Software Reliability in Semantic Web Service Composition Applications

Liviu Adrian COTFAS, Andreea DIOSTEANU
Academy of Economic Studies, Bucharest, Romania
liviu.cotfas@ase.ro, andreea.diosteanu@ie.ase.ro

Web Service Composition allows the development of easily reconfigurable applications that can be quickly adapted to business changes. Due to the shift in paradigm from traditional systems, new approaches are needed in order to evaluate the reliability of web service composition applications. In this paper we present an approach based on intelligent agents for semi-automatic composition as well as methods for assessing reliability. Abstract web services, corresponding to a group of services that accomplishes a specific functionality are used as a mean of assuring better system reliability. The model can be extended with other Quality of Services – QoS attributes.
Keywords: *Software Reliability, Web Service Composition, Intelligent Agents*

1 Introduction

Nowadays, due to the rapid development of the IT&C sector, almost every activity is dependent on and influenced by the technological progress. Therefore, it became a necessity to carefully monitor and evaluate the functionality of both hardware and software. The evaluation consists in determining the probability for a system to function correctly in a predefined context, which includes specified conditions and constraints. This probability is known as system reliability. One of the greatest challenges for researchers is to create models so that to evaluate the level of system reliability. Because, in essence, the reliability evaluation function is a probability based function, the researchers concentrated their efforts in determining suitable statistical models based on probability density for estimating software reliability [1]. Initially, models were empirical and heavily relied on the observation of software failure. Such approaches required a great amount of data in order to offer fairly good predictions. More recent research efforts have led to the development of reliability models addressing fault coverage, testing coverage, and imperfect debugging processes [2].

Many approaches in the last years have focused on developing statistical models, based on Markov model, Halstead's Software Metric, McCabe's Cyclamate Complexity Metric, Poisson probability theory, time series so that to extend the statistical models and identify the dynamics of the software product reliability, semi quasi renewal, etc.

This paper proposes a methodology for evaluating and assuring reliability in semantic web service composition applications based on the concept of abstract web service. The rest of the paper is organized as follows. Section 2 presents several approaches and indicators used when assessing software reliability. The relation between reliability and the Software Development Lifecycle – SDLC is discussed in Section 3. Section 4 presents existing web service composition approaches. An approach based on intelligent agent search is presented in Section 5, followed in section 6 by a model for assessing reliability in composite web service applications.

2 Existing Models for Software Reliability

The reliability definitions given in the literature vary between different practitioners as well as researchers. The generally accepted definition is as follows. *Reliability* is the probability of success or the probability that the system will perform its intended function under specified design limits. More specific, reliability is the probability that a product or part will operate properly for a specified period of time (design life) under the design operating conditions (such as temperature, volt, *etc*.) without failure [2]. In other words, reliability may be used as a measure of the system's success in providing its function

properly. Mathematically, reliability can be expressed as a time dependent probability, where $R(t)$ is the probability that the system will be successful from the beginning until the moment t.

$$R(t) = P(T > t) \quad (1)$$

where:
T – represents the failure time;
In terms of probability density the reliability function can be expressed as follows:

$$R(t) = \int_t^{\infty} f(s)ds \quad (2)$$

where:
f(t) – density function for the failure time T;

$$f(t) = -\frac{d}{dt}[R(t)] \quad (3)$$

In many cases such as when assessing the reliability of a web service, evaluating reliability based on failure time is not the best option. Evaluating reliability as the number or failures versus number of calls can offer a better indication when evaluating web services. In both approaches $Rel(t)$ takes values in the interval [0,1].
As an alternative to the reliability function [3] developed a new mathematical function called systemability, considering the uncertainty of the operational environments in the function for predicting the reliability of systems. Systemability is defined as the probability that the system will perform its intended function for a specified mission time under the random operational environments. In a mathematical form, the systemabililty function is given by:

$$R_s(t) = \int_{\eta} e^{-\eta \int_0^t h(s)ds} dG(\eta) \quad (4)$$

where:
$h(t)$ – hazard rate function;
η – common environment factor;
$G(\eta)$ – cumulative distribution function of η;
Reliability is one of the quality characteristics that consumers require from the product manufacturers. More than 30 factors influencing an application's reliability have been identified including the Difficulty of Programming - PDIF, the Level of Programming Technologies - TLVL and the Percentage of Reused Code – PORC.
Software reliability is commonly considered when assessing the Quality of Service - QoS associated to an application. Other aspects taken into consideration when evaluating QoS include: availability, accessibility, integrity, performance and security [4].
Given the importance of the topic a large number of approaches have been proposed. In 1991, the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) adopted ISO/IEC 9126 as the standard for software quality. The standard specifies external and internal metrics that can be used to evaluate software products. Software quality attributes are divided in six groups (functionality, reliability, usability, efficiency, maintainability, and portability), which are further subdivided into sub characteristics. Software reliability evaluation can take into account metrics such as maturity metrics, fault tolerance metrics, recoverability metrics and compliance metrics. Some of the most common reliability measure indicators are [2]:

