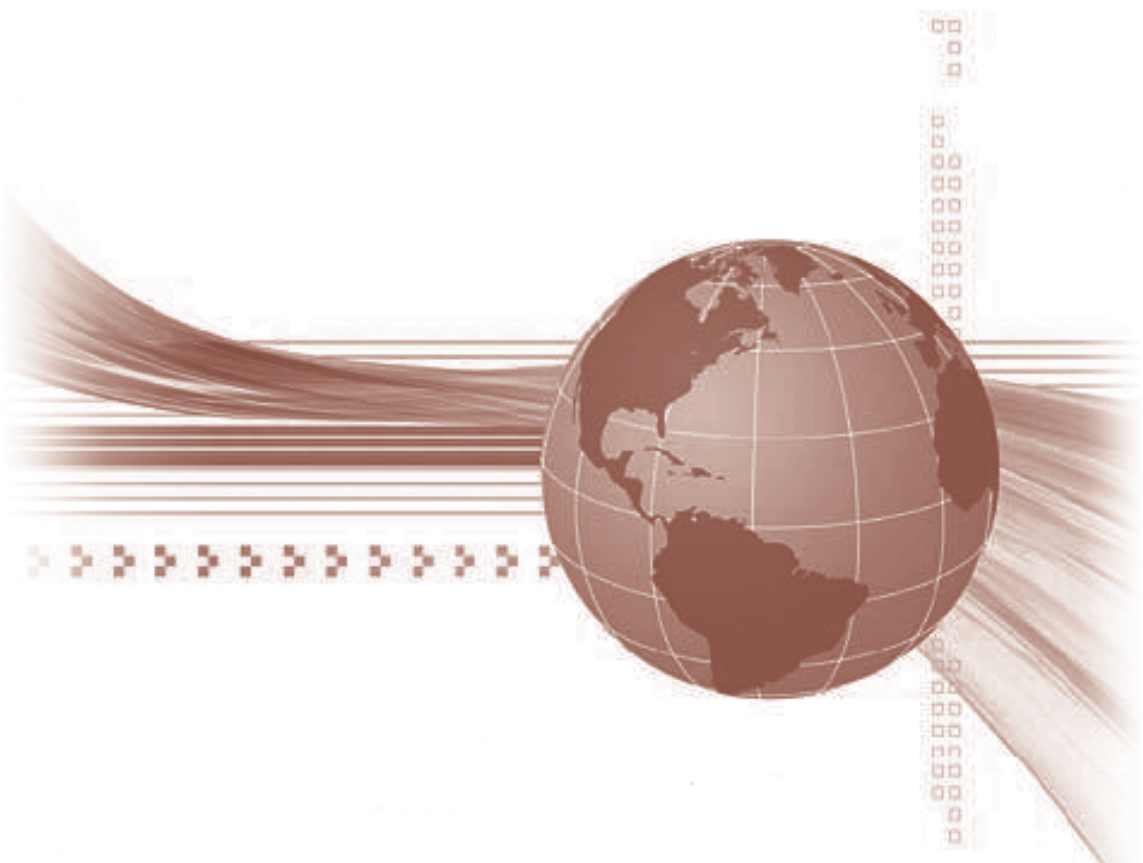
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

3/2013

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

No. 3/2013

July - September

This volume contains papers presented at the International Conference

KEPT2013

KNOWLEDGE ENGINEERING PRINCIPLES AND TECHNIQUES

The conference has been kindly sponsorted by

EDITORIAL BOARD

YEAR **Volume 58 (LVIII) 2013**
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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

AN AGENT BASED APPROACH FOR PARALLEL CONSTRAINT VERIFICATION

RADU D. GĂCEANU[(1)], HORIA F. POP[*,(1)], AND SERGIU A. SOTOC[(1)]

Abstract. The medical procedure result in case of cardiac arrest is highly dependant on several factors including the quality of chest compressions, early defibrillation and advanced life support. Timing and precise procedure compliance are critical in such situations. This paper presents an agent-based approach for detecting erroneous procedure follow-ups. The Erlang programming language is chosen for implementation because of its reputation in outstanding concurrency support thus being a perfect environment for agent-based solutions. We provide experiments on a synthetic dataset as well as on a dataset provided by the Romanian Resuscitation Council.

1. Introduction

In case of cardiac arrest, the result of the medical personnel performance is greatly influenced by the time and accuracy of applying predefined procedures. Even though investments in the infrastructure are continuously being done, the survival rate in case of cardiac arrest remains low [6].

In order to improve this situation the Romanian Resuscitation Council (CNRR) [1] promotes high quality education in resuscitation (basic and advanced life support) acting in Romania at national level since 1998.

Our aim is to help the Romanian Resuscitation Council to detect situations where the standard procedures have been infringed so that they can direct their efforts in such weaker areas. We propose an agent-based approach for detecting breaches in the standard protocols. Since the search space may be large (nation-wide registries), providing scalable solutions is desired. The

Received by the editors: April 15, 2013.

2010 *Mathematics Subject Classification.* 68T42.

1998 *CR Categories and Descriptors.* I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems*.

Key words and phrases. Intelligent Agents, Parallel Constraint Verification, Erlang.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

* Corresponding author.

Erlang programming language advocates the benefits of concurrent programming so it seems a good choice for our agent-based approach.

Approaches that are similar to ours are presented in [7, 8, 16]. Authors from [16] present a review of distributed constraint satisfaction problems and some extensions to the considered models. They propose the use of multi-agent systems in distributed constraint satisfaction problems. In their view an agent is responsible for instantiating several variables and in order to reach a solution agents have to negotiate over the enforced rules. Authors from [7] employ a two dimensional grid for the agents environment in order to solve CSP problems. A solution to the problem is given by the positions of the agents on the grid provided that no constraint is unsatisfied. In [8] propose a methodology called constrained partition and coordinated reaction for distributed constraint satisfaction is presented. The introduced approach partitions the set of constraints and associates an agent to each subset of constraints. The process starts from an initial instantiation of variables and, according to the authors, the solution emerges through incremental local revisions of the variable instantiations. The methodology was applied on the job shop scheduling problem and experiments show better results compared to other methods.

Our approach considers an agent for each variable and instead of finding an assignment for all variables such that all constraints hold (like in a CSP), our aim is to find all variable assignments for which some constraint does not hold. The proposed problem has practical reasons and is clearly motivated in Section 2.

The paper is organized as follows: Section 2 states the motivation of our approach, Section 3 presents the theoretical background. The proposed approach is presented in Section 4 and is followed by Section 5 where we show our experimental evaluation. In Section 6 we present a comparison with related work. Finally, conclusions and ideas for future work are presented in Section 7.

2. Motivation

In cardiac arrest cases, the outcome of the intervention on the patient is greatly influenced by the time and precise compliance to well defined procedures. Technological growth has facilitated the decrease of rescue teams to patient arrival times. Nevertheless survival rates remain unsatisfactory [6]. The focus is now on improving the performance of rescue team procedures as this is thought to be the main cause of low survival rates [9].

In this regard, the Romanian Resuscitation Council (CNRR) [1] aims to promote high quality education in resuscitation (basic and advanced life support). Since 2004, CNRR is the unique partner of the European Resuscitation

Council in Romania. Tasks performed by CNRR since 1998 until today include: promoting the quality of education in resuscitation; translation of the resuscitation guidelines in 2000, 2005 and 2010; formations in resuscitation and trauma at national level; implication in the REMSSy program, a project for the development of the Romanian Emergency Medical Services; editing of text books 2006, and releasing the printed version of the Romanian resuscitation guide lines 2010; active involvement in increasing the quality of the medical care in fields as resuscitation and post-resuscitation care; management of a nation wide network of instructors for advanced and basic life support; implication into the European Registry of Cardiac Arrest (EuReCa) since 2010; main promoter of the Romanian Registry of Cardiac Arrest; Organization of international conferences in the field of resuscitation 2005 and trauma in 2007.

Detecting failures in the standard procedures is highly important in order to improve rescue team performance and possibly save lives. We propose an agent-based approach for detecting breaches in the standard protocols. Ideally, nation-wide registries should be available for analysis so any solution in this direction should be scalable.

Our programming environment is the Erlang programming language. We have chosen Erlang because it is promoted as a language with built-in support for concurrency and distributed programming. Moreover, using a declarative language (like Erlang) seems to be a more natural choice for implementing agents. For example it is much easier to implement a rule-based system in a declarative language than in an imperative one (a rule base could represent the intelligence part of an agent).

Rule based systems are becoming key components in nowadays fields like Semantic Web [10]. In order to make better use of artificial intelligent research experience and the practical application of Semantic Web technologies, the World Wide Web Consortium (W3C) has adopted the Web ontology language (OWL) as a language for processing Web information [4]. Authors from [5] note that rules represent because the standard ontology language OWL provides only basic forms of reasoning. However, ensuring the correctness of a rule-based approach may not always be a trivial task since the actual system behaviour and the resources required to realize such rules may be difficult to predict. These issues are even more disturbing in distributed rule-based systems where several communicating rule-based programs exchange information via messages. Also, a communicated fact may be added asynchronously to the state of the rule based system at run-time, potentially triggering a new strand of computation which executes in parallel with current processing [10]. Since agents in a multi-agent system can operate asynchronously and in parallel their use in a distributed rule-based environment may result in increased overall speed. A multi-agent based approach in general is more desirable compared

to a centralised approach because it makes the system more robust, reliable, scalable, flexible, cost effective, easier to develop and reuse [14].

3. Theoretical background

An agent is an entity that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [13]. An agent that always tries to optimize an appropriate performance measure is called a rational agent. Such a definition of a rational agent is fairly general and can include human agents (having eyes as sensors, hands as actuators), robotic agents (having cameras as sensors, wheels as actuators), or software agents (having a graphical user interface as sensor and as actuator). Software agents are generally thought to act independently of the user intervention. According to [13, 12] agents exhibit the following characteristics:

- autonomy
- reactivity
- pro-activity
- sociability
- intelligence
- mobility
- self-organization.

The most attractive property is self-organization, that is, the ability to improve its behaviour without external influence or guidance.

An agent always has an associated environment, being able to act autonomously in order to accomplish its objectives. It only plays a role and operates in the environment through its sensors and effectors.

An abstract view of an agent is best expressed by Figure 1. However, such a view of an agent is fairly general and hence this abstract architecture should be refined. As described in [15] the agent's decision function may be separated into perception and action subsystems. In this sense, the agent may be seen as a pair $Ag =< see, action >$, where the function *see* maps environment states to percepts and the function *action* maps sequences of percepts to actions:

$$see : E \to Per \tag{1}$$

$$action : Per^* \to Ac \tag{2}$$

An intelligent agent has all the abilities of a software agent plus the skill of communicating with other agents, being reactive at the changes in the environment, having at least a partial representation of it. An intelligent agent is driven by completing its goals, it has its own resources, is able to completely understand the environment and is also able to learn.

Usually agents coexist and interact forming Multi-agent Systems (MAS).

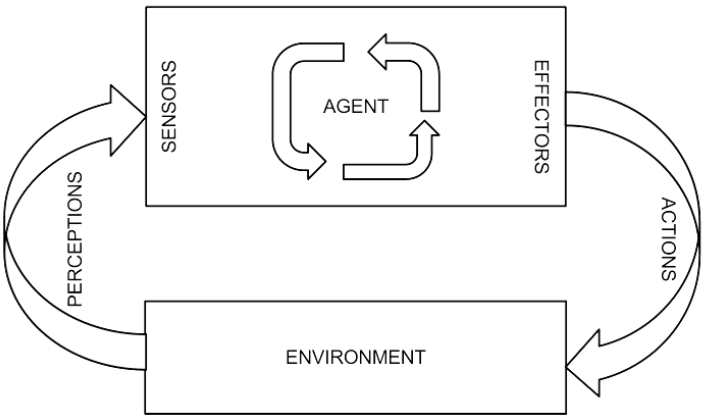


FIGURE 1. An agent perceiving its environment. The agent takes sensory input from the environment, and outputs actions that modify the environment.

Definition 3.1. *In computer science, a multi agent system (MAS) is a system composed of several interacting agents, collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system.*

The precise nature of the agents is not clearly established. MAS may also include human agents. Examples of multi agent systems include human organizations and society in general.

Remark 3.1. *In a multi agent system, agents can be software agents, robots and also humans.*

Remark 3.2. *As a consequence to Remark 3.1 it follows that there is no single agent system.*

A multi agent system has the following advantages:

- it is inherently a distributed architecture and thus critical failures and performance bottlenecks are avoided
- allows interconnection of legacy systems
- problems like task allocation, team planning, complex phenomena simulation are naturally modelled in terms of interacting agents
- efficiently operates with information from spatially distributed sources
- suitable in situations where knowledge is distributed temporally and spatially

- enhances system performance at least in the following aspects of computational efficiency, robustness, reliability, and extensibility.

Some of the important things that need to be taken into consideration when dealing with a MAS are: communication between agents, collaboration and coordination [12]. The interactions between agents in a MAS can be either cooperative or selfish [12, 15, 11]. The information exchange is done using Agent Communication Languages like KQML [2] and FIPA ACL [3].

4. Proposed model

We propose an agent-based model for the parallel constraint verification problem.

Definition 4.1. *Given:*

- *A set of variables* $\mathcal{V} = \{V_i | i = \overline{1, |\mathcal{V}|}\}$
- *A set of domains* $\mathcal{D} = \{D_i | i = \overline{1, |\mathcal{D}|}\}$*, where each* D_i *is finite and discrete,* $|\mathcal{V}| = |\mathcal{D}|$ *and* $V_k \in D_k, \forall k = \overline{1, |\mathcal{V}|}$
- *A set of rules* $\mathcal{R} = \{R(P_i) | i = \overline{1, |\mathcal{R}|}\}$*, where each* P_i *represents a subset of variables from* $\mathcal{V}$ *, and each rule* $R(P_i)$ *represents a relation over the given subset of variables* P_i*.*

The constraint verification problem consists in finding all ordered bags

$$S =< v_1, v_2, \ldots, v_{|\mathcal{V}|} > \tag{3}$$

s.t. $\exists S^{'} \subseteq S, \exists i \in \{1, \ldots, |\mathcal{R}|\}$ *where* $R(P_i)$ *is not satisfied.*

We consider the environment E of an agent as a finite set of states:

$$E = \{e_i | i = \overline{1, |\mathcal{V}|}\}, \tag{4}$$

where $e_i \subseteq < v_1, \ldots, v_{|\mathcal{V}|} >$.

The set of all possible actions an agent may choose from is:

$$Ac = \{\alpha(R_i) | i = \overline{1, |Ac|}\}, \tag{5}$$

where $\alpha(R_i)$ denotes the action taken by the agent when rule $R_i \in \mathcal{R}$ is triggered.

Definition 4.2. *A behaviour, b, of an agent in an environment is a sequence of state transitions:*

$$b : e_0 \xrightarrow{\alpha(R_0)} e_1 \xrightarrow{\alpha(R_1)} \ldots \xrightarrow{\alpha(R_{u-1})} e_u \tag{6}$$

Definition 4.3. *An agent in the environment is:*

$$Ag : \mathcal{B}^E \rightarrow Ac \tag{7}$$

where $\mathcal{B}^E$ is the subset of all possible behaviours b that end with an environment state.

A sketch of our approach to the constraint verification process is given in Algorithms 4.1 and 4.2. The main algorithm spawns a new agent for each considered record of data. By a record we mean an ordered set of values for all variables from $\mathcal{V}$. In its behaviour, each agent is checking the rules consistency against the provided data record.

Algorithm 4.1 Spawn Agents

1: Input: $\mathcal{V}$, $\mathcal{D}$, $\mathcal{R}$
2: **for all** records $L =< v_1, v_2, \ldots, v_{|\mathcal{V}|} >$ **do**
3: $\quad SpawnAgent(AgentBehaviour, L, \mathcal{R})$
4: **end for**

Algorithm 4.2 Agent Behaviour

1: Input: $L =< v_1, v_2, \ldots, v_{|\mathcal{V}|} >, \mathcal{R}$
2: **for all** rules $R \in \mathcal{R}$ **do**
3: $\quad P \leftarrow GetValues(R, L)$
4: $\quad SW \leftarrow CheckConsistency(R, P)$
5: $\quad ProcessResult(P, SW)$
6: **end for**

Algorithm 4.2 describes an agent's behaviour. The agent receives an instance L and the set of rules $\mathcal{R}$. For each rule R from the set of rules the agent checks whether the rule holds given L. $GetValues$ is a convenience function which, given L, returns a list P of variable values occurring in rule R. The $CheckConsistency$ function checks if the rule R holds given P and returns a boolean SW. In the $ProcessResult$ function we only print the values P in a file if SW is $false$, i.e., the rule R did not hold over L.

5. Experiments

In order to evaluate our approach we have performed two experiments. For the first one we have used a dataset provided by the Romanian Resuscitation Council while the second experiment is on a synthetic dataset.

One of the CNRR datasets consists of 698 instances and 49 variables of categorical data recording information such as: date of cardiac arrest, time of collapse, treatment before team arrival, time of CPR (Cardiopulmonary Resuscitation) by rescuer, defibrillation time by rescuer, number of shocks, time of first cardiac rhythm analysis, defibrillator type etc. The data is noisy and

there are a lot of missing values. After a preprocessing phase 62 instances having the following variables were selected for analysis: patient id, time of CA (Cardiac Arrest) establishment, time of CPR, time of first cardiac rhythm analysis, time of defibrillation. Between these variables there is only one constraint: they have to be in increasing order.

In Table 1 we show the problematic instances as resulted from the execution on the CNRR dataset.

Id	CA	CPR	First CRA	Defibrillation	Deceased
73	9:05	9:00	9:00	9:00	6 Feb 2011
19	0:45	0:45	23:00	0:45	None
1225	19:40	19:30	19:27	19:34	5 May 2009
2170	16:25	16:45	16:47	16:46	8 Jul 2011

TABLE 1. CNRR dataset. Column Id represents the patient Id, column CA denotes the time of cardiac arrest, CPR represents the time of cardiopulmonary resuscitation, column $FirstCRA$ represents the time of the first cardiac rhythm analysis, $Defibrillation$ represents the time of defibrillation and column $Deceased$ represents the decease date of the patient.

Instance 73 is not valid because the reported time of cardiac arrest is after the reported time of CPR. In case of instance 19 the time of first cardiac rhythm analysis is far from the other reported times. Instance 1225 is problematic since the times of cardiac arrest, cardiopulmonary resuscitation and first cardiac rhythm analysis are not in chronological order. In instance 2170 reports a defibrillation time before the time of First CRA. We notice that all patients are deceased or the information regarding their survival is missing.

Analysing the considered dataset (after preprocessing and feature selection) we notice that 45.16% of the patients have actually survived. The fact that all the reported problematic instances represent data from patients that have not survived is disturbing. The dataset resulted after preprocessing is indeed small compared to the original and hence a general conclusion regarding a certain correlation cannot be drawn. Nevertheless, the question is if the procedures followed by the rescue team in the case of the reported problems have been accurately followed. Our results show that the procedures may not have been followed in the case of patients from Table 1. Could this be the cause of their death?

Our second experiment was done on a synthetic dataset. We have generated 10000 instances with 10 variables imposing the following rules:

$$R1 : X1 \leq X2 \leq X3 \leq X4, \tag{8}$$

$$R2 : X5 \leq X6, \tag{9}$$

$$R3 : X7 \leq X8, \tag{10}$$

$$R4 : X9 \leq X10. \tag{11}$$

A snapshot of the dataset is shown in Table 2.

Id	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
0	19:18	15:04	8:15	17:09	13:10	5:09	10:42	0:06	19:00	18:35
1	23:06	7:08	3:48	1:57	20:48	22:33	12:42	10:45	6:18	10:08
2	3:47	14:03	12:57	12:36	11:18	8:30	16:49	12:22	22:03	3:25

TABLE 2. Synthetic dataset (snapshot).

Table 3 shows the problematic times (with respect to the considered rules) of the instances considered in Table 2.

Id 0	Id 1	Id 2
19:18 15:04 8:15 17:09	23:06 7:08 3:48 1:57	3:47 14:03 12:57 12:36
13:10 5:09	None	11:18 8:30
10:42 0:06	12:42 10:45	16:49 12:22
19:00 18:35	None	22:03 3:25

TABLE 3. Synthetic dataset (snapshot) — result.

In Figure 2 we show the performance of the execution on the synthetic dataset considering 10 episodes.

6. Comparison to related work

Authors from [7] use a two dimensional grid of agents in order to check that all constraints of a CSP problem are satisfied. As opposed to their approach, our aim is to find all variable assignments for which some constraint does not hold. Thus, even though both approaches deal with software agents and rule-checking, the proposed frameworks are fundamentally different.

In [8], the authors introduce a methodology called constraint partition and coordinated reaction for distributed constraint satisfaction. In their approach the instantiation of a variable depends on a set of agents who can negate or

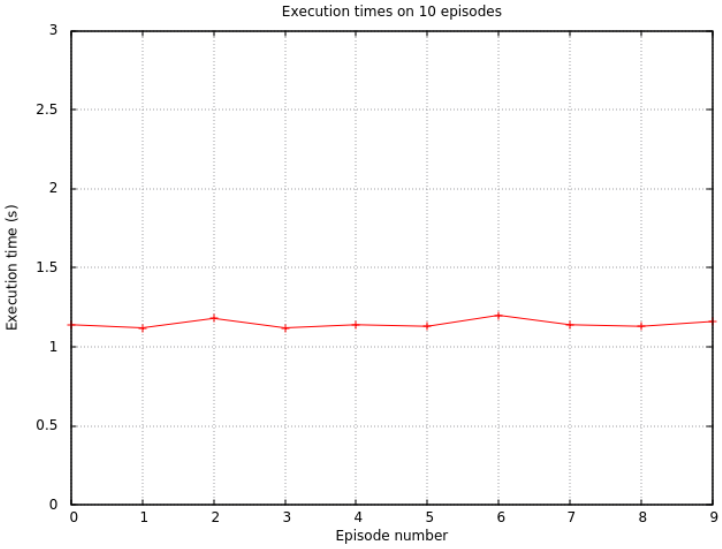


FIGURE 2. Execution performance on 10 episodes.

approve the instantiation of the same variable confirmed by some agent beforehand, and this is what authors from [8] call distributed constraint satisfaction with coordination reaction. A final solution is an instantiation of all variables that all agents agree on, i.e. it does not violate any constraints. The authors from [8] have improved this method by adding the responsibility for enforcing constraints of a particular type to specialist agents. The agents coordinate to iteratively change the instantiation of variables under their jurisdiction according to their specialized perspective. Experiments on a benchmark suite of job shop scheduling problems show promising results. While authors from [8] are focusing on finding variable values that match all rules, our aim is to find all instances that don't match some rule.

Authors from [16] advocate that distributed constraint satisfaction problem solving provide a useful mechanism for formalizing cooperative distributed problems in general. They present the asynchronous backtracking algorithm that allows agents to act asynchronously and concurrently, in contrast to the traditional sequential backtracking techniques employed in constraint satisfaction problems. The authors from [16] use one agent for each variable. Each agent instantiate its variable concurrently and use message passing in order come to an agreement. In contrast to this approach we assign agents to instances, not to variables. As soon as an agent has evaluated an instance, it may resume its execution and check another instance (or it may be replaced

by another newly created agent) and hence the number of agents remains constant with respect to the number of instances. This is not the case in the approach from [16] though, where the number of agents grows linearly with the number of variables; however this is seldom an issue.

7. Conclusions and future work

An agent-based approach to parallel constraint verification is presented in this paper. We use a dataset provided by CNRR and synthetic one.

The CNRR provided dataset contains highly noisy data and hence after preprocessing it we obtain only a small fraction of the original instances. Our agent-based approach is able to detect problematic instances corresponding to patients that have not survived. Actually all the detected instances correspond to patients that are either dead or any information regarding survival is missing.

Our future work will focus on: extending our system with a rule inference engine, fuzzifying the rules, improving the performance of our system.

Acknowledgements

The authors gratefully acknowledge the effort of the CNRR team responsible with collecting and preparing the CNRR data set. We particularily thank Dr. Valentin Georgescu, Dr. Mihaela Godeanu, Dr. Horea Sabău, Dr. Victor Strâmbu, Dr. Oana Tudorache, from the CNRR office in Bucharest.

This work was supported by a grant of the Romanian National Authority for Scientific Research, CNDI–UEFISCDI, project number PN-II-PT-PCCA-2011-3.2-0917.

References

[1] Consiliul Naţional Român de Resuscitare. http://www.cnrr.org/.

[2] T. Finin, Y. Labrou, and J. Mayfield. *Kqml as an Agent Communication Language*. Software Agents, B.M. Jeffrey, MIT Press, 1997.

[3] Foundation for Intelligent Physical Agents. http://www.fipa.org/.

[4] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semant.*, 6(4):309–322, November 2008.

[5] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 723–731, New York, NY, USA, 2004. ACM.

[6] Ian Jacobs, Vinay Nadkarni, the ILCOR Task Force on Cardiac Arrest, Cardiopulmonary Resuscitation Outcomes, Jan Bahr, Robert A. Berg, John E. Billi, Leo Bossaert, Pascal Cassan, Ashraf Coovadia, Kate D'Este, Judith Finn, Henry Halperin, Anthony Handley, Johan Herlitz, Robert Hickey, Ahamed Idris, Walter Kloeck, Gregory L. Larkin, Mary E. Mancini, Pip Mason, Gregory Mears, Koenraad Monsieurs, William

Montgomery, Peter Morley, Graham Nichol, Jerry Nolan, Kazuo Okada, Jeffrey Perlman, Michael Shuster, Petter A. Steen, Fritz Sterz, James Tibballs, Sergio Timerman, Tanya Truitt, and David Zideman. Cardiac Arrest and Cardiopulmonary Resuscitation Outcome Reports: Update and Simplification of the Utstein Templates for Resuscitation Registries: A Statement for Healthcare Professionals From a Task Force of the International Liaison Committee on Resuscitation (American Heart Association, European Resuscitation Council, Australian Resuscitation Council, New Zealand Resuscitation Council, Heart and Stroke Foundation of Canada, InterAmerican Heart Foundation, Resuscitation Councils of Southern Africa). *Circulation*, 110(21):3385–3397.

[7] Jiming Liu, Han Jing, and Y. Y. Tang. Multi-agent oriented constraint satisfaction. *Artif. Intell.*, 136(1):101–144, February 2002.

[8] Jyishane Liu and Katia P. Sycara. Distributed constraint satisfaction through constraint partition and coordinated reaction. In *Proceedings of the 12th International Workshop on Distributed AI*, 1993.

[9] Mary Ann A. Peberdy, William Kaye, Joseph P. Ornato, Gregory L. Larkin, Vinay Nadkarni, Mary Elizabeth E. Mancini, Robert A. Berg, Graham Nichol, and Tanya Lane-Trultt. Cardiopulmonary resuscitation of adults in the hospital: a report of 14720 cardiac arrests from the National Registry of Cardiopulmonary Resuscitation. *Resuscitation*, 58(3):297–308, September 2003.

[10] Abdur Rakib, RokanUddin Faruqui, and Wendy MacCaull. Verifying resource requirements for ontology-driven rule-based agents. In Thomas Lukasiewicz and Attila Sali, editors, *Foundations of Information and Knowledge Systems*, volume 7153 of *Lecture Notes in Computer Science*, pages 312–331. Springer Berlin Heidelberg, 2012.

[11] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[12] Gabriela Şerban. *Sisteme Mutiagent in inteligenta artificiala distribuita. Arhitecturi si aplicatii*. Ed. Risoprint, Cluj-Napoca, 2006.

[13] Gabriela Şerban and Horia Florin Pop. *Tehnici de Inteligenta Artificiala. Abordari bazate pe Agenti Inteligenti*. Ed. Mediamira, Cluj-Napoca, 2004.

[14] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[15] Michael Wooldridge. *Intelligent Agents, An Introduction to Multiagent Systems*. Ed. G. Weiss, 1999.

[16] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, June 2000.

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

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

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

E-mail address: ssie1324@scs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

HEURISTIC ALGORITHMS FOR SOLVING THE BI-DIMENSIONAL TWO-WAY NUMBER PARTITIONING PROBLEM

LEVENTE FUKSZ[(1)], PETRICĂ POP [(2)], AND IOANA ZELINA [(2)]

Abstract. The bi-dimensional two-way number partitioning problem is a generalization of the classical number partitioning problem, where instead of a set of integers we have a set of vectors of dimension 2 that have to be divided into two subsets so that the sums of the vectors in the subsets are equal or almost equal for both coordinates. This work presents three heuristic algorithms for solving the problem. The algorithms are analyzed, implemented and tested on randomly data instances.

1. Introduction

The number partitioning problem (NPP) is a well known combinatorial optimization problem defined as follows: given a finite set of n integers $S = \{a_1, a_2, ..., a_n\}$, find a partition of S into two subsets S_1 and S_2 such that it minimizes the difference of the sums of the elements in the subsets:

$$\Big| \sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i \Big| \rightarrow min.$$

This difference is called *discrepancy* and the partitions with the property that the discrepancy is 0 or 1 are called *perfect partitions*.

The NPP is a difficult problem belonging to the the class of *NP-hard* problems [3] and therefore it is difficult to be solved in both theory and practice.

Received by the editors: April 22, 2013.

2010 *Mathematics Subject Classification.* 90C27, 68T15.

1998 *CR Categories and Descriptors.* G.2.1 [**Mathematics of Computing**]: Discrete mathematics – *Combinatorial Algorithms*; I.2.8 [**Computing Methodologies**]: Artificial intelligence – *Problem Solving, Control Methods and Search*.

Key words and phrases. heuristic algorithms, number partitioning problem, bi-dimensional two-way number partitioning problem.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

The problem finds interesting applications in public key encryption, scheduling problems, minimization of the VLSI circuit size, choosing up fair sides in a ball game, etc.

