Jakub Grzesiak
Łukasz Jędrychowski

MODEL OF RECONFIGURATION IN COMPONENT ENVIRONMENTS

Abstract *The significance of component-based software and component platforms has increased over the last twenty years. To achieve full flexibility, a reconfiguration process is needed that allows us to change system parameters without rebuilding or restarting. In terms of components, such a process should be executed with extraordinary care, since the contracts between components must be preserved. In this article, a model of reconfiguration (including the roles of the components to be used in this process) is proposed. The provided solution is general and could be applied to many component platforms.*

Keywords component, DSL, reconfiguration

Citation Computer Science 16 (1) 2015: 3–15

1. Introduction

The significance of component-based software and component platforms has increased over the last 20 years [15]. Currently, most enterprise applications are developed for environments such as Microsoft .NET, Java EE, or lightweight component frameworks (e.g., Spring). To achieve full flexibility, a reconfiguration process is needed that allows us to change the parameters of a system without rebuilding or restarting. In terms of components, such a process should be executed with extraordinary care, since the contracts between components must be preserved.

The main goal of the reconfiguration process is to realize changes in configuration without rebuilding or restarting an application. It is necessary that, after reconfiguration, all constraints are still met and the system works properly.

There are many reasons why the reconfiguration process is necessary. One of the most important is to customize an application that depends on changes in the environment. Many configuration change requests come from a user or another system, implied from functional or non-functional requirements. In many cases, rebuilding or restarting an application is more expensive than processing a reconfiguration. In terms of high-availability systems, such processes are often impossible. The reconfiguration process should preserve consistency (e.g., contracts between components) and should not trigger any side-effects. Existing solutions are not general, as most of them are useful only for particular components and are lacking in functionality. We have introduced a more-general solution that could be applied to many component platforms.

The structure of the article is described below. In section 1, we introduce the problem. Then in section 2, we present technologies which are currently available. In section 3 and 4, we describe the model of reconfiguration and the framework architecture. In sections 5, 6, and 7, we present the effects of our research.

2. Reconfiguration frameworks – the state of the art

The most important theoretical issues related to reconfiguration are described in article *Issues in the Runtime Modification of Software Architectures* [14]. It lists the characteristics of dynamic architectures and shows the challenges of implementation. Two cases can be distinguished when a new module is required in a system. The first is reactive, due to issues during runtime. The second is proactive – when a new module is an extension of functionality. The introduction of a dynamic architecture requires the coverage of three topics: the time when reconfiguration can be processed, which operations are available, and what constraints are imposed on the system and its components.

The key issue that should be solved is component substitution. This requires a mechanism to check whether or not components are equivalent. This problem is comprehensively described in *Component Reconfiguration in Presence of Mismatch* [4]. The authors proposed a solution to the problems associated with non-compliant modules. This consists of three stages: detection, compatibility check, and substitution. Another approach is based on adapters. It assumes that a module has a set of provided and required operations. In the case of a component, it will be as a contract. An adapter-based solution is more flexible, because it is automatically generated using a component's description with LTS (Labeled Transition System) [5]. To create a more-effective solution, reconfiguration could use only context equivalence between components instead of full equivalence.

To support the reconfiguration of component-based software, some specialized frameworks have been developed. They differ in their scopes of functionality and the technologies upon which they are built. Due to different reconfiguration models, the variety of technologies, and limitations, there is no universal framework for all purposes.

Peyman Oreizy described general concepts of reconfiguration and introducd a prototype solution named ArchShell [14]. It supports the creation and run-time modification of software. ArchShell provides a command line interface for creating and starting systems as well as changing their architecture. A user can use it in a similar fashion as a typical UNIX console. Elements of system architecture are C2 [7] components (implemented in Java) that use connectors to pass messages. ArchShell supports multi-threading, processes, and mechanisms of an operating system.

A second interesting approach was described by Batista et al. in [1]. It is a meta-framework named Plastik, which uses reconfiguration and architecture description language based on ACME [6] and components in OpenCOM [12] model. Plastik supports two types of reconfiguration:

- programmed, in which changes are defined during design of a system as a predicate-action pair; reconfiguration is fired when a given state (described by a predicate) is achieved;
- ad-hoc, in which changes are unpredictable during the architecture design phase; a user can only define constraints which should be met for all components; reconfiguration could be processed only when the integrity and consistency are preserved.