A. System Mean Time to Failure

SMTTF measure is one of the most widely used reliability calculations, but also one of the most misused calculations. It has been misinterpreted as "guaranteed minimum lifetime".

$$SMTTF = \int_0^{\infty} tf(t)dt \quad (5)$$

where:
$f(t)$ – density function for the failure time T;

B. Maintainability

Maintainability is defined as the probability that a failed system will be restored to the specified conditions within a given period of time when maintenance is performed according to prescribed procedures and resources. In other words, maintainability is the probability of isolating and repairing a fault in a system within a given time. Maintainability engineers must work with system designers

to ensure that the system product can be maintained by the customer efficiently and cost effectively. This function requires the analysis of part removal, replacement, tear-down, and build-up of the product in order to determine the required time to carry out the operation, the necessary skill, the type of support equipment and the documentation.

$$M(t) = \int_0^\infty rest(ts)ds \quad (6)$$

where:
$rest(t)$ – density function for the restore time T;

C. Availability

Since, reliability is a measure that requires system success for complete time span that is for the entire functioning time of the system. Over this period of time no changes or fixes are allowed. Availability is a measure that allows for a system to repair when failure occurs.

$$A = \frac{t_{working}}{t_{working}+t_{down}} \quad (7)$$

where:
$t_{working}$ – period of time in which the system is functioning;
t_{down} – period of time in which the system is not functioning;

3 Assuring Reliability in the Software Development Lifecycle - SDLC

Even if in theory the reliability evaluation process doesn't seem very complicated, the development of sound, complex and accurate reliability evaluation systems requires a lot of effort and research. This is due to the fact that each and every software product has its own particularities, flows, architecture, etc. Environmental factors play a very important part as they illustrate the usefulness of the model and its applicability.
However, the reliability models prove their value only if they are taken into account by software developers. Furthermore, the reliability models have an increased level of usability that is users are capable to read and understand their results. If this thing does not happen, the reliability evaluation model can bring disadvantages in the sense that users lose their trust in the software product, thus leading to sever economic and technical problems.

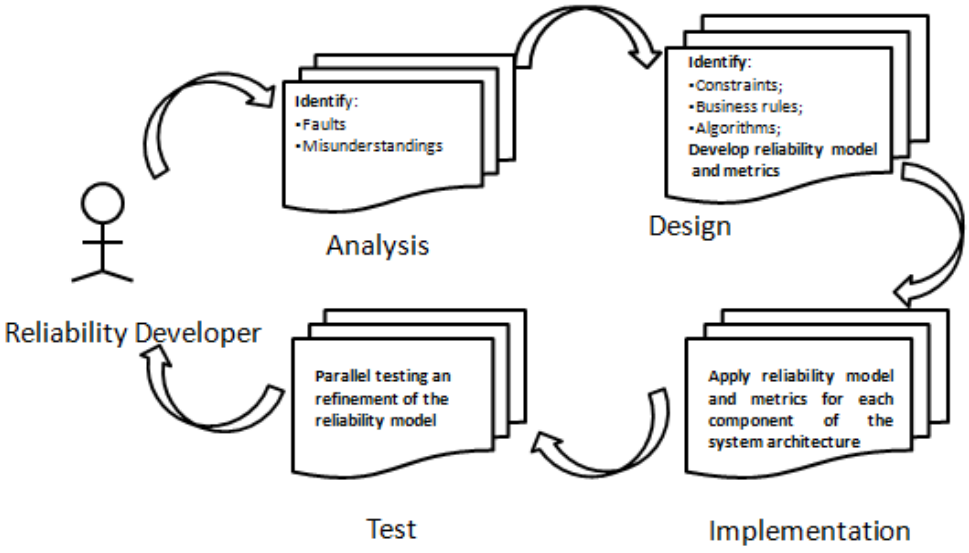

Fig. 1. Methodology for reliability evaluation in connection with software product lifecycle

