likely are equivalent and therefore the refactoring step has been performed correctly, as it has preserved the program behaviour. Such a property can be defined within QuickCheck and can be checked just after a refactoring transformation has been made. User-defined properties along with the described equivalence property can efficiently validate refactoring steps.

Further validation steps. Beside checking whether the original and the refactored program are semantically equivalent, we might verify refactoring-specific properties as well, ensuring a more accurate validation of the refactoring steps. Such properties can be based on software metrics, calculated and compared on both the old and the new version of the source code.

6. Related work

There is a large body of work of software restructuring and refactoring. Accordingly, there exists several state-of-the-art research for making program refactoring safer. In most cases, these works focus on object-oriented programs (for example [2, 6, 11]), but there exists a technique for testing Erlang specific refactoring steps as well [3].

However, in the above researches, randomly generated test data is used and no complex (if any) additional semantic or syntactic information about the program is applied. The method illustrated in this paper uses knowledge based on impact analysis, which makes the generation of more specific and more relevant test cases possible, which may result in a more efficient testing method.

In the case of object-oriented programs there exists also incremental software change supporting tool using impact analysis [1]. Integrating the interactive behaviour of the refered tool and the test case selection method presented in this paper could be a possible direction for future work.

7. Conclusions and future plans

The paper demonstrates how knowledge originating from impact analysis can be used for validation of refactoring steps. The applicability of introduced method is illustrated by a case study based on capabilities of RefactorErl, a refactoring tool for functional programming language Erlang. The paper also outlines a method for assembling the required knowledge in the specific case of Erlang.

Since Erlang is a dynamically typed language, to ensure the safety of statically performed transformations is hard and the derived static semantic knowledge is not enough. Therefore we examined various connections among refactorings and test cases. Therefore we recommend to rerun a subset of selected test cases after the refactorings and we introduce new properties to check the behaviour of the changed system. We use impact analysis to select the code parts that are affected by a transformation and based on this knowledge we can minimise the cost of regression test.

In the future, the described method can be extended to make available refactoring steps driven improvement of existing test databases with semi-automatic extension of existing test cases relevant for program slices changed via given refactoring steps.

Acknowledgement

This work was supported by TECH_08_A2-SZOMIN08, ELTE IKKK and Ericsson Hungary.

References

[1] Buckner, J., Buchta, J., Petrenko, M., Rajlich, V. JRipples: A Tool for Incremental Software Change IEEE International Workshop on Program Comprehension, 2005, 149 - 152.

[2] Daniel, B., et al., Automated testing of refactoring engines. In ESEC/FSE, pages 185194, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-811-4.

[3] Drienyovszky, D., Horpácsi, D., Thompson, S., QuickChecking Refactoring Tools, Erlang 10: Proceedings of the 2010 ACM SIGPLAN Erlang Workshop, ed.: Scott Lystig Fritchie and Konstantinos Sagonas, pages 75-80 ACM SIGPLAN, September 2010,

[4] Erlang Homepage, *https://www.erlang.org*

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

[6] Gligoric, M., et al., Test generation through programming in UDITA, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, pages 225-234, New York, NY, USA, 2010, ACM Press

[7] Horváth, Z., Lövei, L., Kozsik, T., Kitlei, R., Víg A., Nagy, T., Tóth, M., Király, R.: Modeling semantic knowledge in Erlang for refactoring, Knowledge Engineering: Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, volume 54(2009) Sp. Issue, Studia Universitatis Babe-Bolyai, Series Informatica, Cluj-Napoca, Romania, July, 2009

[8] Horwitz, S. and Reps, T. and Binkley, D., Interprocedural slicing using dependence graphs., PhD thesis, University of Michigan, Ann Arbor, MI, 1979

[9] Lövei, L.:, Automated module interface upgrade, Erlang '09: Proceedings of the 8th ACM SIGPLAN workshop on Erlang, ISBN 978-1-60558-507-9, pages 11–22, Edinburgh, Scotland, September, 2009

[10] Quviq QuickCheck, http://www.quviq.com/, 2011

[11] Soares, G. Making Program Refactoring Safer. In ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, pages 521-522, New York, NY, USA, 2010.

[12] Stafford, J.A., A Formal, Language-Independent, and Compositional Approach to Control Dependence Analysis., PhD thesis, University of Colorado, Boulder, Colorado, USA, 2000

[13] Tóth, M., Bozó, I., Horváth, Z., Tejfel, M.:, 1^{st} order flow analysis for Erlang, In Proceedings of 8th Joint Conference on Mathematics and Computer Science, 2010

[14] Tóth, M., Bozó, I.:, Building dependency graph for slicing Erlang programs, Paper submitted to Periodica Politechnica, 2010

[15] Tóth, M., Bozó, I., Horváth, Z., Lövei, L., Tejfel, M. Kozsik, T., Impact analysis of Erlang programs using behaviour dependency graphs., Central European Functional Programming School. Third Summer School. Revised Selected Lectures., 2010

[16] Weiser, M.: Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method., ACM Transactions on Programming Languages and Systems, 12(1):3546, 1990

Department of Programming Languages and Compilers, Faculty of Informatics, Eötvös Loránd University

E-mail address: {bozo_i,toth_m,matej,daniel_h,kitlei,kojqaai,hz}@inf.elte.hu

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

A COMPARISON OF AOP BASED MONITORING TOOLS

GRIGORETA S. COJOCAR AND DAN COJOCAR

Abstract. The performance requirements of a software system are very important for the end user, especially today when everything gets faster and faster. In order to improve the performance of a software system, developers must first identify the parts that take a long time to execute. In this paper we describe how AOP is used for monitoring software systems performance and we present a comparison of the existing AOP based monitoring tools. The comparison is done using different criteria like: AOP extension used, UI support, performance metrics provided, etc.

1. Introduction

1.1. **AOP.** Separation of concerns is an important principle in software engineering [6]. It refers to the ability to identify, encapsulate and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose [12]. Even though separation of concerns seems simple, it is not. System concerns are of two types: core concerns that capture the central functionality of a module, and crosscutting concerns that capture system-level, peripheral requirements that cross multiple modules. Even though the most popular programming paradigms are good for designing and implementing core concerns, they do not provide the means of a clear separation for crosscutting concerns. From the various approaches that have proposed solutions for the design and implementation of crosscutting concerns [1, 5, 9, 11, 15], the aspect oriented programming (AOP) approach has known the greatest success both in industry and academia. AOP introduces four new notions in order to implement a crosscutting concern: *joinpoint*, *pointcut*, *advice*, and *aspect*.

- A *join point* is a well-defined point in the execution of a program. Join points consist of things like method calls, method executions, etc.

Received by the editors: April 10, 2011.

2000 *Mathematics Subject Classification.* 68M20, 68N19.

1998 *CR Categories and Descriptors.* D.1.m [**Programming Techniques**]: Miscellaneous – *Aspect oriented programming*; D.2.8 [**Software Engineering**]: Metrics – *Performance measures*.

Key words and phrases. aspect oriented programming, performance monitoring tools.

- A *pointcut* groups a set of join points, and exposes some of the values in the execution context of those join points.
- An *advice* is a piece of code that is executed at each join point in a pointcut. Usually, an AOP extension supports at least three types of advice: *before*, *around*, and *after*.
- An *aspect* is a crosscutting type that encapsulates pointcuts, advice, and static crosscutting features. An aspect is the modularization unit of AOP.

The aspects are integrated into the final system using a special tool called *weaver*. Nowadays, there are AOP extensions for well-known programming languages (eg. AspectJ for Java [2]) which are used in industry, too.

1.2. **Performance Analysis.** Performance of software systems is a very important topic. It refers to the response time or throughput as seen by the users of the software system. Many different things can impact the performance of a software system: network load, computation time, database query response time, etc.

There are two approaches to performance engineering: a *fix-it-later* approach and an *engineering* approach. The first approach advocates concentrating on correctness and deferring consideration of performance until the testing phase. The detected performance problems are then corrected by adding additional hardware, tuning the software, or both [3]. The second approach, called *software perfomance engineering* (SPE) [14], uses model predictions to evaluate trade-offs in software functions versus hardware costs.

Even though the SPE approach have obtained good results in developing software systems that meet their performance objectives from the beginning, this approach is not used very often. There are still software systems built without considering performance issues, and then, during testing, they are modified to meet the performance requirements. However, modifying the system to discover performance problems is a tedious task. The developers must modify different parts of the software system in order to identify those that cause performance problems.

The paper is structured as follows. In Section 2 we describe the AOP based approach for developing monitoring tools/frameworks and some of the freely available AOP based monitoring tools. A comparison of these tools is presented in Section 3. Conclusions and further work are given in Section 4.

2. The AOP Approach

The goal of AOP based monitoring tools is to develop an easy to use and easy to integrate tool in order to obtain performance data for different parts of a software system. Usually, when performance problems are encountered,

the developers must gathered different kinds of data in order to discover the parts of the software system that caused the problems. In order to do that, using object oriented programming for example, two steps must be performed:

(1) Some parts of the software system or all of it must be manually modified in order to gathered different kind of performance data, like the execution time of each unit of work (method, component, etc.).
(2) The gathered data must then be analyzed in order to discover the parts with performance problems.

If the second step can be automatically performed, the first one is a tedious one, that takes time to complete. Also, if new kind of data must be added for the analysis, all the previous modified parts of the system must be manually modified, again. This leads to decreased productivity, and lost time and effort.

The idea of the AOP based approach is to keep all the source code that gathers the data in one place, and then to integrate it with the software system whenever needed. Also, using AOP, the above described steps are merged. The analysis is performed while the data is gathered and when the execution of the system is finished, the results of the analysis are also ready.

The basic design of AOP based monitoring is as follows: pointcuts that match the chosen joint points are defined [4, 10]. Usually, methods executions or calls are chosen as joint points. Then, the around advice or before and after advices are written to update the performance data.

Some of the advantages of using AOP for monitoring software system performance are:

- The source code for the performance analysis (data gathering and analysis) is kept in one place: the performance analysis module (aspects). The analyzed software system has no reference to this module.
- The monitored system does not need to be modified in order to obtain performance data.
- The module can be easily plug-in and out of the system.
- The performance analysis aspects can be easily used for different software systems.

There are also some disadvantages of using AOP:

- The pointcut(s) definition is very dependent on the signature of the methods. If after a execution run, the signatures of some methods are changed, the developer must be careful to redefine the target pointcut, otherwise the methods may be missed in the following runs.
- In order to define the pointcut(s) the developer that uses the performance module must have an indepth knowledge of the software system. This may take time if he/she is not among the developers of the software system.

- This approach can be used only for programming languages for which an AOP extension exists.

2.1. **AOP based Monitoring Tools.** In the following we present some of the existing monitoring tools that use AOP.

InfraRED is a tool for monitoring performance of a J2EE application and diagnosing performance problems [8]. It collects metrics about various aspects of an application's performance and makes it available for quantitative analysis of the application. It uses AOP to weave the performance monitoring code into the application.

Glassbox is a troubleshooting agent for Java applications that automatically diagnoses common problems [7]. It offers ready to use aspects and tools to help developers get started with application-level monitoring and identify potential problems.

Perf4J is a toolset for calculating and displaying perfomance statistics for Java code [13]. It adds Java server-side code timing statements and it logs, analyzes, and monitors the results.

SpringSource AMS is designed to manage and monitor Spring-based applications, the Spring runtime, and a variety of platforms and application servers [16]. It focuses on application-level monitoring.

3. Monitoring Tools Comparison

In this section we present a brief comparison of the previously described tools. The comparison is performed using the following criteria: language dependecy, weaving approach, source code modification, extendability, computed metrics, and UI support.

Language dependecy. All the previously described AOP based monitoring tools can be used to monitor only Java based software systems. Also, all tools use AspectJ as the AOP extension for Java. However, some of them can be configured to use a different AOP extension, i.e., InfraRED with Aspectwerkz or JBoss AOP, SpringSource AMS with Spring AOP, etc.

Weaving approach. AspectJ supports three different approaches for weaving aspects into the software system:

- *Compile-time weaving (CTW)* where the AspectJ compiler takes as input the software system source code and the aspects source code and produces woven class files as output.
- *Binary weaving* is used to weave existing class files and JAR files.
- *Load-time weaving (LTW)* is binary weaving defered until the point that a class loader loads a class file and defines the class to the JVM.

All the previously presented monitoring tools support compile time weaving, and most of them also support load-time weaving. The latter one is preferred by the developers as it does not add extra step to the building process, it is easy to switch between the original version of the software system and the instrumented one, and the extra time required for load time weaving does not take too long.

Source code modification. Does the user of the tool need to modify the source code of the analyzed software system? Most tools do not require the manual modification of the source code. The only tool that requires source code modification is Perf4J because it uses AspectJ5 annotation facility. The *@Profiled* annotation provided by Perf4J is used to select method calls for monitoring. The annotation step must be performed only once, even if new data is required for performance analysis. However, if new methods are included in the analysis, they must be first annotated, and it is the user responsability to perform the annotation.

Extendability. How easy is for a developer/tool user to define/include new performance metrics using these tools? In order to define a new performance metric using the AOP approach, all the user has to do is to define a new aspect in which he specifies the methods to be monitored and how to compute the new metrics. As such, most tools allow the user to define new performance metrics. The exception is Perf4J, that uses AOP only for gathering performance data.

User Interface (UI) Support. All the tools provide a UI for viewing the performance metrics and for analysis. The UI provided is either a Web UI or a JMX client. Java Management Extensions (JMX) is a standard API for managing Java applications by viewing attributes of managed objects.

Performance metrics. All tools compute at least the following metrics: the number of times a method was executed in a run, and the average execution time for each monitored method. Some tools provide additional performance statistics like the mininum and maximum execution time, and the standard deviation (Perf4J), or the accumulated and the maximum execution time (Glassbox).

4. Conclusions and Further Work

We have presented in this paper the AOP approach for developing performance monitoring tools and a comparison of some AOP based performance monitoring tools. The comparison was performed using different criteria like language dependency, weaving approach, source code modification, and UI

support. In the future we intend to compare the AOP based performance monitoring tools using different case studies.