Fractal [3] is a hierarchically-structured component model that provides reflective features which support dynamic architectural reconfiguration. It was developed by OW2 Consortium. It supports design, implementation, and reconfiguration of a wide class of systems. Components in Fractal consist of two parts: a controller and a content (internal subcomponents). A controller is responsible for the implementation of the subcomponents' semantic. In a particular case, it allows components to be reconfigured. Reconfiguration is processed by an introspection mechanism.

OSGi technology is a set of specifications that defines a dynamic component system for Java [11]. It reduces software complexity by providing a modular architecture for large-scale distributed systems as well as small, embedded applications. OSGi has a few developed implementations: Apache Felix, Eclipse Equinox, and Knopflerfish. The OSGi component model is a dynamic model. Bundles (OSGi component) can be installed, started, stopped, updated, and uninstalled without bringing down the whole system.

Existing solutions are not general. Most of them have many restrictions (e.g., a given component platform) and lacks functionality. The details are described below.

ArchShell [14] assumes that all components and connectors are written in Java, using the C2 [7] framework.

Plastik [1] is an ADL/Component runtime integration meta-framework. In theory, it offers a platform-independent language to describe the software architecture by using an extension of ACME/Armani [6] and mechanisms for loading the application on the underlying platform. However, it is implemented only for the OpenCOM [12] component model. Moreover, components are not allowed to make reconfiguration decisions.

Fractal [3] does not support ad-hoc reconfiguration. Fractal also does not have formal support to ensure consistency.

OSGi is the most-advanced solution. However, it still requires components to follow its own specification. This could be hard to achieve, especially for applications which use class loaders directly or have extremely high coupling between components.

Due to limitations connected with a specific component platform as well as those described above, we decided to propose a general model that could be applied to almost every component environment. In this article, a model of reconfiguration as well as component roles to be used in the process are proposed. The provided solution is general and could be applied to many component platforms. AgE Reconfiguration Framework is provided as an example. It combines the concept of constrained run-time reconfiguration [14] and supports both programmed (via domain-specific language) and ad-hoc reconfiguration [1].

3. Reconfiguration

We use the following notion in this article. We assume that a **component** is an independent unit of deployment that provides a set of functionalities exposed by interfaces. A **component definition** consists of parameters which specify the behavior of a given **component instances** which are nothing more than just deployed components. We distinguished two different scopes:

- singleton – only one instance with a given configuration exists in a system,
- prototype – many instances with a given configuration exist in a system.

Reconfiguration is a process in which component definitions and parameters of singleton-scope component instances are changed. It is possible to define the following elements of this process:

a) change of a property which is simple type of component platform (e.g., Boolean, Integer, Double, String) – parameters of an instance are updated,
b) change of a reference to another component – references from one instance to another are modified,

c) change of a property which is a collection in the component platform,
d) dynamically add or remove instance properties,
e) update a definition of a component in an instance provider (which is a component definition repository) – as defined in a), b), and c),
f) replace an implementation of a component with another one.

Reconfiguration needs to be processed in sync with an instance provider to maintain consistency of components and instances.

From the user's point of view, we can define the following requirements for a Reconfiguration Framework. A user of the component platform (that supports Reconfiguration Framework – proposed in the next section) does not always want to rebuild or restart the application in order to use it with new parameters. The user would like to define the structure and a reconfiguration of the components that are the base of the application. For example, by a declaration in Domain Specific Languages (DSL): Architecture Description Language (ADL, for structure) and Reconfiguration Language (RL, for reconfiguration) files.

Also, the authors of the components would like to define constraints for any given properties. For example, the user may decide that a numeric property needs to have a value within a given range which may break contracts between components. It should be possible to implement a custom code which handles reconfiguration parameters and returns a boolean value determining if all constraints are met. The framework should not allow the user to change a property to a value which breaks the registered constraints. In such a case, an exception should be thrown. This concept is a variation of the Bertrand Meyer's *Design by Contract* [13]. In the original approach, we have preconditions, postconditions, and invariants specified for each component. In the proposed solution, we would be assured that all of them are still met after reconfiguration by evaluating some custom constraint logic. It will not be necessary to create a specific data structure for handling preconditions, postconditions, and invariants.

4. Reconfiguration Framework

We distinguished the following elements of the Reconfiguration Framework as presented in Figure 1.

A **Diff** is used as a representation of component reconfiguration. It contains names and values of properties. In the case when property with a given name exists – a new value is assigned; otherwise, the new property would be added to the component. From an implementation point of view, a Diff could aggregate a dictionary/map structure in which property names are keys and new property values are associated with keys.