There are several ways to solve the NPP in exponential time in n: the most naive algorithm would be to cycle through all the subsets of n numbers and for every possible subset S_1 and for its corresponding complementary $S_2 = S \setminus S_1$ calculate their sums. Obviously, this algorithm is impracticable for large instances, since its time complexity is $O(2^n)$. A better exponential time algorithm which runs in $O(2^{n/2})$ was described by Horowitz and Sahni [5].

There have been proposed several heuristic algorithms in order to provide high-quality solutions for the NPP: the set differencing heuristic introduced by Karmarkar and Karp [7], a Simulated Annealing algorithm described by Johnsonn et al. [6], genetic algorithm by Ruml et al. [11], GRASP by Arguello et al. [1], Tabu Search by Glover and Laguna [4], memetic algorithm by Berretta et al. [2], etc.

Kojic [8] described a generalization of the NPP called *the multidimensional two-way number partitioning problem* (MDTWNPP), where instead of numbers we have a set of vectors and we are looking for a partition of the vectors into two subsets such that the sums per every coordinate should be as close as possible. Kojic provided an integer programming formulation and tested the model on randomly generated sets using CPLEX. The obtained experimental results show that the MDTWNPP is very hard to solve even in the case of medium instances. Recently, Pop and Matei [10] introduced an efficient memetic algorithm for solving the MDTWNPP.

In this work we confine to the case when the vectors have dimension two and the aim of this paper is to describe three heuristic approaches: a greedy algorithm and a novel use of genetic algorithms with the goal of solving the bi-dimensional two-way number partitioning problem and a hybrid GA-VNS heuristic that combines the use of genetic algorithms (GA) and Variable Neighborhood Search (VNS). The results of preliminary computational experiments are presented and analyzed.

2. Definition of the bi-dimensional two-way number partitioning problem

Given a set of n bi-dimensional vectors:

$$S = \{v_i \mid v_i = (v_{i1}, v_{i2}),\ i \in \{1, ..., n\}\}$$

then the *bi-dimensional two-way number partitioning problem* (BTWNPP) consists in splitting the elements of S into two sets, S_1 and S_2 such that

1. $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$;
2. the sums of elements in the subsets S_1 and S_2 are equal or almost equal for both coordinates.

If we introduce the variable t that denotes the greatest difference in sums per coordinate, i.e.

$$t = \max\left\{ \left| \sum_{i \in S_1} v_{ij} - \sum_{i \in S_2} v_{ij} \right| \ : \ j \in \{1, 2\} \right\}$$

then the objective function of the BDTWNPP is to minimize t.

Defined in this way, we can observe that the BDTWNPP is a special case of the multidimensional two-way number partitioning problem introduced by Kojic [8], where the vectors are bi-dimensional.

The BDTWNPP is NP-hard, as it reduces when the vectors have dimension one to the NPP, which is known to be an NP-hard problem.

Next we define the *bi-dimensional multi-way number partitioning problem* (BDMWNPP) as a generalization of BDTWNPP where a set of vectors is partitioned into a given number of subsets rather than into two subsets.

Let again S be a set of n bi-dimensional vectors and $p \in \mathbb{N}$, $p \geq 2$, then the bi-dimensional multi-way number partitioning problem consists in splitting the elements of S into p subsets, $S_1, S_2, ..., S_p$ such that

1. $S_1 \cup S_2 \cup ... \cup S_p = S$ and $S_i \cap S_j = \emptyset$, for all $i, j \in \{1, ..., p\}$ and $i \neq j$;
2. the sums of elements in the subsets $S_1, S_2, ..., S_p$ are equal or almost equal for both coordinates.

In particular, if the set of vectors is partitioned into two subsets we get the BDTWNPP. For partitioning into more than two subsets, the objective function to be minimized is the greatest difference between maximum and minimum subset sums for both coordinates.

Introducing the variable t denoting the greatest difference between maximum and minimum subset sums per every coordinate, i.e.

$$t = \max\left\{ \left| \max\left\{ \sum_{i \in S_l} v_{ij} : l = \overline{1, p} \right\} - \min\left\{ \sum_{i \in S_l} v_{ij} : l = \overline{1, p} \right\} \ \middle| \ j \in \{1, 2\} \right\}$$

then the objective function of the BDMWNPP is to minimize t.

Example Let $S = \{(1, 4), (2, 9), (7, 2), (5, 5), (3, 7), (4, 10), (3, 2)\}$ and we want to partition its elements into two subsets S_1 and S_2. We can do this partition in several ways, some candidates are:

- $S_1 = \{(1, 4), (2, 9), (7, 2)\}$, $S_2 = \{(5, 5), (3, 7), (4, 10), (3, 2)\}$, then the sums are $(10, 15)$ and $(15, 24)$, the difference between the maximum and minimum values per coordinates is $(5, 9)$ and therefore $t = 9$;

- $S_1 = \{(1,4),(2,9),(7,2),(5,5)\}$, $S_2 = \{(3,7),(4,10),(3,2)\}$, then the sums are $(15,20)$ and $(10,19)$, the difference between the maximum and minimum values per coordinates is $(5,1)$ and $t = 5$;
- $S_1 = \{(1,4),(4,10),(7,2)\}$, $S_2 = \{(5,5),(3,7),(2,9),(3,2)\}$ then the sums are $(12,16)$ and $(13,23)$, the difference between the maximum and minimum values per coordinates is $(1,7)$ and $t = 7$;
- $S_1 = \{(1,4),(2,9),(7,2),(3,7)\}$, $S_2 = \{(5,5),(4,10),(3,2)\}$, then the sums are $(13,22)$ and $(12,17)$, the difference between the maximum and minimum values per coordinates is $(1,5)$ and $t = 5$;

Therefore, the minimum of the maximal elements of the listed candidates is in the second and fourth cases with $\min t = 5$.

3. Heuristic algorithms for solving the bi-dimensional two-way number partitioning problem

3.1. **The Greedy algorithm.** For the number partitioning problem, the obvious greedy heuristic is to sort the numbers in decreasing order, place the largest number in one of the two subsets and then place each of the remaining numbers in the subset with the smallest sum.

However, for the two-way number partitioning problem, there are two sums for each subset. In this case, each time we need to put a pair in one of the two existing subsets, we have to analyze both possibilities and then choose the one which gives the smallest difference.

For the given example, $S = \{(1,4),(2,9),(7,2),(5,5),(3,7),(4,10),(3,2)\}$, we put $(1,4)$ in S_1 and $(0,0)$ in S_2. The current sub-sums are $(1,4)$ respectively $(0,0)$. To put $(2,9)$ in one of the two subsets, we calculate the sub-sums and we obtain $(3,13)|(0,0)$, respectively $(1,4)|(2,9)$. In the first case the difference is $max\{|3-0|,|13-0|\} = 13$ and in the second case the difference is $max\{|1-2|,|4-9|\} = 5$. Therefore, $(2,9)$ goes in S_2 and we have the sub-sums $(1,4)|(2,9)$. For $(7,2)$ we have the sub-sums $(8,6)|(2,9)$, respectively $(1,4)|(9,11)$ with the smallest difference $t = 6$ so $(7,2)$ goes in S_1. For $(5,5)$, the sub-sums are $(13,11)|(2,9)$, respectively $(8,6)|(7,14)$ with the smallest difference $t = 8$ and $(5,5)$ goes in S_2. For $(3,7)$, the sub-sums are $(11,13)|(7,14)$ respectively $(8,6)|(10,21)$ with the smallest difference $t = 4$ and $(3,7)$ goes in S_1. For $(4,10)$, the sub-sums are $(15,23)|(7,14)$ respectively $(11,13)|(11,24)$ with the smallest difference $t = 9$ and $(4,10)$ goes in S_1. For $(3,2)$, the sub-sums are $(18,25)|(7,14)$ respectively $(15,23)|(10,16)$ with the smallest difference $t = 7$ and $(4,10)$ goes in S_2. We obtain then the partition $S_1 = \{(1,4),(7,2),(3,7),(4,10)\}$, $S_2 = \{(2,9),(5,5),(3,2)\}$, the sub-sums are $(15,23)$ and $(10,16)$ and the smallest difference is $t = 7$. We can rewrite the

sequence as $(1,4)|(0,0),(1,4)|(2,9),(8,6)|(2,9),(8,6)|(7,14),(11,13)|(7,14)$, $(15,23)|(7,14),(15,23)|(10,16)$ and the smallest difference $t=7$.

Each time we put a pair in one of the two subsets we have to perform 11 computational steps: 8 additions and 3 comparisons, so we perform $(n-1)\times 11$ computational steps for a set with n pairs. For our example, we perform $6\times 11 = 66$ computational steps. If both current subsumes in one of the subsets are smaller or equal to the sub-sums in the other subset and the pairs are pairs of positive integers, then we can put the next pair directly in the subset with the smallest sub-sums. In this case, we perform only 4 additions and 3 comparisons instead of 8 additions and 3 comparisons. For our example, we put $(7,2)$ directly in S_1 because both subsumes in S_1, $(1,4)$ are smaller than those in S_2, $(2,9)$. The number of computational steps becomes $7+7+11+11+11+7=54$.

The suboptimal solution provided by the greedy heuristic algorithm depends on the order the pairs in the set S are chosen.

If we sort the pairs in S in decreasing order according to the second component, we obtain $S=\{(4,10),(2,9),(3,7),(5,5),(1,4),(3,2),(7,2)\}$. The sequence that describes the greedy algorithm becomes: $(4,10)|(0,0),(4,10)|(2,9)$, $(4,10)|(5,16),(9,15)|(5,16),(9,15)|(6,20),(9,15)|(13,22),(12,17)|(13,22)$ and the smallest difference is $t=5$. The subsets are $S_1=\{(4,10),(5,5),(3,2)\}$ and $S_2=\{(2,9),(3,7),(1,4),(7,2)\}$. The number of computational steps is $7+7+7+11+11+7=50$. In this case the solution is better and the number of computational steps is smaller than in the previous case.

3.2. **The genetic algorithm.** The Genetic Algorithms (GA) were introduced by Holland in the early 1970s, and were inspired by Darwins theory. The idea behind GA is to model the natural evolution by using genetic inheritance together with Darwins theory. GA have seen a widespread use among modern metaheuristics, and several applications to combinatorial optimization problems have been reported. Next we give the description of our genetic algorithm for solving the BDTWNPP.

Representation

In order to represent a potential solution to the BDTWNPP, we used a binary representation where every chromosome is a fixed size (n-dimensional vector) ordered string of bits 0 or 1, identifying the set of partition as assigned to the pairs. This representation ensures that the set of vectors belonging to the set S is partitioned into two subsets S_1 and S_2.

Initial population

The construction of the initial population is of great importance to the performance of GA, since it contains part of the building blocks the final solution is made of, which is then combined by the crossover operator.

We considered a novel method for generating the initial population: partially randomly and partially based on the problem structure. In this case, we pick randomly a number $q \in \{2, ..., n\}$ and then for the pairs belonging to $\{2, ..., q\}$ the genes are generated randomly and the other pairs are partitioned iteratively such that by adding each pair we reduce the greatest difference in sums per coordinate.

The fitness value

GAs require a fitness function which allocates a score to each chromosome in the current population. Thus, it can calculate how well the solutions are coded and how well they solve the problem. In our case, the fitness value of the BDTWNPP, for a given partition of the pairs into two subsets is given by the corresponding greatest difference in sums per coordinate and the aim of the problem is to minimize this value.

Selection

Selection is the process used to select individuals for reproduction to create the next generation. This is driven by a fitness function that makes higher fitness individuals more likely to be selected for creating the next generation. We have implemented three different selection strategies: the fitness proportionate selection, the elitist selection and the tournament selection. Several BDTWNPP were tested and the results show that the tournament selection strategy outperformed the other considered selection strategies, achieving best solution quality with low computing times.

Genetic operators

During each successive generation, a proportion of the existing population is selected to produce a new generation. The *crossover operator* requires some strategy to select two parents from previous generation. In our case we selected the two parents using the binary tournament method, where p solutions, called parents, are picked from the population, their fitness is compared and the best solution is chosen for a reproductive trial. In order to produce a child, two binary tournaments are held, each of which produces one parent. We have experimented a single point crossover. The crossover point is determined randomly by generating a random number between 1 and $n - 1$. We decided upon crossover rate of 0.85 based on preliminary experiments with different values.

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state and it helps to prevent the population from stagnating at any local optima and its purpose is to maintain diversity within the population and to inhibit the premature convergence. We consider a mutation operator that changes the new offspring by flipping bits from 1 to 0 or from 0 to 1. Mutation can occur at each bit position in the string with 0.1 probability.

An important feature of our GA, that increased its performance, is that every time a new population is produced, we eliminate the duplicate solutions.

In our algorithm the termination strategy is based on a maximum number of generations to be run if the optimal solution of the problem is not found or no improvement of the discrepancy value is not observed within 15 consecutive generations.

As we will see in the Computational results section, our proposed GA is effective in producing good solutions. However, due to the weakness of GAs to intensify the search in promising areas of the solutions space, we will combine our GA with the local search ability of VNS in order to enhance the exploitation ability of GAs.

3.3. **The GA-VNS hybrid algorithm.** Variable neighborhood search (VNS) is quite a recent metaheuristic for solving combinatorial optimization and global optimization problems introduced by Mladenovic and Hansen [9]. Its basic idea is a systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley.

VNS is based on two simple facts:

- **Fact 1:** A local minimum w.r.t. one neighborhood structure is not necessary a local minimum with another;
- **Fact 2:** A global minimum is a local minimum w.r.t. all possible neighborhood structures.

The hybrid algorithm that we are going to present in this section combines the GA described in the previous section with a Variable Neighborhood Search procedure.

Applying local search to all the individuals of a current population will lead to highly time consuming procedure, therefore we selected a subset of individuals in each generation with a specified probability and then the VNS procedure is applied to each of them separately. In the case better individuals are found they are introduced in the current population.

Our VNS algorithm applies 10 types of neighborhoods, denoted by $\mathcal{N}_i$, $i \in \{1, ..., 10\}$. The neighborhoods are implemented as inversions of bits (representing either 0 or 1) within the chromosome with positions generated randomly. The ten neighborhoods correspond to the number of bits which are inverted $i \in \{1, ..., 10\}$. For each neighborhood the following repetitive loop is applied: we choose randomly the positions within the chromosome made up of bits and then we inverse the corresponding genes by logical negation.

The first neighborhood is the set of candidate solutions that have one bit difference against the current solution. We select randomly an entry from the string representation and then inverse the value of the corresponding gene,

meaning that we assign an integer from one set to the other one. If by changing the value of a gene, we obtain a neighbor having a lower cost than the current solution, than the neighbor becomes the new current solution and the search proceeds. The search process continues until no better solution is found in the neighborhood, then the search switches to the second neighborhood, which consists of candidate solutions having exactly two bits difference against the current solution. This new neighborhood is examined in order to find an improvement solution and the search continues till a better feasible solution cannot be found.

Then the search switches to the new neighborhood and the process goes on iteratively.

The switching of neighborhoods prevents the search being struck at a local optimum. When there is no better solution found in a current neighborhood, it can be a local optimum, but by changing the neighborhood, it is highly probable that a better feasible solution can be found and the local optimum is skipped.

The described strategy show how to use VNS in descent in order to escape from a local optimum of and now we are interested in finding promising regions for sub-optimal solutions.

Our implementation of the VNS procedure is described in Algorithm 1.

Algorithm 1 Variable Neighborhood Search Framework

Initialization. Select a set of neighborhoods structures $\mathcal{N}=\{\mathcal{N}_l \mid l = 1, ..., 10\}$; an initial solution x and a stopping criterion
Repeat the following sequence till the stopping criterion is met:
(1) Set $l = 1$;
(2) Repeat the following steps until $l = 10$:
Step 1 (Shaking): Generate $x' \in \mathcal{N}_l$ at random;
Step 2 (Local Search): Apply a local search method starting with x' as initial solution and denote by x'' the obtained local optimum;
Step 3 (Move or not): If the local optimum x'' is better than the incumbent x,
then move there ($x \leftarrow x''$) and continue the search with $\mathcal{N}_1$
otherwise set $l = l + 1$ (or if $l = 10$ set ($l = 1$);
Go back to Step 1.

According to this basic scheme, we can observe that our VNS is a random descent first improvement heuristic.

The algorithm starts with an initial feasible solution x from the selected individuals from the current population and with the set of the 10 nested

neighborhood structures: $\mathcal{N}_1$,..., $\mathcal{N}_{10}$, having the property that their sizes are increasing:

$$\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \subseteq ... \subseteq \mathcal{N}_{10}(x).$$

Then a point x' at random (in order to avoid cycling) is selected within the first neighborhood $\mathcal{N}_1(x)$ of x and a descent from x' is done with the local search routine. This will lead to a new local minimum x''. At this point, there exists three possibilities:

1) $x'' = x$, i.e. we are again at the bottom of the same valley and we continue the search using the next neighborhood $\mathcal{N}_l(x)$ with $l \geq 2$;
2) $x'' \neq x$ and $f(x'') \geq f(x)$, i.e. we found a new local optimum but which is worse than the previous incumbent solution. Therefore, also in this case, we will continue the search using the next neighborhood $\mathcal{N}_l(x)$ with $l \geq 2$;
3) $x'' \neq x$ and $f(x'') < f(x)$, i.e. we found a new local optimum but which is better than the previous incumbent solution. In this case, the search is re-centered around x'' and begins with the first neighborhood.

If the last neighborhood has been reached without finding a better solution than the incumbent, then the search begins again with the first neighborhood $\mathcal{N}_1(x)$ until a stopping criterion is satisfied. In our case, as stopping criterion we have chosen a maximum number of iterations since the last improvement.

4. Computational results

This section presents the obtained results for solving the BDTWNPP. The experiments were carried out on instances obtained using the randomly number generator Random.org.

The testing machine was an Intel Core i5-2450M and 4 GB RAM with Windows 7 as operating system. The greedy algorithm, GA and GA-VNS hybrid algorithm have been developed in Microsoft .NET Framework 4 using C #.

Based on preliminary computational experiments, we set the following genetic parameters: the size of the initial population consists of 50 individuals generated half randomly, tournament selection with groups of 7, one-point crossover, mutation probability 10% and maximum number of generations 100.

In Table 1, we present the obtained computational results using our proposed greedy algorithm, GA and GA-VNS hybrid algorithm. In our experiments, we performed 5 independent runs for each instance.

The first three columns in the table give the dimension of the instance: the interval from were have been selected the numbers and the number of pairs

Instances			Greedy algorithm		Results of GA		Results of GA-VNS	
Min	Max	# of pairs	Best sol.	Time	Best sol.	Time	Best sol.	Time
1	100	10	15 (15;15)	0.6	11 (3;11)	0.24	3 (3;3)	0.15
1	100	50	44 (44;38)	0.0	4 (4;4)	0.567	0 (0;0)	2.75
1	100	100	59 (59;53)	0.1	3 (3;1)	0.159	1 (1;1)	0.125
1	100	500	243 (228;243)	0.0	1 (0;1)	1.340	1 (0;1)	2.578
1	100	1000	48 (1;48)	0.31	1 (1;0)	6.140	1 (1;0)	6.234
1	1000	10	499 (499;337)	0.0	55 (55;25)	0.359	55 (55;25)	0.109
1	1000	50	285 (235;285)	0.0	269 (269;193)	0.156	15 (15;3)	4.672
1	1000	100	368 (368;330)	0.1	20 (12;20)	0.390	6 (4;6)	5.390
1	1000	500	548 (511;548)	0.0	6 (1;6)	1.734	2 (1;2)	24.266
1	1000	1000	926 (890;926)	0.31	4 (4;2)	7.672	2 (2;2)	39.236
1000	10000	10	736 (736;510)	0.0	2014 (2014;722)	0.343	736 (736;510)	0.93
1000	10000	50	1040 (97;1040)	0.0	1422 (293;1422)	0.234	54 (19;54)	4.765
1000	10000	100	2651 (708;2651)	0.0	227 (134;227)	6.78	23 (12;23)	4.140
1000	10000	500	17663 (17663;17233)	0.15	61 (61;59)	2.843	11 (3;11)	24.485
1000	10000	1000	7169 (7126;7169)	0.31	41 (10;41)	24.329	10 (10;9)	53.80
10000	1000000	10	468796 (387067;468796)	0.0	308916 (306299;308916)	1.93	129927 (129927;127386)	0.46
10000	1000000	50	298653 (298653;155068)	0.0	229024 (81191;229024)	0.359	7691 (7691;6272)	1.453
10000	1000000	100	796974 (771643;796974)	0.1	96800 (4525;96800)	1.109	3019 (3019;10)	6.140
10000	1000000	500	916103 (916103;906447)	0.15	5319 (5319;3875)	13.781	1771 (37;1771)	49.689
10000	1000000	1000	3456189 (3456189;3440327)	0.31	2689 (2357;2689)	17.735	2013 (1505;2013)	7.774
1000000	10000000	10	2356617 (2356617;747913)	0.0	2540301 (2540301;460699)	0.15	407361 (51473;407361)	0.14
1000000	10000000	50	2158720 (2158720;435126)	0.0	1511498 (1511498;848162)	0.140	48618 (48618;3624)	1.125
1000000	10000000	100	392774 (156892;392774)	0.0	322636 (152242;322636)	0.140	28888 (5420;28888)	5.844
1000000	10000000	500	32243919 (31950547;32243919)	0.11	104339 (3359;104339)	20.266	3671 (355;3671)	37.48
1000000	10000000	1000	27098586 (24681685;27098586)	0.31	69835 (69835;32814)	36.783	6677 (6677;4940)	52.867

considered, the next two columns provide the results and computational times obtained by using the greedy heuristic algorithm and the last four columns give the best solutions and required necessary computational times provided by the GA and the hybrid GA-VNS algorithm.

Analyzing the results presented in Table 1, we observe that our proposed hybrid GA-VNS heuristic algorithm performs favorable in terms of the solution quality in comparison with the GA alone and the greedy heuristic algorithm: in 22 out of 25 instances the hybrid GA-VNS provided the best solutions and in the other 3 instances we obtained the same solutions as those obtained using the GA alone. For all the considered instances, the solutions provided by GA and GA-VNS have better quality than the solutions provided by the greedy algorithm.

The running times of our GA and hybrid GA-VNS are proportional with the number of generations. From Table 1, we observe that the greedy algorithm is faster comparing to GA and GA-VNS approaches.

5. Conclusions

This paper deals with the bi-dimensional two-way number partitioning problem, where a set of pairs has to be partitioned into two subsets such that the sums of numbers in each subset should be equal or are close to be equal for both coordinates.

We developed three heuristics for solving the BDTWNPP: a greedy algorithm, a genetic algorithm and an efficient hybrid approach to the problem that combines the use of genetic algorithms (GA) and Variable Neighborhood Search (VNS). Some important features of our hybrid algorithm are:

- using a novel method for generating the initial population: partially randomly and partially based on the problem structure.
- elimination of the duplicate solutions from each population;
- using the VNS procedure along the GA in order to intensify the search within promising areas of the solution space.

The preliminary computational results show that our hybrid GA-VNS algorithm compares favorably in terms of the solution quality in comparison to the greedy algorithm and the genetic algorithm alone.

In the future, we plan to asses the the generality and scalability of the proposed hybrid heuristic by testing it on more instances and to apply it also in the case of multi-way number partitioning problem.

Acknowledgments. This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS - UEFISCDI, project number PN-II-RU-TE-2011-3-0113.

References

[1] M.F. Arguello, T.A. Feo, O. Goldschmidt, *Randomized methods for the number partitioning problem*, Computers & Operations Research, Vol. 23, Issue 2, pp. 103-111, 1996.

[2] R.E. Berretta, P. Moscato, C. Cotta, *Enhancing a memetic algorithms' performance using a matching-based recombination algorithm: results on the number partitioning problem*, in Metaheuristics: Computer Decision-Making (M.G.C. Resende and J. Souza, editors), Kluwer, 2004.

[3] M. R. Garey, D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1997.

[4] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1997.

[5] E. Horowitz, S. Sahni, *Computing partitions with applications to the Knapsack problem*, Journal of ACM, Vol. 21, Issue 2, pp. 277-292, 1974.

[6] D.S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, *Optimization by simulated annealing: An experimental evaluation; Part II: Graph coloring and number partitioning*, Operations Research, Vol. 39, Issue 3, pp. 378-406, 1991.

[7] N. Karmarkar, R.M. Karp, *The differencing method of set partitioning*, Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, 1982.

[8] J. Kojic, *Integer linear programming model for multidimensional two-way number partitioning problem*, Comput. Math. Appl., Vol. 60, pp. 2302-2308, 2010.

[9] N. Mladenovic and P. Hansen, *Variable neighborhood search*, Computers and Operations Research, Vol. 24, Issue 11, pp. 1097-1100, 1997.

[10] P.C. Pop and O. Matei, *A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem*, Applied Mathematical Modelling (to appear).

[11] W. Ruml, J.T. Ngo, J. Marks, S.M. Shieber, *Easily searched encodings for number partitioning*, Journal of Optimization Theory and Applications, Vol. 89, Issue 2, pp. 251-291, 1996.

(1) Indeco Soft, 5 Magnoliei St., 430094 Baia Mare, Romania
E-mail address: levi.fuksz@yahoo.com

(2) Department of Mathematics and Computer Science, North University Center of Baia Mare, Technical University of Baia Mare, 76 Victoriei St., 430122 Baia Mare, Romania
E-mail address: petrica.pop@ubm.ro

E-mail address: ioanazelina@yahoo.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

A MULTI-DIMENSIONAL SEPARATION OF CONCERNS OF THE WEB APPLICATION REQUIREMENTS

CALIN-EUGEN-NICOLAE GAL-CHIS[(1)]

Abstract. The implementation phase of the requirements in web applications is depending on a certain level by the platform of deployment, the architectural choice, the developer expertise and experience, is depending on the programming languages, and of the framework and tools selected to be used during the development process. A methodology of sorting and grouping web application requirements is needed for two reasons: the first is to make sure that the requirement fits a specific form of description in the recording of own characteristics and second, to assign the requirements to the appropriate deployment method, to a certain type of developer in his domain of expertise. An approach based on multi-dimensional separation of concerns is proposed to guide the process of implementing the requirements.

keywords: requirements engineering, web applications, separation of concern, software architecture

1. Introduction

In todays web applications development, requirements engineering (RE) is making important steps towards formalizing the requirements. Requirements are supposed to be implemented into the application. In terms of implementing the requirements the only category of people interacting with the requirements are the developers. By developers we do not understand just programmers, but include here web designers, software architects, testers, tools of deployment or any other actor interacting direct or indirect with the code of the developed product, with the application data or with the system configuration of the

Received by the editors: March 30, 2013.

1998 *CR Categories and Descriptors.* D.2.1 [**SOFTWARE ENGINEERING**]: Requirements/Specifications – *methodologies*; D.2.9 [**SOFTWARE ENGINEERING**]: Management – *Life cycle*.

Key words and phrases. requirements engineering, web applications, separation of concern, software development.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

softwares environment. In the process of implementing one requirement, the developers are affecting at least one of the components of the MVC (Model-View-Controller) architectural pattern in at least one aspect.

In web applications and not only, the software architecture [2] is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

The software architecture is focused to organize functionalities in areas of concern, such as data layer, busyness layer, service layer, presentation layer and other connected systems. The choice of architecture solutions in web applications is vast, a selection of the solution depending of several factors, such as ease of deployment, reduced cost, ease of development, reusability, mitigation of technical complexity.

The RE is providing methodologies to define the requirements. Once they are defined, requirements have to be implemented into the application, as part of the requirements lifecycle. This paper introduces an approach for mapping the requirements to the software architecture by using a separation of concerns based methodology.

2. Related work

Even though one of the uses of RE is exploratory, such as while specifying requirements, the problem to be solved is better understood; the goal of RE in the Software Engineering is to obtain a stable set of requirements, which serves as basis for the further steps in the development process. According to Lowe and Hall, three activities are used to achieve this goal: elicitation, specification, and validation of requirements [1].

The elicitation of requirements is the activity by means of which the functionalities of the system to be built are collected from any available source. The overall requirements elicitation objectives for software engineering remain unchanged when applied to Web systems. However, the specific objectives for Web systems become: (1) the identification of content requirements, (2) the identification of the functional requirements in terms of navigation needs and businesses processes, and (3) the definition of interaction scenarios for different groups of Web users.

Requirements specification consists in producing a description of the requirements. Different techniques can be used for the specification: from informal textual description to formal specification techniques.

Finally, requirements validation consists in checking the requirements specification in order to establish whether the clients and Web application users needs are fulfillefd.

Escalona and Koch [3] are proposing a requirements engineering process with these three main activities for requirements: requirements elicitation, requirements specification and requirements validation. When corrections are applied to requirements, given there are some requirements not validated in the validation phase, the activity flow is returning to the specification phase refining and adjusting the requirements to meet the specifications, as presented in Figure 1. The phases of the requirements lifecycle are handled by various persons, such as: specialists (designers, analysts), groups (e.g. in JAD) or default actors (the project manager, the client).

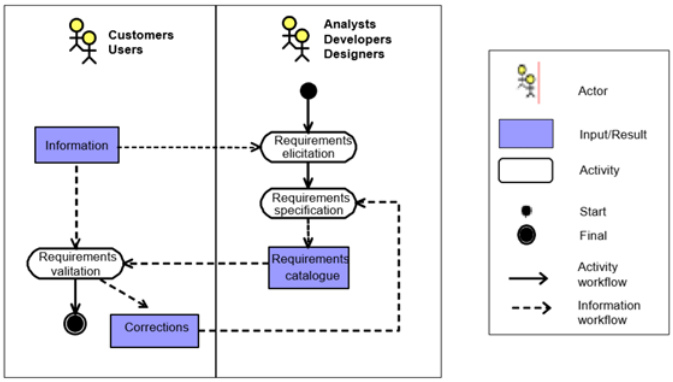


FIGURE 1. The requirements engineering process by Escalona and Koch

Another requirements lifecycle model proposed by Craig [4] specifies the requirements primary source, requirements owner, requirements location, the validation and their focus. The model covers requirements from project concept to testing and to deploy phase.