References

1. Mehmet Aksit, *On the design of the object oriented language sina*, Ph.D. thesis, Department of Computer Science, University of Twente, The Netherlands, 1989.
2. *AspectJ Project*, http://eclipse.org/aspectj/.
3. Ken Auer and Kent Beck, *Pattern languages of program design 2*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996, pp. 19–42.
4. Ron Bodkin, *AOP@Work: Performance Monitoring with AspectJ, Part 1 and Part 2*, http://www.ibm.com/developerworks/java/library/j-aopwork10/, http://www.ibm.com/developerworks/java/library/j-aopwork12/, 2005.
5. Krzysztof Czarnecki and Ulrich Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.
6. Edsger Wybe Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976.
7. *Glassbox.com*, http://glassbox.sourceforge.net/glassbox/Home.html.
8. *InfraRED: Opensource J2EE Performance Monitoring Tool*, http://infrared.sourceforge.net/versions/latest/.
9. Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin, *Aspect-Oriented Programming*, Proceedings European Conference on Object-Oriented Programming, vol. LNCS 1241, Springer-Verlag, 1997, pp. 220–242.
10. Ramnivas Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming (2nd edition)*, Manning Publications Co., 2010.
11. Karl J. Lieberherr, *Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes*, Information Processing '92, 12th World Computer Congress (Madrid, Spain) (J. van Leeuwen, ed.), Elsevier, 1992, pp. 179–185.
12. David L. Parnas, *On the Criteria To Be Used in Decomposing Systems into Modules*, Communications of the ACM **15** (1972), no. 12, 1053–1058.
13. *Perf4J home*, http://perf4j.codehaus.org/.
14. Connie U. Smith, *Performance engineering of software systems*, Addison-Wesley, 1990.
15. *Subject oriented programming*, http://www.research.ibm.com/sop/.
16. *Virtualized Java Application Monitoring*, http://www.springsource.com/products/systems-management, 2009.

(1) Babes-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu str., Cluj-Napoca 400084, Romania

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

INTEROPERABILITY ISSUES OF MDWE METHODOLOGIES

ATTILA ADAMKÓ AND LAJOS KOLLÁR

Abstract. Due to the evolution of Web technologies experienced in the past 10–15 years, the Web has become a primary platform for developing applications. However, as these technologies evolve very fast, they might become obsolete soon. Developers of Web applications need sophisticated solutions that support the whole product lifetime of an application that is able to cope with the skyrocketing changes of the underlying technologies.

Model-driven Web Engineering (MDWE) is a still emerging field aiming at providing sound model-based solutions for building Web applications that try to separate the abstract design (PIM) from the concrete technological platforms (PSMs). However, current MDWE approaches cannot provide solutions for all kinds of the requirements against a software system therefore a lightweight, extensible, loosely coupled set of models for designing applications are needed.

This paper introduces an approach for the interoperability of (some) existing methodologies based on metamodeling, model transformations and model weaving which allows the MDWE methodologies to be extended in a consistent manner where new model kinds are separated and weaved together with the classical models that each approach supports.

1. Introduction

Existing model-based Web Engineering approaches provide different methods and tools for both the design and the development of various kinds of Web applications. In order to reduce complexity, most of the methodologies propose the separation of different views (i.e., models) of the application into 3 levels: structural (or content), navigational (or hypertext) and presentational models. For more information see [9]. Figure 1 shows the most common design dimensions of the currently existing methodologies.

Received by the editors: Aprtil 10, 2011.

2010 *Mathematics Subject Classification.* 68U35, 68M11, 68N99.

1998 *CR Categories and Descriptors.* D.2.2 [**Software**]: Software Engineering – *Design Tools and Techniques*; D.2.10 [**Software**]: Software Engineering – *Design*.

Key words and phrases. MDWE, Web Applications, Metamodeling, Model transformations, Model weaving.

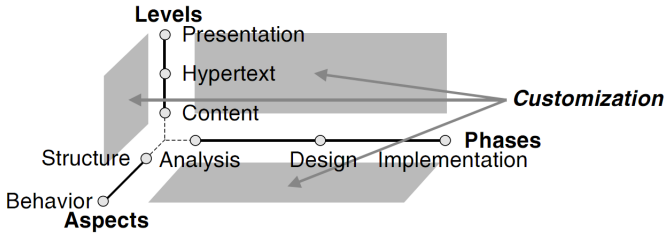


FIGURE 1. Design dimensions of Web applications [9].

In addition, some methodologies add some new models (or refine existing ones) to obtain a more fine-grained solution when modeling the application. Despite the separation, the levels should be interconnected in order to be able to capture the semantics behind the elements of the different models, e.g., the navigational objects are based on certain elements of the content model.

Beyond the creation of the models for the corresponding levels, Web application designers need to be aware of the various aspects of the systems to be modeled. Some applications are providing access to more or less static information hence they require much less behaviour modeling compared to systems that need to perform several complex business processes like e-commerce applications. Both structure and behaviour need to be modeled using a uniform notation that has to cope with the specific characteristic of each of the levels.

Current design methods offer some possibilities for modeling the levels and aspects mentioned above but they all has a unique approach (e.g., offering some model kinds that the others not) so this field is not standardized.

2. RESEARCH BACKGROUND

2.1. **Domain-specific modeling, Metamodeling.** The main goal of domain-specific modeling is to raise the level of abstraction by specifying the solution directly using domain concepts. The final product (and maybe several intermediate artifacts, as well) are generated based upon these high-level specifications. It also allows the stakeholders and domain experts to concentrate to the domain only. Domain-specific languages (DSLs) are built in order to capture domain semantics. A very common (but not the only) way of defining DSLs is metamodeling. The previously mentioned Web application design methods contain notations that can be used for describing a model of a Web application so they can be considered as DSLs for Web applications hence.

Some of the existing Web application design methods (e.g., UWE, WebML) offer a metamodel, as well [**?**, **?**]. This allows model-based development since one need to build models conforming to the appropriate metamodel in order

to capture the structural, navigational or presentational structure of the application to be developed. However, in the most of the cases, these models mix the different levels of Web applications that results in a solution that might be appropriate for the given application domain but makes the reuse of models or model parts almost impossible.

2.2. **Model transformation, Model weaving.** Model transformations are the most important operations in model engineering, describing how elements in the source model are converted into elements in the target model. This is achieved by relating the corresponding metamodel elements in the source and the target metamodels. Transformations can be classified into two categories: vertical transformations (a.k.a. refinements) are defined between models of different abstraction levels (e.g., PIM—PSM mappings), while horizontal transformations are mappings between models of the same level of abstraction (e.g., for improving or correcting a model).

Weaving models are used to explicitly describe fine-grained relationships between models and metamodels (that are models themselves, as well) and execute operations based on them. With the help of applying weaving models, large metamodels that capture all aspects of a system can be avoided and a lattice of metamodels can be constructed instead where each metamodel that focuses on its own domain is maintained independently from the others. The links defined by the weaving model have some associated semantics about the linked elements.

3. Problem statement

Most of the methods mentioned in Section 1 are using different notations for these models, hence the interoperability between them is very hard to achieve. This also deceases reuse as one cannot import, for example, a conceptual model or a part of it when developing an application for a similar domain.

The idea of complete integration of the existing languages and methodologies, i.e., developing a common metamodel and unified phases of development that everyone will use in the future is utopian and (in our opinion) it must not be the goal of any integration or interoperability efforts. The main reason behind it is that different domains and various flavours of Web applications may require different styles of modeling and it is almost impossible to achieve such a common modeling notation which is easy to understand and work with while being flexible enough to solve the uprising issues. Therefore we should work on bridging the different models together that allows (or promises, at least) the interchangeability of models and/or model pieces instead.

New models, processes and transformations should be included into the existing design methods when new aspects arise. However, these changes to a methodology are very risky and can cause several problems. In [6], three categories of concerns were identified:

- dependent concern, that depend on some other (earlier defined) concern(s), e.g., navigation (which depends on the conceptual model);
- replacement concern, that fully replaces a previously defined concern, e.g., presentation;
- orthogonal concern, that is a brand new concern which is completely independent of all the others, e.g. business process models.

However, we are not against the creation of subsequent metamodels and/or methodologies as they can result in better description of system parts or improved development processes. We only claim that a common metamodel is not the Holy Grail of MDWE as each and every "common" one will most probably fail as being a universal solution because the diversity of Web applications will require new answers for such questions that probably had not been asked by the time of developing the common metamodel.

4. Proposed solution

Our goal is to establish an extensible model-based framework which can provide interoperability among the existing Web modeling languages. This task has to be achieved by separating the different concerns (i.e., levels, phases and aspects) of Web applications in order to be able to either reuse relevant model parts or "transfer" a model into another notation (e.g., after a structural model is created conforming the metamodel of language A we decide to build the navigational model in language B since it might be more appropriate for our goals).

Hence, it is extremely important that the metamodels defining the languages for describing the various aspects of a Web application need to be separated from each other as much as possible. So we suggest of decomposing the various methods into a combination of models, each of which conforms to a well-defined part of the whole application domain regardless of the language used for the notation. For example, that allows of describing the structural model either in relational model, Entity Relationship (ER), UML or by using any custom DSL but it requires the separation of the structural model from any other models (e.g., navigational or requirements model). Besides, we suppose that no method uses a notation that does not conform to the MOF metapyramid (in fact, this is not a heavy constraint).

In our proposed solution, model weaving should appear on two levels:

(1) On *intra-method* level, the relationships existed before the decomposition of the concerns need to be defined in a weaving model in order to be able to produce the same level of expressiveness. Let us consider the well-known conference management system as an example! In UWE, for instance, we would have a UML class called *Paper* in the structural model while its derived (and stereotyped) versions would appear in the navigational and presentational model, as well. Instead of the given method's built-in notation for this derivation, weaving links should be established in a weaving model that comprises statements about the relationship between the models in question. This weaving model can also be used later on when the starting point of the design is the building of the structural model as it captures the semantics that structural model elements also become (stereotyped) elements of the navigational model under given circumstances so a transformation might be applied to the structural model in order to create an initial version of the navigational one.
(2) On *inter-method* level, when the relationships described by the weaving model define which model elements of a given model M_a conforms to which model elements in M_b. M_a and M_b here typically have the same level of abstraction (e.g., they both are structural models described by different methodologies) and the weaving model is defined between their corresponding MM_a and MM_b metamodels. For example, if one of the methods uses ER for describing the structural model while the other one applies UML for the same purpose, then the weaving model should contain that the strong entity type of the ER corresponds to a class in a UML class diagram, etc. This approach allows not only the generation of such a model transformation based on the weaving model that can transform a model in a notation into another model of another notation but model traceability is also supported.

Acknowledgement

The work is supported by TÁMOP 4.2.1./B-09/1/KONV-2010-0007/IK/IT project. The project is implemented through the New Hungary Development Plan co-financed by the European Social Fund, and the European Regional Development Fund.

References

[1] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[2] M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: a generic model weaver. In *Proc. of the 1cre Journe sur l'Ingnierie Dirige par les Modcles (IDM05)*, 2005.
[3] M. D. D. Fabro and F. Jouault. Model Transformation and Weaving in the AMMA Platform. In *Pre-proc. of the Generative and Transformational Techniques in Software Engineering Workshop*, pages 71–77, 2005.
[4] J. Gómez and C. Cachero. OO-H method: extending UML to model web interfaces. pages 144–173, 2003.
[5] N. Koch and A. Kraus. Towards a common metamodel for the development of web applications. Cueva Lovelle, Juan Manuel (ed.) et al., Web engineering. International conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003. Proceedings. Berlin: Springer. Lect. Notes Comput. Sci. 2722, 497-506 (2003)., 2003.
[6] N. Moreno, S. Meliá, N. Koch, and A. Vallecillo. Addressing new concerns in model-driven web engineering approaches. In *Proceedings of the 9th international conference on Web Information Systems Engineering*, WISE '08, pages 426–442, Berlin, Heidelberg, 2008. Springer-Verlag.
[7] D. Schwabe. A conference review system. `www.dsic.upv.es/~west/iwwost01/files/ConferenceReviewSystem.doc`, 2001. [Online; accessed 24-April-2011].
[8] D. Schwabe and G. Rossi. An object oriented approach to web-based applications design. *Theor. Pract. Object Syst.*, 4(4):207–225, 1998.
[9] W. Schwinger and N. Koch. *Modeling Web Applications*, chapter 3, pages 39–64. John Wiley & Sons, 2006.

H–4032 Debrecen, Egyetem tér 1.
E-mail address: `adamkoa@inf.unideb.hu`

E-mail address: `kollarl@inf.unideb.hu`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

EXTENDING UML STATE DIAGRAMS WITH BEHAVIORAL PATTERNS

DAN MIRCEA SUCIU

Abstract. State diagrams generated by reverse engineering based on the execution of software applications or those that model Interactive Voice Response (IVR) applications are usually very dense and the benefit of using *super-states* or *orthogonal regions* to increase their readability is poor. We propose to extend state diagrams with *behavioral patterns*, which are powerful constructions that could significantly reduce the number of displayed transitions.

1. Introduction

1.1. **UML State Diagrams.** UML State Diagrams [3] have their origins in finite state machines. Software systems with complex behavior were modeled in dense state machines which were difficult to read and understand. In order to increase their readability the *super-states* and *orthogonal regions* were proposed [1]. Both constructions help designers to build less complex state diagrams, with a smaller number of states and transitions, without affecting their expressibility power. Anyway, there are research fields where state diagrams still need to be refined, the current graphical syntax being insufficient to deal with tens or hundreds of states and transitions. In [5] is presented a method of generating state diagrams by observing the behavior of a software system based on its inputs and outputs. For very simple software systems, which deal with two or three variables, this kind of reverse engineering is straight forward even in the absence of *super-states* or *orthogonal regions*. As far as the number of variables is increasing, the complexity of the resulted state diagrams is exponential. In this context, one idea is to find a way to group together states having similar behavior, and enclose them in a *super-state* [2]. In practice we can find many such groups to which a state belongs, but a *super-state* could

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques* D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements*.

Key words and phrases. UML, state diagrams, finite state machines, behavior modeling.

cover only one group at a time. This is a restriction that is solved by our proposed concept of *behavioral pattern* presented in current paper.

1.2. **IVR applications and state diagrams.** Another popular usage of state diagrams is for designing *Interactive Voice Response* (IVR) applications. IVR is a technology that allows a computer to decode humans requests via a telephone keypad or by speech recognition and which can respond with pre-recorded or dynamically generated audio. While a traditional IVR depended upon proprietary programming or scripting languages, the modern IVR applications are generated in a similar way to Web pages, using standards such as *VoiceXML*,[8] and State Chart XML (SCXML) [7]. SCXML provides a generic state-machine based execution environment based on Harel Statechart elements [1]. Commons SCXML is an implementation aimed at creating and maintaining a Java SCXML engine capable of executing a state machine defined using a SCXML document, while abstracting out the environment interfaces. For professional IVR applications with medium or high complexity, the corresponding state diagrams contain tens of states and hundreds or even thousands of transitions. The graphical view of these state diagrams is in many cases useless, due to their complexity. Our solution does not just increase the readability of such diagrams, but allow designers to define patterns which could be easily reused in subsequent IVR applications.