A **Transaction** is a reconfiguration that could affect many components. It should be processed against all components or none at all, according to the ACID properties (atomicity, consistency, isolation, and durability). Technically, it is a collection of Diff objects. When something goes wrong during the execution of a transaction, the framework will create a reversed transaction based on the initial configuration. A reversed transaction tries to undo changes made by the original one.

A **reconfigurable component** needs only to implement a simple interface (IReconfigurable) via one of two methods: one which gives information if reconfiguration is possible, and another that enables a component to be notified about processed reconfiguration. If the component platform does not support a change of parameters internally, then the IReconfigurable interface should be extended with appropriate methods for updating component properties.

All reconfigurable objects need to be registered within **Reconfiguration Manager**. This is responsible for processing all reconfigurations. ReconfiguraitonManager reconfigures components and also updates their definitions.

Reconfiguration Provider delivers reconfigurations to a selected ReconfigurationManager in an appropriate time (for example, when computation ends). ReconfigurationProvider contains all necessary information about reconfigurations.

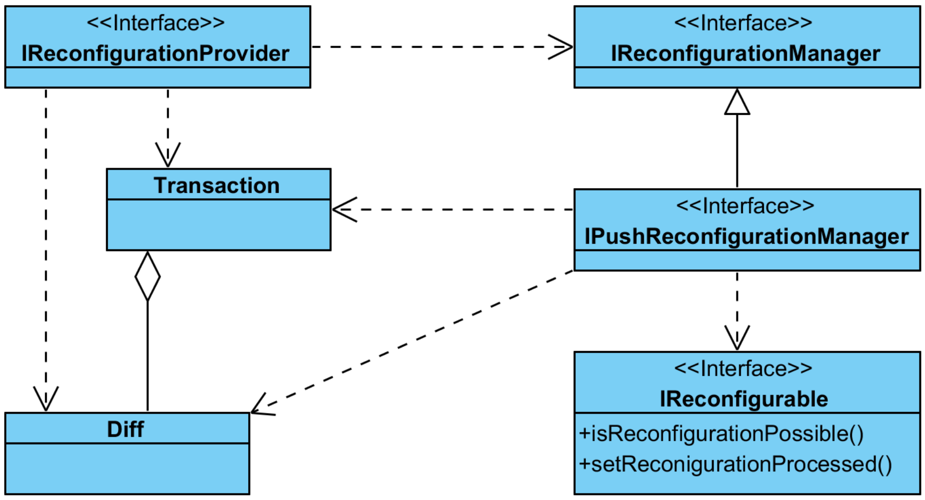

Figure 1. Reconfiguration Framework class diagram.

A reconfiguration process is initiated by the Reconfiguration Framework after the initiating event has occurred and the application is working. The framework checks whether the components are ready to change their parameters. It is possible to postpone or reject the process. Reconfiguration is performed unless constraints are not broken; otherwise, the next reconfiguration is processed or the application is stopped. The first case is only valid for high-availability systems, and the second is valid only for certain kinds of computations. For each reconfiguration for a given component, all provided constraints are checked. The Reconfiguration Framework is responsible for notifying components about definition changes and updating values or references of their properties. After reconfiguration is finished, the application is working with the new parameters.

The provided solution implements **constrained run-time push reconfiguration model** and is handled by **PushReconfigurationManager**. Constrained, because the framework does not allow properties to be changed to forbidden values/references. Run-time as a reconfiguration is processed when the application is working. Push, because the framework is responsible for process initiation, component notification, and updating both instances and definitions. Authors of components have to provide methods for deciding whether to reconfigure or not, changing parameters (if component platform does not support it internally), and processing necessary operations after reconfigurations are finished. They could also define constraints.

Theoretically, it is possible to implement components in such a way that the reconfiguration process could be initiated and handled by them. In this case, the Reconfiguration Framework supports only the notification and update of definitions.