A study conducted by Heijstek and Chaudron [5] in 2008 over industrial practices of software development validates the level of effort assigned to each discipline during of the software development process as is described by the Rational Unified Process (RUP). We can conclude from the Figure 2 [6], which indicates the level of effort for requirements in RUP, that the requirements workflow is important in all the phases of a software project, mentioning here the Inception, Elaboration, Construction and Transition phases.

Notable in the RUP is that the analysis and design discipline is connected with the translation of requirements to a formal design. This type of design models can be considered tracks to be followed for writing the source code. A

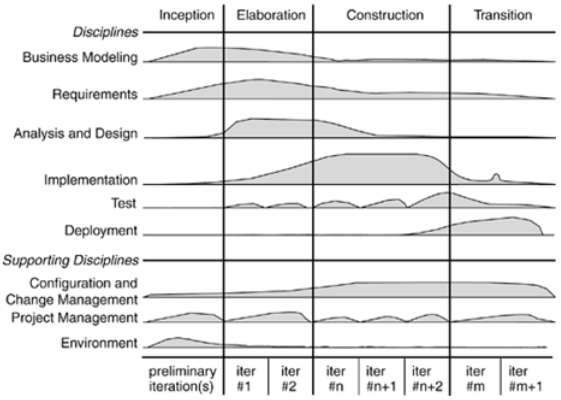


FIGURE 2. The Life Cycle for the Rational Unified Process

modeling language such as the Unied Modeling Language (UML) can be used to design classes and structure them into packages with well defined interfaces.

In order to connect the requirements to the Software Architecture (SA), Liu and Mei [7] are proposing a feature-orientation mapping from requirements to the SA. Their process implies requirements specification being organized as features. Several activities have to be performed on features: feature analysis and organization, feature elicitation, feature refinement. After that a mapping is perform from the feature model to the architecture models. The SA is defined from three viewpoints: as conceptual architecture, as logical architecture and as deployment architecture, each taking on certain types of features.

Another feature-architecture mapping (FArM) is presented by Sochos, Riebisch and Philippow[8]. Their method provides a mapping between features and architecture, which is based on a set of transformations on the initial product line feature.

The separation of concerns method is applied by Chen, Liu and Mencl [9] on Requirements Modelling. Even, though they split the model into several parts their approach is supporting separation of concerns and consistent and incremental modelling of requirements.

The separation of concerns is taken to a higher level by Moreira, Rashid and Araujo[10], in a multi-dimensional separation of concerns. Despite of the fact that they give up on using viewpoints, use cases or themes in representing the requirements, the solution provided is conceptualized in such a way that requirements are no longer scattered in different representations, but are using a unique representation. All requirements are decomposed in a uniform pattern regardless of their functional or non-functional nature.

Concern identification is based on the fact that certain concerns, both functional and non-functional, are repeating during system development. This kind of concerns may include shopping, booking, availability and security, so both functional and non-functional concerns.

The requirements space is divided into the system space and meta concern space. The system space gathers different types of systems that are possible to be realized (i.e. requirements associated to application that are just part of the requirements space); while the meta concern space comprises an abstract set of typical concerns, functional and non-functional, that are found in various systems (i.e. divides requirements into groups associated to an available concern in the space: authentication, navigation, mobility, portability,...).

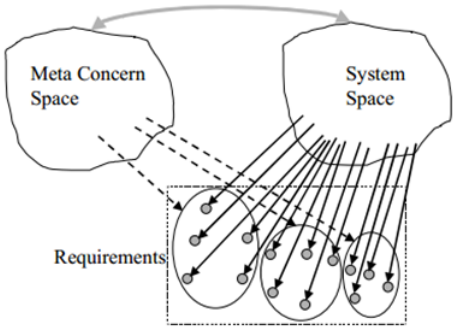


FIGURE 3. The system space and meta concern space

In the Figure 3, we can notice the requirements grouped in concerns. The concerns are part of the abstract meta concern space, while requirements are part of the concrete system space. By grouping the requirements in concerns, a conceptual binding is created between the two spaces. Both spaces are being represented using XML templates as in Figure 4.

```
<?xml version="1.0" ?>
<MetaConcern name="InformationRetrieval">
    <Description>The operation of accessing information from a
        computer system </Description>
    <Examples>Database retrieval, Multimedia retrieval</Examples>
    <Relationships> Availability, Mobility, InformationUpdate </Relationships>
</MetaConcern>
```

FIGURE 4. *InformationRetrieval* meta concern in XML

Requirements are recorded in concerns and are allocated unique ids. Also refined sub requirements are recorded the same way, but they are nested to the parent requirement, like in Figure 5. Also, a set composition rules is established to define relationships between concerns.

```
<?xml version="1.0" ?>
<Concern name="InformationRetrieval">
  - <Requirement id="1">
      It should be possible to retrieve information from the system.
      <Requirement id="1.1">It should be possible to access
          information about the attractions. </Requirement>
      <Requirement id="1.2">It should be possible to access
          information about the current location. </Requirement>
    </Requirement>
    <Requirement id="1.3">It should be possible to obtain a list of
        available preset tours. </Requirement>
</Concern>
```

FIGURE 5. *InformationRetrieval* concern in XML

A separation of the requirements in concerns is providing means in selecting the ideal or most suitable architectural choice for each concern, as presented in Figure 6. The architectural choices are usually not the same one, being possible to be even conflicting from one to another, but and analysis and negotiations with steakholders can lead to best accepted solution.

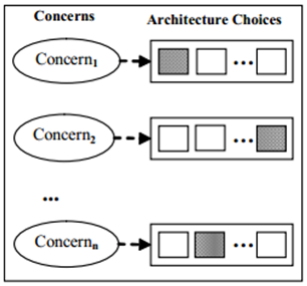


FIGURE 6. Architectural choices to satisfy each concern

3. Separation of Concerns in Web Enginnering

Mapping each requirement to the software architecture and eventually to the related developer is the last phase before the implementation of the requirements. The implementation of the requirements in the application phase should be added in the process. It is a natural step and is performed, indeed after the requirements are defined. So, in the process proposed by Escalona and Koch, the Requirement Implementation (Developing into the code) step is added, as it can be visualize in Figure 7. In this step is also included the process of the mapping of the requirement with the developer. The developer of each requirement is to be selected in conformity with the implementation method (database setup, html design, server-side script, etc.).

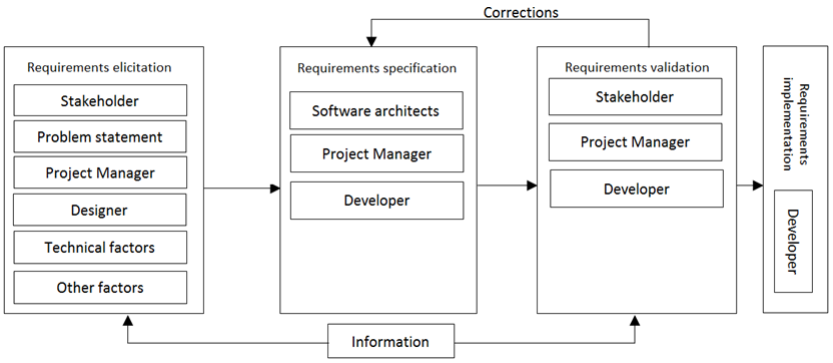


FIGURE 7. The updated requirements life-cycle and the actors involved

Developers are not involved just in the last phase; they are involved in all requirements processes. By developers we understand web designers, software architects, programmers, testers, tools of deployment or any other actor that directly affects the product. Developers are assigned specific roles in the product specification. Web designers can elicit requirements by sketching and storyboarding and they can specify requirements by modeling prototypes, while the programmers are being responsible to code prototypes in the requirements specification and in the requirements validation phase, as is presented below, in Figure 8.

Requirement elicitation	Requirements specification	Requirements validation	Requirements Implementation	
Interviewing	Natural Language	Review		
JAD (Joint Application Development)	Glossary and Ontology	Audit		
Brainstorming	Templates	Traceability Matrix	Tool Modeling	
Sketching and storyboarding	Prototypes	Prototyping for validation	Coding, Configuring, Designing	Developer activities
Use Case modeling	Use Case Modeling			
Questionnaire and Checklists	Scenarios			

FIGURE 8. Developer involvement in some of the main techniques used in the RE

Managing the requirements is a defining activity of the software development lifecycle, from the problem statement to the final product. Requirements have to be reflected in the product, therefore, implemented by the developers. After the requirements are set and ready, they are to be assign to be implemented. The process of implementing the requirements in the final product has to be assigned to a developer. The type of the developer is linked to the specific of the requirement.

Without a specific approach of mapping requirements to the software architecture the process of implementing the requirements is not fluent. Also, by means of implementing, there can be different developers, each one with their own expertise. If in enterprise applications are using a set with just several technologies, in web application there are considerably more technologies, and they are changing and updating constantly. Some technologies are new and in progress to be formalized, others will maybe be developed in the near future, providing solutions to the new communication and devices on the market.

The technologies used in web applications are included but not limited to: basic client side coding (html, javascript, css), advanced client side coding (Ajax, jquery, smarty), server side coding (java, ruby, .NET, php, asp, perl, python), client side and server side (tools to maintain complex javascript front-end applications), database technology (mysql, mssql, apache, sql lite, Microsoft SQL Server), web development software and frameworks (Macromedia Dreamweaver, WebDSL), content management systems CSM (wordpress, drupal, joomla), security (ecommerce, ebanking, networking), Web design tools (photoshop). Some developers master all technologies, but some are mastering just some of them.

A desirable approach into implementing the requirements is to assign the requirements to their appropriate technology, as represented in Figure 9. This will also help the project management in the development process to assign the requirements to developer teams in a methodological manner, so each requirement will be assigned to the best developer or team for implementation.

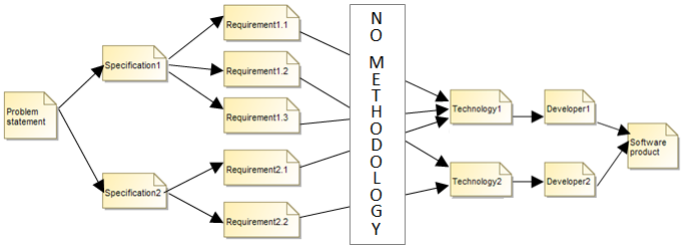


FIGURE 9. Information flow - direct connection from requirements to the technology used in implementation process

To meet this necessity, an approach can be elaborated, based on the multidimensional separation of concerns introduced by Moreira, Rashid and Araujo [10]. Their methodology can be transformed to sustain multiple aspects of the implementing the requirements process and not only. They do also provide tool support through their ARCADE tool.

One way is to create concerns spaces that are related to certain technologies. In order to do that, the requirements had to be assigned concerns into a

Meta Concern Space that can have concerns such technology types used in implementation. Following that join, a certain technology can be easly associated to a certain developer type, as shown in Figure 10.

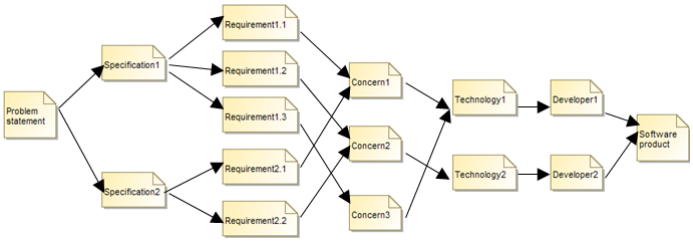


FIGURE 10. Assign requirements to technology using separation of concerns

To support this representation, to the meta concern in XML is added a *Technology* tag to record the technology appropriate to implement the concern (Figure 11).

```
<?xml version="1.0" ?>
- <MetaConcern name="InformationRetrieval">
        <Technology>Ajax calls</Technology>
        <Description>The operation of accessing information from a
                computer system </Description>
        <Examples>Database retrieval, Multimedia retrieval</Examples>
        <Relationships> Availability, Mobility, InformationUpdate </Relationships>
</MetaConcern>
```

FIGURE 11. The *InformationRetrieval* meta concern in XML with Technology details recorded

An upside of this method is that in the case of changing of one requirement, only one technology will be used and only one type of developer or one group will participate in updating the software product. Another plus of this approach is that in certain situations the technologies can have tools to perform the implementation of the requirements. In this way certain implementation does not have to be assigned to developers. Examples of tools for web are: CMS (Content Management System), WebDSL, Eclipse UML Class diagram. In other domains, other tools more appropriate can be used, like DOORS as a tool used in industry. In Figure 12 is presented such a situation.

Another possibility, depicted in Figure 13, is to create a concern space (MVC) that is assigned as concerns: Model, View and Controller, instead of technologies. This approach is useful when the level of conceptualization is high and this need is relevant to the project manager in the life cycle of developing the software product.

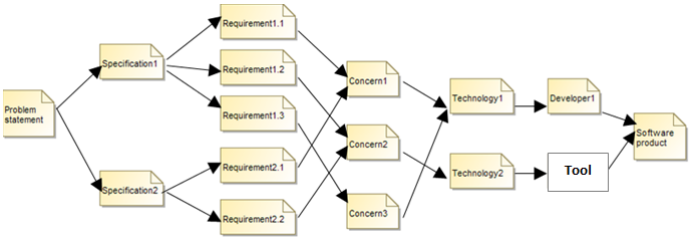


FIGURE 12. Concerns linked to development technologies relying on tools

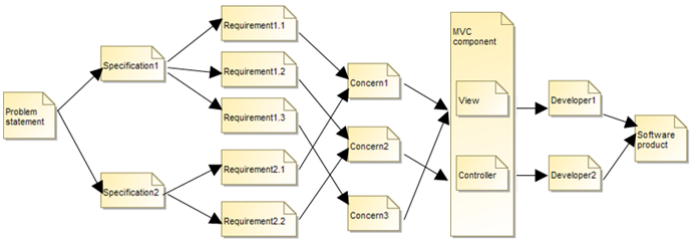


FIGURE 13. Assigning the requirements to MVC components in order to implement requirements

To support this representation, in Figure 14, we added to the meta concern in XML a *MVCComponent* tag to record in the MVC component appropriate to represent the concern

```
<?xml version="1.0" ?>
- <MetaConcern name="InformationRetrieval">
        < MVCComponent >controller</ MVCComponent >
        <Description>The operation of accessing information from a
              computer system </Description>
        <Examples>Database retrieval, Multimedia retrieval</Examples>
         <Relationships> Availability, Mobility, InformationUpdate </Relationships>
</MetaConcern>
```

FIGURE 14. The *InformationRetrieval* meta concern in XML with *MVCComponent* detail recorded

Different meta concern spaces can be created, such as MVCController meta concern space, with the concerns: model, view and controller. So, the requirements space can be altered by adding meta concern spaces. This creates a conceptual binding between the abstract representations of sets of concerns in different meta concern spaces. This binding is basically a mapping from one meta concern space to another, through requirements (Figure 15). These metaconcern spaces and the bidings between can be valuable for traceability purposes.

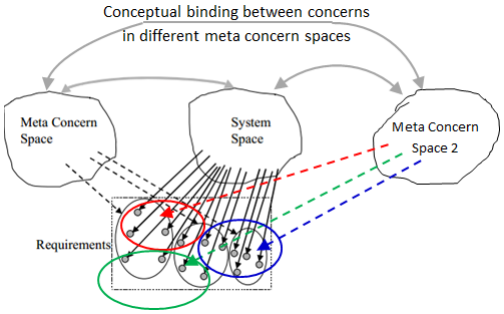


FIGURE 15. The system space and meta concern spaces

This model of dividing the requirement space can be extended to other object spaces (e.i. specification). There is a possibility for different system spaces to share the same Meta concern Space (i.e. specification space and requirements space can share the same Meta concern space, each of them being related to the same concern, such as *DevelopmentPriority*). That way a binding between different systems spaces is created, as shown in 16.

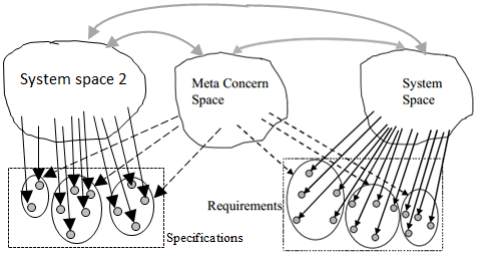


FIGURE 16. System spaces sharing a meta concern space

4. FUTURE WORK

Further investigation and analysis on the approach introduced in this paper could lead to valuable solutions or improvements to software engineering existent activities and methodologies, not only to the requirements engineering field or web application methodologies. One important step is to define rigorously the methodology and then to test it.

Tools can be developed to aid the process flow in using this methodology. Special tools could be developed to visualize the graphs created in the bindings between spaces. In some way, the bindings created in between different systems can be interrogated using a dedicated query language. The definition of sets of algebraic operations and operators on concern spaces can also be investigated.

5. Conclusions

This paper has proposed an approach based on the existing multi-dimensional separation of concerns in requirements engineering, with an impact in the implementation of the requirements for web applications. The proposed approach is subject to improvements, but it offers a solution in the process of selecting the technology of requirements implementing in the application and also can use the resouces and the tools provided by the original approach.

Other uses of the methodology are presented, such as the extension of the approach over multiple meta concern spaces and multiple system spaces. Also, the methodology introduces a different approach of the analysis of the requirements specification from a separation of concerns perspective and the traceability that is conferred to the requirements by this approach.

Acknowledgements: This work was possible with the financial support of the Sectoral Operational Programme for Human Resources Development 2007-2013, co-financed by the European Social Fund, under the project number POSDRU/107/1.5/S/76841 with the title Modern Doctoral Studies: Internationalization and Interdisciplinarity.

References

[1] Lowe, D., Hall, W., *Hypermedia and the Web. An Engineering approach.* John Wiley and Son, USA, 1999.
[2] Microsoft, *Microsoft Application Architecture Guide.* 2nd edition USA, 2009.
[3] Escalona and Koch, *Requirements Engineering for Web Applications A Comparative Study - Journal of Web Engineering.* Vol. 2, No.3, Rinton Press, USA, 2004.
[4] Chris Craig, *The requirements lifecycle.* http://businessanalyst.wikia.com/wiki/the requirements lifecycle USA, 2007.
[5] Werner Heijstek and Michel Chaudron, *Evaluating RUP Software Development Processes Through Visualization of Effort Distribution.* Leiden, Netherlands 2008.
[6] Aked, Mark, *RUP in brief.* IBM, 2003.
[7] Dongyun Liu, Hong Mei, *Mapping requirements to software architecture by feature-orientation.* Beijing China 2009.
[8] Periklis Sochos, Matthias Riebisch, Ilka Philippow, *The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines.* Germany, 2006.
[9] Xin Chen, Zhiming Liu, Vladimir Mencl, *Separation of Concerns and Consistent Integration in Requirements Modelling.* Macao, China, 2007.
[10] Ana Moreira, Awais Rashid, Joo Arajo, *Multi-Dimensional Separation of Concerns in Requirements Engineering.* IEEE, 2005.

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

E-mail address: calin.gal-chis@ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

FORMAL DEFINITION OF FUML IN $\mathbb{K}$-FRAMEWORK

SIMONA MOTOGNA, FLORIN CRĂCIUN, IOAN LĂZAR, BAZIL PÂRV [1]

Abstract. The Alf language was introduced as a simpler, textual definition of fUML executable models. The operational semantics of Alf is defined by mapping the Alf concrete syntax to the abstract syntax of fUML. The operational semantics of fUML is described in a semi-formal way, focusing on its implementation in Java. Our paper addresses two problems regarding this issue: i) semantic integration, namely semantics mappings should be defined using platform independent constructions, and ii) the correctness of the execution engine must be guaranteed. We propose an approach to give a formal definition of Alf in the $\mathbb{K}$-semantic framework. Executable $\mathbb{K}$-definitions will specify a reference virtual machine that can gain access to $\mathbb{K}$'s tools for formal analysis and verification.

1. Introduction

Software systems have recorded a spectacular evolution in the last years: they become more complex every day, and are used in a lot of domains. This evolution puts a lot of pressure on the development cycle of software systems, such that we would like to automate the development process as much as possible. Constructing software automatically from high-level models is one of the challenges in software engineering nowadays.

In this context, models should be easy to built, but should also encapsulate a complete and precise behavior description. An executable model, in addition, has an associated formal action semantics such that the model can be executed an tested in this early stage of development. fUML [15] is the OMG proposal for such an approach.

Received by the editors: April 15, 2013.

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

1998 *CR Categories and Descriptors.* D.2.4 [**Software/Program Verification**]: Subtopic – *Validation*; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Subtopic – *Logics of programs*.

Key words and phrases. fUML, K-Framework, Formal Methods.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

In a visionary article, Harel & Marron [8] argue that the abstraction level provided by models and specifications will make them more expressive and intuitive and the focus in software systems development will shift to model construction, composition and adaptivity. Considering the important role of models in the description of the system's behavior, it is clear that such an approach must assure at least some essential attributes of the system, such as correctness, safety, completeness.

Formal methods represent the mathematical instrument that can be used to assure these attributes, as they offer the appropriate mechanism to detect if a model has errors or how they can be avoided. The main drawback in the use of such methods is the fact that they are difficult to be understood by software engineers.

The main goal of this paper is to propose an approach that will take advantage of the characteristics of formal methods in order to construct a complete semantical definition of Alf and to provide an execution engine for it. The tool that we have chosen is the $\mathbb{K}$-framework proposed in [19], based on its main features. The $\mathbb{K}$-framework provides the necessary constructions to define the Alf semantics and its features can be used for reaching our purpose [20]:

- executability: the definitions are directly executable in order to be experimented with and analysed;
- unique definition: there is only one definition for a language, and several analysis tools that are sound with respect to this definition;
- program logic: the framework serve as a program logic with which the programs can be verified and analysed.

The rest of the paper is organized as follows: the next section gives an short introduction to fUML, Alf and $\mathbb{K}$-framework discussing related work concerning these subjects. Section 3 is dedicated to the presentation of our approach to constructing a virtual machine for fUML, while Section 4 deals with conclusions and future research directions.

2. Background

2.1. **fUML and Alf.** Approaches in which modeling is at the core of the development activities also simplify the component construction process [5]. One of the main component based development's challenge is to provide a general, flexible and extensible model, for both components and software systems. This model should be language-independent, as well as programming-paradigm independent, allowing the reuse at design level. Well-known such approaches are based on UML and MDA.

MDA framework [12] provides an approach for specifying systems independently of a particular platform and for transforming the system specification

into one for a particular platform. The most important benefits are higher abstraction level in program specification and increase of automation in program development. The availability of such tools and the easiness of their use has contributed to the success of MDA. But development processes based on MDA are considered heavy-weight processes since they cannot deliver (incrementally) partial implementations to be executed as soon as possible.

In this context, executing UML models became a necessity for development processes based on extensive modeling. For such processes, models must act just like code, and UML 2 and its Action Semantics [13] provide a foundation to construct executable models. In order to make a model executable, it must contain a complete and precise behavior description. Unfortunately, creating such a model is a tedious task or an impossible one because of many UML semantic variation points. Executable UML [9] means an execution semantics for a subset of actions sufficient for computational completeness. Two basic elements are required for such subsets: an action language and an operational semantics. The action language specifies the elements that can be used while the operational semantics establishes how the elements can be placed in a model, and how the model can be interpreted. Again, creating reasonable sized executable UML models is difficult, because the UML primitives from the UML Action Semantics package are too low level.

The Executable Foundational UML (fUML [15]) is a computationally complete and compact subset of UML, designed to simplify the creation of executable UML models. The semantics of UML operations can be specified as programs written in fUML. 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.

The complete static and operational semantics of fUML is still in its early stages, and although several proposal have been issued, this problem is still open. In our opinion, the difficulties risen in the complete semantical definition of fUML lie in the following three aspects:

- The fUML standard enforces a data flow abstract representation for the behavior of the methods. For example, accessing the values of parameters or variables from certain reserved locations, the values (or the references) of these types of entities will flow as tokens on edges. If a parameter or variable is used in multiple places, its value is copied using a fork node and sent to each action that needs it. To assign a

new value to the entity, a new point that provides the value is created (with a new fork node).

- The number of fUML constructs and the relations between them; most of the research carried in this field concentrate on a subset of fUML, especially on actions and activities, but the integration of all syntactical elements in the formal specification is not an easy task.
- fUML suffers from the same problem as several well-known programming languages, namely they are not designed or analyzed using formal semantics. The current implementation of fUML uses a Java virtual machine, and the correctness of the semantical constructions and their corresponding behavior cannot be guaranteed. In a formal environment, using a mathematical mechanism we can prove different properties regarding model execution.

The language Alf has been adopted as Action Language for fUML in 2010 [14] providing a concrete syntax for describing fUML models. Alf semantics is mapped to fUML abstract syntax metamodel. Alf syntax is inspired from well known programming languages such as C++ and Java, and "acts as the surface notation for specifying executable behaviors within a wider model that is primarily represented using the usual graphical notations of UML" [14]. In this way, creating reasonable sized executable UML models is much easier based on Alf constructs, instead of low level UML primitives.

There has been a lot of reaserch in the field of verification an semantic specification for fUML and Alf, based on diverse mathematical mechanisms. Relevant results have been obtained in using CSP (Communication Sequential Processes) [3, 1, 2], considering a lightweight verification method for strong executability [18, 17], or using Petri Nets [22]. However, all these approaches fail to offer a complete specification, since they impose restrictions on used UML diagrams and notations. The fUML virtual machine is under investigation in [21].

The execution for fUML activities is concurent, the nodes within activities may be executed concurrently according to the control and data flow model defined by the UML specification. Figure 1 (a) shows an fUML activity fragment which computes the value 7. According to fUML, an action may begin execution when it has been offered control tokens on all its incoming control flows and all its input pins have been supplied (via object flows) object tokens sufficient for their multiplicities. Figure 1 (a) does not contain control flows. When this fragment is executed, three control tokens are offered for the value specification actions, because these actions do not have incoming control flows. These actions may be executed concurrently and they provide the values 1, 2, respectively 3 to the other actions. Because the next two call behavior actions

(+) require the object token offered by the fork node a, they will wait until that token will be offered by that fork node (synchronization mechanism). When the fork a offers the object token, the next two + actions may be executed concurrently and their results will be offered (via the fork nodes b and c) to the last + action which will provide the final value.

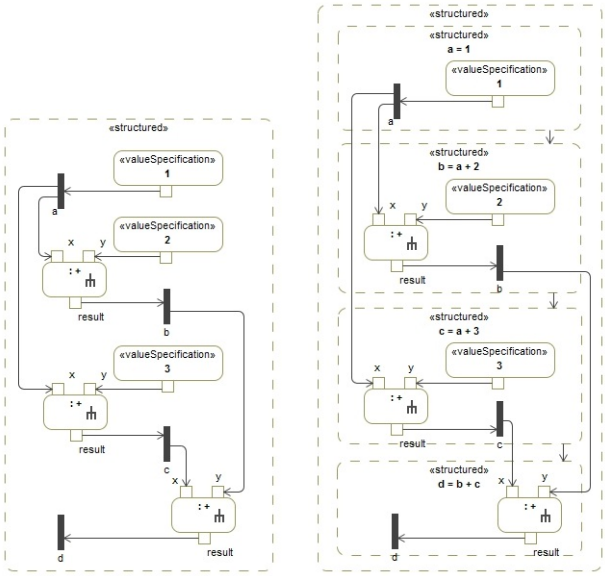


FIGURE 1. (a) fUML activity fragment; (b) Alf to fUML mapped activity fragment.

Using a textual notation, the activity fragment from Figure 1 (a) may be written as shown in Figure 2 (a). Taking into account the concurent execution semantics of fUML, after a is computed, the statements *2* and *3* may be executed in parallel, then d will be computed after the previous executions have been completed. We may notice that the statements presented in Figure 2 (a) are common to the languages that use a sequential model of computation. So, if this textual representation would be compiled into a parallel fUML representation without explicit mechanisms for specifying the concurrency aspects, then it would be difficult to write and control the concurrency aspects.

```
a = 1;        //Statement 1
b = a + 2;    //Statement 2
c = a + 3;    //Statement 3
d = b + c;    //Statement 4
```

```
a = 1;        //Statement 1
//@parallel
{b = a + 2;   //Statement 2
c = a + 3; }  //Statement 3
d = b + c;    //Statement 4
```

FIGURE 2. (a) Block of statements written using a textual notation; (b) Alf textual notation

Alf defines a textual representation inspired from such languages, but also adds features which allows users to write parallel programs using very simple textual constructs mapped to the powerful abstract syntax of fUML. For example, Figure 1 (b) shows the fUML activity corresponding to the textual representation from Figure 2 (a). Because the textual representation contains a sequence of statements (sequential model of computation), all statements are mapped to *structured* activities having control flows between them. But, the control flow introduced between the structured activities corresponding to the statements *2* and *3* will prevent the concurrent execution of these two statements.

In order to indicate parallel execution, Alf provides the *parallel* annotation. Figure 2 (b) explicitly shows that the statements *2* and *3* must be executed in parallel, and after both will be completed, statement *4* will be executed. The compiled fUML model of this textual notation contains a *structured* activity containing the compiled statements *2* and *3*, but without control flows between them.