1.3. **Paper structure.** The next section describes the concept of *behavioral pattern* and makes a comparison with the *super-state* concept introduced by Harel. The third section presents a case study using the state configuration of a medium-size IVR application and shows how *behavioral patterns dramatically* decrease the number of transitions, while the *super-states* are not useful in this context. The last section describes the potential of behavioral patterns and some ways of extending them.

2. Behavioral patterns

The left diagram from figure 1 shows a simple state-diagram containing 3 states and 7 transitions. We can observe that every time an event *reset* occurs, the system enters in state **A**. At the same time, if the system is in state **B** or **C**, the event Event occurs and the logical expression cond is evaluated to TRUE, the system enters again in state **A**. We can say that in the first case the system behaves in the same way when an event reset occurs, and in the second case it behaves in the same way if it is in some particular states (**B** or **C**) and the event Event occurs. These are, in fact, two examples of *behavioral patterns*. If we want to use super-states in order to make the initial diagram more readable, we can group the states **B** and **C** under the

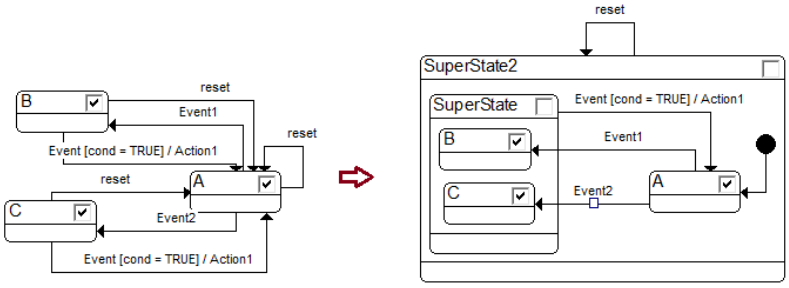


FIGURE 1. Using super-states to decrease the number of transitions

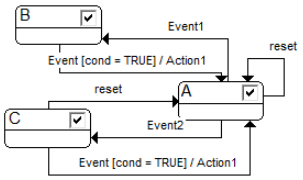


FIGURE 2. Sample state diagram

same *super-state* (let's call it **SuperState**) and nest the result together with the state **A** into another *super-state* (called **SuperState2**). The result of this operation in presented in the right diagram of figure 1. Consequently, the number of transitions decreased to 4 and, even if the final number of states is bigger now, the diagram is clearer than the initial one.

In Figure 2 we have the same initial state-diagram with only one modification: the transition triggered by event *reset* from state **B** to state **A** was deleted. This change makes impossible the usage of both *super-states* identified in the previous example. For the initial behavioral pattern we need to group together the states **A** and **C** and for the second one we need to group the states **B** and **C**. Because we cannot define super-states which have only a part of their sub-states in common, we are forced to use only one *super-state*. As a consequence, the final number of transitions we obtain is 5, more than in the previous example, even if we initially cut one transition.

Such anomalies are excluded in case of using specific graphical constructions to model behavioral patterns. Our proposal is shown in figure 3: we use doubled rounded rectangles to specify behavioral patterns and they are positioned in a separate area (called pattern area) together with modeled transitions and their state targets. In our example, state **A** was copied two times inside the pattern area to help the definition of two patterns: **Pattern1**, which describes the behavior of the system when *reset* event occurs, and **Pattern2**,

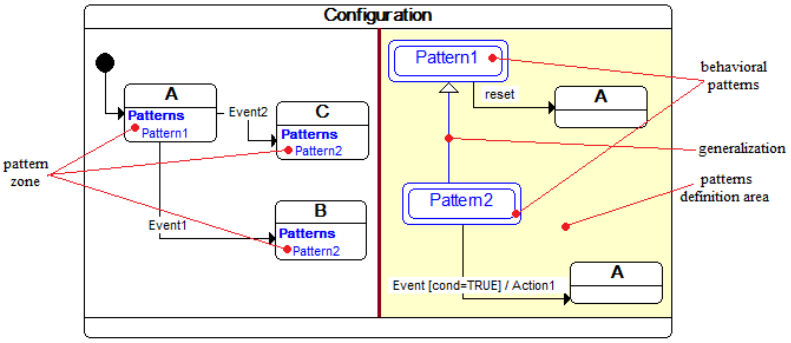


FIGURE 3. Using behavioral patterns to decrease the number of transitions

which describes the behavior of the system when event *Event* occurs and the system is in states **B** or **C**. In order to specify that a state follows a specific pattern, a new section is introduced in the graphical representation of a state called Patterns.

It makes sense to define also a generalization relation between patterns. In our example, there is a generalization relation between **Pattern2** and **Pattern1**, which means that the behavioral pattern **Pattern2** inherits the entire behavior described by **Pattern1**. Because the states **C** and **B** react at *reset* event in the same manner as A but, at the same time, react in a similar way in the presence of event *Event* and when the logical expression *cond* is evaluated to FALSE, they implement both **Pattern1** and **Pattern2**. But **Pattern2** inherits the entire behavior of **Pattern1**, so only **Pattern2** is enough to be specified as implemented pattern by **C** and **B**. If we are in conditions described by the state diagram from figure 2, there is no generalization relationship between **Pattern2** and **Pattern1**. In this case, **Pattern1** should be explicitly specified for state **C**, and the total number of displayed transitions remains the same. Therefore, the anomaly which appears in case of using super-states is solved.

3. CASE STUDY

In order to validate the efficiency of behavioral patterns, we extended *Active CASE* tool ([5]) with the following features:

- the root state of each class behavioral model has by default a pattern definition area (in order to not be confounded with an orthogonal region, it has a distinct background color);
- two new graphical elements are available for statechart editing: *behavioral patterns* and *pattern generalization* - each regular state was

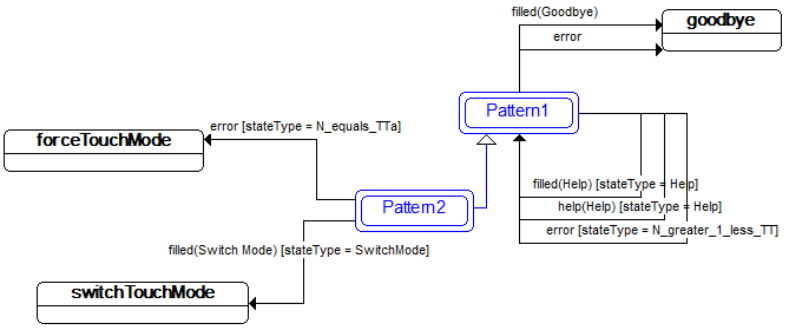


FIGURE 4. Automatically detected *behavioral patterns*

extended with an optional compartment for displaying the list of implemented patterns;

- a pattern generator, which automatically detects and creates *behavioral patterns* for a given statechart.

As a case study we selected the state diagram used for implementing a medium-size IVR application, containing 47 states and 708 transitions. The graphical representation of this state diagram shows a tangled net of transitions without any value for IVR designers. The pattern generator detected 30 distinct behavioral patterns, and the resulting statechart had just 195 transitions. So, more than 500 transitions where removed from the graphical representation of the state diagram using *behavioral patterns*.

Figure 4 shows two of the 30 generated patterns. **Pattern1** describes the behavior implemented by 35 states, so using this pattern 175 transitions were extracted from original state model. The second pattern, **Pattern2**, is implemented by 34 states, which means a number of 68 transitions replaced. Both patterns, together, cover almost a half of the total amount of replaced transitions. Not all generated patterns must be preserved, because many of them could not have a relevant impact in for the initial state diagram.

4. CONCLUSIONS AND FUTURE WORK

We proved that *behavioral patterns* are sensitive superior to *super-states*, especially in those software domains which deal with state diagrams having a high level of complexity. Anyway, *behavioral patterns* and *super-states* can leave together inside the same state diagram. It is up to the system designer when to use super-states and when to use behavioral patterns in order to make the model clearer and with a high level of readability. Usually, normal state diagrams having up to 10 concrete states do not need behavioral

pattern definition. There are some other interesting applications of *behavioral patterns*, which worth to be subject of future research, like:

- re-definition of state diagrams inheritance using *behavioral patterns*;
- support for automatic detection of rare events using statecharts ([4], [6]) by detecting any anomaly in *behavioral patterns* detected in different moments in time;
- definition of *behavioral pattern* at orthogonal region level, not at state level.

References

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

[2] Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve Easterbrook, Pamela Zave, "Matching and Merging Statechart Specifications", International Conference on Software Engineering, Proceedings of the 29th international conference on Software Engineering, pp 54-64, 2007

[3] Object Management Group, "OMG Unified Modeling Language Specification, Superstructure version 2.3", May 2010 (available at http://www.omg.org/spec/UML/2.3/)

[4] Vasile-Marian Scuturici, Dan-Mircea Suciu, Romain Vuillemot, Aris Ouksel, Lionel Brunie, "Detecting Anomalies in Data Streams using Statecharts", Extraction et Gestion des Connaissances (EGC'10), Revue des Nouvelles Technologies de l'Information, RNTI-E-19, Hammamet, Tunis, January 2010, pp 635-636

[5] Dan Mircea Suciu, "Reverse Engineering and Simulation of Active Objects Behavior", Knowledge Engineering, Principles and Techniques - "KEPT-2009" Selected Papers, "Babes-Bolyai"University of Cluj-Napoca, pp. 283-290 , July 2-4 2009

[6] Dan Mircea Suciu, Romain Vuillemot, Marian Scuturici, "Visual Detection of Rare Events Using Statechart", IEEE VisWeek Compendium, VAST Contest, Piscataway, NJ, IEEE. October 10, 2009

[7] World Wide Consortium, "State Chart XML (SCXML): State Machine Notation for Control Abstraction", 16 December 2010 (available at http://www.w3.org/TR/scxml/)

[8] World Wide Consortium, "Voice Extensible Markup Language (VoiceXML) 3.0", 16 December 2010, (available at http://www.w3.org/TR/voicexml30/)

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

OWL-BASED MODELING OF RPG GAMES

MONICA ROMAN, IOANA SANDU, SABIN C. BURAGA

Abstract. In this paper we propose a general method of using RDF and OWL models to represent the knowledge within an electronic game. Following this approach, we are able to separate programming from resource allocation (creating and allocating the AI procedures and dialog, structuring and manipulating game elements, etc.). This work demonstrates the integration of the DL-based ontology in a game by using Jena, a Java semantic framework. As a case study, we are implementing the base for a fantasy RPG type game that includes intelligent methods for several key activities: character creation, NPC generation, a simple battle system, etc.

1. Introduction

This paper proposes a new approach for creating a game architecture and storing and retrieving data from a knowledge base by using the existing semantic Web standards: RDF (Resource Description Framework) and OWL (Web Ontology Language).

Ontologies have been previously used in games for creating Artificial Intelligence, dialog or NPC generation. For this project, we set out to create a knowledge base that will help us better define a fantasy RPG (Role Playing Game) software and help us analyze possible situations that can come up at any point in the game.

In a game ontology, rather than organizing a game by its characteristics or elements, the elements themselves are organized. This approach makes exploring issues regarding games and game play much faster and easier [12].

As games have taken an important place in mass media, they have become more complex at the architectural level. They have moved from being standalone programs built for single platforms into multi-platform, networked

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68T30, 68U35.

1998 *CR Categories and Descriptors.* code [**I.2.4**]: Knowledge Representation Formalisms and Methods – *Representation languages*; code [**I.2.3**]: Deduction and Theorem Proving – *Inference engines*.

Key words and phrases. knowledge representation, game computing, RDF, OWL.

and/or Web-based systems [8] Therefore, linking data sources in a coherent manner has also become more important.

To accomplish our goals, first we created the RPC game ontology by using the Protégé system [3]. We modeled the basic elements needed by an RPG type game that are loosely influenced by the Game Ontology Projects classification [12] – for related work, see section 2. Also, we took into account the famous classical electronic games like Dungeon and Dragons, Heroes of Might and Magic III, Diablo, World of Warcraft, Dragon Age: Origins, The Elder Scrolls: Morrowind and Oblivion, etc. Details about the concrete modeling are given by section 3.

The second step was to use the Pellet reasoner [10] in order to obtain various inferences to assist users in creating game characters and selecting a certain type of a character. Additionally, several AI activities (NPC generation and behavior, battle scenarios, etc.) could be performed on the basis of these inferences.

The third phase was to implement a software prototype – briefly described in section 4 – in order to access the models and query the inferred ones. For this, we chose the well-known platform-independent Jena framework [13].

In our opinion, this proposal opens interesting possibilities in developing semantic Web-based games.

2. Related Work

On the subject of using semantic Web technologies in the context of computer games, the Game Ontology Project (GOP) [12] is proposed that started analyzing, studying and classifying games from an ontological point of view. Because terms and concepts are not very well defined in games, GOP identifies their important structural elements and the relationships between them, organizing them hierarchically. Its main subclasses are *Interface*, *Rules*, *Entity*, *Entity Manipulation* and *Goals*.

There are several disparate approaches of using ontologies concerning different aspects of game development. For example, [5] gives a proposal of using ontologies to position NPCs in an environment based on their behavior is given, [6] presents a method to generate correct stories on the basis of an ontological model, and [9] demonstrates that ontology-based retrieval for strategies are easier to adapt than those returned by classical retrieval methods (case-based planning) [7].

So far, ontologies have been used for various purposes in games, but none offered a full, detailed structure of an RPG game – including elements like: characters, story and resources –, that was integrated using Jena in a Java based game.

3. Ontological Model of the RPG Game

3.1. **Generic Resource Modeling via RDF and OWL.** In the semantic Web, everything is asserted via RDF (Resource Description Framework) [1] statements: $\langle s, p, o \rangle$ triples, where s – subject, p – predicate, and o – object are simply Web addresses (Uniform Resource Identifiers). Using this model, any kind of resources could be denoted, These are the basic elements of an ontology.

Resources are identified by an unique URI – for example:

`http://www.semanticweb.org/ontologies/2011/0/RPGontology#Warrior`

An object can be a resource or a literal (having a certain datatype), but subjects and predicates are always resources.

The OWL (Web Ontology Language) [1] is a broadly accepted ontology language used to model vocabularies for a certain domain. Its main components are classes, properties and relations.

Classes contain all the "things" that have the same properties and restrictions. Relations can be defined between classes or individuals. Properties are attributes that characterize classes or individuals (they can be either datatype properties that define a value or object properties that have as return value an object).

There are three types of OWL that vary in complexity, expressivity and flexibility (OWL Lite, OWL DL and OWL Full).