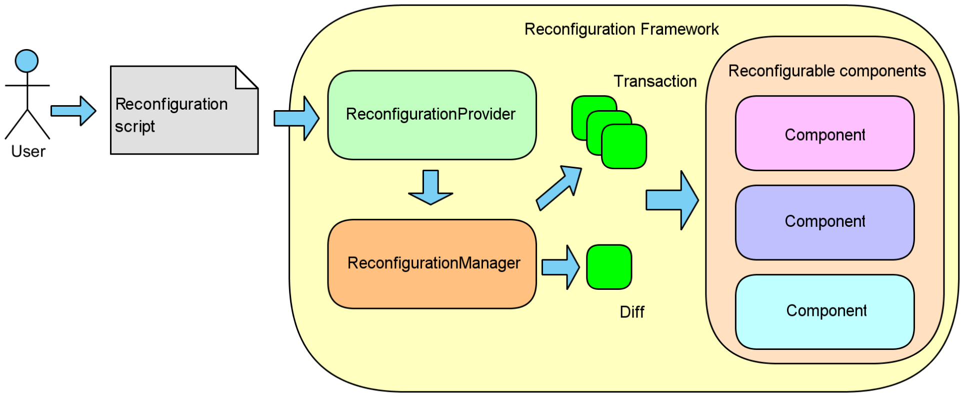

Figure 2. Reconfiguration Framework overview.

Architecture Description and Reconfiguration Languages are used to describe system architecture and all reconfigurations (see Fig. 2). To support the creation of an application, its structure and component parameters should be described with the ADL, as proper definitions are inferred from such files. A reconfiguration should be stored in the RL files, containing both definitions of application changes (Diffs and Transactions) and information about the time in which the process should be fired. The reconfiguration language should offer the following constructs:

- an assignment instruction which could be used to assign new values of properties or references (on the left side of instruction we need to have property of a given component identified),
- instructions for manipulating properties which are collections,
- an instruction which allows the grouping of reconfigurations in transactions,

- arithmetic and boolean operators,
- structural constructs (e.g., loops or conditional statements).

In the reconfiguration mentioned above, language changes within the component-system architecture could be described. Due to this fact, we can conclude that this language could be also used as ADL.

5. Result of the project

The **AgE**[1] is a computational platform for solving a wide range of optimization and simulation problems by multi-agent systems based on heuristics; e.g., evolutionary algorithms. [8] It is developed by the Department of Computer Science of AGH University of Science and Technology in Krakow, Poland. Computational agents in the AgE platform are its components.

Components in the AgE platform are nothing more than standard Java classes. There are no restrictions concerning component classes – no need to implement interfaces or extend another class. Components are registered within an instance provider, which is implementation of the service-locator pattern. It allows us to obtain services by type, parameterized-type, or identifier. After this operation, a component instance is returned. The instance is created only during the first request and then cached (similar behavior to the singleton pattern).

Running a simulation or a computation based on heuristics requires us to change one or more parameters of computational agents, because the obtained results might not be acceptable. For example, optimization results could be divergent from expected extremum. Without a reconfiguration framework, the user has to stop computation, change parameters in an appropriate file with computation structure, and restart the AgE platform. This operation is time-consuming (updating a file, starting JVM) and needs to be processed by a human or an external program. AgE Reconfiguration Framework solves all of these problems.

The **AgE Reconfiguration Framework** (see Fig. 3) is an implementation of the reconfiguration framework model, which is described in the previous sections.

Currently, the AgE Reconfiguration Framework allows us to process all reconfiguration aspects. It enables adding, removing, or updating component properties – both simple types and references. The Reconfiguration Framework is responsible for maintaining consistency of component definitions.

A user of the AgE platform only declares a computation structure in an XML file (compliant with AgE configuration XSD schema) and appropriate reconfigurations in a DSL file using the AgE Reconfiguration Language. The Reconfiguration Framework changes parameters after a computation is finished and the platform is still working. Then, the computation with new parameters is restarted. Authors of the components do not need to take care of implementing a mechanism for property changes, as framework handles it for IPropertyContainers.

[1]Documentation of the AgE framework `http://age.iisg.agh.edu.pl/`.