Alf allows us to use the *parallel* annotation for the block written in Figure 2 (a). In this case, the compiled model would be the model from Figure 1 (b) but without control flows between structured activities. The synchronization between actions follows the control and data flow model. (There are some restrictions for parallel blocks, e.g. a local name used in the lhs of assignments may be part of only one assignment.)

```
a = 1; //Statement 1
//@parallel
{
    b = a + 2; //Statement 2
    c = a + 3; //Statement 3
}
d = b + c; //Statement 4
```

FIGURE 3. Alf textual notation

2.2. **K-framework.** Developed as a collaborative effort between several research groups, $\mathbb{K}$ is a rewrite-based framework supporting definition and execution of programming languages. The $\mathbb{K}$-semantics can be executed and tested, and the underlying matching logic and language Maude can be used for program analysis and verification.

A $\mathbb{K}$ definition consists of configurations, computations and rules. Several notable results, such as the formalization of C [6], Scheme [10], and Verilog [11], type checking [7] and symbolic execution [4] have contributed to the confidence in the $\mathbb{K}$-framework capabilities. Recent approaches [20] suggest that $\mathbb{K}$ may be a suitable tool for formalization and analysis of fUML and Alf.

3. Virtual Machine for fUML

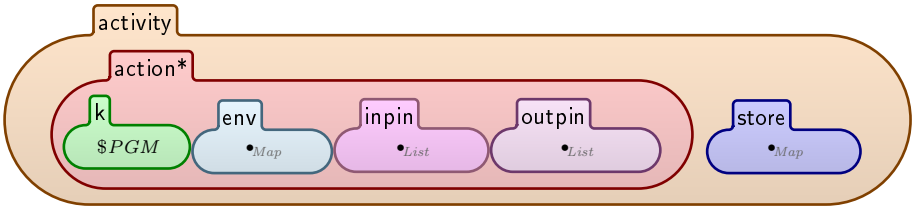
In this section we illustrate our formalization of fUML in K-framework. The fUML standard includes class and activity diagrams to describe a system's structure and behaviour respectively. In this paper we mainly focus on the formalization of the activity diagrams. A formalization of class diagrams has already been given in [20] and can be later integrated with our current approach.

We start with the syntax of ALF simple expressions which are going to be mapped to actions nodes in fUML.

```
MODULE ALF-SYNTAX
  SYNTAX  AExp ::= Id
                 | Int
                 | AExp + AExp
                 | AExp * AExp
  SYNTAX  Ids ::= List{Id,","}
  SYNTAX  Stmt ::= Id = AExp
                 | Stmt ; Stmt
  SYNTAX  AExps ::= List{AExp,","}
END MODULE
```

In order to define the K-semantics of fUML we introduce the main K-configuration. A configuration consists of a pool of actions ($cell_{action^*}$) which forms an activity ($cell_{activity}$). Each action has a list of input ($cell_{inpin}$) and output pins ($cell_{outpin}$) and the program that is going to be executed ($cell_K$). Input and output pins are named. Action pins names are mapped to global names through the action environment ($cell_{env}$). The global names are kept in an activity store ($cell_{store}$). The activity store allows the actions to communicate in a asynchronous way. An action can read an input pin only if its name is not mapped to an undefined value ($\perp$). After the action execution is completed the action output pins are mapped to values. For the moment we do not make a distinction between data and control pins (since they can have the same formalization). The action nodes are executed concurrently.

CONFIGURATION:



ALF program is stored in one of the action nodes and is translated into one or more fUML action nodes. Note that ALF action nodes and fUML action nodes are treated in a similar way and their execution runs in parallel.

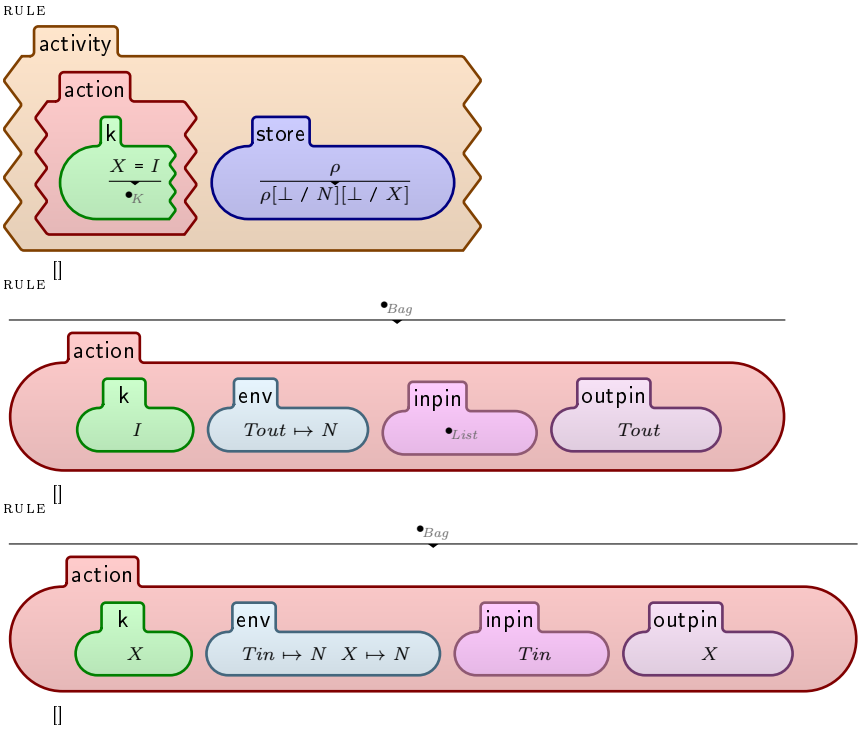
The execution of an ALF action node generates a fUML activity diagram (a pool of interconnected fUML action nodes), while the execution of the fUML action nodes do the *model execution* namely the propagation of object flow and control flow through the fUML model.

We adopted a small-step operational semantics for the translation of ALF to fUML. Therefore we introduced the following intermediate operations to be executed inside a fUML action node (in $cell_K$): *read* to allow the fUML action to wait for the input pins (data and control input flow) to have valid values; *write* and *writeVar* to propagate the values on the output pins (data and control output flow); *lookup* to find the value assigned to a global name which usually denotes a variable name or an input pin name; and *callBehaviour* to invoke a fUML behaviour (For simplicity in this paper we consider only the simple arithmetic operations to illustrate the concepts of our formalization). The undefined value $\perp$ is used to model the fUML diagram edges on which data or control flow has not been propagated yet. Thus the activity global names (corresponding to fUML activity edges) are mapped to $\perp$ in the activity store.

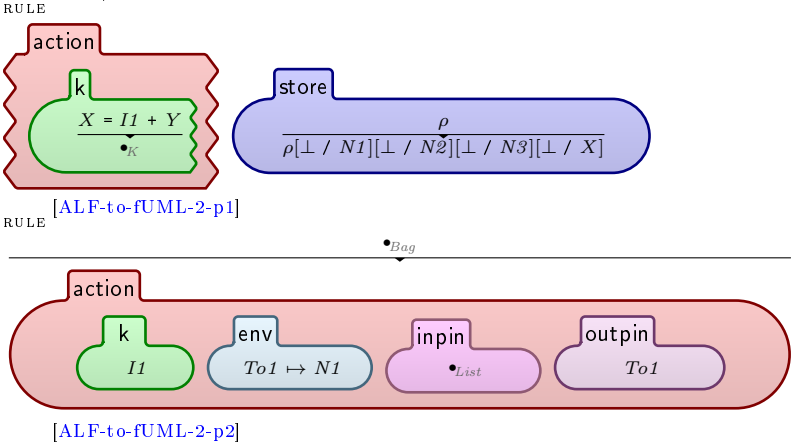
SYNTAX K ::= `read` (*Ids*)
| `write` (*Int*, *Ids*)
| `writeVar` (*Id*)
| `callBehaviour` (+, *Id*, *Id*)
| `lookup` (*Id*)
| $\perp$

The following rules illustrate the translation of ALF action nodes into fUML action nodes. The graphical presentation of the rules is directly generated by $\mathbb{K}$-framework compiler. Note that in this paper we split a rule on many lines due to the page width limitation such that the notations *p1...pn* denote the components of a splitted rule.

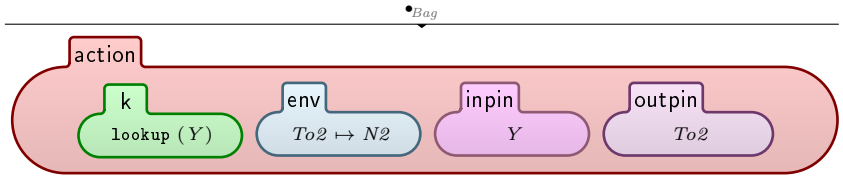
The first rule generates two new fUML actions from an ALF assignment $X = I$ (where X is a variable name and I is a constant) as follows: a value specification action (*ALF-to-fUML-1-p2*) and a variable action (*ALF-to-fUML-1-p3*). Note that new fUML action nodes are added to the existing action nodes in that activity. The rule assumes that inside the activity exists an ALF action node (*ALF-to-fUML-1-p1*) that has that assignment as the current instruction in its $cell_K$. The rule consumes that assignment of the ALF action node and let the ALF action $cell_K$ to continue with the next instruction. The fUML actions input and output pins are kept separately and are linked together through the local action environments and global activity store. Those two new generated fUML actions and ALF initial action are executed concurrently. fUML actions communicate in an asynchronous manner through the store variable N.



The following rule generates four new fUML action nodes from an ALF assignment $X = I1 + Y$ (where X is a variable name and $I1$ is a constant) as follows: a value specification action, a waiting actions (lookup for the variables valid values), one call behaviour action and a variable action.

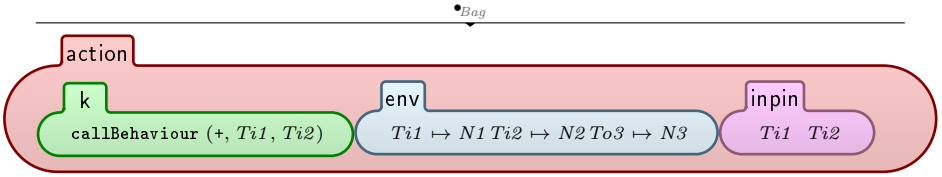


RULE



[ALF-to-fUML-2-p3]

RULE



[ALF-to-fUML-2-p4]

A fUML action node is destroyed when its executable code is completed (namely $\mathbb{K}$-cell is empty). We also provide rules for simplification of ALF expressions like the following:

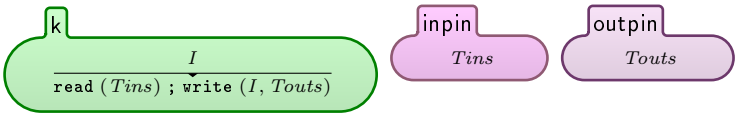
RULE

$$\frac{X = E1 + E2}{Y = E1 \ ; \ X = Y + E2}$$

when $E1 =/=_{\mathtt{K}} Id \wedge_{Bool} E1 =/=_{\mathtt{K}} Int$

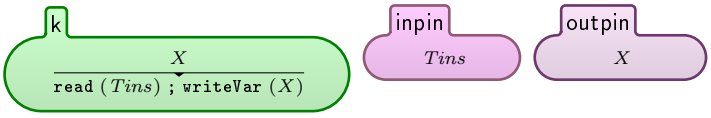
Next rules define the execution of the fUML node actions: value specification, variable and call behaviour. Each rule first waits for the valid values on its input pins and then writes an appropriate value on its output pins.

RULE



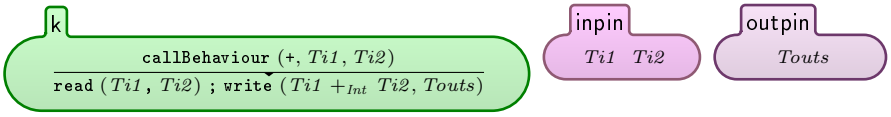
[fUML-Action-ValueSpec]

RULE



[fUML-Action-Var]

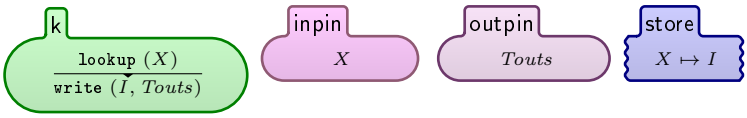
RULE



[fUML-Action-CallBehaviour]

The rule for *lookup* is waiting for a global variable to have a valid value in the store. When the assigned value is valid, that value is written on the output pins.

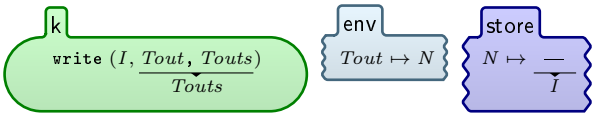
RULE



when $I =/=_{Int} \perp$
[fUML-Action-Lookup]

The next rules are writing appropriate valid values on the output pins.

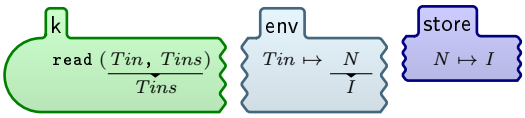
RULE



[fUML-Action-Set-Outpins]

The rules for *read* guarantee that all input pins have received valid values.

RULE



when $I =/=_{Int} \perp$
[fUML-Action-Wait-Inpins]

4. Conclusions and Future Work

We propose a novel formalization of ALF and fUML in $\mathbb{K}$-framework. The $\mathbb{K}$-framework allows us to directly define a concurrent semantics and to execute fUML models conform to OMG specification [15].

The main contributions of this paper are: i) the definition of $\mathbb{K}$-configurations corresponding to Alf syntax (fragment for simple arithmetic expressions) and ii) the specification of the $\mathbb{K}$-rules that simultaneously transform Alf artifacts into fUML constructs and execute them.

This paper is to be consider a proof of concepts for an approach of defining a virtual machine for Alf and fUML in the $\mathbb{K}$-framework, virtual machine that will be based on the executable $\mathbb{K}$-rules.

In contrast to the existing fUML virtual machine implemented in Java our approach is declarative and allows us to directly support a higher degree of genericity (specified by explicit semantics variation points in OMG specification [15]).

The main benefit consists in a higher level of platform and language independence, since the approach will not be based on a Java virtual machine, but on a more formal definition.

Another advantage is the extensibility. It is well known that Alf syntax, and implicitly its applicability, are quite restrictive [16]. Our approach may offer the necessary instrument to ease the extension of Alf for specific constructs.

Our future investigations will concentrate on a complete definition for Alf and fUML in $\mathbb{K}$. Next important step is to integrate class diagrams and OCL constraints [20] in our current proposal. We also plan to use the $\mathbb{K}$-framework tools to perform different forms of analysis and verification (e.g. inconsistency, deadlock free like in [3, 1]) on our $\mathbb{K}$-rules.

References

[1] Islam Abdelhalim, Steve Schneider & Helen Treharne (2011): *Towards a Practical Approach to Check UML/fUML Models Consistency Using CSP*. In: *ICFEM*, pp. 33–48. Available at `http://dx.doi.org/10.1007/978-3-642-24559-6_5`.

[2] Islam Abdelhalim, Steve Schneider & Helen Treharne (2012): *An Optimization Approach for Effective Formalized fUML Model Checking*. In: *SEFM*, pp. 248–262. Available at `http://dx.doi.org/10.1007/978-3-642-33826-7_17`.

[3] Islam Abdelhalim, James Sharp, Steve A. Schneider & Helen Treharne (2010): *Formal Verification of Tokeneer Behaviours Modelled in fUML Using CSP*. In: *ICFEM*, pp. 371–387. Available at `http://dx.doi.org/10.1007/978-3-642-16901-4_25`.

[4] Irina Mariuca Asavoae, Mihail Asavoae & Dorel Lucanu (2010): *Path Directed Symbolic Execution in the K Framework*. In: *SYNASC*, IEEE Computer Society, pp. 133–141. Available at `http://synasc10.info.uvt.ro/[SYNASC]`.

[5] Krishnakumar Balasubramanian, Aniruddha S. Gokhale, Gabor Karsai, Janos Sztipanovits & Sandeep Neema (2006): *Developing Applications Using Model-Driven Design Environments*. *IEEE Computer* 39(2), pp. 33–40. Available at `http://doi.ieeecomputersociety.org/10.1109/MC.2006.54`.

[6] Chucky Ellison & Grigore Roşu (2012): *An Executable Formal Semantics of C with Applications*. In: *Proceedings of the 39th Symposium on Principles of Programming Languages (POPL'12)*, ACM, pp. 533–544, doi:`10.1145/2103656.2103719`.

[7] Chucky Ellison, Traian Florin Şerbănuţă & Grigore Roşu (2009): *A Rewriting Logic Approach to Type Inference*. In: *Recent Trends in Algebraic Development Techniques*, *Lecture Notes in Computer Science* 5486, Springer, pp. 135–151, doi:`doi:10.1007/978-3-642-03429-9`. Available at `http://dx.doi.org/10.1007/978-3-642-03429-9`. Revised Selected Papers from the 19th International Workshop on Algebraic Development Techniques (WADT'08).

[8] David Harel & Assaf Marron (2012): *The quest for runware: on compositional, executable and intuitive models*. *Software and System Modeling* 11(4), pp. 599–608. Available at `http://dx.doi.org/10.1007/s10270-012-0258-8`.

[9] Stephen J. Mellor & Marc Balcer (2002): *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[10] Patrick Meredith, Mark Hills & Grigore Roşu (2007): *A K Definition of Scheme*. Technical Report Department of Computer Science UIUCDCS-R-2007-2907, University of Illinois at Urbana-Champaign.

[11] Patrick O'Neil Meredith, Michael Katelman, José Meseguer & Grigore Roşu (2010): *A Formal Executable Semantics of Verilog.* In: *Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'10)*, IEEE, pp. 179–188, doi:doi:10.1109/MEMCOD.2010.555863.

[12] OMG (2003): *MDA Guide Version 1.0.1.* . Available at http://www.enterprise-architecture.info/Images/MDA/MDA20Guide20v1-0-1.pdf.

[13] OMG (2009): *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure Version 2.2.* Available at http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/.

[14] OMG (2010): *Concrete Syntax for UML Action Language (Action Language for Foundational UML - ALF).* Available at http://www.omg.org/spec/ALF/1.0/Beta1.

[15] OMG (2011): *Semantics of a Foundational Subset for Executable UML Models (FUML).* Available at http://www.omg.org/spec/FUML/Current.

[16] Isabelle Perseil (2011): *ALF formal. ISSE* 7(4), pp. 325–326. Available at http://dx.doi.org/10.1007/s11334-011-0168-x.

[17] Elena Planas, Jordi Cabot & Cristina Gómez (2011): *Lightweight Verification of Executable Models.* In: *ER*, pp. 467–475. Available at http://dx.doi.org/10.1007/978-3-642-24606-7_37.

[18] Elena Planas, David Sanchez-Mendoza, Jordi Cabot & Cristina Gómez (2012): *Alf-Verifier: An Eclipse Plugin for Verifying Alf/UML Executable Models.* In: *ER Workshops*, pp. 378–382.

[19] Grigore Roşu & Traian Florin Şerbănuţă (2010): *An Overview of the K Semantic Framework. Journal of Logic and Algebraic Programming* 79(6), pp. 397–434, doi:10.1016/j.jlap.2010.03.012. Available at http://dx.doi.org/10.1016/j.jlap.2010.03.012.

[20] Vlad Rusu & Dorel Lucanu (2011): *A K-Based Formal Framework for Domain-Specific Modelling Languages.* In: *FoVeOOS*, pp. 214–231. Available at http://dx.doi.org/10.1007/978-3-642-31762-0_14.

[21] P. Langer T. Mayerhofer & G. Kappel (2012): *A Runtime Model for fUML.* Available at http://publik.tuwien.ac.at/files/PubDat_210111.pdf.

[22] Yann Thierry-Mieg & Lom-Messan Hillah (2008): *UML behavioral consistency checking using instantiable Petri nets. ISSE* 4(3), pp. 293–300. Available at http://dx.doi.org/10.1007/s11334-008-0065-0.

(1) Babeş Bolyai University
E-mail address: {motogna,craciunf,ilazar,bparv}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

A DECORATOR BASED DESIGN FOR COLLECTIONS

V. NICULESCU[(1)] AND D. LUPSA[(1)]

Abstract. We propose in this paper a design for a framework dedicated to collections data structures, based on which we are able very easily to use, to adapt, to transform, and to extend the collections. A certain collection type is seen as a set of features which are added to a storage support. The design is based on Decorator together with Proxy and Template Method design patterns. This design choice allows features to be dynamically added or removed and from this, a high degree of flexibility in creating and managing the collections is achieved. The framework could be easily extended, but in an organized and reliable manner.

1. Introduction

In a previous paper [10] we have analysed the general requirements for a framework dedicated to collections data structures, based on an analysis of the related work. We propose in this paper a framework for collections data structures, in which we are able, very easily to use, to adapt, to transform, and to extend the collections. The design of the framework relies on defining collections using features, and on a design infrastructure based on *Decorator* design pattern together with others such as *Proxy* and *Template Method* [5]. A certain collection type is seen as a set of features that are added to a storage support. These features could be dynamically added or removed and this leads to a high degree of flexibility in creating and managing the collections.

Received by the editors: May 9, 2013.

2010 *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, collections framework, design patterns, genericity, representation.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

2. Theoretical approach

The problems that could be emphasized related to the formal and accurate defined types that characterize the data structures are related to the fact that universal and overall accepted definitions could not be found. In the literature there are different classification and definitions for the types corresponding to different containers. And because of this, the existing implemented solutions - frameworks - are also very different [10, 8].

We may start from a general definition:

Definition 1 (Collection)**.** *A* ***collection*** *– sometimes called a container –is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.*

There are two general and important aspects related to collections [9]:

(1) *storage capability* – the elements that are grouped together have to be stored into the memory in an accessible way; usually the term *container* reflects more this aspect;
(2) *specific behavior* – the operations that are allowed for a specific type of container have different specifications; usually the term *collection* is chosen to emphasize this aspect.

The first aspect is directly connected to the data structure used for storing the elements. For storage, we may use a continuous block of memory or a set of discontinues blocks of memory (nodes) connected one to another using links (references). A linked representation may have different structures: linear, tree-like, or others.

The set of operations that could be applied to a container may be different, but also their specification may be different from one collection type to another. In order to emphasize these differences from behavior point of view, we may identify a set of features that could be applied to a container. So, our approach is not based on abstract data types, but on specific behaviors defined with features.

Definition 2 (Feature)**.** *We consider a* **feature** *as being a distinctive property that characterizes the behavior of a collection – an operation or a set of operations with defined arguments, together with their semantic expressed by a clear specification. It is something that fundamentally characterizes the collection behavior.*

2.1. Storage capability.

Each collection has to be stored in the memory, in a way that allows elements to be added, removed, and retrieved. The storage capability of a collection could be considered as a basic, compulsory, implicit feature. It is an implicit feature that characterizes any collection; the set of operations of this basic feature could be seen in Figure 1.

FIGURE 1. `IStorage` interface.

The formal specification that characterizes the storage capability could be given using Hoare style specifications.

- **-:** The postcondition of the method `add(e:Element)` assures just the fact that the element e is in the storage;
- **-:** The postcondition of the method `remove(e:Element)` assures that one instance of the element e has been removed from the storage if such an instance exists;
- **-:** The number of the elements in the storage is returned by the method `size()`, and we may obtain an reading iterator over the elements of the storage using the method `iterator()`.
- **-:** The methods `copyFrom()` and `copyTo()` are import/ export operations; they allow the elements of an entire other container to be added into, and also to insert all the elements into another container.

Instead of considering *Iterable* as an independent feature we have considered the existence of an iterator on the storage as being implicit. The reason of this decision is based on the fact that many features could be easily added and implemented based on iterators. If the iterability is implicitly considered then the correctness of other features definitions is assured easier. Also, there are implicit implementations of certain operations based on iterators.

In conclusion, we have considered that:

- memory representation,
- iterability, and
- searchability

are implicit properties of each collection type of our framework.

Our decision to consider all of them as basic properties is based on the fact that in the proposed framework the first level data structures (which are used for storage) are not created as a combination of their basic properties.

Two important memory representation categories have to be considered:

- **-:** block representation,
- **-:** linked representation.

The block representation means that a single continuous block of memory cells is used for storing the elements. There are no many options for achieving this: simple array implementation is the main choice.

A linked representation means that we may use memory locations at different addresses – nodes, and the elements could be retrieved based on using link information between these nodes. Examples of this category are the linked list and the linked trees.

2.2. **Specialized behavior – specialized containers.** Starting from a concrete storage structure we may create different collection types, by adding different behaviors.

Definition 3 (Behavior)**.** *A* behavior *is defined as being formed of a combination of a set of basic features.*

In Table 1 the considered features are presented.

Generally, a set is characterized only by the fact there are no duplicate elements in the container. The feature `unique` defines the operation `add` with the same argument list as in the basic storage type, but changes the postcondition of the operation, by assuring the fact that the argument is added only if its value is not yet present in the container. How these elements are stored, is not a fact that characterizes the set.

`Searchable` feature certifies the fact that there is an operation for searching an element in the container with a time-complexity less than $O(n)$, which is the time-complexity of the implicit searching operation. Sorted arrays or the binary searching trees are example of searchable containers.

`Ranked` is a feature that specifies an added behavior that allows the access to the elements based on their rank. A rank of an element in a collection is equal to the rank of it in the traversal executed by the implicit iterator. And so, this could be added not only to sequences.

`FlagDeletion` allows logical deletion of the elements. This means that an element is not really removed from such a collection, but it is just logically marked for deletion. A `purge` operation will do the real deletion of all marked elements.

A container could have the `DeepOwnership` over the elements that it collects. From the implementing point of view this means that when an element is added into, a copy of it is created and this copy is stored, and when an element is deleted, it is destroyed, too.

`Sequence` assures the fact that the elements are in a particular linear order; so there are first and last elements, and each elements in between have a previous and a succesive element.

Stacks and queues specify specific behaviors, and because of that, they could be seen as features.

If the elements that are stored form pairs $(key, value)$ this means that we have an *associative* container. There are several variants to achieve this.

`Synchronized` feature assures the fact that the container could be used in multiprogramming, by several threads of execution.

Many other features could be defined, and this represents the main modality of extending the framework.

2.3. **Features classification.** The features could be classified depending on how they change the behavior of the container:

(1) features that preserve the default container operations, but changes their specifications; ex. `Unique`;
(2) features that add new operations; ex. `Ranked, Sequence`;
(3) features that restrain the set of operations; ex. `UnmodifiableStorage`;
(4) features that restrain the implicit set of operations, but add some other new operations; ex. `Stack, Queue` – they eliminate `remove`, and introduce `extract()`;

Features like `Stack, Queues` or `PriorityQueue` have all in common the fact that they use a special rule (LIFO, FIFO, etc.) in order to extract the elements from their storage. So, they are specializations of a more general feature `RuleBasedExtraction`.

As we have mentioned before one goal of the framework is to allow features to be added and removed dynamically. Still we may identify some restrictions; for example there are features, which could not be added after we have already added some elements into the support container. All these features are specializations of `EmptyStorage` feature.

Generally, between features we may establish specialization/generalization relationships.

In order to create new kind of containers a linear combination of features can be used. We consider that each feature is wrapped around the previous feature, or storage (storages could be seen as basic features).

Some of these features are *symmetric* – could be combined in any order without changing the result. Examples of this type are: `Unique` and `DeepOwnership`.

The features that add new operations are not symmetric with the rest of the features. `Stack`, for example, add the operation `extract()` that allows the

elements to be extracted based on the LIFO rule; it should be the last added feature in order to allow this operation to be accessible.

Because of these, we introduce levels for all features, based on which we will impose an order to combine the features. Table 1 presents the levels, the features of each level, and the existence or not of the symmetry property of each level.

Level	*Features*	*Symmetry*
4	`Ranked, Stack, Queue, PriorityQueue, Map, DMap, OMap`	no
3	`Synchronized,Unmodifiable`	no
2	`Unique, FlagDeletion, DeepOwnership, Searchable`	yes
1	`Sequence, SortedSequence, Heap, BSTree, Hashing`	no
0	all the storages types	no

TABLE 1. Features and their level based classification.

A level is *symmetric* iff all features defined inside it, could be added in a symmetric way. This means that we can add as many features we want of that level, in any order. From a *non-symmetric* level we may add only one feature; there is mutual exclusion between the features of such a level.

The feature `Unmodifiable` is applied when we want to use an existing collection only for storing and searching. We add this feature to assure that the collection state will not be changed; these kinds of collections do not need synchronization. So, the features `Unmodifiable` and `Synchronized` could belong to the same unsymmetric level.
In order to assure a proper synchronization of all the collection operations, we have considered the `Synchronized` feature in a level as high as possible. For example, `Synchronized` feature should be added after `Unique` feature, since if the verification of the existence of a value in the container would not have done in a synchronized manner then the result will probably not be correct.
Still, the features that change the basic `IStorage` interface should be on the highest level.

Since we have introduced levels for each feature, we may formally define the notion of *a well defined collection*:

Definition 4 (Well-defined collection)**.** *If the collection C is defined as*

$$C = F_1 \circ F_2 \circ \cdots \circ F_n \circ S$$

where F_i are features, and S is a storage instance, then the collection C is **well-defined** *if:*

- level condition*:*

$$level(F_1) \geq level(F_2) \geq \cdots \geq level(F_n)$$

and

- mutual exclusion condition*:*

$$\forall\ level\ l \neq 2;\quad \exists!\ F_i (0 < i \leq n)\ such\ that\ level(F_i) = l$$

We may consider few concrete examples:

- `Searchable` ∘ `Unique` ∘ `SortedSequence` ∘ `Array` represents a searchable sorted set stored into a storage of block memory representation type – `Array`; the collection is a well-defined collection since $level($`Searchable`$) = level($`Unique`$) = 2$; $level($`SortedSequence`$) = 1$; and the level 2 is symmetric.
- `Ranked` ∘ `Unique` ∘ `DeepOwnership` ∘ `LinkedList` represents a ranked set that is the owner of its elements, and a `LinkedList` is used for storage; the collection is a well-defined collection since $level($`Ranked`$) = 4$; $level($`Unique`$) = level($`DeepOwnership`$) = 2$; and the level 2 is symmetric.