We based our work on OWL DL that supports more complex ontologies, without losing the computational completeness – the formal model is based on several species of Description Logics (DL) [2]. In OWL the main ontological constructs are classes, properties and objects. In DL, we are using concepts, roles and individuals. DL are having a special importance in providing a logical formalism for ontologies, in general, and the semantic Web, in particular [4].

Using RDF and OWL DL, we can form a knowledge base, where TBox (terminological box) component is composed by OWL constructs, and ABox (assertional box) component is including RDF statements.

3.2. **Proposed Ontological Model in the Context of RPG Games.** Our ontology focuses on characters, their quantitative features (e.g., level, health points) and the items they can use and equip, depending on their class and their race.

Using the ontological model, we can easily create a character or a fight in our game by retrieving and saving data in the ontology. For reasoning aspects, we focused our attention on Pellet [10]. Our proposed ontological model has the $\mathcal{ALCHIF}(\mathrm{D})$ logic expressivity (attributive language with complements, including functionality, role inverse, and role hierarchy constructs) [11].

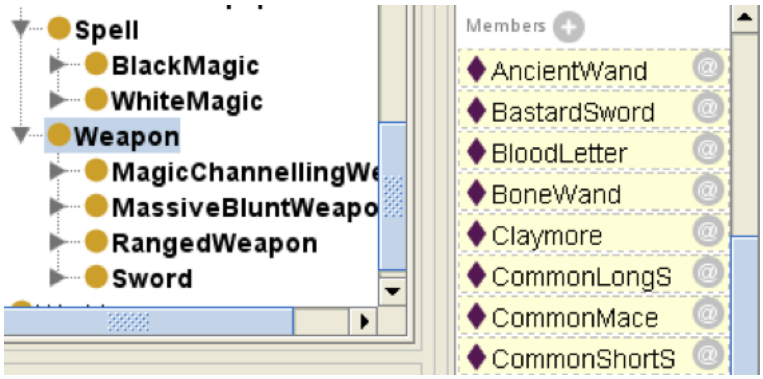


FIGURE 1. Using DL inferences to obtain individuals that are members of the class *Weapon*

3.2.1. *Population of the Game.* We started by giving a character the possibility of being one of 4 disjoint races: *Human*, *Elf*, *Orc* and *Goblin*. Each race can only use certain objects. We defined the ontological class *UsableEntiy* that contains all the entities in the game that a character can use (the equipment): weapons, armor, items and spells.

We then specified the classes *HumanEquipment*, *ElfEquipment*, *OrcEquipment* and *GoblinEquipment*, respectively, and gave them the properties that they contain objects that only humans, elves, orcs and goblins can use.

Using the Pellet reasoner, the model was enriched with the inferred appropriate subclasses and individuals:

- $HumanEquipment \equiv$ canBeUsedBy **some** *Human*
- $Human \equiv Race \sqcap \neg(Elf \sqcup Goblin \sqcup Orc) \sqcap$ canUse **only** $(Armor \sqcup HealthPotion \sqcup MagicPotion \sqcup PoisonPotion \sqcup RangedWeapon \sqcup Sword)$

A character can also have a class (group) that can only use certain objects. We defined 6 disjoint character classes: *Warrior*, *Paladin*, *Wizard*, *Warlock*, *Rogue*, and *Barbarian*. Afterwards, we created the classes *WarriorEquipment*, *WizardEquipment* and so on and then the reasoner populated them with objects that only the respective group can use. For example, *WarriorEquipment* $\equiv$ canUsedBy **some** *Warrior*.

Because class equipments have more restrictions than race equipments, they will be inferred as subclasses of the appropriate *RaceEquipment* classes. This means that we linked what objects a certain race can use and what objects a certain class can use.

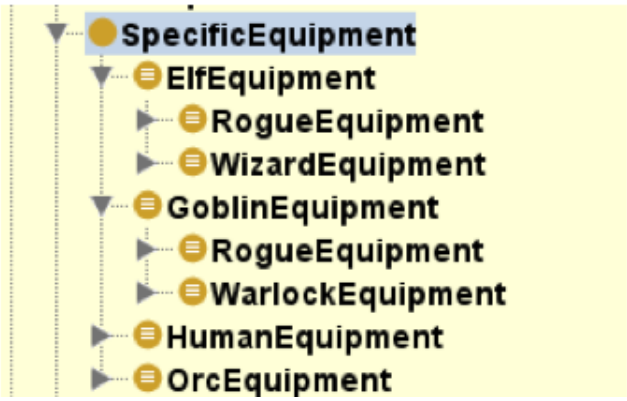


FIGURE 2. Classifying (inferring) specific equipment based on given restrictions

3.2.2. *Modeling Characters.* When creating a new character, we can choose a race for it and – based on what we specified earlier – we can get all the classes it could have (e.g., if we have an elf, then it will show us that he/she can only be a member from *Rogue* or *Wizard* classes; she/he can not be a *Warrior* because warriors can equip heavy armor, but elves can not).

Each character can have certain common quantitative features (statistics) that we modeled via datatype properties. For example, a character hasHP (Health Points), hasMP (Magic Points), hasDP (Dexterity), hasPD (Physical Defense), hasPO (Physical Offense), hasMD (Magical Defense), hasMO (Magical Offense), hasXP (Experience Points) and hasLevelNumber.

Also, every character has a gender (male or female) and a type (playable or non-playable). A NPC (Non-Playing Character) can additionally have a trait – hostile, coward and neutral – and can inhabit certain places in the world: abandoned ruin, forest, swamp, etc.

3.2.3. *Places of Interest.* For our ontology, we also created classes for shops and quests.

In shops, characters can go buy and sell usable entities – for example, a Potion Shop can have as items: some Medium Health Potions and some Minor Magic Potion, while an Armor Shop can have various types of armor.

We defined quests as a class that can have instances with data type properties like has Dialog (has Quest Accepted Dialog, has Quest Rejected Dialog, has Quest Completed Dialog, etc).

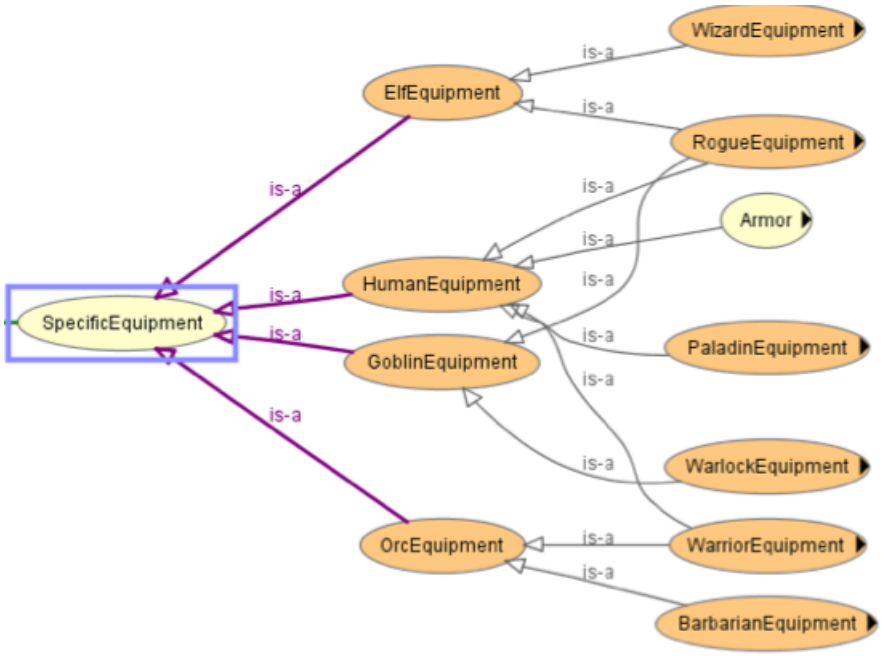


FIGURE 3. Visualizing inferred classes about equipments by using OWLViz plug-in

4. Implementation Details

We started by analyzing RPG games in general and studying what elements we would need for our game.

We first built a knowledge base that contains almost all of the games information and searched for ways it could help us faster develop a game. For the construction of the ontology, the best choice was to use Protégé system [3], the most widely used knowledge-base editor. It allows creating, editing and visualizing RDF or OWL specifications and automatically checks for inconsistencies. It also automatically populates classes with instances. It offers a substantial number of plug-ins and can be extended by third-party ones. The OWLViz plug-in enables a clear visualization of the asserted and inferred classes hierarchy.

We then searched for the best suited programming language that could give a proper support regarding the use of ontologies in a pragmatic manner. One of the most important aspect is to create and integrate an ontology model into a computer game.

We chose Java platform and Jena framework [13] built to be used within semantic Web applications. It allows working with RDF, RDFS, OWL, SPARQL and includes a rule-based inference engine. It connects our OWL ontology to the Java-based game prototype and enables reading and writing data in our ontological model (via OWL API). The Jena Ontology API in conjunction with a powerful reasoner (a built-in or an external) becomes a very useful instrument.

For our prototype, we stored information about individuals (the ABox compartment of the knowledge base) into RDF documents. For instance, the below mentioned RDF file holds the default values for all the game character classes: *Warrior*, *Paladin*, *Wizard*, etc. These values are specified for level 1 characters that well be used when a new playing character is created.

```
<rdf:Description rdf:ID="Warrior">
   <!-- A warrior is a game character -->
   <rdf:type rdf:resource="#Character"/>

   <hasHP rdf:datatype="xsd:byte">35</hasHP>
   <hasMP rdf:datatype="xsd:byte">0</hasMP>
   <hasDP rdf:datatype="xsd:byte">5</hasDP>
   ...
   <hasLevelNumber rdf:datatype="xsd:byte">1</hasLevelNumber>
</rdf:Description>
```

We used both the Jena implicit reasoner and the Pellet one. While trying to list classes and subclasses from our ontology in Jena, we encountered a problem. The ontology seemed to be too big or complex to process and with the default reasoner we got the results in approximately 30 seconds (unacceptable in a real game). When we imported Pellet, we got the inferred results in about 2 seconds. For our ontology, Pellet seems to be a more powerful and faster reasoner.

5. Conclusions and Further Work

This paper demonstrates the integration of an ontology in an RPG game using the Jena framework. By being able to define game rules ontologically and reason with the instances created against that ontology, our proposal could be view as an encouraging approach of using semantic Web technologies in the context of computer games, as an alternative method for the classical AI methods.

In the near future, we intend to extend our research by continuing to expand the ontology, to study certain complexity aspects, and creating a full working game with it.

References

[1] Allemang, D., Hendler j. *Semantic Web for the Working Ontologist*, Morgan Kaufmann, 2008

[2] Baader F., Horrocks, I., Sattler, U. "Description Logics", *Handbook of Knowledge Representation*, Elsevier, 2007

[3] Gennari, J. et al. "The Evolution of Protégé: An Environment for Knowledge-Based Systems Development", *International Journal of Human-Computer Studies*, Volume 58, 2002

[4] Horrocks, I. "Ontologies and the Semantic Web", *Communications of the ACM*, Volume 51, Issue 12, 2008.

[5] Machado, A., Amaral, F., Clua, E. "A Trivial Study Case of the Application of Ontologies in Electronic Games", *Proceedings of the VIII Simposio Brasileiro de Jogos e Entretenimento Digital*, Sociedade Brasileira da Computaçao, 2009

[6] Peinado, F., Gervas, P., Daz-agudo, B. "A Description Logic Ontology for Fairy Tale Generation", *Forth International Conference on Language Resources and Evaluation: Workshop on Language Resources for Linguistic Creativity*, 2004

[7] Menkovschi, V., Metafas, D. "AI Model for Computer games based on Case Based Reasoning and AI Planning", *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*, ACM Press, 2008

[8] Mepham, W. "Semantically enhanced games for the Web", *Proceedings of the WebSci'09: Society On-Line*, Athens, Greece, 2009

[9] Sanchez Ruiz-Granados, A. et al. " Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based Retrieval", *Proceedings of the ICAPS-07 Workshop on ICAPS 2007 Workshop on Planning in Games*, AAAI Press, 2007

[10] Sirin, E. et al. "Pellet: A Practical OWL-DL Reasoner", *Web Semantics: Science, Services and Agents on the World Wide Web*, Volume 5, Issue 2, Elsevier, 2007

[11] Zolin, E., *Complexity of Reasoning in Description Logics* – Available online at `http://www.cs.man.ac.uk/ ezolin/dl/`

[12] * * *, *Game Ontology Project* – Available online at `http://www.gameontology.org/`

[13] * * *, *Jena – A Semantic Web Framework for the Java Platform* – Available online at `http://jena.sourceforge.net/`

Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iasi, Berthelot Street, 16 – Iasi, Romania

E-mail address: `busaco@info.uaic.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

STEREOMATCHING USING RADIOMETRIC INVARIANT MEASURES

ALINA MIRON[1,2], SAMIA AINOUZ[1], ALEXANDRINA ROGOZAN[1], ABDELAZIZ BENSRHAIR[1], AND HORIA F. POP[2]

Abstract. Stereo Matching can provide important information about the environment for many autonomous systems. Usually one of the demands for these systems is real-time or almost real-time running. This is why we studied how to adapt two stereo matching measures to this demand, by providing an implementation on a GPU . For stereo matching algorithm a simple local approach based on window aggregation was chosen, and as pixels measures were used two radiometric insensitive measures: Census transform and Daisy features. Also, in response to the on going debate of whether to use color information in stereo matching, experiments were performed using different color spaces for both of the radiometric insensitive measures considered.

1. Introduction

Stereo Vision is a technique for obtaining 3D information from a given scene by using a pair of cameras. There exist many applications for this technique especially in the field of automated systems. For example in the field of robotics or autonomous driving systems, stereo vision can be used to extract relative position of objects or to separate occluding image components, such as one person in front of another one.

In this paper the focus is on the stereo matching step of the stereo vision flow. An important challenge for finding reliable correspondences in a pair of stereo images of a scene is the fact that images could present radiometric distortions and great baseline distance (if the cameras have a significant distance of each other). The stereo matching algorithms could be divided in local and global approaches. Local approaches rely mostly on a support window (typically square), which can be implemented efficiently using a sliding window technique [7]. In the global approaches typically the problem is expressed as

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68T45.

1998 *CR Categories and Descriptors.* I.4.8 [**Computing Methodologies**]: Image Processing and Computer Vision – *Scene Analysis*.

Key words and phrases. stereo vision, census transform, daisy features, GPU.