Therefore, developers need to first understand the software product life cycle and, for each phase, to perform empirical observations that lead to the reliability model:

- **ANALYSIS**: this is the most important phase because it has as output the specifications based on which the software product is developed. During this phase, the reliability models developers have to clearly understand the specifications and for each item they have to identify possible faults or misunderstandings with regard to the business logic.
- **DESIGN**: consists of translating the business specifications into technical ones, to identify and create algorithms and models. This phase has two stages: general system architecture and detailed architecture. In the general system architecture, designers create the overall architecture of the product, identify the main components and provide a description of the manner in which the components interact with each other. In the detailed design they analyse each component and provide a complete description of the algorithm the component will implement, constraints, requirements, etc. During this phase the reliability developers have to identify the business rules, constraints, algorithms and create already a theoretic model that will evaluate the risk of failure for each independent component and also for the entire product as a whole, considered based on the interaction of independent components. Furthermore, in this phase test cases and test scenarios are also developed.
- **IMPLEMENTATION**: this phase consists in writing the appropriate code. With regard to reliability, the metrics and models developed in the previous phase have to be applied after each component is completed. If results are not satisfactory, then code review and code refectory are required.
- **TEST**: Over this phase, the test cases and scenarios are applied. Furthermore, the reliability developers perform a parallel testing and refining of the reliability evaluation model

Figure 1 summarizes the main steps of the proposed methodology in relation to the software development life cycle. In web service based application it is very important to have a good understanding of the system both as a whole, but also of each individual component. If this is achieved, the reliability evaluation model will be consistent.

4 Web Service Composition

While individual web services can be used to accomplish specific tasks, there is a growing need to integrate multiple services into complex chains of web services. Applications based on this approach, offer business the possibility to quickly reconfigure their software systems to take advantage of new market opportunities. Compared to traditional software approaches, web service composition allows modifying the functionality of an application just by changing the involved web services, without having to rewrite the code of the application, thus reducing the need to wait for long software release cycles or for internal software development projects. Besides avoiding long development times, associated development costs are also greatly reduced [5]. Complex applications can be seen as collections of independent web services provided by different companies. Interoperability is thus an important aspect as fully integrated traditional enterprises are being replaced by business networks in which every company is specialized in certain services or products.

Existing approaches for web service composition can be classified in:

- Manual - Is considered both time-consuming and error-prone. The person performing the composition should have advanced domain knowledge [6].
- Semi-automatic - The user is involved to ensure the validity of the generated flow at various stages of the composition process. This usually assures a higher QoS.
- Automatic - In this approach the user intervention is minim and the system has to perform all validations.

Semi-automatic and automatic approaches are mostly used. In these approaches the system identifies a chain of web services that together accomplish a certain task. The main approaches are using workflows, templates or AI planning. The workflow approach re-

quires extensive domain knowledge and implies a certain degree of manual implementation [7]. The template approaches relies on OWL-S to store an outline of the required actions [8]. Once created, templates have the advantage of being reusable. AI planning approaches are usually automatic so the user is not required to have prior domain knowledge. Such techniques try to automatically generate the web service chain based on the initial state and the desired goal state using forward-chaining [9], backward-chaining [10]. Petri-net based algebra, space search, graph based planning, HTN - Hierarchical Task Network planning, approaches based on logical programming and others can also be used [11].

Existing software reliability models were developed for statically built software applications. Given the differences between monolithic approaches and approaches based on creating applications from web services many techniques developed for traditional software, including the software reliability models are no longer valid.

A common problem in the above mentioned approaches is that multiple web services might offer the requested functionality. Usually one of the services is selected based on cost, performance criteria, reputation or a combination of these attributes [12], [13].

In order to increase the overall web service chain reliability we propose considering them as built from abstract services. In the discovery step, besides selecting the web service which offers the best characteristics we also store all the web services that offer the required functionality. If for any reason the best web service will not be available at runtime, one of the other web services will be used.

An *Abstract Service* can be defined as a group of web services that provide a specific functionality. The abstract service is considered to fail only if all the services in the group fail. A similar approach is taken in [5]. Therefore, the reliability of an abstract service can be computed as:

$$R_{abstract} = 1 - \prod_{i=1}^{p}(1 - R(ws_i)) \quad (8)$$

where p is the number of web services that offer the required functionality. If $p > 1$ then $R_{abstract}$ is closer to 1 than the reliability of any of the web services in the corresponding group. In other words, $R_{abstract} \geq R(ws_i)$.