3. Framework Design

The main idea and advantage of the framework is the following:

Anytime a feature could be added to a collection data structure and then could be removed.

Decorator design pattern [5] fits very well to this way of creating new types of containers, and this is why we have chosen to used it in framework implementation. The *Decorator* pattern is combined with *Proxy* pattern [5], since the decorations could be easily implemented by adding prefix and suffix operations that precede and succeed the initial operations. Also, *Template Method* pattern [5] is useful for implementing and using these operations.

Generally, Decorator pattern allows responsibilities to be added to an object by modifying the existing methods, not by adding methods to the object's interface. This means that in a classical usage of the pattern, the interface presented to the client must remain constant as successive layers are specified. This corresponds to symmetric features. For the highest level features the same infrastructure is used, but the new interfaces of these features are accessible if they are added as the last decoration, but also they take the benefits of the lower levels decorations.

The symmetric features modify the specifications of the basic feature operations. Prefix proxy operations could modify the preconditions, and suffix proxy operations could modify the postconditions.

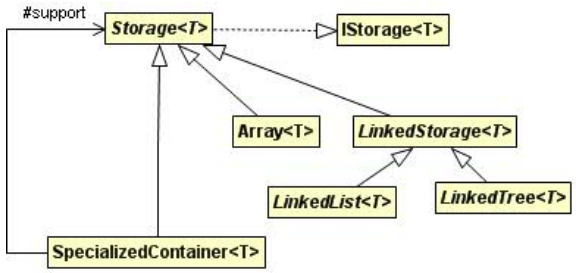


FIGURE 2. Decorator design for defining specialized collections.

A decoration that correspond to a feature that restrains the basic features implements prefix proxy operations that block the execution of the operations that have to be excluded.

3.1. **Specialized Containers.** The root decorator class is `SpecializedContainer<T>` that extends `Storage<T>`, but which also uses a storage of type `Storage<T>`. Each feature will be introduced as a decoration of the storage (Figure 2).

This class defines template methods for each independent methods of the class `Storage<T>`. These methods call the *proxy methods* that precede and succeed the calls of the `Storage<T>` methods. We have:

- `prev_add` and `post_add`,
- `prev_remove` and `post_remove`,
- `prev_search` and `post_search`,
- `prev_it` and `post_it`.

The postcondition of the method `add` assures the fact that if the element has been added, then a not null reference on the added element is return; usually this reference is of iterator type.

The proxy methods for adding are used also for other similar operations such as `insert` and `set` of iterators.

When we define a container with several decorations, the proxy methods of each decoration is called in a chain. The following example illustrates these calls.

Example[Proxy methods]

```
Storage<Integer> s =
            new Deco1<Integer>(new Deco2<Integer>(new Array<Integer>()));
s.add(4);
```

First, the method `s.add(4)` calls the method `prev_add` defined in `Deco1`, then the method `prev_add` from `Deco2` is called, and then (if it is the case) the

element is included using the method `add` from `Array` class. After this, the methods `post_add` from `Deco2` and `Deco1` are called in this order.

The proxy methods are very important for operations specializations. The following examples emphasizes their use.

- : In order to define a set we have to assure the fact that no duplicates are included into the container. `Unique` defines a decoration that assures this fact. This could be easily implemented using these proxy methods, more precisely by defining the method `prev_add` in such a way that if the element is already into the container, the add operation of the storage support is not longer called.
- : Another example is for `DeepOwnership` feature that creates a copy of the element to be added, inside the own method `prev_add`. In the method `post_remove` the reference to the copy of the element could be set to null. If the framework would be ported into a language as C++, here the destructor could be explicitly called.
- : In order to define synchronous access to a container we use (`Synchronized` feature). The same proxy methods are used: a `prev` method locks the storage container, and a `post` method releases it.

A container could be modified not only directly by sending corresponding methods to it, but also through an iterator built over it. Because of this the proxy methods have to be used by the iterator operations, too.

This design of the framework allows a very flexible and dynamic adaptation: one decoration could be added and used for a while, and then could be dynamically removed.

4. Related Work

There are many others collections frameworks as well. We have analyzed in [10] some of them [4, 6, 12, 13, 14, 15, 3], and emphasies the different approaches and some general requirements.

Another related approach is connected with feature-based programming and generative programming [1, 2]. Generative programming has important advantages such as: static composition and adaptation, which lead to efficiency, and external and internal adaptations, which leads to flexibility. Still, there are also important drawbacks; such as using as composition operator only the parameterization of specific language constructs (types, classes, functions) , but the most important is the lack of dynamic composition.

Our approach embraced the same idea of using features, but we have considered only behavior features. We have imposed a delimitation of the storage aspects of a container from the behavioral aspects. Another important difference is at the design level, and it is given by the *Decorator* pattern (and

the other connected design patterns). Based on this, the features could be added and removed dynamically.

5. Conclusions and Further Work

Decorator pattern has been used in order to allow the creation of a new collection based on dynamic composition of the features that characterize the corresponding data structure. In this way we may add or remove features dynamically. Also, *Proxy* design pattern could help us for defining specialized operation based on their basic variants.

The fact that only linear combinations of features are allowed could be seen as a disadvantage since the features with changed interface are visible only if they are finally added. Still, when working with a collection at one moment only one such feature is used. Because we allow features to be added and remove dynamically, linear combination is not longer a disadvantage.

Scalability is an important issue related to collections libraries, or framework. There are so many categories of data structures with so many variants, such that a classical approach would lead to an enormous number of classes to cover all. Our approach masters the incrementation of the necessary number of classes.

This approach has been directed by the reason of creating a framework easy to use and extend. The design of the framework induces an order in using the collections but also in developing new extensions. This is achieved while the scalability is preserved.

As a further work we intend to develop a concrete implementation of the proposed design and analyse the achieved usability, flexibility and performance of the resulted framework.

References

[1] D. Batory,B.J. Geraci:*Composition Validation and Subjectivity in GenVoca Generators.* IEEE Trans. Software Engineering'97.

[2] K. Czarnecki, U. Eisenecker: *Generative Programming.* Addison Wesley, 2000.

[3] J. Bloch: *The Java Tutorial. Trail: Collections* `http://docs.oracle.com/javase/tutorial/collections/`.

[4] M. Evered, G. Menger, J. L. Keedy, A. Schmolitzky: *A Useable Collection Framework for Java*, 16th IASTED Intl. Conf. on Applied Informatics, Garmisch Partenkirchen, 1998.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1994.

[6] D.R. Musser, A. Scine:*STL Tutorial and Reference Guide: C++ Programming with Standard Template Library*, Addison-Wesley, 1995.

[7] V. Niculescu: *A Uniform Analysis of Lists Based on a General Non-recursive Definition.* Studia Universitatis "Babeş-Bolyai", Informatica, Vol. LI, No. 1 pp. 91-98 (2006).

[8] V. Niculescu, G. Czibula: *Fundamental Data Structures and Algorithms. An Object-Oriented Perspective*, Casa Cărţii de Ştiinţă, 2011 (in Romanian).

[9] V. Niculescu,: *Storage Independence in Data Structures Implementation*, Studia Universitatis "Babeş-Bolyai", Informatica, Special Issue, LVI(3), pp. 21-26, 2011.
[10] V. Niculescu, D. Lupsa, R. Lupsa: *Issues in Collections Framework Design*. Studia Universitatis "Babeş-Bolyai", Informatica,Vol. LVII, No. 4 (Dec. 2012), pp. 30-38.
[11] C. Szypersky, S. Omohundro, S. Murer: *Engineering a Programming Language: The Type and Class System of Sather*, in Programming Languages and System Architectures, ed. J. Gutknecht, Springer-Verlag, pp. 208-227, 1993.
[12] P. Sestoft , N. Kokholm: *The C5 Generic Collection Library for C# and CLI* `http://www.itu.dk/research/c5/`
[13] *Fastutil: Fast & compact type-specific collections for Java*, `http://fastutil.dsi.unimi.it/`
[14] *Guava project*,`https://code.google.com/p/guava-libraries/`
[15] *YACL - Yet Another Collections Library*,`http://sourceforge.net/projects/zedlib`

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

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

COMPARATIVE STUDY OF TASK DELEGATION MODELS IN SOFTWARE AS A SERVICE PROJECT MANAGEMENT APPLICATIONS

BOGDAN POP[(1)] AND FLORIAN BOIAN[(1)]

Abstract. Task delegation and resource allocation are two of the most important aspects of project management. Bad judgement and errors with regard to task delegation can result in loss of time, resources and a lack of successful project outcomes. Most of the currently available project management applications, no matter what their platform or distribution model is, offer a wide range of tools to ease task delegation. However, none of them have successful, automated task delegation mechanisms, although an automated process would help by reducing losses caused by human error or poor decisions, thus improving overall project results. This paper presents a comparative study between commercial, publicly available project management applications and a proposed application that automates task delegation and showcases the benefits found by using an automated task delegation process.

1. Introduction

Project management is a complex activity that requires proper application of skills, knowledge, different tools and techniques in order to reach or surpass project requirements. Project management consists of five main process groups: 1) the initiation of the project, 2) it's planning, 3) the execution of the project, 4) the proactive management of the project and 5) it's closing. Successful projects can be defined in many ways, mainly because project success can be measured based on a number of factors. Such a complex activity, usually managed by humans, is easily subjected to errors and losses. A study

Received by the editors: April 5, 2013.

2010 *Mathematics Subject Classification.* 90B50, 90B90.

1998 *CR Categories and Descriptors.* K.6.1 [**Management of Computing and Information Systems**]: Project and People Management – *Life Cycle*.

Key words and phrases. project management, tasks, task delegation.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

in 2005 [7] shows that 27% of a manager's time and more than USD 100 billion is spent each year to counteract the effects caused by improper project management, especially problems created by workers that are not suited for their tasks.

The paper is structured as follows: section 2 briefly describes project management and the different methods and factors one can use to measure a project's success, section 3 presents the proposed real-life project that is used within the proposed applications. The 4^{th} section shows the results of the study performed and whether the proposed model is successful or not, while the 5^{th} section presents possible future studies or tests that may be performed and future developments.

2. Background: Project Management and Project's Success

Before presenting the real-life study and its results, a simple classification of project types, how their success or failure is measured and how one can tackle the management of the project via different techniques is required. This allows for a better understanding of the actual study and the performance of each of the tested applications. Most project management applications are tailored towards a specific domain or field and few of them are geared towards project management in a general sense.

A project, disregarding its scope, can be viewed as a series of tasks that have start and end dates, sometimes even times, budget limits, and a specific objective. Usually, this objective must be met while keeping the work within certain specifications and constraints. The tasks that compose a project require resources, both human and non-human [3].

Projects can be in-house and be developed within the company. Others can be contracted projects where a business to business relationship is formed between the project owner and the developer. Projects can be subcontracted where the whole project or a part of it is sold for development outside the company. In this case, the seller may be a contractor as well, not necessarily the project owner. Larger projects are consortium-based, as multiple companies or organisations are forming a joint venture to have the tasks completed with well or ill defined responsibilities.

Project management is the complex set of activities performed by an individual or a group of individuals that requires proper application of skills, knowledge, tools and techniques in order to reach or surpass project requirements.

Project management usually includes the following [1]:

- Identifying the requirements and objectives of the project

- Proactively addressing the concerns and expectations of the project owners / stakeholders since the project is started and even after it is completed
- Balancing of the project constraints:
  - Scope
  - Budget
  - Schedule
  - Resources
  - Quality
  - Risks

Project management is a proactive activity since many factors can change at any given point in time and countermeasures must be performed to preserve the scope and the end goal of the project.

Sometimes it is even the case that a simple change in one of the important factors or constraints determines a chain reaction modifying a lot of other variables in the project.

Project owners or stakeholders can each have a different grasp on the most important factors creating added pressure on the project manager(s) and the workers. Changing the terms and environment of the project can also add additional risks and the development team must be able to asses the situation quickly and make the proper adjustments in order to deliver the project successfully.

Proactive management involves continuously improving and detailing a plan of action as more detailed and specific information and accurate estimations become available during the project's lifecycle. This allows a project management team to manage to a greater level of detail as the project evolves [1].

There are many ways to measure a project's success. Such an assessment is usually made based on the most important factors of the project and its desired outcome. One of the generally accepted set of measurable objectives that are taken into consideration are showcased in Table 1. Some potential benefits of proper project management are shown in Table 2 [3].

2.1. **Project Management Applications.** This subsection briefly describes the most popular project management applications available on the market as of March, 2013. The popularity is measured by the number of users each application has and the user-base growth over the past 12 months. Data was collected from the official website of each service provider, or by manual inquiries, if not otherwise specified. Based on data collected and shown in Table 3, one can infer that proprietary web based project management applications are more popular than the self hosted ones.

TABLE 1. Measurable objectives taken into consideration while assesing a project's success

Objective	Importance
Customer acceptance	High
Time constraints	High
Budget constraints	High
Effective and efficient usage of resources	Medium
Desired outcome, quality and performance	Medium

TABLE 2. Potential benefits of project management

Benefit	Importance
Identification of functional responsibilities to ensure that all activities are accounted for, regardless of personnel turnover	High
Minimizing the need for continuous reporting	High
Identification of time limits for scheduling	High
Identification of a methodology for trade-off analysis	Medium
Measurement of accomplishment against plans	Medium
Early identification of problems so that corrective action may follow	High
Improved estimating capability for future planning	Medium
Knowing when objectives cannot be met or will be exceeded	High

Most project management applications are geared to a specific domain a group of domains. Others are designed for a wider range and are very general in what they can perform. One domain-oriented application is Trac [18], which is a simple application that makes issue tracking easy for software projects and uses a minimalistic approach in its design, focusing on actual development rather than imposing techniques and policies on the people using it.

Launchpad [13] is also tailored towards software development. Its main features include code hosting and reviewing, bug, issue and specification tracking, and more. It was launched as a proprietary application in 2005, but since 2009 the license has been modified and it is now open source under AGPL.

Redmine [16] isn't designed specifically for software development and can be used to manage projects in a more general sense, being bundled with a lot of features and tools. It is cross-platform and cross-database, supports multiple projects, has flexible role based access control and issue tracking systems,

TABLE 3. User base and user growth for SaaS project management applications

Application	License	User Base	User Base Growth
Trac	BSD	16.200	Low
Launchpad	AGPL	30.759	Low
Redmine	GPL	N/A	Low
MantisBT	GPL	1.500.000	Very low (-40%)
Jira	Proprietary	14.500	Medium
Basecamp	Proprietary	200.000	High
Mavenlink	Proprietary	125.000	High
Assembla	Proprietary	500.000	Very High
TeamworkPM	Proprietary	N/A	Medium

has Gantt charts and calendars, per project wikis and forums, advanced time tracking tools and more.

MantisBT [14] is a self-hosted issue tracking system that can support multiple projects per instance, sub-projects and categories. Users in MantisBT can have a different access level per project, with no limit on the number of users, issues, or projects. MantisBT was launched over a decade ago and is quite popular, with more than 1.500.000 downloads. However, statistics show a decrease of those numbers of almost 35% from month to month, with unclear information on whether what percentage of the downloads are actually used in production. Figure 1 shows download statistics within the past 12 months (March 2012 - February 2013).

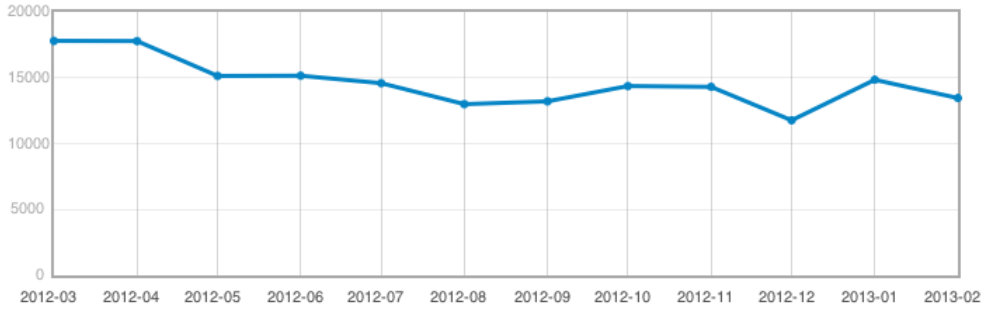


FIGURE 1. Monthly downloads of MantisBT in the past year [8]

Jira [11] is a project management tool used to track teams, planning, building and launching great products. It is one of the most complex applications with more than 150 tools such as capturing any types of issues, from bugs and features to stories and requirements to simple tasks and action items.

Jira can be extended by using one or multiple add-ons out of more than 400 that are available. These add-ons ca be used for agile project planning or to simplify planning and reporting when developers use Scrum [6] or Kanban [4]. Jira also has time tracking add-ons as well as Gantt chart plugins, which are all very important throughout the development and management of any project.

Basecamp is [12] one of the most popular project management applications on the market and is developed by 37signals which have also pioneered the RubyOnRails framework. With Basecamp, projects can be stored safely in the same system. Basecamp has advanced reporting tools that allows project managers to easily grasp the status of their projects, no matter how many there are. Basecamp makes easy for all team members, clients, contractors, and vendors to interact using the same platform. Project managers have full control over advanced users permissions, from project access to user interactions. Basecamp feature-set is very granular and allows users be organized in groups.

One can also see everyone's schedule on a visual advanced calendar, provided permissions are granted. Tasks can easily be assigned and delegated to team members. Additionally, Basecamp allows the creation of to-do lists and items, and all the features are integrated within a central email based notification system. Basecamp can be integrated with third party applications that allow easy time tracking, budget monitoring and more.

Mavenlink [15] is a business management application that extends basic project management techniques and activities to a wider whole business coordination. Mavenlink can easily be used for collaboration purposes, tracking tasks and time, budget management and even accounting with invoice generation.

The project management applications chosen for the task delegation model study are the following: Basecamp, Teamwork PM and Automated.PM [10], the later being the application developed based on the model described in [5].

The choice was made based on popularity and feature set of the applications. The proposed study should reveal and point the advantages of the described model over the ones in use by current applications. The complete scenario is described in the following section while the result of the study and whether the described model is successful or not are presented in section 4 and 6.

3. Approach: Proposed Real-Life Study Scenario

In order to study the performance of the selected project management applications, a real-life development project was chosen. Specifically, the project that is used as a test subject is building a treehouse. This choice was made because the size of the project isn't large to pose any serious issues and it is easier to measure its outcome. One can also measure the performance of the applications and their task delegation models that are being studied and tested.

A secondary reason why a treehouse project was selected for the study is the fact that every treehouse project is different than the next one. Most treehouses' bases are different from each other because usually a treehouse base is composed out of at least 2 or 3 large tree branches which always have irregular shape, size and growth angles. Apart from the base structure, most treehouses have different surrounding environments which also affect their own structure, planning, required materials and completion times [2].

The branches used for the support of any treehouse should always be in close proximity to each other. It is also recommended that the wood used to build the treehouses is pressure-treated and coated in order for it to support the structure and the floor. The roofs can be made out of various materials, from wood to thick film, depending on overall design, weather conditions and climate etc. One could also use recycled wood instead of fresh treated timber. No matter what type of wood is used, the structure of the treehouse shouldn't be rigid and fixed. Movement and tree growth should be allowed for long term durability. If the treehouse is fixed in its structure and binding points with the tree, it is likely that during time the structure will suffer greatly due to integrity changes from the base up [2].

Although only one solid tree with multiple thick branches can be used, it is recommended that more are used, resulting in bigger treehouses and a healthier environment for each of them as the actual treehouse would be supported by multiple plants instead of one. The average height of the base structure of a treehouse is usually at around 3 meters high, but this varies due to field conditions and age-group destination. Another important aspect in treehouse building is choosing the species of trees used for the support structure. Strong, slowly growing trees should be chosen against others that may grow faster or may not be sturdy [9].

If treehouses are built high off the ground movement caused by weather such as wind and gusts have to be taken into consideration and the structure be built even more elastic to allow for movement. If the base trees are strong, sturdy and the treehouse isn't built high off the ground, a fixed platform can also be used [2].

The height of the treehouse should be decided based on manpower available for the construction. If the project is built by only one individual, it's recommended the house isn't very high to reduce the complexity of the tasks, since most of the wood bolting and nailing is done overhead. The higher the distance from the ground, the harder and riskier the construction tasks are, especially when working with structure joists which may be even 50 kg in weight. A common-sense rule in treehouse constructions is making them lightweight.

Treehouse building can be simplified by using additional manpower or by using mechanical tools, such as chain hosts, most often used by car mechanics, with or without ratcheted brakes, power drills and screwdrivers, preferably electric, cordless ones, sabersaws, circular saws, hand saws, hammer drills, different sized bits, nails, screws, preferably galvanized, tape measures, metal yardsticks, carpenter's squares and levels and a tall ladder [9].

Other parts or tools that may be required, depending on design, layout and terrain details are: plastic sheathed galvanized steel cables that usually support up to 200 kg, heavy duty cable tension adjusters, flat washers and lock washers, bevels to protect against sharp edges, rivets and more.

3.1. **The treehouse project.** This subsection describes the general steps and development process of a treehouse while tables 4 and 5 present the tasks required to complete the project, their description, estimated duration and minimum recommended manpower. The initial phase in building a treehouse is selecting the trees it will rest upon. After the location is chosen, the area must be cleared, by tree surgeons or at their recommendations. The following task is to buy the necessary materials and equipment, if none is available. This includes lumber, bolts, cables and required machinery.

Once the necessary materials and tools are on the construction site, the main beams are raised, leveled and bolted in the trees. The base platform of the treehouse is composed out of 2 individual pieces, one being built on the ground and then raised into the tree on the structure, while the other should be built directly on the main beams. The next step required to complete the entire base structure of the treehouse is to add the plywood to the structure and hold it in place with galvanized screws.

The construction continues by cutting an opening into the base structure to form an access door and to build the ladder needed to climb into the treehouse. Once the base platform is completed and easy access is obtained, the external walls are built by using joists for the structure and plywood to coat the wall. Cuts are made into the wall's plywood in order to obtain windows. Joists are then placed in position to form the roof structure at an angle with splinters applied on top to protect the structure against rain, snow or debris.

TABLE 4. Treehouse project design & base structure tasks

Task name	Task description	Estimated duration (hours)	Recommended manpower
Treehouse design	Selecting the location of the treehouse, structure design and layout	20	1
Clearing the construction site	Clearing the area around the trees, by or at the recommendations of a tree surgeon	24	2
Lumber acquisitions	Buying lumber, ranging from large joists to splinters and plywood	4	2
Hardware acquisitions	Buying bolts, cables etc	2	1
Tool acquisitions	Buying all tools and machinery needed for the construction	8	1
Leveling base beams	Level base structure's main beams between trees using carpenters level	4	2
Assemble central base platform	Cut, trim, sand and bolt together the central area of the platform	4	1
Build the perimeter base platform	Raise joists in the tree and bolt them to the base beams	4	2
Raise central base platform	Raise central base platform and bolt it to the main beams	3	3
Add plywood to the entire base structure	Measure, cut and raise plywood and screw it tightly to the base structure using galvanized screws	4	1
Create access door	Cut a whole in the base structure and the applied plywood, add hinges and lock	2	1
Build ladder	Measure, cut and bolt together joists and poles to form a ladder; put ladder in position below access door	3.5	1

TABLE 5. Treehouse project walls and roof tasks

Task name	Task description	Estimated duration (hours)	Recommended manpower
Build exterior walls' structure	Measure, cut, position, screw exterior joists for treehouse walls	10	2
Apply plywood to exterior walls	Measure, cut and bolt plywood to exterior walls	6	2
Cut windows	Position, measure and cut windows within the walls	3	1
Build roof structure	Measure, cut and screw into position joists for the roof	6	2
Apply splinters to roof	Measure, cut and bolt splinters on top of the roof structure	10	2

3.2. **Testing and studying the proposed project management applications.** Based on the data presented within tables 4 and 5 content was generated and introduced into the project management applications in order to thoroughly manage the tasks presented. There were 3 workers assigned to develop the project, a carpenter, a taxi driver and an unqualified individual. The crucial design and structure tasks were assigned to the carpenter while the rest of the tasks to any worker that was available. Section 4 presents results gathered from the project management applications after 3 consecutive treehouse development projects. This was needed in order to train the proposed model (Automated.PM [10]) and have it assign tasks automatically.

4. Study Results and Successfulness of the Studied Model

Although Basecamp has many features as described in section 2, it has many drawbacks. For instance, using Basecamp with Projectite, its Gantt diagram add-on, had its issues, mainly because a task cannot be assigned to more than one person. Another drawback was that one cannot set the start and end times as well, only dates are available. This is a major issue since some tasks may be solved within minutes or hours instead of entire days. Further more, the Gantt charting add-on didn't allow to set multiple predecessors for a single task. This was needed for instance in the case of leveling the base beams which required both lumber, tools and hardware, and the presence of the carpenter and an additional assistant.

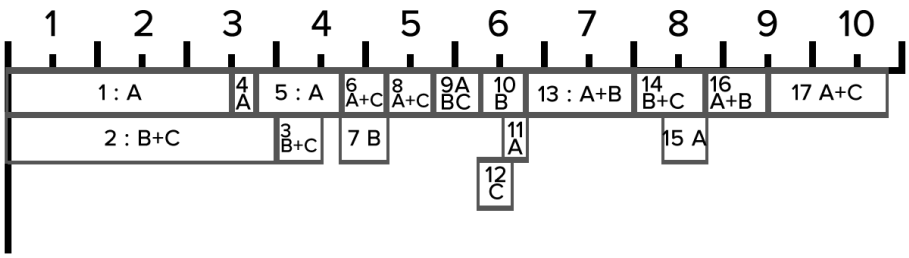


FIGURE 2. Chart showing ideal task delegation within the project. Numbers 1 to 17 are task numbers as denoted in tables 4 and 5. Capital letter A, B and C represent carpenter, taxi and unqualified workers

This resulted in a project estimated time to complete of 8 days for the design and base structure alone. The ideal task delegation scenario only needs roughly 5 days and a half for the first part of the construction, while the whole project estimated time to complete is just under 10 days, as shown in figure 2. The overall estimated time according to Basecamp and Projectite is 16 days, almost double than the ideal scenario.

Teamwork PM [17] has the same issue as Basecamp, not allowing time input, only dates. However, Teamwork PM allows to assign multiple people to the same task. A drawback for Teamwork PM is that it allows project managers to assign multiple tasks to the same individuals, although it is clear that they are already working on another task within the same time slot. Teamwork PM also has time loggers which can be used by workers to effectively track worked hours on each task they are assigned to. It also shows improved features but extra time is required on project manager's part when assigning tasks to workers with their start and end dates. Teamwork PM project plan estimated the time to complete at around 14 days, still 4 days behind the ideal schedule.

While testing and studying Automated.PM, better results and potential for improvement was detected. As the proposed model uses past task delegation to automatically assign tasks, the first phase of the project was initially introduced and delegated manually. Automated.PM allows time input instead of date constraints the other applications had, resulting in more accurate estimations. When creating the tasks for the second part of the construction, the software automatically assigned them as they were added to available workers, based on their past success or failures. Since the carpenter is more experienced with structural engineering and design of buildings, the software automatically assigned this individual to the two most important tasks of the second phase,

namely building the walls' and roof's structure. On the initial run of the project, Automated.PM yielded an estimation of just under 12 days for the completion of the project, while the third and last run yielded an estimation of 11 days and 5 hours.

5. Future Studies, Tests and Work

As the study has shown, with each similar project managed using the proposed model, the estimated time to complete slightly improved. However, the study has been performed on a small scale project. Future studies and tests should be designed with larger, more complex projects in mind, within larger companies and with multiple teams working on the project.

Future developments may include enhancements of the current model, such as assigning tasks based on users skill set not just past performance, time tracking, taking user preference into consideration when automatically assigning the tasks, the ability to assign tasks to multiple individuals or to select task predecessors for newly created ones in order to create proper dependencies.

The proposed model performed better than the ones currently on market with overall completion times being 26.77% faster than Basecamp and 15.45% faster than TeamworkPM. However, the estimation of the proposed model is 19.23% longer than the ideal project plan leaving room for improvements.

References

[1] *A guide to the project management body of knowledge*, 4^{th} Ed, P.M.I., 2008
[2] A. Aikman, *treehouses*, CreateSpace Independent Publishing Platform, 2010
[3] H. Kerzner, *Project Management a Systems Approach to Planning, Scheduling, and Controlling*, Tenth Edition, John Wiley & Sons, 2009
[4] P. Klipp, *Getting Started with Kanban*, Amazon Digital Services, 2011
[5] B. Pop, *Building an Automated Task Delegation Algorithm for Project Management and Deploying It As Saas*, Studia Univ. Babes-Bolyai, Informatica, Vol LVIII, No 1, 2013
[6] K. H. Pries, *Scrum Project Management*, CRC Press, 2012
[7] J. Skabelun, *Are non-performers killing your bottom line?*, Credit Union Executive, Vol 31, No 13, 2005
[8] Sourceforge, *Statistics for MantisBT 01.03.12 - 28.02.13*, http://goo.gl/TXUkX
[9] L. Tai et all, *Designing Outdoor Environments for Children: Landscaping School Yards, Gardens and Playgrounds*, McGraw-Hill Professional, 2006
[10] http://automated.pm
[11] http://www.atlassian.com/software/jira/overview
[12] http://www.basecamp.com
[13] https://www.launchpad.net
[14] http://www.mantisbt.org
[15] https://www.mavenlink.com/
[16] http://www.redmine.org
[17] http://www.teamworkpm.net/
[18] http://trac.edgewall.org

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

E-mail address: popb@cs.ubbcluj.ro, bogdan.pop@webraptor.eu

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

A SERVER-SIDE SUPPORT LAYER FOR CLIENT PERSPECTIVE TRANSPARENT WEB CONTENT MIGRATION

DARIUS BUFNEA[(1)] AND DIANA HALIŢĂ[(1)]

Abstract. The migration process of a website's content within a Content Management System almost always implies changes in the site structure as seen by search engines and web clients. This variation leads to some disadvantages, such as misdirecting search engines visitors to old, unavailable, URLs. Even if, over time, search engines adapt to changes in the site's structure, the problem remains unsolved for visitors landing from 3rd party referrers. This paper presents a server-side support layer for client perspective transparent web content migration, layer that automatically maps the old visible structure of a website to the new one implied by the migration process. Some of the advantages of such a mechanism are: reducing incoming dead links from 3rd party referrers, assisting search engines for properly redirecting users or page rank and SERP conservation.