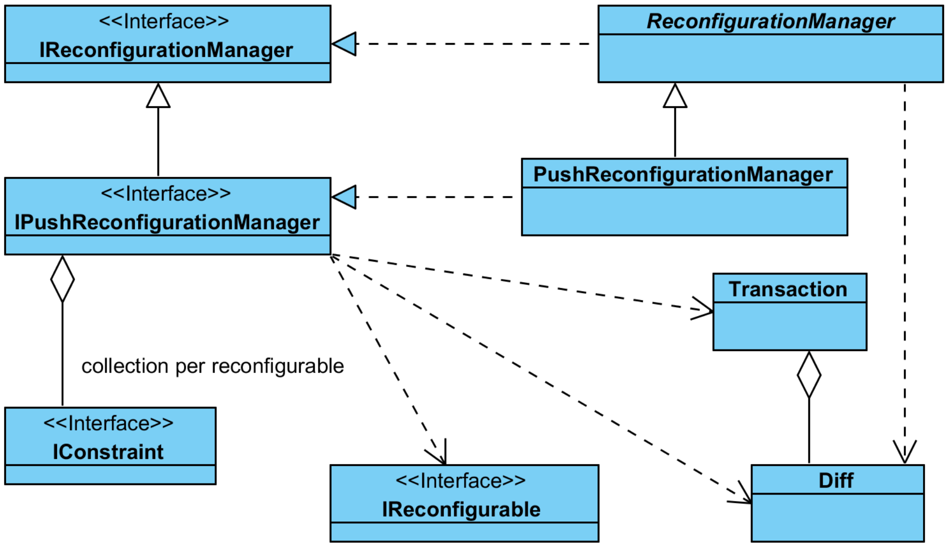

Figure 3. AgE Reconfiguration Framework class diagram.

The AgE Reconfiguration Language was created with Xtext[2]. It has a structure very similar to Java and offers features described in section 4. In this reconfiguration language, it is possible to define reconfigurations in loops. This allows a computation to be described for any configuration of parameters in several lines.

6. Case study

The concept of component reconfiguration can be utilized in many various areas of component-based systems, such as high-availability systems, computations, or test automation. In this paper, the proposed concept is illustrated on reconfiguration and load-balancing of multi-agent systems.

We use our enhancement to run a computation that tries to solve an optimization problem according to the defined criteria. It is important that the goal can be easily achieved. We only have to implement two additional, simple methods for reconfigurable components. We modify its behavior without altering its old source code. Now, it is possible to change a stop condition or any other component during computation. The computation is running in the meantime. Without a reconfiguration framework, every time that computation parameters or strategy should be changed, the whole computation has to be stopped, the configuration has to be changed, and then the computation started again.

[2]Documentation of the Xtext framework `http://www.eclipse.org/Xtext/`.

To illustrate the proposed concept, let us present a sample multi-agent system that solves the Optimal Golomb Ruler [2]. It is complex combinatorial optimization problem. The Golomb Ruler is an ordered sequence of n nonnegative integers:

$$\langle a_1, a_2, \ldots, a_n \rangle \text{ where } a_i < a_{i+1}, a_j - a_i (1 <= i < j <= n) \text{ are distinct}$$

The goal is to discover a ruler with a minimum length. It is challenging for optimization methods. Firstly, it requires proper problem encoding. Next, decision refers to handling constraints and designing genetic operators. It could be improved by using some local search methods. Every approach to solve Golomb Ruler problem requires other configurations, and each of them should be in many versions with different parameters. In AgE, the Optimal Golomb Ruler is modeled by an agent structure with the strategy design pattern applied. For example, we have a shift and inversion mutation strategy, no crossover, one-point crossover, or two-point crossover variation strategy.

Reconfiguration enables us to run multiple computations with different parameters, population size (with and without local optimization). Before the AgE Reconfiguration Framework has been implemented, users had to manually modify the configuration of computation in an XML file or create a script/program which changes the configuration file and then restart the platform (stop and start JVM, creation of computation). Now, they can just create a simple script in Reconfiguration Language and let the framework do all of this work. For the Golomb Ruler exercise: If a user would like to make 20 attempts for each of three mutation strategies and for every 100 from 100 to 1000 computation steps, he had to restart the platform 600 times. Now it is possible to write a RL script like below:

Listing 1. RL script for OGR problem.

```
computation "ogr.xml"
for (strategy :
     [invertionMutation,
      shiftMutation,
      segmentShiftMutation]) {

    mutationStrategy = strategy

    for (int steps = 100;
             steps <= 1000;
             steps = steps + 100) {

         stopCondition.noOfSteps = steps

         for (int attempt = 1;
                  attempt <= 10;
                  attempt = attempt + 1) {
              run
         }
    }
}
```

The second case where we use dynamic reconfiguration is load balancing. To create a mechanism responsible for load balancing, we use the reconfiguration framework and query mechanism provided by the AgE platform, which allows us to check the current load. Thanks to reference reconfiguration, we are able to distribute computation tasks between computation nodes. We just need to implement an algorithm which decides if migration of a task is profitable. In our solution, we used a diffusion-based gradient-scheduling algorithm [10].