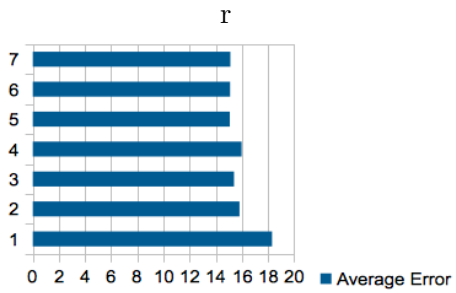


FIGURE 1. Average Error for different masks configuration. (1) Dense mask, (2) Star, (3) Check 2, (4) Check with fortified core, (5) Check 3, (6) Check 4, (7) Check 5.

a energy function that must be optimized. This can be done using dynamic programming [2], graph-cuts [8] or belief-propagation [5].

As is often the case for real time applications, there must be found a trade-off between computational cost and the disparity map precision. That is why in practice local methods for stereo matching are used. For these, there exist different pixels descriptors. The simpler would be the color intensity, and probably the most complex are the local region descriptors (like SIFT). For our experiments we chose to use the following pixels descriptors: census transform [11] and daisy features [10].

The rest of the abstract is organized as follows: Section 2 presents the results of the implementation on GPU and experiments performed with different color spaces, and in Section 3 are presented the conclusion and further work.

2. Experiments

2.1. **GPU computation.**

2.1.1. *Census Transform.* The first algorithm which would be suitable for a GPU implementation is the Census Transform. This is because the values computed for each pixel are independent, thus the algorithm is highly parallelizable. If on the CPU implementation the algorithm runs in 0.5 sec, in the GPU implementation the algorithm runs almost 35 times faster, in 0.015 sec.[1] Like we previously stated , for the census transform a check like census mask was used where it is taken into consideration each 3^{rd} pixel at each 3^{rd} row.

The size of the census window was chosen in an empirical way to be 15. This is because a window size of 15 will give in a check like configuration a total

[1]All the experiments were performed on laptop having a CPU with 2.4 GHz Intel Core 2 Duo, and a GPU NVIDIA GeForce 320M with VRAM 256MB

TABLE 1. Comparison between computation time between CPU and GPU per major operations for Daisy features

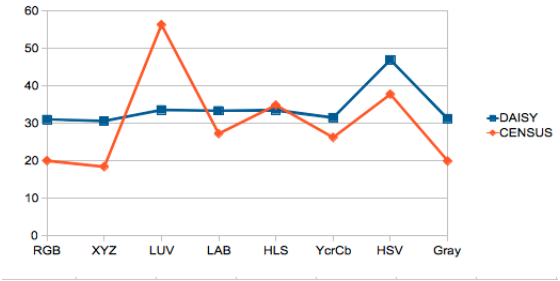
Operation	CPU time (sec)	GPU time (sec)
Image loading from disk	0.01	0.01
Copy Image from CPU RAM to GPU RAM	-	0.045
Daisy Set Parameters	0.000150	0.000184
Daisy Initialize Descriptor	0.36	0.08
Compute Daisy features	0.532	0.22
TOTAL time	0.9	0.35

of 25 relevant pixels. This bit string of 25 elements can be represented as a simple number of integer datatype. In figure 1 are shown comparative average errors obtained for different configuration masks. As it can be seen keeping the same number of relevant pixels but using a check like configuration better results are obtained. The dataset used to run this experiment was formed from 4 pairs of stereo images taken from the Middlebury database [9]. The distance between two bit strings is a simple Hamming distance that can be implemented efficiently in the form of a bitwise exclusive or.

2.1.2. *Daisy Features.* According to the authors of [10] for an image of 800×600 the computation time for Daisy features is around 3.8sec. In our implementation of Daisy features the running time for an image of 450×375 is around 1sec. This is still not fast enough for a real time application and that is why we decided to try a GPU implementation of these features. At the time of writing of this paper there didn't exist a GPU implementation for daisy features. For the GPU implementation we started implementing all the basic operations. One of the basic operations that is used intensively in the daisy algorithm is a 1D gaussian convolution of the image. The convolution time in our CPU implementation is about $4.6 * 10^{-3}$. After the GPU implementation the new time of the convolution operation is about $5 * 10^{-5}$, therefore the GPU implementation of the convolution runs almost a hundred times faster than the CPU counterpart.

In table 1 can be saw a comparison between the computation time of the CPU and GPU implementations for Daisy features. Even if the computation of Daisy features is still not fast enough for a real-time running, with more optimization (more use of the shared memory on GPU, loading the image directly in the GPU RAM) the running time could be decreased even more to reach near-real time levels.

FIGURE 2. Mean Error across different color spaces



2.2. **Color Information for radiometric invariant measures.** In stereo matching, there exist different opinions in what concerns the use or not of color information. There exist different studies that prove the performance of color-based stereo in comparison with grey-scale matching [3], [7] . According to [6] by using color information instead of gray values the improvement obtained is around 25%. In another study [4], the authors evaluate different matching measures that are insensitive to radiometric distortions and they stated that color information didn't help in stereo matching. Other study [1] claims the fact that color information is less robust with the radiometric insensitive measures, leading to a degradation, so they state the fact that color should not be used in stereo matching. This is why we tried to reproduce the experiments made by [1] with census measure, but with two important differences: the first one is the fact that there is no energy minimization function used (because this could interfere with the actual results obtained), and the second difference is that the results are validated on another radiometric invariant measure, that provided by the daisy features.

In figure 2 can be saw the results of stereo matching experiments using Census measure and Daisy features. There were used for the experiments eight different color spaces: *RGB, XYZ, LUV, LAB, HLS, YCrCb, HSV, Grayscale.* The first remark for this figure is that this is not intended to be a comparison between Census transform and Daisy features, because they address different problems: Daisy can be used for wide baseline stereo matching while Census is used for standard stereo matching. This is why for the set of images from Middlebury database used (four images with small baseline) the Census measure obtains better results than Daisy. Instead of making a comparison between the two measures we are more interested on the color space trend comparison. For the Census transform the results obtained when using for pixel color comparison a simple sum over the channels are: the smallest mean error was

obtained when using the XYZ color space: *18.4%*, followed by RGB and Grayscale with *19.9%*. The interesting fact is that the same order is preserved when using as measure Daisy features: for XYX color space the error is *30.58%*, followed by RGB with *30.96%* and Gray with *31.21%*. Therefore, color didn't lead to a degradation of results when using in combination with radiometric invariant features, but in fact it improved the results.

The presented results were obtained by using a simple sum over the channels. This fact may not do justice to all the color spaces and that is why an experiment was performed with optimized weights of the three channels using genetic algorithm (there were not optimized the weights for RGB color space). The results improved for almost all the color spaces but the best results were still obtained by XYZ color space. Also because the initial database of testing images was small, we performed the same experiments on the TNO/MARS PRESCAN [2] but the results remained consistent.

3. Conclusion

The power of parallel computation given by the GPUs can be used to reach real-time or near real-time running for some stereo matching algorithms. Here, there were presented two radiometric insensitive measures that are suitable for a GPU implementation. Even if in numerical error the Census transform performs better than Daisy features for the dataset of images used, the latter could represent a solution for a general approach for both standard and wide base line stereo matching. In what concerns the use of color for radiometric insensitive measures, based on our results we can conclude that color information taken as a simple sum over channels from XYZ color space performs better than Grayscale.Some could argue that the improvement brought by the color information is not significant enough to compensate a greater running time needed for the transformation in that color space. But stereo vision does not represent the final goal: color information may also be useful for further object detection. As future work we plan to integrate a way for fast image segmentation because this can provide important information in low texture areas (that appear widely for example in a traffic situation).

References

1. M. Bleyer and S. Chambon, *Does Color Really Help in Dense Stereo Matching?*, Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT), vol. 2010, 2010.
2. M. Bleyer and M. Gelautz, *Simple but effective tree structures for dynamic programming-based stereo matching*, International Conference on Computer Vision Theory and Applications (VISAPP), Citeseer, 2008.

[2]http://staff.science.uva.nl/ wvdmark/StereoVisionMarsPrescan/

3. S. Chambon and A. Crouzil, *Colour correlation-based matching*, International Journal of Robotics and Automation **20** (2005), no. 2, 78–85.
4. H. Hirschmuller and D. Scharstein, *Evaluation of cost functions for stereo matching*, Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, IEEE, 2007, pp. 1–8.
5. A. Klaus, M. Sormann, and K. Karner, *Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure*, Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, vol. 3, IEEE, 2006, pp. 15–18.
6. R. Klette, A. Koschan, K. Schluns, and V. Rodehorst, *Surface reconstruction based on visual information*, TR95/6, July (1995).
7. K. Muhlmann, D. Maier, J. Hesser, and R. Manner, *Calculating dense disparity maps from color stereo images, an efficient implementation*, International Journal of Computer Vision **47** (2002), no. 1, 79–88.
8. N. Papadakis and V. Caselles, *Multi-label depth estimation for graph cuts stereo problems*, Journal of Mathematical Imaging and Vision (2010), 1–13.
9. D. Scharstein and R. Szeliski, *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms*, International journal of computer vision **47** (2002), no. 1, 7–42.
10. E. Tola, V. Lepetit, and P. Fua, *DAISY: An Efficient Dense Descriptor Applied to Wide-Baseline Stereo*, IEEE transactions on pattern analysis and machine intelligence (2009), 815–830.
11. R. Zabih and J. Woodfill, *Non-parametric local transforms for computing visual correspondence*, Computer VisionECCV'94 (1994), 151–158.

(1) Laboratoire d'Informatique, de Traitement de l'Information et des Systemes , INSA Rouen, Avenue de l'Universite, 76800, Saint-Etienne-du-Rouvray, France, web:www.litislab.eu,

(2) Babeş-Bolyai University, Faculty of Mathematics and Computer Science,, 1, Mihail Kogalniceanu St., 400084, Cluj-Napoca, Romania, web:www.cs.ubbcluj.ro,

E-mail address: alina.miron@insa-rouen.fr

E-mail address: samia.ainouz@insa-rouen.fr

E-mail address: alexandrina.rogozan@insa-rouen.fr

E-mail address: abdelaziz.bensrhair@insa-rouen.fr

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

AVERAGE BANDWIDTH REDUCTION IN SPARSE MATRICES USING HYBRID HEURISTICS

LIVIU OCTAVIAN MAFTEIU-SCAI, VIOREL NEGRU, DANIELA ZAHARIE, AND OVIDIU ARITONI

Abstract. This paper proposes a hybrid heuristic aiming the reduction of the average bandwidth of sparse matrices. The proposed heuristic combines a greedy selection of rows and columns to be interchanged with an approach based on a genetic-inspired crossover. Several ideas of limiting the search tree are also investigated. Preliminary numerical experiments illustrate the fact that the proposed heuristic leads to better results with respect to the average bandwidth than the classical Cuthill-McKee algorithm.

1. The Matrix Bandwidth Reduction Problem

Reducing the bandwidth of sparse matrices is important in solving large linear systems of equations and is especially useful in designing efficient parallel implementation of the solving procedures. The bandwidth of a matrix is related to the concentration of nonzero elements around the main diagonal and can be measured in different ways. The most frequently used bandwidth measure is the maximum of the absolute value of the difference between the row and column indices of nonzero elements, i.e. $bw = \max\{|i-j|; a_{ij} \neq 0, i = \overline{1,n}, j = \overline{1,n}\}$ for a square matrix A of size n. However, the bw measure does not provide too much information about the grouping pattern of non-zero elements around the main diagonal, which is particularly important in balancing the load on different processors in the case of parallel implementations [1]. This motivates the interest in defining and using other measures of the compactness of the non-zero elements around the main diagonal. In this work we use the average bandwidth proposed in [8] and defined by Eq. (1), where m

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68T20, 65F30.

1998 *CR Categories and Descriptors.* I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Heuristic Methods*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra – *Sparse, structured, and very large systems (direct and iterative methods).*

Key words and phrases. bandwidth reduction, sparse matrices, heuristic algorithms.

denotes the number of non-zero elements of the matrix A.

$$mbw = \frac{1}{m} \sum_{a_{ij} \neq 0} |i - j| \tag{1}$$

If the distribution of the non-zero elements is symmetric with respect to the main diagonal, the average bandwidth can be computed by taking into account only the non-zero elements which are in the upper triangular part of the matrix, i.e. $i < j$. Minimizing the average bandwidth instead of bw has the advantage of leading to a more uniform distribution of non-zero elements around the main diagonal.

The bandwidth reduction problem consists of finding a permutation of rows and columns such that the resulting matrix has a smaller value for the bandwidth measure. Most methods designed to solve the matrix bandwidth reduction problem are based on the corresponding graph formulation: find a labeling of the vertices of a graph such that most connections are between vertices having close labels.

The methods for bandwidth matrix reduction can be grouped in two main classes: exact and heuristic ones. The exact methods are mainly based on branch and bound search for the optimal labeling of vertices (which is equivalent with an optimal permutation of rows and columns in the matrix). The branch and bound approach proposed by DelCorso and Manzini [4] is based on the concept of upper partial ordering and on an estimation of the lower bound used to prune some nodes in the search tree [1]. Caprara and Salazar [2] improved the lower bound given in [4] and introduced a control mechanism of the enumeration process by introducing new pruning criteria. A more recent approach is proposed in [9]. Its main particularity is the fact that it combines the branch and bound search with some information collected based on a heuristic search. The main disadvantage of the exact methods is related to their computational cost. This computational cost can be reduced by using heuristic techniques (which, on the other hand, do not guarantee the solution optimality). One of the best known heuristic technique, used in many comparative studies, is that proposed by Cuthill and McKee [3]. It is based on generating labelings based on some rooted level structures and has the disadvantage of being costly for some configurations and of providing solutions with a bandwidth which can be far from the optimal value. In order to improve the behavior of the Cuthill-McKee heuristic, another technique, also based on rooted level structures, has been proposed in [5]. In the last years approaches based on various metaheuristics have been proposed: tabu search [10], genetic algorithms [7], particle swarm optimization [6] etc.

[1] An implementation is available at www.mfn.unipmn.it/~Manzini/bandmin.

The method proposed in this paper does not use neither the graph formulation nor the rooted level structures but relies on applying successive transformations directly to the matrix. Other particularities of the proposed method are: (i) it uses the average bandwidth as primary quality measure; (ii) it combines a greedy-like selection of rows/columns to be swapped with a genetic inspired crossover; (iii) it uses an elitist selection of the active nodes in the search tree, leading to an effective pruning technique.

2. The Proposed Hybrid Heuristic

The main idea of the proposed heuristic is that of reducing the average bandwidth by applying successive perturbations based on swapping pairs of rows and corresponding pairs of columns. The key elements of the heuristic are the choice of the rows/columns to be swapped and the decision of stopping the sequence of perturbations applied to a given configuration.