1. Introduction

In today's Internet, more and more web sites are being built using a Content Management System (CMS). From the top one million websites, as classified by Alexa.com, 22.5% of them are built on a CMS platform and WordPress occupy a percentage of 12.4% of the total[1]. And these numbers continue to grow. More and more websites are being powered by Content Management Systems, among their advantages, we can highlight the following:

Received by the editors: April 16, 2013.

2010 *Mathematics Subject Classification.* 68U35 - Information systems, 68M11 - Internet topics.

1998 *CR Categories and Descriptors.* H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Web-based interaction*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia – *Navigation* .

Key words and phrases. CMS migration, web content migration, transparent 404 not found redirection.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

- Easy maintenance: content's maintainer must not have knowledge of design / HTML, he does not need to be a web programmer to handle website's content;
- Consistency of design is preserved: the content from all authors is presented with the same, consistent design;
- Easy migration from one presentation to another, from one design to another: site's graphic can be changed without need of rebuilding the website;
- Regular security updates;
- Full control over the elements related to search engine optimization. Content freshness is a factor that helps a lot because search engines prefer a site with a content updated on a daily or at least weekly basis;
- Multiple roles for users with different levels of rights;
- Additional functionality offered: 3rd party plugins;
- Web based administration: editing anywhere, anytime removes bottlenecks;
- Sites are self-organized based on categories, pages, post links. Navigation is automatically generated adjusted: site's menus are typically generated automatically based on the database content and links will not point to non-existing page.

All above advantages are the main reasons that more and more websites are being built based on a Content Management System. More over, based on the same favors, old sites built on static technologies are being migrated within a CMS, considering also the scalability of such a system.

There are also situations, when a CMS based web site must be migrated to a different, new CMS. In many cases, the current CMS can not support new goals or does not have all the functionality or features needed to achieve them. Migrating from one CMS to another is necessary from several points of view:

- the old CMS might be considered deprecated;
- there is the possibility of choosing an open source CMS granting easier access to support;
- CMS scalability;
- highlighting the newest web technologies: CSS3, Ajax, HTML5.

2. Migration process challenges

Migration of a web site to a new CMS can be done either manually or automatically. In the case of a static technologies based web site, the migration process will involve most likely manual operations. On the contrary, migration between two popular CMS will imply some automatically performed actions.

The migration process it's not only a problem related exclusively to content migration. One of the most difficult part of migrating to a new CMS is mapping the old content to a new one, in addition to mapping the old logical organization model to the new one as required by the new CMS.

The complexity of the migration process can have a gradual solution, following three steps: dividing the content into categories, estimating needed time for migration and migration reevaluation based on guidance.

The first step of the migration process is about classifying different types of content, depending on the outcome of its analysis and derived types - these types shall mean all categories in which the content is subdivided. This creates two types of rules that must be identified: those rules which relate specifically to particular content, needed also after migration, and rules relating to what can and what can not be automatically migrated. These rules are useful in defining the priorities and the efforts which must be made. At this point, a decision can be taken about what can be automatically migrated and what can't. Generally it is desired to automate as much of the content that must be migrated.

The second step is about guiding and estimating migration needed time, i.e. compare automatic migration with manual migration required time.

The final step involves evaluating the automatic migration process. The biggest problem which arises when choosing automatic migration is content's structure and its regularity. On the other hand, manual migration to another CMS requires in some cases viewing, editing and manually moving the content, which probably will lead to a waste of resources, time and effort.

How difficult is migration of a website's content within a CMS or from one CMS to another?

- Competition between CMS providers translates into a difficult transition between them.
- Using the same CMS for many years makes migration difficult; a consequence could be the ability to run an inconsistent code.
- Consolidation content management platforms in a single system can be logistically a nightmare.
- Content that is meant to be migrated is not limited to text / html, but may include graphic content resources (images), multimedia content, which does not appear in the database structure (that graphic content resides as a file in a file system).
- In some cases, content migration is hindered by its organization. For example, a page in PHP Website is organized as sections reunion, including an order relation which is timeless. Such multisection page

turns naturally into a series of temporal posts (lacking a time attribute).
- Once the content migration is complete, it is necessary to change internal link's meaning, so that it would point to the appropriate page from the new site structure.
- If the site is very dynamic, i.e. the information on it are updated regularly, what happens when the site is in a state of transition? Time spent on updates will be doubled, the information must be renewed also in the old site and the new one (which has not yet been released).
- The new site structure will not be instantly visible in Internet by search engines; however search engines have the ability to adapt (sooner or later) to this new structure. On the other hand, third party referrers that link to the migrated website probably will never adapt to the new site structure, this operation implying manual intervention.

The migration process implies not only moving the content within the new CMS, the migration, either automatically or manually performed, should also take into consideration the transparent behavior of the web site from visitor perspective, either a web browser or a search engine crawler. One common problem induced by such a migration is exposing to the web a certain content to a new, different URL as it was presented in the old site. After migration, this will be reflected in lots of misleading visitor coming from search engines or third party referrers. Besides not serving the proper content to visitors, the newly migrated website can lose page rank or other in-time gather benefits induced by back links or social shares. This paper presents a server-side support layer for client perspective transparent web content migration, layer that automatically maps the old visible structure of a website to the new one implied by the migration process. Some of the advantages of such a mechanism are: reducing incoming dead links from 3rd party referrers, assisting search engines for properly redirecting users or page rank and SERP conservation.

Similar solutions as the one presented in this paper to the above problems have been proposed and implemented, mostly as plugins, inside all major CMS today ([5], [6], [7]). However, these solutions, in order to properly redirect a user or a crawler in case of a 404 not found request, only take into consideration some in-time gather data based on user behavior, such as target or exit pages in users' session or users' chosen pages from a custom search that follows the 404 response. Our server-side support layer is trying to match new URLs to old ones based on content similarity, or other semantic related information such as: URLs, the query given to the search engines or referrer's content similarity with the linked content. Also, such a solution has the advantage

of being implementable prior to the release of the new web site, having no dependence on in-time gather statistical data based on user behavior.

3. Types of content migration

The following formal notations are general enough for any CMS (or static content website) to any CMS migration, but we'll highlight them on our real life scenario CMSes implied in the migration process: phpWebsite as the source CMS and Wordpress as the target CMS.

The content which appears both in the old site and in the new site, is presented in two different ways. We are dealing either with static content (i.e. a file that exists in the file system of the web server, usually an external resource such as a .pdf or .jpg file) or with dynamic content, taken from a CMS' database.

A) Static content migration. When we migrate such a content, even if its URL is changing, the base name of the file remains the same.

Example:

```
oldsiteURL: http://oldsite/oldpath/filename
newsiteURL: http://oldsite/newpath/filename
```

We define it as perfect match, $similarity($`oldsiteURL, newsiteURL`$) = 1$.

B) Dynamic content migration. We describe the content which is found either at an old or new URL as consisting of two parts. The first part is represented by the content corresponding to the page template (i.e. the master page), and the second part is the absolute content which is usually stored in the database. Actually, the absolute content is the one being migrated. When comparing the similarity of old and new content, we'll take into discussion only the absolute content, in order to avoid the noise induced in the similarity algorithm by the master pages' HTML code.

Example:

For Wordpress the absolute content can be extracted either from the database or directly from its associated URL. Within a Wordpress' master page, the absolute content can be retrieved as the inner HTML of the div having the `content` id.

We'll designate by OP and by NP a page from the old site and a page from the new site, respectively. In Wordpress, absolute content is represented by:

- pages $NP_{\neg T}$ (timeless articles, i.e. pages)
- posts NP_T (temporal articles, i.e. posts)
- reunion of posts (i.e. a category)

- subset of posts in a category

In phpWebsite, absolute content is represented by:

- single section page: OP_{SS}. Such a page is usually migrated in a $NP_{\neg T}$.
- multiple section page: $OP_{MS} = \bigcup content(section_i)$. Such a page is migrated either in a single $NP_{\neg T}$, if its sections are time independent, or each of its sections becomes a separate post NP_T tagged in the same category. The migration of the temporal posts in Wordpress has encountered difficulties because at least in phpWebsite, its data model stores no time related information, in relation to the date of creation or modification of a page section.

When we compare the similarity of a content from the old site which corresponds to a page with multiple sections (page which in terms of presentation is shown as a category of posts in the new site) it is useful to compare the similarity of the absolute content of the old site content with the absolute content (in the new site) of a subset of posts in that category. The substantiation for this claim is presented in section 4 of our paper.

C) A static content of the old site, especially an HTML file, is migrated as dynamic content within the new CMS. This is useful for integrating file's content in the new look and feel of site. In this situation, the similarity function should be computed based on the absolute content from the new site and on the file's content from the old site.

A possible approach when we have to migrate a site between two CMSes would be to compare the absolute content within the databases maintained by the two CMSes. We have rather prefer to determine the absolute content or the custom content involved in the similarity comparison based on its web presentation because:

- This method is more general, in comparison with the one which effectively compares content from databases. It can also be taken into consideration when migrating a static web site.
- We would not cover cases listed above.
- There are no additional privileges required, for examples access rights to the CMS database. Querying content via its web presentation URL is much easier since no knowledge is required about the organization model or the database structure of the CMS.
- Matching process should be done from the web client perspective.

4. Algorithm and implementation

This section of our paper presents the algorithm (and its implementation details) used for matching old site URLs to new ones. In order to match these URLs, our algorithm makes use of a similarity function, but the method

is not dependent of a certain similarity function. Future work may imply comparison of the results obtained with different similarity functions, from their speed and matching accuracy point of view. Our approach is general enough to be applied to any similarity function, or to allow the replacement of this function with minimal effort.

Information processing is not done in real time (i.e. while the user gets a 404 not found response). Rather than trying a real time computation, we choose instead a batch processing approach: a previously run program implements a similarity algorithm in order to identify URLs with similar content. There are several reasons to do this. First of all, in both sites there are thousands of valid URLs. This would imply that the complexity of the algorithm to be at least equal to the cardinal of the cartesian product of the two sets: old site's and new site's URLs. Running a real time computation will induce delays in serving a response to the web client. Secondly, the batch program runs completely independent of the CMS core, its language of implementation does not necessarily depend on an API exported by the CMS.

For a quick match we use the cosine similarity algorithm [3]. This algorithm has a fast and well-implemented implementation within Apache Lucene [2]. Apache Lucene is a high-performance, full-featured text search engine library that provides a fast and tested implementation for at least the similarity algorithm. It is an open source project available for free download. Moreover, its API allows users to modify the function used in the similarity algorithm by implementing matching providers, in order to get better or faster results when comparing two absolute contents.

The formal presentation of the algorithm implemented by the batch process follows:

```
For each oldsiteURL having static content do
   Identify the newsiteURL which points to the same static)
      content (based on base filename) (section 3, A)
   If this newsiteURL was identified then
      eliminate the oldsiteURL from the URLs list which must be processed
   EndIf
EndFor
For all unprocessed oldsiteURLs do
   If it is a static content URL then
      absolute_content = the effective content (section 3, C)
   else
      absolute_content = content(oldsiteURL) - content(page template)
         (section 3, B)
   EndIf
   Identify the newsiteURL so the absolute_content has the best
```

```
      similarity with the oldsiteURL's absolute_content
   The matching pair is inserted in a table, together with the current
      date, the similarity and the best similarity ever
EndFor
```

One might wonder why we stored in a database the current similarity and also the best similarity ever. A category evolves in time, because new announcements might appear in it. In such a case, the similarity obtained when comparing the category presentation URLs is decreasing. We will have maximum similarity if we make the comparison with a subset of older announcements (i.e. the newest announcements from the old site).

Example: In Student News category, semantically speaking, a visitor might want to see the newest articles and he may want to be redirected to the newest ones, not to a snapshot page that match perfectly (from the content point of view) the news in the way they appear in the old site.

In order to speed up the running time, a threshold experimentally determined can be used. If the similarity of two compared URLs is greater than a threshold, both old and new URLs can be removed from their sets and not get used anymore in the comparison process.

The deployment of the results map in a real live situation such as a production web site, was done directly in the Apache configuration file (i.e. the appropriate .htaccess or *.conf file) using the `Redirect permanent` directive [4]. This method guarantees the independence of the CMS and its technology. Another advantage is the fact that this file can be deployed by the user for most web hosting service providers.

More over, this Apache configuration file can be overwritten after future runs of the algorithm and it can be easily integrated in the `web client -> web server -> web application -> backend database server` workflow, practically without any intervention to the new CMS configuration, core, plugins or content.

5. Results and evaluation

In order to demonstrate the benefits of our approach we have performed some experiments related to the number of not found (i.e. 404 HTTP response code) pages requested by users landing on our site from an external referrer. By external referrer we understand either a search engine or a 3rd party referrer that links our site. These experiments were run for 30 days, both on the old site before migration and on the new site as well.

The number of not found requests addressed to our test site before migration was relatively small, around 1.9% of the request from an external referrer

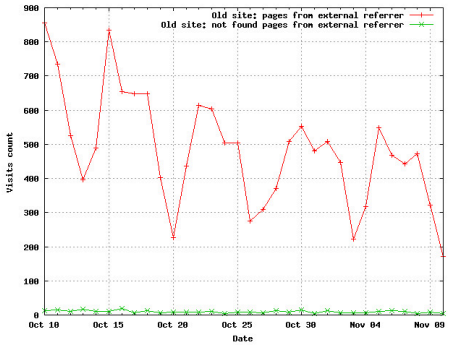


FIGURE 1. Old site: number of pages accessed from external referrer and number of not found pages accessed from external referrer

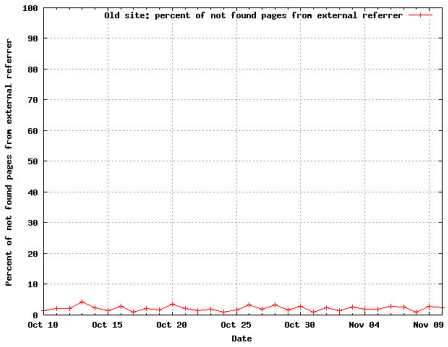


FIGURE 2. Old site: percent of not found pages accessed from external referrer

generating a 404 response (Fig. 2). These responses where mainly generated for requests coming from a 3rd party referrer.

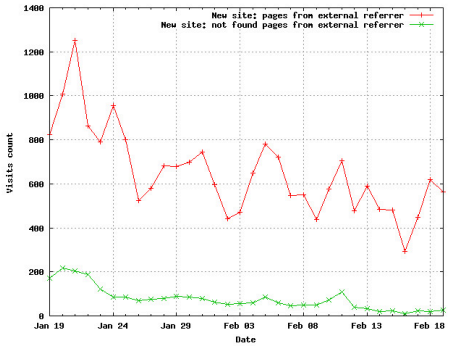


FIGURE 3. New site: number of pages accessed from external referrer and number of not found pages accessed from external referrer

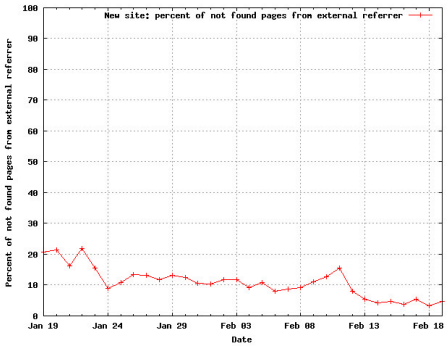


FIGURE 4. New site: percent of not found pages accessed from external referrer

After migration, the number of not found requests made via an external referrer increased naturally to an average of 11.2% for 30 days (Fig. 4). Search engines are quickly adapting to modifications in a site's structure, about one week after migration they were correctly redirecting the traffic to proper landing pages (Fig. 5, Fig. 7).

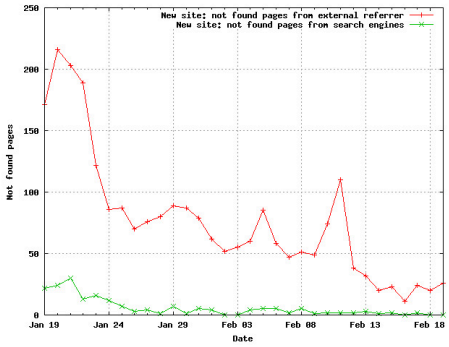


FIGURE 5. New site: adaptation of the search engines to the new structure of the site

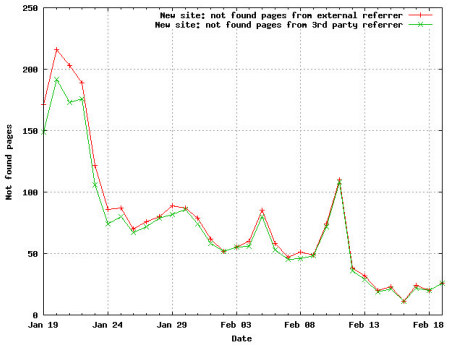


FIGURE 6. New site: Most of not found pages are coming from 3rd party referrer

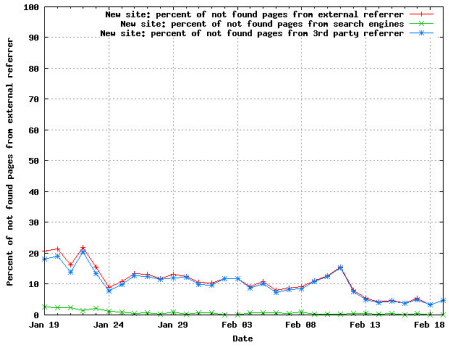


FIGURE 7. New site: percent of not found pages, not found pages from search engines, not found pages from 3rd party referrer

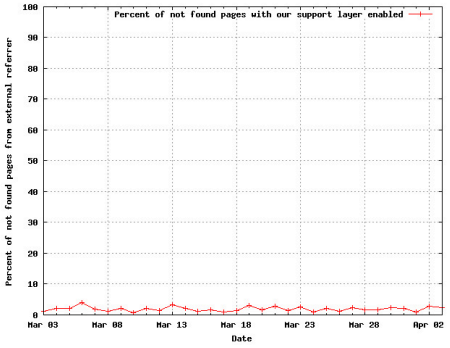


FIGURE 8. Percent of not found pages having an external referrer with our support layer enabled

After migration, most of not found request were made via a 3rd party referrer (6).

Figure 7 depicts the percent of not found requests made via external referrers in a 30 days time interval, after the migration process had been completed.

In the first week after migration, around 1.9% of the requests made via a search engine (from the total number of requests made via an external referrer) were generating a 404 response code, this value dropping to 0.23% in week four. In the matter of requests made via a 3rd party referrer, in the first week after migration we measured that 14.7% percent of these requests were for not found

pages, this value dropping to 4.25% in week four, still above the average of not found responses that where generated before migration.

Figure 8 shows the percent of not found response after we embedded our support layer inside the newly migrated version of our test site. The 404 not found responses have dropped to an average of 1.6% counted for a 30 days time interval, bellow the average of the old site. In the batch run of the similarity algorithm we have identified pairs of URLs having the cosine similarity higher than 0.7 for around 93% of the migrated content. This threshold was experimentally chosen, in fact the similarity distribution in the [0, 1] interval was either above 0.8 for matching URLs (i.e. correctly identified pairs of URL), or bellow 0.3 for arbitrary pairs.

6. Conclusions and future work

In this paper we have advanced a server-side support layer for client perspective transparent web content migration within a CMS. We have pointed out the need of such a layer and the challenges in implementing it. Also, we have proposed a method, covering both the theoretical and practical aspects, for implementing this layer by mapping URLs from the old site to the ones in new site based on their content similarity.

We are currently focused in evaluating different similarity functions in order to improve the matching process and its speed. An improvement which can be brought into discussion refers to giving different weights in the similarity algorithm to the various properties of the content such as: URL, page headings, page title, key words (when the user is coming from a search engine). Another option to be taken into consideration is redirecting the user to a similar page, from the content point of view, as the page where he is coming from (i.e. the page within our site most similar with the referrer). The main goal remains the same: succeeding in user's redirection, no matter where he comes from, even if he comes from a third party referrer.

References

[1] Leena Rao, *WordPress Now Powers 22 Percent Of New Active Websites In The U.S.*, August, 2011, TechCrunch

[2] Apache Lucene, http://lucene.apache.org/, *A high-performance, full-featured text search engine library*

[3] Amit Singhal, *Modern Information Retrieval: A Brief Overview*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2011, 24 (4): 35-43

[4] mod_alias - Apache HTTP Server, http://httpd.apache.org/docs/current/mod/mod_alias.html

[5] Zyxware Technologies, *Search404: Automatically search for content when a 404 error occurs*, http://drupal.org/project/search404

[6] *Dynamic404 - Logical error-pages*, http://www.yireo.com/software/joomla-extensions/dynamic404

[7] *404 Redirected Wordpress plugin*, http://wordpress.org/extend/plugins/404-redirected/

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

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

E-mail address: diana.halita@ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

XRDL: A VALID DESCRIPTION LANGUAGE FOR XML-RPC

DIANA TROANCĂ[1] AND FLORIAN BOIAN[1]

Abstract. XML-RPC standard does not define any Description Language for this web service. XRDL started as an open source project with the purpose of being XML-RPC Description Language. The open source project already includes some applications for XRDL (automatic XRDL generation for servers and clients written in PHP or C++/Qt [3]), but it has not yet been demonstrated that XRDL can be used as a Description Language without any exceptions. This paper intends to prove that XRDL is a valid XML-RPC Description Language, so the focus can move back on its applicability and further development.

1. Introduction

The purpose of this paper is to demonstrate that XRDL is a valid Description Language for XML-RPC services. This includes the fact that any XML-RPC service can be described by a XRDL document and that any web service described by a XRDL document is a valid XML-RPC service. Therefore the demonstration will consist of two parts.

XRDL [3] started as an open source project. It has not yet been accepted as a standard and it is not widely accepted as the XML-RPC description language [8]. Some researchers accept it as XML-RPC Description Language and others argue its utility. The open source project already implements some applications for XRDL, but they do not demonstrate the validity of XRDL as a Description Language for XML-RPC services.

Received by the editors: April 17, 2013.

2010 *Mathematics Subject Classification.* 03B48, 03F60.

1998 *CR Categories and Descriptors.* H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services*;

Key words and phrases. XRDL, XML-RPC, Web Services.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

XRDL is defined by a XSD-schema that describes the structure of a XRDL valid document. As defined in the XSD-Schema it has two wide sections for describing the service:

- a section where complex types used by the service can be defined and
- a section that describes the methods that the service offers.

In the types section of a XRDL document we can define complex types used by the web service. The second section describes all the methods offered by the service with the parameters needed and the result that the method will return. The data type of the parameters and the result is specified using the "type" attribute. According to the definition of XRDL the value of this attribute is a string. In order to generate a valid XML document it can take any value that represents a valid data type in XML, but XML has a lot of built-in data types.

The specification of XML-RPC [4] defines six simple types and two compund types that can be defined as a collection of more simple or compound elements. The basic data types are presented in Table 1 [5, 8].

TABLE 1. XML-RPC Data Types

Type	Value	Example
int or i4	32-bit integers	<i4>35</i4> <int>2</int>
double	64-bit floating-point numbers	<double>-3.456</double>
boolean	0 (false) or 1 (true)	<boolean>0</boolean>
string	ASCII text or Unicode	<string>How are you</string>
dateTime.iso8601	Dates in the following format: CCYYMMDDTHH::MM:SS	<dateTime.iso8601>20130903T13:11:05 </dateTime.iso8601>
base64	Binary data encoded as base 64 (as defined in RFC 2045)	<base64>base64 encoded data. </base64>

The two compound types used by XML-RPC are *array* and *struct*. Array is a sequence of elements with the same or with different types. The *struct* data types are defined as pairs in form of name-value and are similar to hashtables.

For the purpose of this paper, we cannot allow the *type* attribute to take any valid XML value. For example, if we have an element with *type="decimal"*, we have no equivalent for that particular type in XML-RPC [6, 7]. Taking all this into consideration we will add a restriction to XRDL. The *type* attribute from a valid XRDL can have as a value either a simple data type that is also defined in the XML-RPC specification or a complex type that is defined in the types section of that XRDL document. We will prove in the next paragraphs that the compound data types from the XML-RPC specification can be defined in XRDL as a complex type using the simple types available.

2. XML-RPC described by a XRDL document

The first part of the equivalence is to prove the following Theorem:

Theorem 2.1. *Any XML-RPC can be characterized by a XRDL document.*

The characterization of a web service consists of describing the methods offered by that service. In order for a potential user to know how to call the methods, he needs to know the name of the function, the input of the function (the parameters) and the result that it returns [4]. XRDL has a simple way of describing the functions of a web service, defined in the following part of the XSD document:

```
<xs:element name="methods">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="method" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType mixed="true">
                        <xs:attribute name="type" type="xs:string" use="required" />
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
               <xs:attribute name="result" type="xs:string" />
               <xs:attribute name="name" type="xs:string" />
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

From that schema we can see that any method can be described in XRDL and the demonstration in this case comes down to proving that any data type from XML-RPC has an equivalent in XRDL. Moreover, considering the restriction we put on the XRDL *type* attribute, we still have to prove that the compound types from XML-RPC can be defined in XRDL.

In the following Sections we will discuss the two compound data types *array* and *struct*.

The *array* type in XML-RPC contains an element named *data* that has one or more children. These children are each defined by a *value* tag which contains another tag specifiying the data type and the value. An array can be unidimensional if it contains only elements of simple data types or multi-dimensional if it also contains elements of other compound types.

Lemma 2.2. *A onedimensional array with n elements can be defined as a data type in XRDL,* $\forall n \in \mathbb{N}$.

We will prove Lemma 2.2 using an induction argument. For the case n=1, an array with one element will have the following general form:

```
<array>
   <data>
      <_simpleType>_customValue</_simpleType>
   </data>
</array>
```

_simpleType will be replaced by any of the types: string, int/i4, double,boolean, dateTime.iso8601 or base64 and *_customValue* will be replaced by a suitable value according to the data type chosen. In XRDL we define a complex data type:

```
<type name="_1_ElementArray">
   <member type="_simpleType"> _firstMember</member>
</type>
```

_1_ElementArray is the name given to the custom data type, *_simpleType* is the same type used in the XML array and *_firstMember* is the name given to the member of the array (Obs. The names given to the data type or to the members are irrelevant). As we can see this custom type is equivalent to the array given in XML. For the inductive step we assume that a onedimensional array with k elements can be defined as a complex type in XRDL. We have to prove that a onedimensional array with k+1 elements can be defined as a complex type in XRDL. Let the k+1-length array be:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
      <value>
         <_simpleType_2>_value_2</_simpleType_2>
      </value>
      ...
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
      <value>
         <_simpleType_k+1>_value_k+1</_simpleType_k+1>
      </value>
   </data>
</array>
```

If we build an array containing only the first k elements, acoording to the induction hypothesis that array can be expressed as a complex type in XRDL as follows:

```
<type name="_k_ElementArray">
   <member type="_simpleType_1">_ member_1</member>
   <member type="_simpleType_2">_ member_2</member>
   ...
   <member type="_simpleType_k-1">_ member_k-1</member>
   <member type="_simpleType_k"> _member_k</member>
</type>
```

Now we can build a similar array with k+1 elements just by extending this type with one more element:

```
<type name="_k+1_ElementArray">
   <member type="_simpleType_1">_ member_1</member>
   <member type="_simpleType_2"> _member_2</member>
   ...
   <member type="_simpleType_k-1">_ member_k-1</member>
   <member type="_simpleType_k"> _member_k</member>
   <member type="_simpleType_k+1">_ member_k+1</member>
</type>
```

This is the k+1-length array that we had to transcribe in XRDL and in conclusion the hypothesis is proven for any $n \in \mathbb{N}$.

Now we will consider multi-dimensional arrays, which are arrays that contain other *arrays* or *struct* elements.

Lemma 2.3. *A n-dimensional array can be expressed in XRDL as a complex data type, for any* $n \in \mathbb{N}$.