5 Intelligent Agents Web Service Composition

We summarize bellow a web service composition approach based on intelligent agents search that we presented in [14]. The main components of the platform are presented in Figure 2.

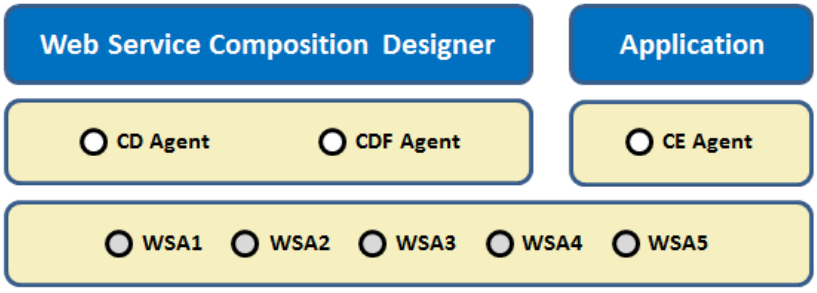

Fig. 2. Agent based Web Service Composition Framework

The Web Service Composition Designer allows the user to create and modify web service chains in an interactive manner. The Composition Designer – CD Agent assists the composition process. The Composition Directory Facilitator – CDF Agent maintains a list of all available agents. Applications that want to execute a specific chain of web services use the Composition Execution – CE Agent. In order to facilitate the search, all web services are represented as agents with the associated input and output characteris-

tics, named WSA.

The design of the newly created chain starts in the WSC Designer Module where the user has the possibility to either create a completely new chain or to modify an existing one. From the user's perspective the web service chain is composed from a succession of actions or sub-goals that must be performed in order to achieve a certain goal.

Step 1: The CDA agent semantically queries the CDF agent searching for Web Service Agents that can perform the requested action. The CDA agent stores the list of all found agents and begins a parallel negotiation.

Step 2: Every candidate WSA agent compares the request's input data with the internally mapped web service or web service chain input parameters. In case any input parameters don't match, the WSA agent will itself query the CDF agent in order to find an agent or a chain of agents capable of performing the required transformation. If still needed the found agents can themselves repeat the procedure. In order to limit the search space a maximum number of agents used to model an action can be specified.

Step 3: Each caller agent, including the CDA verifies the correspondence between the called WSA web service output parameters and the requested output parameters. In case any output parameters don't match, an agent capable of performing the required transformation will be searched using the CDF agent similar to Step 3.

Step 4: All found services and chains are displayed in the WSC Designer together with their estimated reliability and execution time.

The following section of the paper expands upon reliability evaluation of the web service composition chains.

The resulting chain is stored so it can either be executed using the Composition Execution – CE Agent or incorporated in new chains. A corresponding WSA agent is created and registered with the CDF agent.

6 Evaluating Reliability in Web Service Composition Applications

The overall reliability of a system depends on the reliability of its subsystems, which in turn is influenced by the reliability of the components and the reliability of the connections between them. The expected Quality of Service - QoS is an important criterion when building a web service chain in most approaches. It is commonly used when several services matching the requirements a found. In order to select one of the several services QoS is taken into consideration. Different approaches are employed to select the services in a manner that will assure the highest global QoS. [15] proposes an algorithm for web service chaining in which the optimization method is based on Multi-objective Chaos Ant Colony Optimization - MCACO that offers better results than previous approaches based on Multi-objective Genetic Algorithms - MOGA [16]. Some approaches propose extending the existing standards to incorporate reliability information. One such extension to the Web Service Description Language – WSDL is called Q-WSDL (QoS enabled WSDL) presented in [17] that adds QoS information to the standard WSDL files.

The reliability prediction of the composite service can be carried out:

- At design time – static prediction. The reliability of the web service chain is calculated at design time and the best web services are selected. This approach offers a better performance by performing the evaluation at design time, rather than slowing down the execution phase. Complex algorithms can be used to assess the reliability;
- At execution time – dynamic prediction. The reliability prediction and selection of web services is performed in the execution phase. An advantage over the previous approach is that dynamic prediction is not affected if the available web services change over time. The downside is related to the additional execution time, which leads to the necessity of using less complex algorithms.

In our approach a mixed evaluation model is used. Reliability is first evaluated at design time when creating the web service chain in order to choose the most reliable web service chain. Storing all web services that match the

required functionality allows changing at runtime the selected web service if it is no longer available. Thus, the reliability of the web service chain is improved without slowing down the execution.