The frameworks described in section 2 are not capable of supporting AgE in solving the aforementioned problems. Most of the existing solutions do not have required functionality. Moreover, they introduce high coupling to external frameworks, like ArchShell and C2 framework, or Plastik, which is implemented only for the OpenCOM component model. For both solutions, it will require us to change all existing AgE components. While the proposed Reconfiguration Framework can be easily implemented in the majority of component technologies, as it only distinguishes a few interfaces and is not coupled with a component environment. The next benefit of the AgE reconfiguration framework is related to usability. A Reconfiguration Language script is quite easy to understand without any knowledge of the internal implementation of AgE.

7. Conclusions

Existing solutions do not solve the problem of reconfiguration in the component environment in general. Most of them have too many restrictions and lack in functionality. Moreover, their coupling with particular frameworks is too high. We present a model of reconfiguration that can be applied to the majority of component platforms. We created AgE Reconfiguration Framework as an example of a solution based on that model. We provide constraints to avoid incorrect reconfiguration. Some other frameworks with mechanism to check constraints do not allow the reconfigurable component to decide for itself. This is possible in the AgE Reconfiguration Framework. If constraints are not broken, a component can decide whether to accept, postpone, or reject reconfiguration. This feature makes our solution more flexible and gives users more possibilities.

An extension of this work is to consider the reconfiguration of nested components. In particular cases, reconfiguration of nested components could be done without polling them. The decision whether to reconfigure subcomponent or not is made by its parent, as reconfiguration (Diff) is not passed to nested component. Such a feature would be useful, for example, when a parent is an active or autonomic component and all of its children are passive.

References

[1] Batista T., Joolia A., Coulson G.: Managing Dynamic Reconfiguration in Component-Based Systems. In: *EWSA'05 Proceedings of the 2nd European conference on Software Architecture*, 2005.

[2] Bloom G., Golomb S.: Applications of numbered undirected graphs. In: *Proceedings of the IEEE*, vol. 65(4), pp. 562–570, 1977.

[3] Bruneton E., Coupaye T., Leclercq M., Quéma V., Stefani J. B.: The fractal component model and its support in Java. *Software: Practice and Experience*, vol. 36 (11–12), pp. 1257–1284, 2006.

[4] Canal C., Cansado A.: Component Reconfiguration in Presence of Mismatch. *Informatica (Slovenia)*, (35), pp. 29–37, 2011.

[5] Canal C., Poizat P., Salaün G.: *Model-Based Adaptation of Behavioural Mismatching Components*. Tech. rep., IEEE Transactions on Software Engineering, 2008.

[6] Documentation of the ACME project. `http://www.cs.cmu.edu/~acme/`.

[7] Documentation of the C2 style.
`http://www.isr.uci.edu/architecture/c2.html`.

[8] Faber Ł., Piętak K., Byrski A., Kisiel-Dorohinicki M.: *Advances in intelligent modelling and simulation: simulation tools and applications*. In: *Agent-based simulation in AgE framework*, pp. 55–83. Springer, Berlin Heidelberg, 2012.

[9] Fowler M.: Inversion of Control Containers and the Dependency Injection Pattern 2004. `http://martinfowler.com/articles/injection.html`.

[10] Grochowski M., Schaefer R., Uhurski P.: Diffusion based scheduling in the agent-oriented computing systems. *Lecture Notes in Computer Science*, (3019), pp. 97–104, 2004.

[11] Hall R. S., Pauls K., McCulloch S., Savage D.: *OSGi in Action – Creating Modular Applications in Java*. Manning, 2011.

[12] Home page of the OpenCOM dynamic software component model.
`http://opencomc.sourceforge.net/`.

[13] Meyer B.: *Design by Contract*. Prentice Hall PTR, 2002.

[14] Oreizy P.: *Issues in the Runtime Modification of Software Architectures*. Technical report, vol. 96, issue 35 (University of California, Irvine. Dept. Information and Computer Science), 1996.

[15] Szyperski C.: *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., 2002.

Affiliations

Jakub Grzesiak
AGH University of Science and Technology, Department of Computer Science, Krakow, Poland

Łukasz Jędrychowski
AGH University of Science and Technology, Department of Computer Science, Krakow, Poland

Received: 8.04.2014
Revised: 16.10.2014
Accepted: 14.10.2014