An example of a two-dimensional array is:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
      <value>
         <array>
            <data>
               <_simpleType_2>_value_2</_simpleType_2>
               <_simpleType_3>_value_3</_simpleType_3>
            </data>
         </array>
```

```
      </value>
   </data>
</array>
```

Before proving this hypothesis let us treat the second compound data type *struct*.

The *struct* type in XML-RPC contains a sequence of elements of type *member*. Each member has two children *name* and *value*. The *value* tag has another child that specifies the data type of that member and the actual value. A struct can also be multidimensional if it contains other elements of compund data types.

Lemma 2.4. *A onedimensional struct with n pairs of elements name-value can be defined in XRDL as a complex type, for any* $n \in \mathbb{N}$.

For the base case we define the general form of a struct with one pair of elements name-value:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType>_value_1</_simpleType>
      </value>
   </member>
</struct>
```

where *_simpleType* will be replaced by any of the types: string, int/i4, double, boolean, dataTime.iso8601 or base64. In XRDL we define a complex type accordingly:

```
 <type name=_1_PairStruct>
   <member type="string">_member_1</member>
   <member type="_simpleType">_value_1</member>
</type>
```

For the inductive step we assume that a onedimensional struct with k pairs of elements can be defined as a complex type in XRDL. We still have to prove that a onedimensional struct with k+1 pairs of elements can also be defined in XRDL. Let the struct with k+1 pairs of elements be:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
   ...
```

```
   <member>
      <name>_member_k</name>
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </member>
   <member>
      <name>_member_k+1</name>
      <value>
         <_simpleType_k +1>_value_k+1</_simpleType_k+1>
      </value>
   </member>
</struct>
```

If we consider the first k pairs of this struct they would build another struct but with k pairs of elements, which according to the earlier assumption can be defined in XRDL as a complex type. Let that type be *_k_PairStruct*:

```
<type name="_k_PairStruct">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_k-1</member>
   <member type="_simpleType_k-1">_value_k-1</member>
   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
</type>
```

Taking this into consideration we can build another complex type just by extending this type with two other members:

```
<type name="_k_PairStruct">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
   <member type="string">_member_k+1</member>
   <member type="_simpleType_k+1">_value_k+1</member>
</type>
```

And this is exactly the definition of our struct with k+1 pairs. So the hypothesis is proven for any $n \in \mathbb{N}$.

Now we go back to the demonstration of Lemma 2.3. For the base case we observe that a two-dimensional array would have to contain at least an element of a one-dimensional compound type and all the another elements can either be of a simple type or of a one-dimensional compound type. But we have already proven in Lemma 2.3 and Lemma 2.4 that any one-dimensional array

or struct can be defined in XRDL as a complex type. The next step would be to define another complex type in XRDL to build this two-dimensional array. The general form of a two-dimensional array in XRDL is:

```
<type name="_twoDimensionalArray">
   <member type="_type_1">_member_1</member>
   <member type="_type_2">_member_2</member>
   ...
   <member type="_type_m">_member_m</member>
</type>
```

where *_type_i* is a simple type or a one-dimensional compund type for each $i \in \{1, 2, ..., m\}$ and there is at least one $j \in \{1, 2, ..., m\}$ so that *_type_j* is a one-dimensional array or struct. The base case for the hypothesis in Lemma 2.3 is obviously true. For the inductive step we assume that any k-dimensional array can be defined in XRDL. We observe that a k+1-dimensional array has l-dimensional elements with $1 \leq l \leq k$. The inductive assumption says that for each element of this array we can define a complex type in XRDL, so for defining the k+1-dimensional array we can just build a complex type with elements of those types defined before.

For multi-dimensional struct we consider the following Lemma:

Lemma 2.5. *A n-dimensional struct can be expressed in XRDL as a complex data type, for any* $n \in \mathbb{N}$.

The exact same reasoning from Lemma 2.3 can be applied for proving this hypothesis with the observation that one member of the new added pair will always be of type string (the name of that pair) and the other member representing the value can be of any multi-dimensional compound type or of a simple type.

3. XRDL document as a valid XML-RPC service

This Section will consist of the demonstration for the following Theorem:

Theorem 3.1. *Any XRDL document describes a valid XML-RPC web service.*

As we have seen in Section 2 the structure of a XRDL document allows it to describe the methods of a web service and if necessary define compound data types. In XRDL the service methods are described by their name, input and output data. We can conclude that any method described is a valid XML-RPC method if the input and output data types are valid XML-RPC data types. Considering the restriction we added to XRDL in Section 1, any

simple type defined in XRDL will be a valid data type in XML-RPC. So we have to prove the following Lemma:

Lemma 3.2. *Any compound data type defined in XRDL is a valid XML-RPC type.*

For the proof of this Lemma we will assume that the XRDL document respects certain naming conventions specified in the following Proposition:

Proposition 3.3. *The XRDL compound types defined to describe the XML-RPC struct type will reflect the pairs of the struct by a naming convention: one member will always be of type string and its name will start with "_member" and the following member will be of any valid type and its name will start with"_value". The XRDL compound types defined to describe the XML-RPC array type will only contain members named with the word "_member".*

This Proposition ensures the fact that the compound data types defined in XRDL reflect the equivalent complex type from XML-RPC (*struct* or *array*). Otherwise this compound types could not be differentiated, since both XML-RPC complex types, *struct* and *array*, are transcribed in XRDL as a compound data type with an enumeration of members without any structural difference between them.

Considering Proposition 3.3 we can divide the demonstration of Lemma 3.2 into two parts, treating the compound types that contain members named with *"_member"* and *"_value"* separately from the other compound types.

The first case we treat is for compound types that contain members named with *"_member"* and *"_value"*. We consider the following Lemma:

Lemma 3.4. *A compound type from an XRDL document that contains members named with "_value" and has only members of a simple type can be transcribed in XML-RPC into an element of type struct.*

From the hypothesis of Lemma 3.4 we deduce that the compound element will have the general form:

```
<type name="_compoundElement_n">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_n</member>
   <member type="_simpleType_n">_value_n</member>
</type>
```

where *_simpleType_i*, for $\forall i = \overline{1, n}$, will be replaced by any of the types: string, int/i4, double, boolean, dataTime.iso8601 or base64.

For the proof of Lemma 3.4 we will use an induction for $n \in \mathbb{N}$. For the base case n=1 we have a compound element in the form of:

```
<type name="_compoundElement_1">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
</type>
```

This can be transcribed in XML-RPC as a *struct* element as follows:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
</struct>
```

For the inductive step we assume that Lemma 3.4 is true for $n = k$, meaning a compound type with $2 * k$ members can be transcribed as a *struct* element. Let the compound type be:

```
 <type name="_compoundElement_k">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
</type>
```

and the equivalent *struct* element:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
   ...
   <member>
      <name>_member_k</name>
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </member>
</struct>
```

We observe that for the step k+1 the compound type will be:

```
<type name="_compoundElement_k+1">
   <member type="string">_member_1</member>
```

```
   <member type="_simpleType_1">_value_1</member>
   ...


   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
   <member type="string">_member_k+1</member>
   <member type="_simpleType_k+1">_value_k+1</member>
</type>
```

Considering the equivalent *struct* for the compound type with $2 * k$ members we observe that in this case we can extend that struct by one more member in order to get an equivalent for this compund type. So we transcribe it into the following struct:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
   ...
   <member>
      <name>_member_k</name>
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </member>
   <member>
      <name>_member_k+1</name>
      <value>
         <_simpleType_k+1>_value_k+1</_simpleType_k+1>
      </value>
   </member>
</struct>
```

This structure is a valid element in XML-RPC. This concludes the proof of Lemma 3.4.

As a generalization for Lemma 3.4 we have:

Lemma 3.5. *A compound type from an XRDL document that contains members named with "value" can be transcribed in XML-RPC into an element of type* struct.

We observe that Lemma 3.5 looses the restriction from Lemma 3.4 that specified that members can only be of a simple type. To prove Lemma 3.5 we

need to treat the cases of compound types that contain elements of another compound type.

Before proving Lemma 3.5 we need to treat the case of compound types that have equivalents in XML-RPC *array* elements. Let us consider the following Lemma:

Lemma 3.6. *A compound type from an XRDL document that contains only members named with the word "_member" and that has only members of a simple type can be transcribed in XML-RPC into an element of type array.*

Given the hypothesis of this Lemma the general structure of the compound element is:

```
<type name="_n_compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
   ...
   <member type="_simpleType_n">_member_n</member>
</type>
```

where *_simpleType_i*, for any $i = \overline{1,n}$, will be replaced by any of the types: string, int/i4, double, boolean, dataTime.iso8601 or base64. Let us prove this Lemma by applying an induction on n, the number of members of the compound type. For the base case $n = 1$ we have the following compound type:

```
<type name="_1_compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
</type>
```

This can be described in XML-RPC as an *array* element as follows:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </data>
</array>
```

For the induction step we assume that a compound element with k members can be described in XML-RPC as an *array* element and prove that we can also describe a compound element with k+1 members. Let the compound element with k members be:

```
<type name="_k__compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
   ...
   <member type="_simpleType_k">_member_k</member>
</type>
```

and let its equivalent in XML-RPC be:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
      ...
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </data>
</array>
```

We have to prove that we can find an equivalent array element in XML-RPC for a compound type from XRDL with k+1 members. Let that compound type be:

```
 <type name="_k__compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
   ...
   <member type="_simpleType_k">_member_k</member>
   <member type="_simpleType_k+1">_member_k+1</member>
</type>
```

We observe that, considering only the first k members of this compound type, we can build an array in XML-RPC given the previous assumption. We can easily extend that array by one element to obtain an equivalent array for the compound type with k+1 members:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
      ...
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
      <value>
         <_simpleType_k+1>_value_k+1</_simpleType_k+1>
      </value>

   </data>
</array>
```

This concludes the induction that proves Lemma 3.6.

Now we can go back to prove Lemma 3.5. As we mentioned before, in order to prove Lemma 3.5, we need to demonstrate that compound types containing

members named with the word *"_value"* and that contain elements of another compound type have equivalents in XML-RPC as an element of type *struct*.

We observe that a compound type element can include recursively members of other compound types. In the following example *1 compoundArrayElement1* includes *_1_compoundArrayElement2*:

```
<type name="_1_compoundArrayElement1">
   <member type="_1_compoundArrayElement2">_member_1</member>
</type>
<type name="_1_compoundArrayElement2">
   <member type="_simpleType_1">_member_1</member>
</type>
```

We define the dimension of a compound type as the number of compound types involved recursively (including the defined compound type itself) in the definition of that particular compound type. In the previous example *_1_compoundArrayElement1* has dimension 2 and *_1_compoundArrayElement2* has dimension 1.

The proof for Lemma 3.5 can be understood as an induction on $n \in \mathbb{N}$, where n represents the dimension of the compound type. We observe that Lemma 3.4 can be seen as a particular case for Lemma 3.5 with $n = 1$. So the base case for the induction is proven by Lemma 3.4. Furthermore, we know that in XML-RPC we can recursively define elements of type *struct* with any finite dimension. Considering the previous observations a simple induction step proves that any k-dimensional compound type that respects the hypothesis in Lemma 3.5 can be described in XML-RPC as a k-dimensional element of type *struct*. This concludes the proof for Lemma 3.5.

Finally, we consider the following Lemma that treats the general case for Lemma 3.5 without the restriction that members can only be of a simple type.

Lemma 3.7. *A compound type from an XRDL document that contains only members named with the word "_member" can be transcribed in XML-RPC into an element of type array.*

The exact same reasoning used in proving Lemma 3.5 can be applied for proving Lemma 3.7 with the observation that the XML-RPC type *array* can be n-dimensional for any $n \in \mathbb{N}$.

4. Conclusions and future work

Sections 2 and 3 demonstrate that any XML-RPC document can be described by a XRDL document and any XRDL document that respects the restrictions mentioned in this article will also describe a valid XML-RPC service. This leads to the conclusion that XRDL is a valid Description Language for XML-RPC. The higher motivation of this demonstration is to be able to

use XRDL for Web Service Matching [10] and for automatic generation for servers and clients written in different programming laguages.

In a future article we plan to propose an extension to the XSD of XRDL, so that it reflects the restrictions imposed on the *type* attribute in Section 1 of this article.

Many researchers still argue the utility of XRDL as a Description Language for XML-RPC web services. This paper started to prove its utility by proving it is a valid Description Language for XML-RPC services.

For future work we plan to exploit further utilities of XRDL. We will study ways of generating the XRDL description automatically[1]. Furthermore we will find ways of generating a XML-RPC service skeleton using the corresponding XRDL description and ways of generating a client that calls the methods of the XML-RPC service described in a XRDL document[2].

References

[1] F. Boian, B. Jancso, Uniform Solutions for Web Services, Studia Univ. Babe-Bolyai, vol. 57, no. 3, 2012
[2] F. Boian, D. Chice, D. Ciupeiu, D. Homorodean, B. Jancso, A. Ploscar, WSWrapper A Universal Web Service Generator, Studia Univ. Babe-Bolyai, vol. 55, no. 4, 2010
[3] XRDL Project Home, http://code.google.com/p/xrdl/
[4] XML-RPC Specification, http://xmlrpc.scripting.com/spec.html
[5] XML-RPC Data Model, http://www.tutorialspoint.com/xml-rpc/xml_rpc_data_model.htm
[6] XML-RPC, Ken Slonneger, 2006, http://homepage.cs.uiowa.edu/ slonnegr/xml/10.XML-RPC.pdf
[7] XML Schema Tutorial, http://www.w3schools.com/schema/
[8] XML-RPC, http://en.wikipedia.org/wiki/XML-RPC
[9] XML Matters: XML-RPC as object model, David Mertz, 2001, http://www.ibm.com/developerworks/xml/library/x-matters15/index.html
[10] F.M. Boian, A. Ploscar, R.F. Boian, Web Service Matching, Knowledge Engineering Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2013 Cluj-Napoca (Roamania), July 4-6, 2013, this volume

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

E-mail address: dianatroanca@yahoo.com

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

WEB SERVICE MATCHING

FLORIN M. BOIAN(1), ADINA PLOSCAR(1), AND RAREŞ F. BOIAN(1)

Abstract. The paper addresses the heterogeneity of implementing web services of different types on various platforms, and the need for a way to determine whether two web services are matching. Starting with an analysis of the implementation differences and theoretical base for determining matching automatically, the paper proposes a novel architecture. The architecture offers a unified way of designing and creating web services, which eliminates the existing differences plaguing the available frameworks. It also offers support for automatic formal detection of web service matching.

1. Introduction

Web pages were, until recently, the main form in which data was offered on the World Wide Web for human consumption. Therefore the Web is mostly used for browsing using a Web browser to read news/articles, to buy goods and services, to manage on-line accounts and so on.

From a publishing perspective, this is realized by transforming the information from a database, for example, into HTML or similar language so that it can be rendered in a user readable format. Many Web sites put together information extracted by other sites via Web pages, which is an inconsistent process involving decoding and parsing human-readable information not intended for machine processing at all.

Received by the editors: April 17, 2013.

2010 *Mathematics Subject Classification.* 68N25, 68U35.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks – *Distributed systems*; D.2.12 [**Software**]: Software Engineering – *Interoperability*; H.3.5 [**Information Systems**]: Information Storage and Retrieval – *On-line Information Services*; I.2.2 [**Computing Methodologies**]: Artificial Intelligence – *Automatic Programming.*

Key words and phrases. web services, client, matching, XML-RPC, SOAP, REST, XRDL, WSDL, WADL.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

This scenario works well in many cases but it does not support software interactions very well. What is really needed is a mechanism through which the raw data from a database can be accessed in a similar fashion by machines as humans read Web pages now. To reach this goal we need a specialized client that knows how to perform true machine-to-machine communication thus creating a Web that is machine process-able.

In the last 15 years, three models of web services were studied:

- XML-RPC model
- SOAP (+ WSDL and UDDI) model
- REST (RESTfully) model

For all three models, there are many implementations, open source as well as commercial. In this context, four languages came to our attention: Java, C♯, Python and PHP.

2. The Purpose of XRDL, WSDL, WADL

In [4] we presented a method for generating uniform web services and clients. We discussed in that paper the WSWrapper package and how it has as its central component the descriptor generator: XRDL - for XML-RPC services, WSDL for SOAP services, and WADL for REST services. We also proposed there a method for generating client proxies from those descriptors.

A major problem related to Web Services is matching them. There are many web services available on the Internet and APIs offering functionalities that are identical, similar or very close to each other. This leads to a natural attempt to automatically detect such similar services. Studies on the similarity problem are practically impossible to perform without formal descriptors of the services.

Khorasgani et. al. studies in [9] the matching problem for RESTful services. They propose the SFM (Semantic Flow Matching) method for studying the equivalence of two services. Unfortunately, the results are far from being applicable. Their analysis is practically impossible without the use of a WADL.

The WADL[19] purpose is to create a contract between the method exporter (i.e. the service) and the calling clients. However, such a contract is not mandatory [14]. WADL is not a standard and it is not maintained since 2009 but is important in the matching problem. If the service is to be integrated in a complex system, extremely precise communication contracts are mandatory, which turn the WADL in a necessary annoyance.

The matching analysis for SOAP web services is a very complex problem because equivalent services can generate a wide variety of WSDL descriptors, with significant differences between them.

XML-RPC web services are much simpler than SOAP, yet the matching analysis is still very complex. The lack of a formal descriptor makes the matching analysis inapproachable. For the time being, the only proposed descriptor format is XRDL. Unfortunately, the adoption and pro/cons opinions on this format are divided evenly among experts [16].

Our experience in Web service matching is the proposal in [2] of twelve equivalent services: four XML-RPC services, four SOAP services, and four REST services. Every such group of services was implemented in C♯, Java, PHP, and Python. For every group and every language we implemented clients for accessing the services. In [5] we published an extended version of this work that includes Android clients as well.

All web services are identified with their URLs. These have the format: `http://host:port/path` from which `port` and/or `path` may be missing. Working with web services, one quickly realizes that there is a wide variety of constraints and restrictions regarding these URLs. In some cases the `http://` must be present, while in other it doesnt need to be there. In some situations `port` is mandatory, while in others it is not required or even forbidden. In some cases `path` must end with /, in other cases / is forbidden, and so on. Basically, every web service distribution comes with its own conventions.

We will detail this analysis in the next sections.

3. XML-RPC Case Study

The XML-RPC web service used in [2, 5] as a practical example for our work was implemented in every of the four programming languages listed above. The figure 1 shows the XRDL descriptor of this service.

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="Exec" ns="ro.ubbcluj.cs.Exec"
          url="www.scs.ubbcluj.ro:1001">
   <methods>
      <method name="ping" result="string"/>
      <method name="upcase" result="string">
         <param type="string">s</param>
      </method>
      <method name="add" result="i4">
         <param type="i4">a</param>
         <param type="i4">b</param>
      </method>
   </methods>
</service>
```

Figure 1. XRDL descriptor for the Exec web service

The four implementations use very similar XRDLs as the one above, differing only in the values of attributes name, ns, and url of the service element.

The four implementations are:

- C♯ using **XmlRpcCS-1.2**
- Java using **apache-xmlrpc-3.1.3**
- PHP using **xmlrpc-2.2.2** deployed on a web server running a PHP 5 engine
- Python using **xmlrpclib** which is packaged by default in Python 2.7

The localization of these services is relatively consistent. The PHP service is identified by the host of the web service, while the other three create their own web server which listens on the port specified in the URL. The Python implementation requires the specification of both a port and a host address.

The registration of the exported methods is done differently in the implementations. C♯ and Java simply provide the object which implements the methods along with a name under which the clients see this object. In PHP and Python each method is registered along with an associated name to be used by the client.

The PHP implementation is done at the lowest level of all the four languages. Here one needs to specify both on the client and the server the encoding and decoding functions needed for converting the parameters and the messages exchanged between the client and the service.

The clients were implemented on five platforms: C♯, Java, PHP, Python, and Android. Each client gets the URL of the web service which includes the name with which the client identifies it (the name under which the methods are exported). The method calls are then done consistently and without issues.

Although these web services were implemented as equivalent, there is no practical way to automatically prove their equivalence. Consequently, it is necessary to automatically generate the XRDL descriptors which then can be compared in order to prove that the web services match.

4. SOAP Case Study

In the work presented in [5] we used a single web service implemented in multiple programing languages. The methods exported by the web service are the same as those presented in the previous section. The descriptions are similar with those in fig. 1 using the WSDL standard, but instead of `param s` and `param a` we use `param arg0`, and instead of `param b` we use `param arg1`. In the source code of the three web services we changed the definition of method `upcase` to use parameter `arg0` instead of `s`, and in method `add` the two parameters are named `arg0` and `arg1` instead of `a` and `b`. This was necessary due to constraints imposed by some of the clients calling the web

service. The C♯, Python and Android clients, and some PHP clients access the parameters using their *name* while some Java and PHP clients use implicitly declared names such as: *arg0*, *arg1*, *arg2* , and so on. The four services we implemented are:

- C♯ using a file of type ***.asmx** on an IIS web server[2].
- Java using apache-cxf-2.2.7 of type **JAX-WS2**.
- PHP using **nusoap** 0.9.5 with the web service deployed on a PHP 5 engine.
- Python using **spyne**. In [2] we used **jkp-soaplib** which could not be accessed from C♯ and Java.

The localization of these services is relatively consistent. The C♯ and PHP web services are identified by their host web servers. Java publishes an object Endpoint which is configured with the service URL and a reference to the object which exports the methods. PHP and Python require the developer to specify the `targetNamespace` parameter and also an application name.

The method registration is done differently among the implementations. In C♯ and Java the service object is annotated as `WebService`, and the exported methods are annotated as `WebMethod`. The web service Python extends the standard `spyne.service.ServiceBase` object, and the methods are prefixed with the `@rpc` decorator. PHP requires that each method be registered individually along with its prototype and namespace.

The client needs to be treated independently for each service to be accessed.

For each C♯ or Java client that accesses a web service we must generate the source code of a static proxy that accesses the web service. Thus there are four C♯ clients and four Java clients created for every of the web services involved in our analysis. The proxies are generated in a consistent manner. In C♯ this is done using the WSDL utility on the web service URL. The utility generates the C♯ code of the client and compiles it into a DLL which is integrated in the client.

The client sources differ usually only through the line that defines the proxy object named by us service. The definition of these proxies for each web service is given below:

- `Ss1Serv service = new Ss1Serv(); // for C♯`
- `Sj1ServService service = new Sj1ServService(); // for Java`
- `Sh1Serv service = new Sh1Serv(); // for PHP`
- `Sy1Serv service = new Sy1Serv(); // for Python`

This proves that our proposal in [1, 4] to use a single `WebServiceClient` object for any platform is feasible.

The Java clients are of two types: JAX-WS for the C♯, Java and Python web services; and AXIS1 for PHP. This is because `nusoap` does not support (yet) JAX-WS. The JAX-WS proxies are generated using the `wsimport` utility based on the web service URL. The generated sources are compiled and built into a JAR which is then integrated in the client. For PHP, the proxy generation is done using the `Wsdl2Java` utility. As for C♯, the clients differ only in the proxy definition line:

- `Ss1ServSoap service =`
  `(new Ss1Serv()).getSs1ServSoap(); //for C♯`
- `Sjj1Proxy.Sj1Serv service =`
  `(new Sj1ServService()).getSj1ServPort(); //for Java`
- `Sh1ServPortType service =`
  `(new Sh1ServLocator()).getSh1ServPort(); //for PHP`
- `Application service =`
  `(new Sy1Serv()).getApplication();// for Python`

These similarities reinforce the feasibility of our [4] proposal for using a unified client object for any platform.

There are two types of PHP clients. The **nusoap** clients can only access **nusoap** web services. The PHP 5 distribution supports the creation of SOAP clients using the PHP SOAP module which is part of the core PHP distribution. This library is good enough for building clients, but not sufficiently strong yet for building web service. The proxy definition lines are:

- `$service = new nusoap_client($urlServ . "?wsdl", true);`
  `//for PHP nusoap`
- `$service = new SoapClient($urlServ . "?wsdl");`
  `//for any service`

The SOAP PHP clients calling services other than PHP **nusoap** require a special parameters transmission approach: the developer must create a PHP object containing the parameters as properties and these are then accessed either by name or by value.

The SOAP Python clients are the simplest of all. The suds project is a recently released Python distribution specialized on web service clients. The suds library creates dynamically a proxy object starting from the WSDL, without generating any source code. This aspect simplifies significantly the client implementation.

For writing Android web service clients, the only library available is ksoap2. Essentially the client requires the WSDL and gets from there the `targetNamespace` parameter. If this parameter is not specified, then the client goes directly to the web service, gets the WSSDL from there and extracts the `targetNamespace` from it. This peculiar behavior is necessary because there is a problem in

the connection to the Python web service. The connection fails if the client fetches the WSDL from the web service, extracts the `targetNamespace` parameter and then starts calling methods. However everything works fine if the `targetNamespace` is specified directly to the client. The explanation is that the Python web service enforces stateless access, which means that a WSDL delivery closes the connection.

5. REST Case Study

The service used in [2, 5] as example implemented in several programming languages is used here as well. The fig. 2 presents WADL for describing this service.

The four web services implement the same functionalities as those in the previous sections. To support a wider range of examples we choose to make the `upcase` method accessible through both GET and POST calls, and the `add` method through both PUT and DELETE calls. The implemented web services are:

- C♯ using a ***.ashx** file deployed in an IIS web server.
- Java using the JAX-RS **jersey**[17] library deployed in a servlet container.
- PHP using the **Da Silva distribution** [6] which offers two objects: `RestServer.php` and `RestClient.php` which support method mapping using URL regular expressions.
- Python using the **CherryPy** [18] library.

The localization of these services is variable, depending on the method used to pass the parameters: as a query string at the end of a GET method, in the body of a POST or DELETE method, or as variables in the URL path.

The method registration is done differently among the web service implementations. In C♯ the method is given control upon the completion of an analysis of the path in the `HttpContext` URL. In Java the methods to be exported are annotated as such. In PHP the mapping of each method is specified in the `RestServer` object. In Python the methods are annotated using the `@cherrypy.expose` decorator.

The clients were implemented in: C♯, Java, PHP, Python, and Android. Each of them gets the service URL and then the service is accessed simply over URL connections: `HttpWebRequest` and `HttpWebResponse` for C♯, `HttpUrlConnection` in Java, using CURL and possibly the Da Silva `RestClient` for PHP, and the standard `urllib` and `httplib` modules of Python.

Although these services were built to be equivalent, there is no practical way to demonstrate their equivalence automatically. To do so, we need to generate the WADL descriptors and compare them to prove the equivalence.