Based on the reliability value for each web service, the aggregate reliability can be calculated as shown in [17]. The formulae were introduced for evaluating web services, but can be used to evaluate *Abstract Services* as well:

- R_{Seq} – Reliability for a sequence:

$$R_{Seq} = \prod_{i=1}^{n} R(Block_i) \qquad (9)$$

where n is the number of blocks in the sequence. Blocks can be either individual web services or chains composed of multiple web services

- R_{Switch} – Reliability for switch blocks:

$$R_{Switch} = R_{SwitchTest} * \sum_{i=1}^{m} p_i * R(Branch_i) \quad (10)$$

where m is the number of branches, $R_{SwitchTest}$ is the reliability of the condition test and p_i is the probability that branch i will be selected. Therefore $\sum_{i=1}^{m} p_i = 1$. For a switch with only two branches, the formula can be rewritten:

$$R_{Switch} = R_{SwitchTest} * \sum_{i=1}^{n} [p * R(Branch_{True}) + (1-p) * R(Branch_{False})] \quad (11)$$

- R_{Loop} – Reliability for loop blocks:

$$R_{Loop} = (R_{CondTest} * R_{LoopBlock})^{LoopCount} \quad (12)$$

where $R_{CondTest}$ is the reliability of the loop condition, $R_{LoopBlock}$ is the reliability of the repeated block and $LoopCount$ is the number of times the block will be executed.

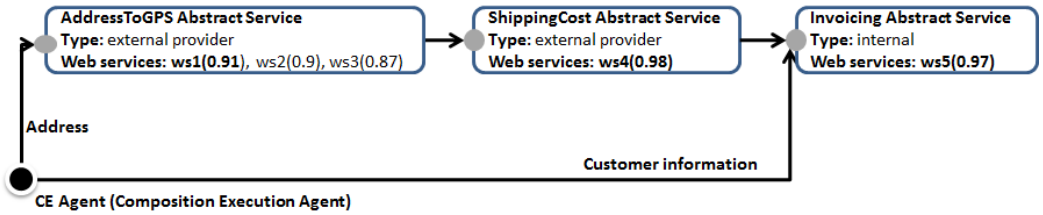

Fig. 3. Web Service Composition Chain Example

Figure 3 presents a simple web service chain that generates invoices based on address where products will be shipped. The first two web services are from external providers, while invoicing service is owned by the company. For each abstract service, matching web services are stored together with their associated reliability. If only the best service was stored and the framework was not able to use an alternative service in case of failure, the reliability would be equal to $R = 0.91 * 0.98 * 0.97 = 0.87$. In our approach, based on the fact that multiple services providing a translation between an address and GPS coordinates have been found, the reliability equals

$$R = [1 - (1 - 0.91) * (1 - 0.9) * (1 - 0.87)] * 0.98 * 0.97 = 0.95.$$

A drawback of the proposed approach is that in some cases there can be only one web service that matches the functionality requirements. In such a situation the system can check the list of web services added after the web service chain was designed to see if a

matching service can be found. If no service can be found, the execution of the chain will fail.

Additionally if the application based on web service composition will be used in a mobile environment, several other reliability issues should be taken into consideration [18]:

- Unreliable communication translates in possible loss of signal during the execution of the web service chain or variations in the connection speed.
- Unreliable location information can affect applications that require the users' location in order to provide customized information.
- Many methods used for reliability prediction rely on past information regarding the system's behavior. Given the fact that in mobile applications such as location based services – LBS the context is variable both in time and space, past information might not allow a good prediction of the system's behavior in the future.

7 Conclusions

In this paper we presented a model for assessing reliability in web service composition applications based on intelligent agents. We plan to extend the model in the future to take other parameters into consideration such as price and availability. We take into consideration the implementation of a multi objective genetic algorithm as an approach to select the web services from each group in order to optimize the whole chain based on the selected QoS attributes. Genetic algorithms would allow us to take an unlimited number of criteria into consideration, by adding parameters to the evaluation function. Assuring a transactional behavior for the entire chain is another important aspect including the possibility to roll back the effects of the web service chain if an error occurs.

Acknowledgement

This article is a result of the project „Doctoral Program and PhD Students in the education research and innovation triangle". This project is co funded by European Social Fund through The Sectorial Operational Program for Human Resources Development 2007-2013, coordinated by The Bucharest Academy of Economic Studies.

References

[1] X. Zhang, H. Pham, "An analysis of factors affecting software reliability", *Journal of Systems and Software,* vol. 50, pp. 43-56, 2000.