Unlike most heuristic bandwidth reduction algorithms which explore a space of permutations, the proposed heuristic directly explores the space of matrices. The search tree is constructed by identifying at each step the appropriate perturbations. More specifically, for each configuration (node in the search tree) the following operations are executed:

Step 1. Find all zero elements which are placed inside the band defined by the bandwidth measure, i.e. the set $S = \{(i,j); a_{ij} = 0, i < j, |i-j| \leq imbw\}$, where $imbw$ denotes the ideal average bandwidth.

Step 2. For each $(i,j) \in S$ find j' which maximizes the difference $|j'-i|$ and satisfies $|j'-i| > imbw$ and $a_{ij'} \neq 0$, construct a swap pair (i,j').

Step 3. For each swap pair selected at the previous step the corresponding rows and columns of the current matrix are swapped leading to new matrices. The bandwidth measure(s) are computed for all newly generated matrices.

Step 4. From the set of matrices constructed at the previous step is selected the subset of so-called elites consisting of all matrices having a smaller average bandwidth than the parent matrix.

Step 5. Starting from the pairs of indices used in obtaining the elite matrices, new pairs of indices are constructed by a genetic inspired crossover. More specifically if (i_1, i_1') and (i_2, i_2') are two pairs which led to improved matrices, then the following pairs are constructed: (i_1, i_2') and (i_2, i_1'). The matrices corresponding to all pairs obtained by crossover are constructed and their bandwidth measures are computed.

The search tree is further developed by branching only the nodes corresponding to improved matrices (elites). The branching process can be based either on a breadth first strategy or on a depth first strategy. In both cases if all newly generated matrices from a current node have a worse structure (i.e. higher value of the average bandwidth), random swap pairs can be generated

(similar to a mutation operation). This would avoid a premature stopping of the search process. On the other hand, before proceeding with the exploration of the new nodes, the elements of the elites subset generated from a given node are increasingly sorted based on the bandwidth measures (mbw is the primary sorting criterion and bw is the secondary sorting criterion). This could favor an earlier generation of good configurations.

The set of criteria used to stop the search contains: (i) finding a matrix with the desired value of the bandwidth; (ii) reaching a maximal number of steps without improvement; (iii) generating the entire search tree.

This heuristic technique for average bandwidth reduction has a structure similar to that proposed in [8] but it also has some particular characteristics: elitist selection of nodes to be further processed and generation of swap pairs using crossover and mutation operators.

3. Preliminary Results

Experiment 1. The aim of the first set of experiments was to check if the algorithm is able to reach the ideal solution ($imbw$ and ibw), if it exists. To this end, symmetric band matrices were generated and then perturbed using a pre-established perturbation ratio. For instance, for a band matrix of size 50×50, containing 100 non-zero values, having an initial bandwidth $ibw = bw_0 = 3$ and an initial average bandwidth $imbw = mbw_0 = 1.54$, the perturbation process generated a new matrix characterized by $bw_0 = 37$ and $mbw_0 = 13.5$. Three algorithms were applied: the Cuthill-McKee algorithm, the heuristic algorithm described in [8] and the hybrid algorithm proposed in this paper. In all tested cases the proposed algorithm constructed the optimal solution and the row/columns swap operations used in the perturbation step were rediscovered by the algorithm. This experiment also illustrated the importance of sorting the nodes in the search tree. An illustration of the results obtained for a symmetric matrix of size 10×10 containing 32 nonzero elements is presented in Figure 1. As can be seen in the figure the results produced by the hybrid algorithm (Hybrid a) and Hybrid b) correspond to different branches in the search tree) are characterized by a compact grouping of non-zero elements around the main diagonal. These results also illustrate the impact on using the average bandwidth as primary selection criterion in the search process. The results obtained for several randomly generated sparse matrices having sizes between $n = 15$ and $n = 100$ are presented in Table 1. For all matrices the hybrid algorithm led to the smallest value of the average bandwidth.

Experiment 2. In this experiment we used matrices from the DWT set included in the collection Harwell-Boeing collection [2]. Some results are presented in the Table 2. The hybrid heuristic led in almost all cases to the smallest

[2]Available at http://math.nist.gov/MatrixMarket/data/Harwell-Boeing.

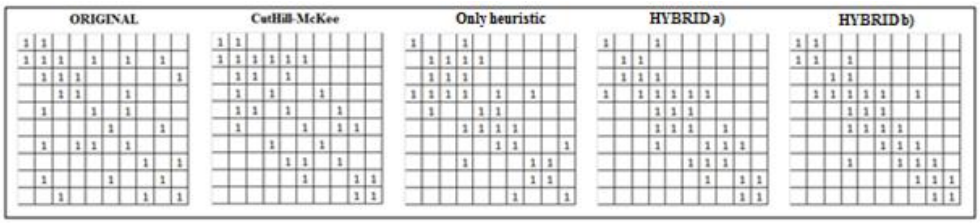


FIGURE 1. Matrices obtained by applying the Cuthill-McKee algorithm, the heuristic algorithm described in [8] and the hybrid algorithm.

TABLE 1. Results obtained for randomly generated sparse symmetric matrices (n is the matrix size, m is the number of non-zero elements.)

Matrix	Initial		Ideal case		Cuthill-McKee heuristic		Greedy heuristic [8]		Hybrid heuristic	
n/m	bw_0	mbw_0	ibw	$imbw$	bw	mbw	bw	mbw	bw	mbw
15/30	11	5.16	3	1.63	3	2.00	4	2.00	3	**1.80**
20/24	18	7.16	2	1.20	5	2.62	6	2.41	5	**2.20**
20/95	18	6.94	6	3.21	12	5.57	10	4.42	10	**4.35**
51/133	46	16.40	3	1.88	23	11.80	32	8.98	21	**6.02**
100/1650	98	34.00	19	9.32	33	38.15	78	16.13	38	**14.00**

TABLE 2. Results for sparse matrices selected from the Harwel-Boeing collection.

Matrix	Initial		Ideal case		Cuthill-McKee heuristic		Greedy heuristic [8]		Hybrid heuristic	
	bw_0	mbw_0	ibw	$imbw$	bw	mbw	bw	mbw	bw	mbw
dwt59	25	5.65	2	1.44	8	4.30	20	5.11	20	**4.09**
dwt66	44	8.34	2	1.49	3	1.88	3	1.51	4	**1.50**
dwt72	12	2.40	2	1.05	8	4.68	12	2.40	9	**2.35**
dwt87	63	19.51	3	1.86	18	**6.36**	56	10.88	31	6.40
dwt162	156	13.43	4	2.11	20	7.61	97	13.34	24	**6.65**

value of the average bandwidth. However the obtained results suggest that the effectiveness of the hybrid approach can be still improved.

4. CONCLUSIONS

The hybrid heuristic which combines the greedy choice of the pairs of rows/columns to be swapped with pairs generated by crossover starting from

elite configurations proved to lead to better results than the heuristic variant proposed in [8]. The experimental results show that the proposed technique can be improved both with respect to the quality of provided solutions and with respect to its efficiency. Therefore further research will address the investigation of other strategies for exploring the search space and the parallel implementation. We will deveat the proposed technique can be improved both with respect to the quality of provided solutions and with respect to its efficiency. Therefore further research will address the investigation of other strategies for exploring the search space and the parallel implementation. We will develop in the future an recommendation system that will help us to indicate the most useful bandwidth reduction method for our matrix.

References

1. P. Arbenz, A. Cleary, J. Dongarra, and M. Hegland, *Parallel numerical linear algebra*, Nova Science Publishers, Inc., Commack, NY, USA, 2001, pp. 35–56.
2. A. Caprara and J. Salazar-González, *Laying out sparse graphs with provably minimum bandwidth*, INFORMS J. on Computing **17** (2005), 356–373.
3. E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, Proc. of ACM (New York, NY, USA), ACM, 1969, pp. 157–172.
4. G. M. Del Corso and G. Manzini, *Finding exact solutions to the bandwidth minimization problem*, Computing **62** (1999), 189–203.
5. N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM Journal on Numerical Analysis **13** (1976), 236–250.
6. A. Lim, J. Lin, and F. Xiao, *Particle swarm optimization and hill climbing for the bandwidth minimization problem*, Applied Intelligence **26** (2007), 175–182, 10.1007/s10489-006-0019-x.
7. A. Lim, B. Rodrigues, and F. Xiao, *Heuristics for matrix bandwidth reduction*, European Journal of Operational Research **174** (2006), no. 1, 69 – 91.
8. Mafteiu-Scai L.O., *Bandwidth reduction on sparse matrix*, West University of Timişoara Annals (Timişoara, Romania), vol. XLVIII fasc. 3, 2010.
9. R. Marti, V. Campos, and E. Pinana, *A branch and bound algorithm for the matrix bandwidth minimization*, European Journal of Operational Research **186** (2008), 513–528.
10. R. Marti, M. Laguna, F. Glover, and V. Campos, *Reducing the bandwidth of a sparse matrix with tabu search*, European Journal of Operational Research **135** (2001), 450–459.

West University of Timişoara, Department of Computer Science, 4 Blv. Vasile Parvan, Timişoara, România

E-mail address: {lscai,vnegru,dzaharie,oaritoni}@info.uvt.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

APORT: WEB PORTAL FOR ASSISTING TEACHING

R.F. BOIAN, S.M. DRAGOŞ, AND C. COBÂRZAN

Abstract. Traditional teaching methods must deal today with challenges such as handling effectively a large number of students, keeping the interaction meaningful and focused on the course material, insuring student's academic integrity, etc. The paper presents a work-in-progress system that draws its approach from the way collaboration takes place in the business world, as well as from the experience gathered implementing and using two older e-learning systems. The proposed approach provides features that help make interaction and communication a first class element of the teaching process.

1. Introduction

The interaction between students and professors during regular academic activities such as lectures, seminars, and lab hours would benefit the student more if it were focused on the aspects that matter most to the student's understanding of the course material. This process is unfortunately often affected negatively by issues such as: lack of interaction means, insufficient time during the scheduled hours, checking the integrity of the students' solutions, performing class administrative work. The problem becomes more acute when the ratio between the number of teaching faculty members and the number of students is low.

The use of a computerized system to assist teaching is a widely used approach today. Extensive work has been done in this field and numerous e-learning systems exist both as research projects and as commercial products. The essence of the problems presented above is a need for solid and consistent mean of collaboration between professors and students, for both frontal and remote interaction. The collaboration requires support for presenting new material, for discussing the presented material, for reviewing and commenting on the requirements and solutions, and most of all for keeping the focus on

Received by the editors: April 10, 2011.

2010 *Mathematics Subject Classification.* 68U35, 68M14, 68M11.

1998 *CR Categories and Descriptors.* K.3.1 [**Computers and Education**]: Computer Uses in Education – *Collaborative learning*.

Key words and phrases. e-learning, web portal, multimedia.

the matter at hand without the need to divert attention to aspects unrelated directly to the discussion. These requirements are quite similar to what IT companies need in order to insure smooth collaboration between remote offices. The Aport system presented in this paper takes its inspiration from such business approaches and provides features that help make interaction and communication a first class element of the teaching process. Another key aspect of the proposed system is the focus on continuous improvement based on direct user feedback as well as information provided by an integrated web analytics module.

The work presented here relies on the experience gathered by implementing and using two other such portals in our department. The Aport system is meant to replace the older systems [6, 4] but it is currently only in a development stage.

2. Collaborative Features

The collaborative features are meant to assist the professor and the students in collaborating remotely, thus reducing the time wasting traveling to the meeting point while keeping the information exchange alive and meaningful.

2.1. **Audio/Video Conferencing.** The most intuitive way of exchanging information is though a audio/video conference. In the teaching context, such a conference will have a large number of participants which will require high bandwidth availability [5]. To reduce the need for such bandwidth, the video streaming will be going exclusively from the professor to the students. The audio interaction will have the professor acting as a moderator and main speaker. The students will have to ask for permission to speak before being given control of the microphone.

Video is already used by many e-learning platforms, however it is used for publishing pre-recorded content that the students can view offline.

2.2. **Threaded Instant Messaging.** Instant messaging is a very effective tool for real-time information exchange. The system will provide a specialized instant messaging feature, which will help the participants organize the discussion into threads, thus avoiding the usual confusion that appears in such conferences with regard to whom talks to whom and what answers belongs to which question. Figure 1 shows a concept drawing of how the threaded chat looks like. participants can ask side questions on existing topics, which can be either replied in their thread or linked to the answer already existing in another thread.

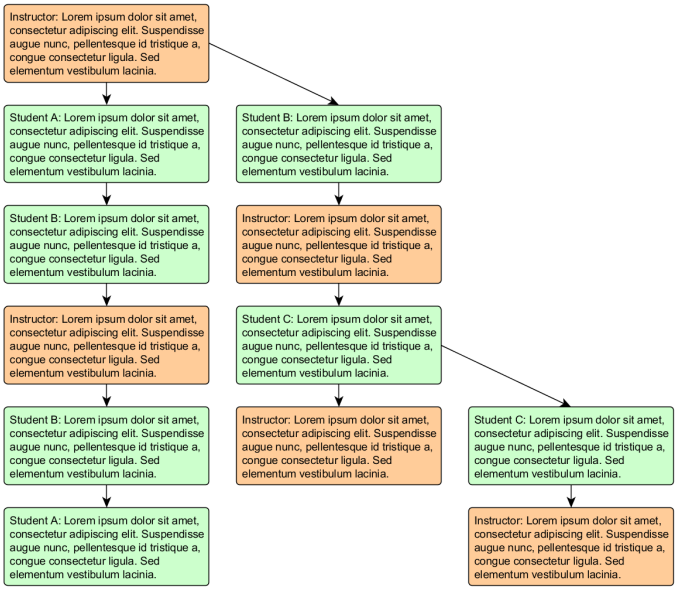


FIGURE 1. Threaded instant messaging

2.3. **Remote Assistance.** Often enough students working independently at home face problems that they cannot solve, yet getting help from a professor proves time-consuming since they must schedule a meeting and sometimes they even have to bring their computer to show the issue. Web-based remote assistance tools are a proven technology which can be found as open source and commercial products. The Aport system will include such a feature to allow fast and effective student assistance.

There are already free solutions for remote desktop access (e.g. Mikogo [2]) but they are usually standalone applications.. We would like such a solution to be integrated with our portal without the need for additional plugins.

2.4. **Support for Students with Disabilities.** As technology becomes available, the access to higher education becomes available easier to students with disabilities such as lack of eye sight. Teaching computer science to a blind person raises serious challenges, but it is all the more rewarding. To assist such students in easily obtaining their information, the portal will be designed with an alternative hierarchical interface which is easier to comprehend through the use of an automated screen reader.