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
   xmlns:tns="urn:yahoo:yn" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:yn="urn:yahoo:yn" xmlns:ya="urn:yahoo:api"
   xmlns="http://wadl.dev.java.net/2009/02">

   <resources base="http://www.scs.ubbcluj.ro/Exec">
      <resource path="Rj1Serv/ping">
         <method name="GET" id="ping">
            <request />
            <response status="200">
               <representation mediaType="application/xml"
                  element="xsd:string" />
            </response>
         </method>
      </resource>
      <resource path="Rj1Serv/upcase">
         <method name="GET" id="upcaseQ">
            <param name="s" type="xsd:string" style="query"
               required="true" />
            <response status="200">
               <representation mediaType="application/xml"
                  element="xsd:string" />
            </response>
         </method>
      </resource>
      <resource path="Rj1Serv/upcase/{s}">
         <method name="POST" id="upcaseP">
            <param name="s" type="xsd:string" style="path"
               required="true" />
            <response status="200">
               <representation mediaType="application/xml"
                  element="xsd:string" />
            </response>
         </method>
      </resource>
      <resource path="Rj1Serv/add">
         <method name="PUT" id="addB">
            <param name="a" type="xsd:int" style="body"
               required="true" />
            <param name="a" type="xsd:int" style="body"
               required="true" />
            <response status="200">
               <representation mediaType="application/xml"
                  element="xsd:int" />
            </response>
         </method>
      </resource>
      <resource path="Rj1Serv/add/{a}/{b}">
         <method name="DELETE" id="addP">
            <param name="a" type="xsd:int" style="body"
               required="true" />
            <param name="a" type="xsd:int" style="body"
               required="true" />
            <response status="200">
               <representation mediaType="application/xml"
                  element="xsd:int" />
            </response>
         </method>
      </resource>
   </resources>
</application>
```

FIGURE 2. WADL descriptor of the Exec web service

6. Our Proposal

Considering the aspects presented above we propose to extend the WSWrapper published in [1], as shown in the fig. 3.

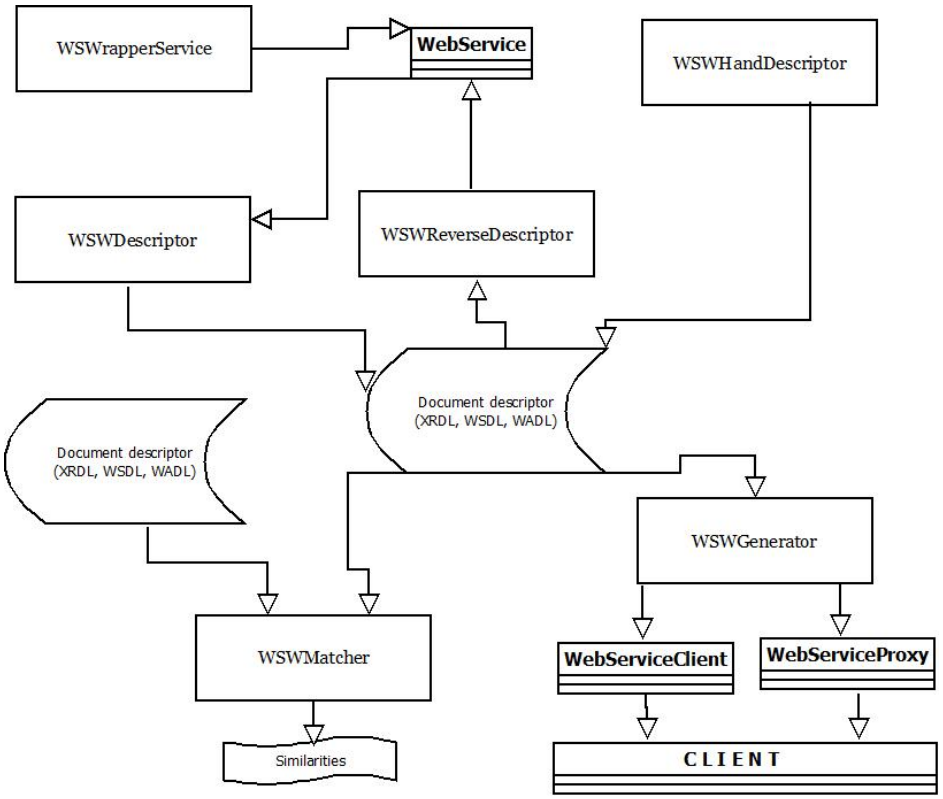


Figure 3. Extended WSWrapper architecture

The main components of this new architecture are:

WSWrapperService processes the **WebDescriptorService** descriptors and generates the actual web service for the desired platform.

WSWDescriptor processes WebService objects using reflection (inspection, introspection) and, depending on the web service type, generates a descriptor of type XRDL, WSDL, or WADL. This is possible because all the languages involved support reflection, and specifically introspection, which is essential to the extraction of meta-data, exported methods, and calling paths, from the web service objects. [12, 13, 15].

WSWHandDescriptor is an auxiliary manual tool which allows the designer to customize the generated descriptor as necessary.

WSWReverseDescriptor generates source code from a given descriptor of any of the three supported types: XRDL, WSDL, or WADL. The generated source code will contain the method prototypes for the web service on a desired platform.

WSWGenerator [8] generates a static proxy based on a web service descriptor. The proxy will be specific to the web service it was generated for and to the platform being used. The proxy is then packaged in a library (i.e. **dll**, **jar**, **phar**, **zip**) which will become part of the client.

WSWMatcher takes as input two web service descriptors and detects the similarities between them, trying to determine whether they match or not.

As web services of different types have different descriptions and even various WSDL files may eventually have (and often do) different descriptions for equivalent services we have the following proposition. We describe each service regardless of the type of the web service in an abstract format that we will call WSAD (Web Service Description Abstract). This WSAD will describe only web method name, parameters whose types must comply with XML types and return type of the method with the same restrictions as for parameters. Because it is an abstract description of the service, the location of the web service is not needed.

WSWMatcher takes as input two WSAD files describing two web services and will compare them syntactically. The matching that we propose is at a syntactic level, and at this point in our research we do not consider the semantic. If two services match in a syntactically way we can continue their matching semantically.

7. Conclusions

The paper presents a detailed analysis of the differences between web service implementations on various platforms and the theoretical possibilities for automatically determining whether web services are matching. The proposed architecture unifies web service development and eliminates the differences which get in the way when using the existing frameworks. It also provides support for automatically detecting web service matching.

References

[1] Boian F., Chinces D., Ciupeiu D., Homorodean D., Jancso B., Ploscar A., *WSWrapper - A Universal Web Service Generator*, Studia Universitatis Babes-Bolyai Series Informatica, Volum LV, nr. 4, 2010, pp 59-69, ISSN: 2065-9601

[2] Boian F.M., *Servicii web; modele, platforme, aplicaii*, Ed. Albastra, Cluj, 2011

[3] Boian F.M., *A uniform approach to define and implement the web services; case studies for indexing huge file systems*, ZAC2012, pp 85-90

[4] Boian F.M., Jancso B., *Uniform solutions for web services*, Studia Universitatis Babes-Bolyai Series Informatica, Volum LVII, nr. 3, 2012, pp 13-23
[5] Boian F.M., http://www.cs.ubbcluj.ro/ florin/books/SWMPA/*New.zip, 2013
[6] DaSilva S.D., *PHP Classes*, http://diogok.users.phpclasses.org/browse/author/529977.html
[7] Jancso B., *RESTful Web Services*, ZAC2010, pp. 158-163
[8] Jancso B., *Web Service proxy Generator*, ZAC2012, pp 124-129
[9] Khorasgani R.R., Stroulia E., Zaiane O.R., *Web service Matching for RESTful Web Services*, http://webdocs.cs.ualberta.ca/ zaiane/postscript/wse2011.pdf
[10] Ploscar A., *A Java Implementation for REST-style web service*, ZAC2010, pp 140-146
[11] Takase T. Makino S. Kawanaka S. Ueno C.F. Ryman A. *Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0.*, http://www.ibm.com/developerworks/library/specification/ws-wadlwsdl/index.html
[12] * * * http://scripts.incutio.com/xmlrpc/introspection.html
[13] * * * http://xmlrpc-c.sourceforge.net/introspection.html
[14] * * * http://stackoverflow.com/questions/1312087/what-is-the-reason-for-using-wadl/1314357♯1314357
[15] * * * http://www.codeproject.com/Articles/235269/Using-Introspection-in-Java, C♯ reflection Java reflection PHP reflection Python inspect
[16] * * * XRDL: XML-RPC Description Language. http://code.google.com/p/xrdl/
[17] * * * http://jersey.java.net/
[18] * * * http://www.cherrypy.org/
[19] * * * http://www.w3.org/Submission/wadl/

(1) Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania
E-mail address: florin@cs.ubbcluj.ro
E-mail address: adina_ploscar@yahoo.com
E-mail address: rares@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

NETWORK INTERFACE AGGREGATION FOR IP TUNNELING

ADRIAN SEREDINSCHI[(1)] AND ADRIAN STERCA[(1)]

Abstract. We present in this paper a network interface aggregation framework for IP tunneling. Our framework periodically measures the level of congestion on routes going through each network interface and based on these measurements, it decides how many flows to send on an outgoing network interface at a time. In this way, we can achieve better inter-flow fairness regarding bandwidth utilization.

1. Introduction. IP Multihoming and IP Tunnels

IP multihoming means any form of providing reliable network connectivity to a service point in the network. It implies two or more redundant network connections from this node to the core network. IP multihoming can be realized in a static setup (i.e. when specific flows are statically mapped/routed to a specific uplink interface) or in a more dynamic setup using BGP.

IP tunnels are used to transparently connect two distant networks that normally do not have a native routing path between them. There are several types of IP tunnels, from simple ones like IP-in-IP encapsulation to GRE and to IPSec tunnels which create a virtual private network composed of two remote sites.

In this work we would like to combine multihoming routing algorithms with IP tunnels in order to provide reliable, load-balanced virtual private network links between two remote sites. More specific, the network architecture we consider is depicted in Figure 1.

Received by the editors: April 30, 2013.

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

1998 *CR Categories and Descriptors.* C.2.2 [**Computer-Communication Networks**]: Subtopic – *Network Protocols*; C.2.3 [**Computer-Communication Networks**]: Subtopic – *Network Operations*.

Key words and phrases. network interface aggregation, IP tunneling, IP multihoming.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

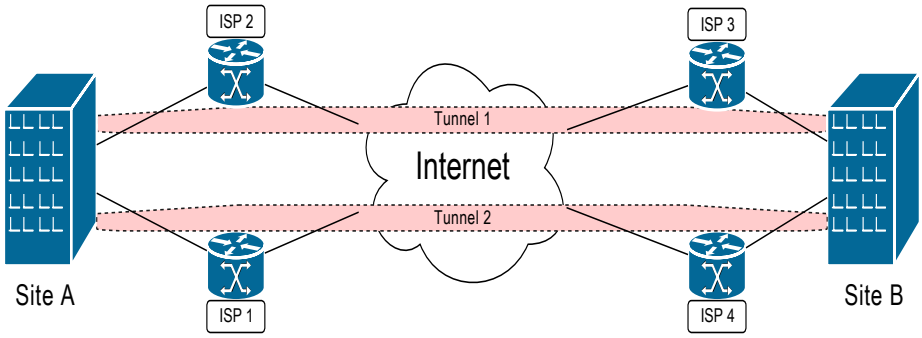


FIGURE 1. The network architecture

We consider a network setup where two remote local area networks denoted in the figure by *Site A* and *Site B* have each several uplink connections to the Internet (generally, we consider a small number of network links, two or three, per network site) and we want to connect them in a virtual private network. For this, we need an IP tunnel (i.e. IPSec or something else) between these two sites, but we also want to spread the outgoing tunnel traffic on all uplink connections of a site, so that the network bandwidth of a site is optimally used. In Figure 1, *Site A* has two uplink connections, one through ISP1 and the other through ISP2 and *Site B* has also two uplink connections, one through ISP3 and the other through ISP4. There are four possible network paths between *Site A* and *Site B*: a path going through ISP2 and ISP3, another path going through ISP1 and ISP4, another path going through ISP2 and ISP4 and, finally, the last path going through ISP1 and ISP3. We depicted in Figure 1 and consider in our framework only the first two paths because we want the network paths to be as independent as possible (i.e. to have a number of common links close to zero), at least theoretically. Of course, in reality there is no guarantee that the network route going through ISP2 and ISP3 and the network route going through ISP1 and ISP4 do not share any intermediate network link. When the network paths between *Site A* and *Site B* share a large number of network links our algorithm would be less effective, but this situation will be discussed further in section 3. Our algorithm routes packets sent by the local area network through the tunnel logical interface on one of the existing paths from the source site to the destination site based on the level of congestion experienced by each network path. The goal is to achieve a better utilization of the network bandwidth (between the source site and the destination site) and to achieve a higher degree of inter-flow fairness between the flows originating from the same network site and sharing the tunnel.

The rest of the paper is organized as follows. In the next section related work is reviewed. Then, section 3 describes the main algorithms of our framework of network interface aggregation for IP tunnels followed by section 4 which shows some experiments performed using our framework which will validate its usefulness. The paper ends with conclusions in section 5.

2. Related work

Dynamic IP multihoming is normally realized using BGP [1]. But since BGP is a general distance-vector routing protocol and BGP routing tables are normally quite large, BGP makes its route selection decision based on local preferences or weights and hop-count costs and it does not use real-time link state information like the congestion level of a whole route in this selection process; this is because it is very costly to probe every possible destination network it knows and it does not have control over the whole network path.

Policy-based routing is another technique used for IP multihoming. In policy-based routing, the next hop is determined based on the source IP address of the packet and routing policies are set by a human operator. A more general form of policy-based routing is source based routing in which the network route on which a packet will travel is specified partially or totally in the packet header. Source based routing is especially used in the context of wireless ad hoc networks, but it requires modifications in the TCP/IP stack currently deployed in Internet routers [2], [3].

Another routing technique related to our framework is multipath routing used also in the context of wireless ad hoc networks [4]. In multipath routing a flow's packets are sent to the destination over multiple routes, possibly overlapping, in order to improve the efficiency of the transport. An important problem with multipath routing is that sending a flow over different network routes can cause many out of order packets at the receiver which can degrade the throughput of the upper level protocol, i.e. TCP.

All the above techniques function at level 3 in the TCP/IP stack and they are all meant to increase the throughput efficiency of the data at the router. Similar approaches exist also at level 2, namely Ethernet link aggregation/bounding (LACP protocol in IEEE 802.1ax or open-source Linux bounding driver).

The same idea of using different network paths between the source network and the destination network depending on the level of congestion on each path is explored at a higher level, level 4 (i.e. the transport level), in Multipath-TCP [5] where a TCP flow is split over several routes depending on the level of congestion on each route. In Multipath-TCP selecting the current route for a packet is done in an end-to-end fashion at the source end node, while in

our case is done inside the network by the source router which does not split a single TCP flow on different routes, but considers all the flows originating from the source network/site. Also, in Multipath-TCP an end host relies on some support from the network (i.e. from network routers) in order to route the packet on a specific route established by the source end node.

Probably, the work closest to ours is presented in [6]. In [6] authors present a multi-homing solution using IP tunnels in order to achieve higher throughput for TCP flows originating on a specific site and having a specific destination site. TCP flows are split over several network interfaces at IP level so that the resulted total bandwidth is the bandwidth aggregated over these network interfaces. Our work is different than the one presented in [6], because we estimate differently the congestion (i.e. available bandwidth) on each network path and our paper considers all flows originating in the source sites while the author of [6] only deal with splitting a TCP flow across the outgoing network links.

3. Network Interface Aggregation for IP Tunneling

The goal of our network interface aggregation framework for IP tunneling is to split outgoing traffic on the IP tunnel so that a link that is more congested than other should receive less traffic in order to maintaining a high degree of inter-flow fairness among all flows originating at the same source. In order to achieve this, our framework periodically measures the current level of congestion of each outgoing link and sends a corresponding fraction of the total traffic on this link.

The architecture of our framework is depicted in Figure 2. The framework runs at both sides of the tunnel and at each side it is composed symmetrically from two modules:

- IP mangler module
- statistics gatherer module

The IP mangler module is responsible with choosing an outgoing link for the current packet. It chooses an outgoing link for a packet by setting (mangling) the source IP address and the destination IP address in the outer IP header of the packet. Using the statistics gatherer module, our framework computes a congestion metric for each (considered) route between the source site and the destination site. If we consider the routes from Figure 1 and we consider that the route going through ISP2 and ISP3 has the congestion metric $cong\_metric\_0$ and the route going through ISP1 and ISP4 has the congestion metric $cong\_metric\_1$ then $traffic\_weight\_0$ of the total traffic will be sent on the route ISP2-ISP3 and $traffic\_weight\_1$ of the total traffic will be sent on the route ISP1-ISP4 where

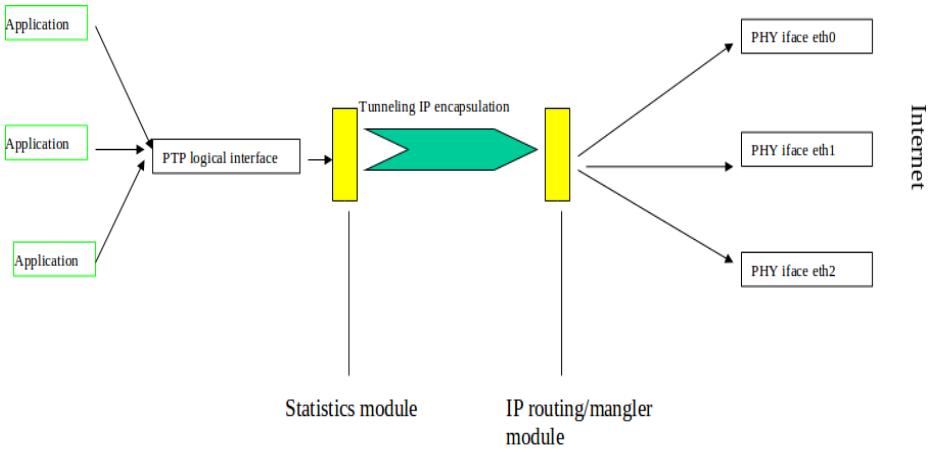


FIGURE 2. Our IP Tunneling Aggregation Framework

$$traffic\_weight\_0 = 1 - \frac{cong\_metric\_0}{cong\_metric\_0 + cong\_metric\_1}$$

$$traffic\_weight\_1 = 1 - \frac{cong\_metric\_1}{cong\_metric\_0 + cong\_metric\_1}$$

and $traffic\_weight\_0, traffic\_weight\_1 \in [0,1]$ and $traffic\_weight\_0 + traffic\_weight\_1 = 1$.

Considering that there are N flows going out through the tunnel and all flows are non-limited TCP flows (i.e. they use as much bandwidth as it is available), because we want to send, as much as possible, all the packets of a flow through the same outgoing link (to avoid packet reordering), we approximate $traffic\_weight\_0$ of the total traffic with $traffic\_weight\_0 \cdot N$ flows and $traffic\_weight\_1$ of the total traffic with $traffic\_weight\_1 \cdot N$ flows. In order to ensure a correct functioning of the statistics gatherer module, we enforce that there exists at least one flow on each outgoing link (if the total number of flows is greater than 1).

The statistics gatherer module monitors the congestion level on the outgoing links of the site. This module uses passive measurements on the incoming traffic in order to determine the level of congestion. More specifically it probes incoming acknowledgment packets on each network interface for the timestamp value in the TCP header (i.e. the TSecr field) and it measures the RTT on that interface. Computing the current congestion level on a link is done in a way similar to how TCP Vegas does [7]. For each link a smoothed

weighted average value of the round-trip time is kept in the state variable *srtt*:

$$srtt = srtt \cdot 0.975 + rtt\_curr \cdot 0.025$$

where *rtt_curr* is an approximation of the current RTT obtained by subtracting the TSecr value of a packet from the current time. The congestion level on a link is computed like this:

$$cong\_metric = cong\_metric \cdot 0.9 + (srtt - min\_srtt)^{+} \cdot 0.1$$

where *min_srtt* is the minimum *srtt* value measured so far on this interface.

The statistics gatherer module relies on the fact that all the packets of a flow (i.e. data packets and acknowledgment packets) follow (approximately) the same outgoing route, so that the RTT measurement per link is consistent. For this reason, our framework maintains a list of connections/flows for each outgoing network interface. An entry in this connections list (i.e. a flow) contains the following data:

- source IP address
- destination IP address
- source port
- destination port
- switch flag

where the first four items are taken from the inner packet, after the outer IP and Ethernet headers are removed and the switch flag is set to 'true' if this flow is sent through a different network route by the remote site (e.g. the flag is set to 'true' if site A sends a flow/connection through the route ISP2 - ISP3 and site B sends the same flow/connection (i.e. the ACK packets) back through the ISP4 - ISP1 route in Figure 1). The switch flag is used by our framework when a new decision to reallocate flows on network interfaces is taken so that if some flows need to be moved from one interface to the other, the flows having the switch flag set to true are chosen first.

From time to time, depending on the traffic weights set by the statistics gatherer module and if the congestion level changed on at least one of the routes, the IP mangler module reallocates flows on the network interfaces (i.e. moves flows from the connections list of an interface to the connections list of another interface and mangles the IP addresses of every packet belonging to that flow accordingly). The IP mangler module reallocates flows on interfaces if the following conditions are met for at least one outgoing network interface:

$$|srtt - min\_srtt| > 0.5 \cdot min\_srtt$$

$$|srtt\_last\_reallocation - srtt| > min\_srtt$$

where *min_srtt* is the minimum SRTT value measured on that network interface, ever and *srtt_last_reallocation* is the *srtt* measured on this interface

when the last reallocation took place. When both conditions from above are met for an interface, approximately $traffic\_weight\_0 - traffic\_weight\_1$ flows are moved from interface 1 to interface 0 if $traffic\_weight\_0 > traffic\_weight\_1$ and $traffic\_weight\_1 - traffic\_weight\_0$ flows are moved from interface 0 to interface 1 if $traffic\_weight\_0 < traffic\_weight\_1$.

A final observation about our framework's functioning is needed and that is the initialization of the framework's state. When started, our framework does not have initial values for the $srtt, cong\_metric$ and $traffic\_weight$ variables for any interface, so the IP mangler module distributes flows equally among existing network interfaces. Once there is at least 1 flow on each interface, the IP aggregation framework can compute these values and it can distribute flows on interfaces according to the level of congestion on each route/interface. The *srtt* value is computed only from packets belonging to flows that follow the same network path in both directions (i.e. flows that have the switch flag set to 'false') - this is required for the measurement to be relevant for that network path.

4. Experiments

In order to evaluate our network interface aggregation framework, we have set up a test scenario as depicted in Figure 3. We have two machines, Site A and Site B, connected to the network by 2 network interfaces each and a router with 4 network interfaces which connects Site A and Site B. An IP-in-IP tunnel is established between Site A and Site B. We started 32 TCP senders on Site A and 32 TCP receivers on Site B and on each flow we sent random data as fast as the network allows from site A to site B. We used a linux traffic shaper on the network interfaces eth2 and eth3 of the Router to limit the bandwidth on each route (i.e. simulate congestion). The traffic shaper alternates the bandwidth of network interfaces eth2 and eth3 of the Router between 1500 Kbits/sec and 3000 Kbits/sec at 30 seconds time intervals. The bandwidth pattern is the following: in the first 30 seconds of a cycle both interfaces are limited to 3000 Kbits/sec each, in the next 30 seconds eth0 is limited to 1500 Kbits/sec while eth1 is still at 3000 Kbits/sec, in the following 30 seconds both interface have again 3000 Kbits/sec, and in the last 30-second period of a cycle interface eth1 is limited to 1500 Kbits/sec and interface eth0 keeps sending at 3000 Kbits/sec. This cycle of 4 states repeates indefinitely.

We have run 2 experiments on this network setup, each lasting for 300 seconds. In the first experiment we have used a static algorithm for splitting the 32 TCP flows at site A on 2 network interfaces; this static algorithm sends 16 flows on one interface/route and the remaining 16 flows on another interface/route and it never realocates the flows on a different interface. In the

second experiment we have used our framework at site A to perform dynamic reallocation of flows on outgoing interfaces depending on the level of congestion on each route. We were only interested in the traffic flowing from site A to site B, in this direction only. We measured at site A the flows per interface allocation, the average RTT (i.e. *srtt*) measured on each interface in both experiments and the average bandwidth per second received by a flow on each interface in both experiments. Our findings are shown in Figures 4, 5, 6 and 7.

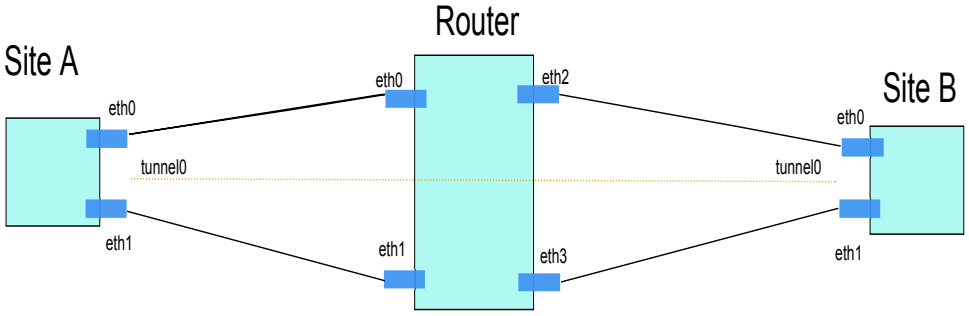


FIGURE 3. Network setup used for evaluation

The *rtt* and *srtt* values measured on interface eth0 of site A in the dynamic reallocation experiment is pictured in Figure 4. The values of *rtt* and *srtt* measured on interface eth0 of A in the static allocation experiment is shown in Figure 5. In both experiments, similar values of *rtt* and *srtt* were obtained on interface eth1 of A, so these plots are not included here. The *srtt* values are measured in milliseconds. It can be seen in figures 4 and 5 that out framework does not change significantly the RTT and SRTT measurement on a network interface.

Figure 6 depicts the allocation of flows per interface performed by our network interface aggregation framework at site A.

For each of the two experiments, we have computed the average bandwidth obtained by a flow on each of the two network interfaces (i.e. interfaces eth0 and eth1 of site A). Then we computed a bandwidth-allocation-per-flow fairness index which is equal to $\frac{min\_bw_i}{max\_bw_i}$ where min_bw_i is the minimum of the two bandwidth-per-flow values on each network interface in second i and max_bw_i is the maximum of the two bandwidth-per-flow values (on each network interface) in the same second i.

The bandwidth allocation per flow fairness index is depicted as a function of time in Figure 7 for each of the two experiments: the experiment when our

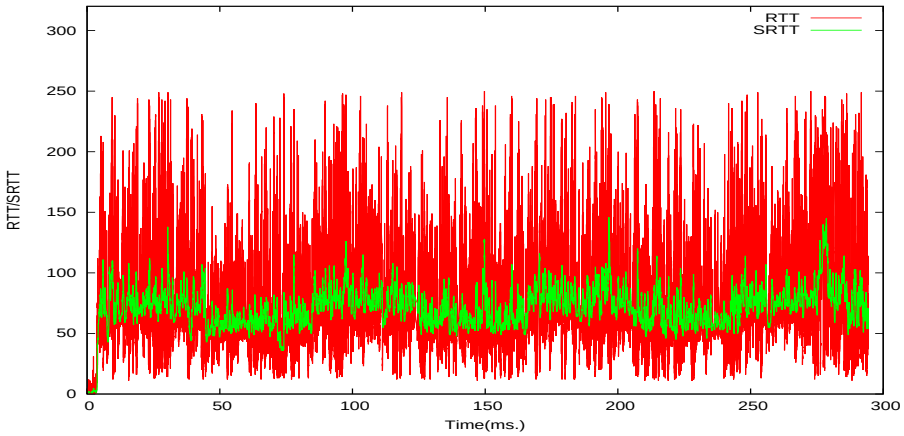


FIGURE 4. The *rtt* and *srtt* measured on interface eth0 of A when our framework was used

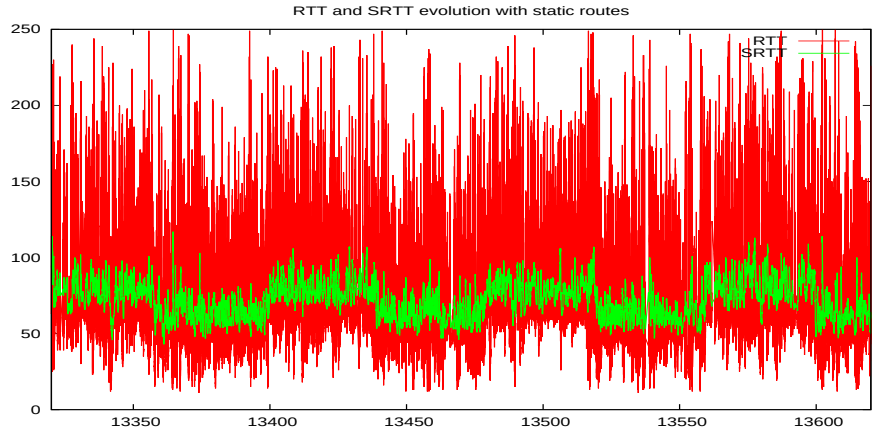


FIGURE 5. The *rtt* and *srtt* measured on interface eth0 of A when static allocation of flows was used

framework was used and the experiment when static allocation of flows was employed (i.e. 16 flows on interface eth0 and 16 flows on interface eth1).

We can see that, in general, our framework attains a higher Bandwidth-per-Flow fairness index and on average, across all 300 seconds of the experiments, our framework obtains a Bandwidth-per-Flow fairness index of 0.843688 while the static allocation scenario obtains a values of 0.741942, so an improvement of slightly over 10%.

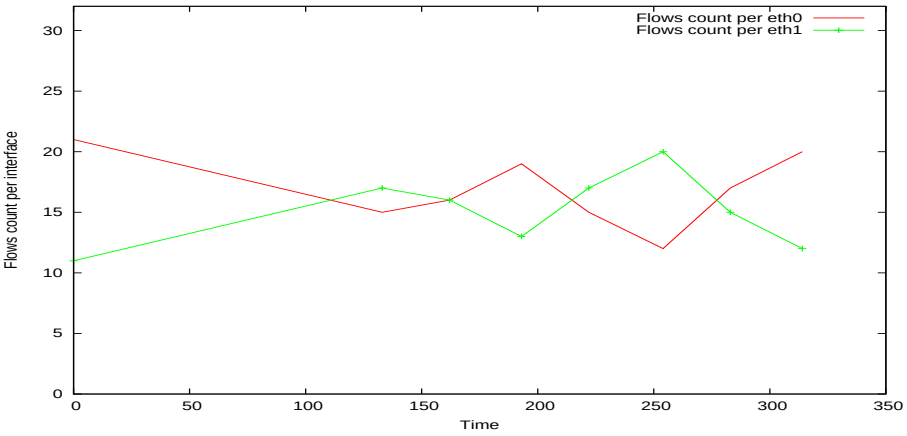


FIGURE 6. Distribution of flows on interfaces eth0 and eth1 of A when our framework was used

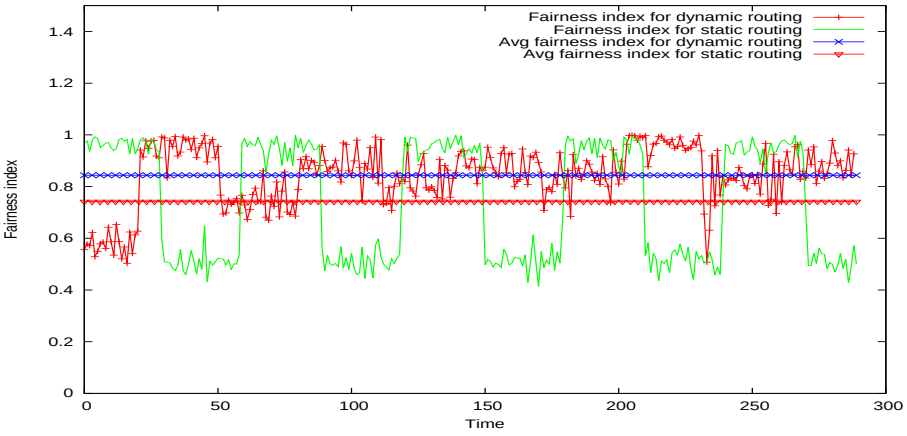


FIGURE 7. Bandwidth-per-Flow fairness index obtained by our framework vs. static allocation scenario

5. CONCLUSION

We have presented in this paper a framework for network interface aggregation for IP tunneling. Our framework periodically measures the level of congestion on routes going through each network interface and based on these measurements, it decides how many flows to send on an outgoing network interface at a time. The utility of our framework is validated in section 4.

References

[1] Y. Rekhter, T. Li, S. Hares, *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, January 2006.

[2] Andrews, M., Fernandez, A., Goel, A., Zhang, L., *Source routing and scheduling in packet networks* Journal of the ACM (JACM), 52, pp. 582-601, 2005.

[3] Johnson, D. B., Maltz, D. A., Broch, J., *DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks*, in Ad hoc networking, 5, pp. 139-172, 2001.

[4] Nasipuri, A., Castaneda, R., Das, S. R., *Performance of multipath routing for on-demand protocols in mobile ad hoc networks*, in Mobile Networks and applications, 6(4), pp. 339-349, 2001.

[5] Han, H., Shakkottai, S., Hollot, C. V., Srikant, R. and Towsley D., *Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet* in IEEE/ACM Transactions on Networking 14, 6, pp. 1260-1271, 2006.

[6] Phatak, D. S., Goff, T., Plusquellic, J., *IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping*, in Computer Networks, 43(6), pp. 787-804, 2003.

[7] Lawrence S. Brakmo , Sean W. Omalley , Larry L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*, in Proc. of ACM SIGCOMM Conference, 1994.

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

E-mail address: sdsd0494@scs.ubbcluj.ro

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