[2] H. Pham, *System software reliability*. Springer, 2006.

[3] H. Pham, "A new generalized systemability model", *International Journal of Performability Engineering*, vol. 1, no. 2, pp. 145-155, 2005.

[4] G. Canfora, M.D. Penta, and R. Esposito, "An approach for QoS-aware service composition based on genetic algorithms", *Genetic And Evolutionary Computation Conference*, pp. 1069-1075, 2005.

[5] H. Zo and D. Nazareth, "Measuring reliability of applications composed of web services", *System Sciences, 2007. HICSS*, pp. 1-10, 2007.

[6] R. Mohanty, V. Ravi, and M.R. Patra, "Web-services classification using intelligent techniques", *Expert Systems with Applications*, vol. 37, pp. 5484-5490, Jul. 2010.

[7] J. Pathak, S. Basu, R. Lutz, and V. Honavar, "MoSCoE: A Framework for Modeling Web Service Composition and Execution", *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, Ieee, p. x143-x143, 2006.

[8] G. Jie, C. Bo, C. Junliang, and Z. Lei, "A Template-Based Orchestration Framework for Hybrid Services", *2008 Fourth Advanced International Conference on Telecommunications*, pp. 315-320, 2008.

[9] R. Peng, Z. Mi, and L. Wang, "Research on Converged Service Composition Based on Extending CSTA Services with OWL-S", *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1-4, Sep. 2009.

[10] H.N. Talantikite, D. Aissani, and N.

Boudjlida, "Semantic annotations for web services discovery and composition", *Computer Standards & Interfaces*, vol. 31, pp. 1108-1117, 2009.

[11] V. Agarwal, G. Chafle, S. Mittal, and B. Srivastava, "Understanding approaches for web service composition and execution," *Proceedings of the 1st Bangalore annual Compute conference on - Compute '08*, 2008, pp. 1-8.

[12] J. Yan and J. Piao, "Towards QoS-Based Web Services Discovery", *Lecture Notes In Computer Science*, p. 200–210, 2009.

[13] J. El Hadad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS-Aware Selection Algorithm for Automatic Web Service Composition", *IEEE Transactions on Services Computing*, vol. 3, pp. 73-85, Jan. 2010.

[14] L.A. Cotfas, A. Diosteanu, and I. Smeureanu, "Fractal web service composition framework", *2010 8th International Conference on Communications*, Bucharest: IEEE, pp. 405-408, 2010.

[15] W. Li, "A Web Service Composition Algorithm Based on Global QoS Optimizing with MOCACO", *Algorithms and Architectures for Parallel Processing*, pp. 218-224, 2010.

[16] S. Liu, Y. Liu, N. Jing, and G. Tang, "A dynamic web service selection strategy with QoS global optimization based on multi-objective genetic algorithm", *Grid and Cooperative Computing*, vol. 3795, pp. 84 – 89, 2005.

[17] P. Bocciarelli and A. D'Ambrogio, "A model-driven method for describing and predicting the reliability of composite services", *Software & Systems Modeling*, Feb. 2010.

[18] S. Malek, R. Roshandel, D. Kilgore, and I. Elhag, "Improving the reliability of mobile software systems through continuous analysis and proactive reconfiguration", *2009 31st International Conference on Software Engineering - Companion Volume*, pp. 275-278, May. 2009.

Liviu Adrian COTFAS is a Ph.D. student and a graduate of the Faculty of Cybernetics, Statistics and Economic Informatics. He is currently conducting research in Economic Informatics at Bucharest Academy of Economic Studies and he is also a Pre-Assistant Lecturer within the Department of Economic Informatics. Amongst his fields of interest are location based services, geographic information systems, genetic algorithms and web technologies.

Andreea DIOŞTEANU has graduated the Faculty of Economic Cybernetics, Statistics and Informatics in 2008 as promotion leader, with an average of 10. She is currently conducting research in Economic Informatics at Bucharest Academy of Economic Studies. She is .NET programmer at TotalSoft. During the bachelor years she participated in many student competitions both at national and international level obtaining a lot of first and second prizes. The most important competitions she was finalist in were Microsoft International Imagine Cup Competition, Software Design section (national finalist); Berkley University and IBM sponsored ICUBE competition were she qualified for the South Eastern Phase-Novatech. Furthermore, she also obtained the "N.N Constantinescu" excellence scholarship in 2007-2008 for the entire student research activity.