3. Dealing with Plagiarism

The easy access to information through the World Wide Web and the numerous tools for file sharing, make plagiarism a very tempting solution for students who want to solve their assignments fast and without effort. Accurate evaluations in this context are difficult and time consuming tasks, which take the professor's focus away from the main goal of the course.

3.1. **Prevention.** Our approach to discouraging plagiarism is to create a large set of problems for each project and to arbitrarily assign them to students. This reduces the number possible solution sources, but also can lead to unfair assignment of several difficult problems to the same student. The system will provide means for marking each problem's difficulty and an algorithm for arbitrary assignment that keeps the overall difficulty level balanced among students.

3.2. **Detection.** The system will also feature an integrated plagiarism detector which will compare each student's solution to an already existing set of solutions. The detector will be able to ignore certain blocks of source code that were provided by the professor as example in class and which consequently are naturally used by students in their solutions.

The plagiarism algorithm we are designing is inspired from the work of Goel, Rao et al in [3] and Schleimer, Wilkerson, and Aiken in [7]. We will analyze the source code in blocks with and without comments, and with and without taking into account the names of the variables and functions. The analysis result will report an overall similarity score as well as similarity reports between the major blocks. A mandatory feature is the possibility to ignore certain blocks when performing the comparison.

4. Assisting Frontal Teaching

4.1. **Presentation Slides.** A major problem faced by students during frontal teaching is taking accurate and detailed notes while still following the flow of the lecture. While many succeed in this task there are even more who fail. The Aport portal will provide the professor with means of creating presentation slides that can be shown during lectures and on which the students can make on-line annotations at the same time. This will also provide the students with the possibility of sharing class notes in a uniform manner. The professor can also consult this notes and get a good insight into the student's understandings and confusions.

4.2. **Online Solution Review.** The portal will provide support for online solution review and annotation, with the possibility of attaching forum-like discussions to each annotation. This will reduce the amount spent in class explaining the issue, and leaving time for answering more specific questions from the students.

One challenge of the solution review is the possibility of annotating any file of the loaded solutions, regardless of their type and format. The solution will be to provide converters for all the accepted file types, that will transform the files into HTML code. Unknown files will be treated as test files by default, unless their type is known to be binary.

5. Automatic Grade Calculation

Calculating the grade can mean as little as performing a mean average of several scores or as complicated combining tens of grades and scores into a final grade. A large number of grades appears sometimes out of the need to keep the students working constantly during the semester and consequently providing weekly projects, quizzes, and homework. The portal will provide the professor with means of specifying the grade calculation algorithm either graphically or programmatically.

The easiest and most flexible implementation of this feature would offer the user a text box in which to right the grading algorithm in a programming language such as JavaScript. While this approach would be suitable for users familiar with programming, it would be unusable by the majority of users. We are investigating graphical solutions such as the Lily JavaScript Visual Programming Tool [1].

6. Effectiveness Evaluation And Feedback

The impact of an e-learning system on the students is determined among other things by the ease of use and consistent use cases. Finding the best way to present and navigate the information is a difficult problem, when lacking data to analyze. The portal will feature an integrated user feedback module and a web analytics engine which will help the authors evaluate and improve its effectiveness through interface changes.

Acknowledgment

This material is partially supported by the Romanian National University Research Council under award PN-II IDEI 2412/2009.

References

1. *Lily quick start*, http://www.lilyapp.org.
2. *Mikogo*, http://www.mikogo.com.
3. Christian Arwin and S. M. M. Tahaghoghi, *Plagiarism detection across programming languages*, Proceedings of the 29th Australasian Computer Science Conference - Volume 48 (Darlinghurst, Australia, Australia), ACSC '06, Australian Computer Society, Inc., 2006, pp. 277–286.
4. F. Boian, R. Boian, and A. Vancea, *Ams: An assignment management system for professors and students*, Proceedings of the Symposium "Colocviul Academic Clujean de Informatic" (Cluj, Romania), 2006, pp. 137–142.
5. Claudiu Cobarzan, *Node ranking in a dynamic distributed video proxy-caching system*, KEPT2009 Knowledge Engineering Principles and Techniques, Selected Papers, Presa Universitara Clujeana, 2009, pp. 328–334.
6. Sanda Dragos, *PULSE Extended*, The Fourth International Conference on Internet and Web Applications and Services (Venice/Mestre, Italy), IEEE Computer Society, May 2009, pp. 510–515.
7. Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken, *Winnowing: local algorithms for document fingerprinting*, Proceedings of the 2003 ACM SIGMOD international conference on Management of data (New York, NY, USA), SIGMOD '03, ACM, 2003, pp. 76–85.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania

E-mail address: {rares,sanda,claudiu}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

A STUDY OF TCP-FRIENDLINESS ON THE SHORT TERM WITH AN APPLICATION TO MEDIA-FRIENDLY CONGESTION CONTROL

ADRIAN STERCA AND ALEXANDRU VANCEA

Abstract. TCP-friendliness is a desired quality of any congestion control algorithm used in the Internet because it expresses fairness towards TCP flows. However, all studies refer to the TCP-friendliness as a long term characteristic of a flow, i.e. they consider only the long-term fairness to TCP. In this paper we introduce the notion of *TCP-friendliness on the short term* and apply it to a multiplicative-type media-friendly congestion control algorithm.

1. Introduction

TCP's AIMD (Additive Increase Multiplicative Decrease) congestion control is not well suited for multimedia streams due to its highly fluctuating throughput. Consequently, other congestion control algorithms which offer a smoother throughput were developed, the so-called TCP-friendly congestion controls algorithms [1]. All these smooth congestion control algorithms have a more stable throughput than TCP's AIMD because they are less aggressive than TCP in using new available bandwidth, but they are also slower responsive to congestion than TCP. Because they offer a more stable throughput, multimedia streams, especially CBR (*Constant Bit Rate*) ones, but also VBR (*Variable Bit Rate*) ones, can be better adapted to predictable bandwidths by the streaming servers. However, although smooth congestion controls improve the delivery of multimedia streams, they are not the optimal solution, because they don't take into consideration media characteristics of the stream (i.e. they are not media-friendly). This led to the development of media-friendly

Received by the editors: May 10, 2011.

2010 *Mathematics Subject Classification.* 68M20, 68U99.

1998 *CR Categories and Descriptors.* C.2.2 [**Computer-Communication Networks**]: Network Protocols – *Protocol architecture (OSI model)*; C.2.3 [**Computer-Communication Networks**]: Network Operations – *Network monitoring*.

Key words and phrases. TCP-friendly congestion control, congestion control, media-friendly congestion control, multimedia streaming.

congestion control [7, 8]. It is important to note that a congestion control algorithm for multimedia streaming must have both characteristics: it must be TCP-friendly (i.e. fair with the network) and it also must be media-friendly to maximize the application benefit.

The contributions of this paper are the following:

- we try to give a classification of media-friendly and TCP-friendly congestion control algorithms
- and we refine the concept of "TCP-friendliness" and distinguish between two types of TCP-fairness: long-term TCP-fairness and short-term TCP fairness.

The rest of the paper is organized as follows. In Section 2 we review TCP-friendly congestion control, focusing on the TCP-Friendly Rate Control (TFRC). Then in Section 3 we define TCP-friendliness on the short term and its utility to media-friendly congestion control algorithms. The paper continues with Section 4 which presents simulations for assessing TCP-friendliness on the short term of UTFRC (Utility-driven TCP-Friendly Rate Control), a multiplicative-type of media-friendly and TCP-friendly congestion control. Finally, the conclusions from Section 5 end the paper.

2. TCP-friendly congestion control

A TCP-friendly congestion control algorithm is a congestion control algorithm which exhibits the same throughput as employed by the TCP's AIMD congestion control algorithm in the same network conditions [1]. This makes TCP-friendly flows fair to TCP flows when consuming network bandwidth. And because TCP-friendly congestion control algorithms typically have a smoother throughput than TCP [2], they are favored over TCP for multimedia streaming applications. There are several proposals for TCP-friendly congestion control [3, 4, 5], but probably the most well known is TCP-Friendly Rate Control (TFRC) [2].

The TCP-Friendly Rate Control [2] is a rate-based congestion control that has two main components: the throughput function and the WALI (i.e., Weighted Average Loss Intervals) mechanism for computing the loss rate. The throughput function is the throughput equation of a TCP-Reno source [6]:

$$X_{TFRC}(p) = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}, \tag{1}$$

where X is the sending rate in bytes/sec, s is the packet size, R is the round-trip time (RTT), p is the steady-state loss event rate and $t_{RTO} = 4 * R$ is the TCP retransmit timeout value. This throughput function is responsible for the TCP-friendliness of TFRC.

The way parameter p from the equation is computed is what gives TFRC its smoothness of throughput. p is computed using WALI as an average loss rate over a time interval including the most recent 8 loss events. The number of packets sent by the sender between two loss events is called a loss interval and the loss rate p is computed using the WALI mechanism as a weighted average of the loss rates in the last 8 loss intervals, where more recent loss intervals get a higher weight [2]:

$$s = \frac{1 * s_0 + 1 * s_1 + 1 * s_2 + 1 * s_3 + 0.8 * s_4 + 0.6 * s_5 + 0.4 * s_6 + 0.2 * s_7}{1 + 1 + 1 + 1 + 0.8 + 0.6 + 0.4 + 0.2}$$

$$p = \frac{1}{s}$$

In the above equation s_i is the length (in packets) of the i-th most recent loss interval, $i \in 0..7$ and the weights are 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2 starting from the most recent loss interval to the oldest.

Because the loss rate, p, used in the TFRC throughput formula is averaged over time intervals much greater that one round-trip time using the WALI mechanism, the throughput achieved by TFRC is much smoother than the throughput of TCP. Several studies give evidence of this fact [9, 10, 11]. A direct consequence of this is that TFRC reacts slowly (more slowly than TCP) to an increase of the level of congestion in the network, but it reacts also slowly when additional bandwidth becomes available in the network.

Having a throughput smoother than TCP's makes TFRC valuable for multimedia streaming applications because of the increased predictability of its throughput. However, several studies documented some of its limitations [7, 8, 12, 13], its lack of media-friendliness to be more specific.

3. Media-friendly congestion control and TCP-friendliness on the short term

Media-friendly congestion control is a type of congestion control which incorporates also media characteristics like bitrate, buffer fill level, quality measurements etc. in the throughput computing formula besides just network parameters (like round-trip time and loss rate) [7, 8].

To the best of our knowledge, media-friendly and TCP-friendly congestion control algorithms are all based on TFRC and fall into two categories: *multiplicative* and *additive*. Both types consider a media-friendly function $\alpha(q(t))$ which embodies the usefulness of increasing TFRC's throughput passed the rate computed with equation (1) to the streaming application and $q(t)$ is a n-dimensional function giving the values of various media characteristics across time. For simplicity, from now on we will discard the media characteristics $q(t)$ from our notation and write simply $\alpha(t)$.

Multiplicative media-friendly congestion control algorithms use a formula like the following to compute the transmission rate [8]:

$$X(t) = \alpha(t) * X_{TFRC}(t) \tag{2}$$

where $X_{TFRC}(t)$ is given by equation (1). In this type of congestion controls the transmission rate computed by TFRC is altered in a multiplicative way.

Additive media-friendly congestion control algorithms use a formula like the following to compute the transmission rate [7]:

$$X(t) = X_{TFRC}(t) + \alpha(t) \tag{3}$$

where $X_{TFRC}(t)$ is given by equation (1). In this type of congestion controls the transmission rate computed by TFRC is altered in an additive way by the media-friendly function $\alpha(t)$. The media-friendly function, $\alpha(t)$ may include network characteristics (like the loss rate) as parameters, but it does not include the throughput computed by TFRC, $X_{TFRC}(t)$.

It is important that media-friendly congestion control algorithms are also TCP-friendly. However, all the media-friendly congestion control algorithms we are aware of [7, 8] and all TCP-friendly algorithms view the *TCP-friendliness* characteristic as a long term characteristic. We argue that it is important to distinguish between two types of TCP-friendliness: *long-term TCP-friendliness* describing the bandwidth usage and relation to other flows during the duration of the whole streaming session and *short-term TCP-friendliness* describing the local impact on other flows in a small period of time. The short-term TCP-friendliness can be important for flows with a short life time (e.g. web connections). For a short lifespan flow, it is not fair if a flow that is TCP-friendly on the long term consumes twice as much bandwidth as the short lived flow, during its short existence. Of course, by its very nature, a media-friendly flow occasionally consumes on short timescales more bandwidth than a TCP-friendly flow. This slight unfairness is inevitable in any media-friendly congestion control algorithm. It is even present in TFRC and also in TCP. On short time scales, 2 TCP flows get different throughputs even if they share the same network conditions (i.e. round-trip time, loss rate). The idea is to limit this "short timescale unfairness" so that other flows (especially those with a short lifespan) are not affected too much.

At this point we can define the two concepts of TCP-friendliness.

Definition 1. The definition of TCP-friendliness on the long term is the definition of the original TCP-friendliness concept from [1] : A flow is termed *TCP-friendly on the long term* if its long-run average transmission rate, $X(t)$, does not exceed the transmission rate of a TCP flow in the same network conditions.

Definition 2. A flow is termed *TCP-friendly on the short term* if during any 8 loss events time interval (i.e. the time interval spanning over 8 consecutive loss events) the flow does not exceed twice the transmission rate of a TCP flow during the same time interval and in the same network conditions.

4. Simulation study of UTFRC's TCP-friendliness on the short term

Simulation results will be presented in the extended version of this paper.

5. Conclusions

In this paper we presented a classification of media-friendly congestion control algorithms and we underlined the need for the concept of TCP-friendliness on the short term. Consequently we introduced the concept of *TCP-friendliness on the short term* and explained its usefulness for multimedia flows. Simulations characterizing the TCP-friendliness on the short term of a media-friendly congestion control algorithm, UTFRC[8], will be presented in the extended version of this paper.

References

[1] S. Floyd, K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*, IEEE/ACM Transactions on Networking, 7(4), pp. 458-472, Aug. 1999.

[2] S. Floyd, M. Handley, J. Padhye, J. Widmer, *Equation-Based Congestion Control for Unicast Applications*, ACM SIGCOMM 2000.

[3] R. Rejaie, M. Handley, and D. Estrin, *An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet*, In Proceedings of INFOCOMM 99, 1999.

[4] D. Sisalem and H. Schulzrinne, *The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaption Scheme*, In Proceedings of NOSSDAV98, 1998.

[5] I. Rhee, V. Ozdemir, and Y. Yi, *TEAR: TCP Emulation at Receivers Flow Control for Multimedia Streaming*, April 2000. NCSU Technical Report.

[6] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, *Modeling TCP Throughput: A Simple Model and its Empirical Validation*, SIGCOMM Symposium on Communications Architectures and Protocols, Aug. 1998.

[7] J. Yan, K. Katrinis, M. May, B. Plattner, *Media- and TCP-Friendly Congestion Control for Scalable Video Streams*, IEEE Transactions on Multimedia, Vol. 8, No. 2, April, 2006.

[8] A. Sterca, *UTFRC - Utility-driven TCP-Friendly Rate Control for Multimedia Streams*, Proc. of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society, Germany, February 2009.

[9] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, *Dynamic behavior of slowly-responsive congestion control algorithms*, In Proc. of ACM SIGCOMM01, San Diego, California, USA, August 2001.

[10] Y. Richard Yang, M. Sik Kim, and Simon S. Lam, *Transient Behaviors of TCP-friendly Congestion Control Protocols*, In Proc. of IEEE Infocom2001, March 2001.

[11] S. Floyd, M. Handley and J. Padhye, *A Comparison of Equation-Based and AIMD Congestion Control*, ACIRI, February 2000, http://www.aciri.org/tfrc/.
[12] I. Rhee, L. Xu, *Limitations of Equation-based Congestion Control*, In Proc. of ACM SIGCOMM'05, Philadelphia, Pennsylvania, USA, August 2005.
[13] Z. Wang, S. Banerjee and S. Jamin, *Media-Friendliness of A Slowly-Responsive Congestion Control Protocol*, In Proceedings of NOSSDAV'04, 2004.

Babes-Bolyai University, Faculty of Mathematics and Computer Science, 1, M. Kogalniceanu St., 400084-Cluj-Napoca, Romania
E-mail address: forest@cs.ubbcluj.ro

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

DYNAMICS WITHIN A LAN DISTRIBUTED VIDEO PROXY-CACHE

CLAUDIU COBÂRZAN AND DIANA MAN

Abstract. We introduce a fuzzy clustering approach that uses a well known algorithm to partition the clients in a LAN. This is done in the context of a distributed video proxy-cache that is able to vary the number of active nodes based on LAN conditions and client behavior.

1. Introduction

Video proxy-caching within LANs is an often deployed solution that tries to tackle some of the problems due to increasing demand for multimedia and especially audio-video content from the Internet.

We defined a distributed system that aims at providing better support for clients within a LAN by varying the number of active nodes depending on a number of conditions.

In the following we will make an overview of the system dynamics and introduce a new approach to determining the node that should host a new proxy-cache in case another one is needed. The new approach uses fuzzy clustering to cluster the existing nodes in the LAN with the cluster centers hosting the running proxy-caches. The number of clusters equals the number of running proxy-caches minus one, since we consider that the systems starts with one active proxy-cache.

2. Overview of the Proxy-Caching System

In [3] we introduced a video-proxy-caching system that starts with a single active node, but can add and then remove nodes from the system as necessary. Adding a new node to the proxy-caching system is done when new computing and/or storage resources are needed to service client requests. Removing nodes from the system takes place when the request volume drops and fewer nodes can deal with the existing clients.

Received by the editors: May 15, 2011.

2000 *Mathematics Subject Classification.* 68-06, 68M14.

1998 *CR Categories and Descriptors.* H.3.4 [**Information Storage and Retrieval**]: Systems and Software – Distributed systems.

Key words and phrases. video proxy-cache, fuzzy, fuzzy clustering.

2.1. **System Dynamics.**

The system is implemented by using two entity types: `dispatchers` and `daemons`. The `dispatchers` run on every proxy-cache node and have coordinator responsibilities: they contact their siblings or origin servers in order to solve requests they can't service from the local cache. Also, they select the node that will host a new proxy-cache, when such an operation is required. The `daemons` run within a LAN on the nodes that can become proxy-caches (potentially on every node). They also act as on-site proxies by selecting one `dispatcher` from the list of active ones to which a client request is forwarded.

The conditions that trigger a *split* operation (that adds a new node), as well as a *hibernate* or *shut down* operation (that remove a node) are detailed in [3] and in [4]. When performing a *split* operation, a `dispatcher` selects the "best" `daemon` to host the proxy-Cathe code and possibly some of the data from the local cache. Leaving the system is done by a *shut down* operation that stops the proxy-cache and its `dispatcher` and discards any locally stored data. A proxy-cache can enter a dormant state in which it only services requests for objects it already stores by performing a *hibernate* operation. If a *split* operation has to be performed and there are hibernating nodes, one of those nodes will be reactivated.

Deciding which nodes will be shut down, put in hibernation or reactivated is done by their *rank* value according to the conditions in [3] and [4]. The ranking mechanism was introduced in [4] and refined in [2] at node level. The idea behind it is to rank the active or hibernating proxy-caches by the volume of served data. In [2] we propose differentiating between data sources - local cache, siblings or origin servers - so that the largest rank is assigned to the node(s) serving the largest amount of data from within the local cache when reported to the amount of data served from sibling caches and remote servers.

2.2. **Cache Operations.**

All cache specific operations (add a new object, discard, move or replicate an object) are done according to the *utility* of the objects. The *utility* value is computed when an object first enters the cache, but varies over time according to several characteristics: size, number of requests, time of the last request etc. The considered characteristics have different weights. Those weights are either static (fixed for the life time duration of the proxy-cache) or are dynamically generated (the values may change during the life time of the proxy-cache). In [9] and [5] we proposed using genetic algorithm when determining the weighting values so that the system's efficiency (measured with *byte-hit-rate* as metric) is maximized. The byte-hit-rate is computed as ratio between the data volume served from within the cache and the data amount served from the cache, siblings and servers (total amount of served data).

A video object $o_{ij} \in LC_i$ held in the cache $P_i \in P$ ($1 \leq j \leq q$, q - the number of cached objects at node i) is defined in [4] as:

$$o_{ij} = (size(o_{ij}), duration(o_{ij}), bitRate(o_{ij}), qualityValue(o_{ij}), TLA, HC, COST) \tag{1}$$

(LC_i stands for the contents of the local cache while P represents the set of active proxies).

The proxy-cache holds information on the size, duration, encoding bit rate and quality value of the video, where the quality value (a real number between 0 and 1) is the measure of the object's quality (based on characteristics of the video object like resolution, color information etc.) to different clients.

Also, three vectors are used to hold additional data:

- the moments in time when the object was last requested - the *TLA (Time of Last Access)* vector
  (e.g. $timeLastAccess(o_{1,3})_{N_2}$ represents the moment the 3^{rd} object cached at node P_1 has been last requested from the node N_2);
- the number of requests for the object - the *HC (Hit Count)* vector
  (e.g. $hitCount(o_{1,3})_{N_2}$ represents the number of times the 3^{rd} object cached at node P_1 has been requested from node N_2);
- the cost of streaming the object from the cache to the requesting client - the $COST$ vector
  (e.g. $cost(o_{1,3})_{N_2}$ represents the cost of streaming the 3^{rd} object in the cache at node P_1 to the node N_2. It is computed as follows:

$$cost(o_{1,3})_{N_2} = \frac{\alpha_{P_1,N_2}}{bitRate(o_{1,3})} duration(o_{1,3}) \tag{2}$$

  meaning "the cost of streaming the 3^{rd} object from the cache at node P_1 to the node N_2 equals the amount of bandwidth needed to stream that particular object" - α_{P_1,N_2} denotes the amount of bandwidth available between node P_1 and node N_2).

3. A New Approach to System Expansion

We propose a new approach for determining the node that will host a new proxy-cache. This approach is based on fuzzy clustering.

Clustering is the division of a data set into subsets (clusters) such that, similar objects belong to the same cluster and dissimilar objects to different clusters. In real applications, there is no obvious boundary between clusters so that, fuzzy clustering is often better. In fuzzy clustering the object can belong to more than one cluster so, it has a degree of belonging to clusters, as in fuzzy logic.

A hard clustering algorithm allocates each object to a single cluster. Thus, traditional clustering divides data objects in partitions. *Fuzzy clustering* extends this notion to associate each data object with every cluster using a membership function. The output of the hard clustering algorithm is a partition, whereas the one of fuzzy algorithm is a clustering. In fuzzy clustering each cluster is a fuzzy set of all the patterns. In general the performance of fuzzy clustering algorithms is superior to that of the corresponding hard algorithms. Fuzzy clustering has proved successful in many relevant applications from real world.

The most popular fuzzy clustering algorithm is *FCM - Fuzzy c-Means* [6, 1]. It is a data clustering technique in which each data point belongs to a cluster to some degree that is specified by a membership grade. The algorithm is also known as *Fuzzy ISODATA* or *Fuzzy K-Means*.

Given a set $X = \{x_1, x_2, ..., x_n\} \subset \Re^p$ of sample data, the aim of the algorithm is to determine the prototypes (cluster centers) in such a way that the objective function is minimized.

In our case the set X is composed by the set of nodes (N).

The objective function $J : NX[1, \infty) \to R$ is:

$$J(c, q) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c}(\sum_{k=1}^{n} u_{ik}^q d_{ik}^2)), q \in [1, \infty) \tag{3}$$

subject to:

$$\sum_{k=1}^{n} u_{ik} > 0, \forall i \in \{1, 2...c\}, \sum_{i=1}^{c} u_{ik} = 1, \forall k \tag{4}$$

where u_{ik} stands for the membership degree of datum x_k to cluster i, d_{ik} is the distance of datum x_k to cluster i, represented by the prototype p_i and c is the number of clusters. The parameter q is a weighting exponent (fuzziness exponent). Usually $q = 2$ is chosen. At $q = 1$, *FCM* collapses to *HCM algorithm* [7].

The first constraint guarantees that no cluster is empty and the second condition ensures that the sum of the membership degrees for each datum equals 1.

Also, this constraint corresponds to a normalization of the membership per datum. As a consequence of both conditions no cluster can contain the full membership of all data points. Thus, the membership degrees for a given datum formally resemble the probabilities of its being a member of the corresponding cluster.

The output of the *FCM* algorithm is not a partition, thus: $C_i \bigcap C_j \neq \emptyset, i \neq j$.

There are two necessary conditions for J to reach a minimum:

$$p_i = \frac{\sum_{k=1}^{n} u_{ik}^q x_k}{\sum_{k=1}^{n} u_{ik}^q} \tag{5}$$

$$u_{ik} = \frac{(\frac{1}{d_{ik}})^{1/(q-1)}}{\sum_{j=1}^{c} (\frac{1}{d_{jk}})^{1/(q-1)}} \tag{6}$$

where d_{ik} is the *distance* between object x_k and the center of cluster C_i [8].
`The FCM Algorithm:`

```
1. Initialize the membership matrix U with random values between
0 and 1 within the constraints of (2).
2. Calculate c cluster centers $p_i, i=1..c$ using (3).
3. Compute the objective function according to (1). Stop if
either it is below a certain threshold level or its improvement
over the previous iteration is below a certain tolerance.
4. Compute a new U using (4).
5. Go to step 2.
```

The *FCM* algorithm assumes that the number of clusters c is known. In real cases, this parameter is not known and must be determined. The traditional approach to determining c is to evaluate a certain global validity measure of the c-partition for a range of c values and then pick the value of c that optimizes the validity measure. An alternative is to perform progressive clustering where clustering is initially performed with an over specified number of clusters. After convergence, bad clusters are eliminated, compatible clusters are merged and good clusters are identified [10].

The system we defined starts with one proxy-cache that will be permanently active but it can add proxy-caching nodes whenever necessary. That is why we set the number of clusters to 1 when the first additional proxy-cache is added. In this situation we have to determine the center of the cluster formed by all the nodes in the LAN (the set N) which will be the node hosting the second proxy-cache. However there is a constraint: in order to host the new proxy-cache, the center has to have a running `daemon`. Whenever there are more than one active proxy-cache, the number of clusters equals the number of active proxy-caches minus one, and the nodes hosting them are the centers of those clusters (if the designated centers have running `daemons`). Whenever a *split* operation is performed, the nodes in the LAN are re-clustered and the number of clusters increases by one.

We will consider the *distance* between the node x_k and the center of cluster C_i the report between the total cost of streaming the stored all objects and the total number of requests for the objects stored on the node. This report must be minimized.

4. Conclusions and Future Work

We introduced a new method of determining the nodes that will host a new proxy-cache in a dynamic distributed video proxy-caching system. This methods clusters the existing clients based on the cost of delivering objects to those clients as well as client activity (number of requests). As future work to intend to refine the conditions for determining those nodes by also considering the time when the client requests were made and also to develop algorithms for moving data between the active nodes (data move, data replicate) that consider client clusters.

References

1. James C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*, Kluwer Academic Publishers, Norwell MA, USA, 1981.
2. Caudiu Cobârzan, *Node Ranking in a Dinamic Distributed Video Proxy-Caching System*, KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEEPT2007, 6-8 June 2007, Cluj-Napoca, Romania (Str. Haşdeu nr. 45, 400371, Cluj-Napoca, Romania), Presa Universitară Clujeană/Cluj University Press, 2007, pp. 298–306.
3. Claudiu Cobârzan, *Dynamic Proxy-Cache Multiplication inside LANs*, Euro-Par 2005, Lecture Notes in Computer Science, vol. 3648, Springer, 2005, pp. 890–900.
4. Claudiu Cobârzan and László Böszörményi, *Further Developments of a Dynamic Distributed Video Proxy-Cache System*, Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007), IEEE Computer Society, 2007, pp. 349–357.
5. Claudiu Cobârzan, Alin Mihăilă, and Cristina Mihăilă, *Dynamics of a Utility based Distributed Video Proxy-Cache*, 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC2008), September 26-29, 2008, Timisoara, Romania, 2008, (accepted for publication in IEEE post-proceedings).
6. J.C. Dunn, *A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters*, Journal of Cybernetics **3** (1973), no. 3, 32–57.
7. A.K. Jain, M.N. Murty, and P.J. Flynn, *Data clustering: A review*, ACM Computing Surveys **31** (1999), no. 3, 264–323.
8. Jan Jantzen, *Neurofuzzy modelling*, Technical Report 98H874 (nfmod), Technical University of Denmark, 1998.
9. Cristina Mihăilă and Claudiu Cobârzan, *Evolutionary approach for multimedia caching*, 19th International Workshop on Database and Expert Systems Applications (DEXA 2008), 1-5 September 2008, Turin, Italy, IEEE Computer Society, 2008, pp. 531–536.
10. Horia F. Pop, *Data analysis with fuzzy sets: a short survey*, Studia Universitatis Babes-Bolyai, Series Informatica **XLIX** (2004), no. 2, 111–122.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania
E-mail address: {claudiu,mandiana}@cs.ubbcluj.ro