# SimCo – Hybrid Simulator for Testing of Component Based Applications

Richard Lipka, Tomáš Potužák, Premek Brada, and Pavel Herout

Department of Computer Science and Engineering,
University of West Bohemia Plzeň, Czech Republic
{lipka,tpotuzak,brada,herout}@kiv.zcu.cz

**Abstract.** Testing of component-based applications is important in order to ensure that third-party components do not compromise the functionality or properties of the whole system. However, thorough testing of functionality, behaviour and extra-functional properties is a tedious and time consuming process. In this paper we present an approach to discrete event simulation testing of components and component sets. Its unique feature is the ability to execute a mixture of real, implemented components and simulated mock-ups of the remaining parts of the application. Together, this approach allows faster testing on a wide scale of different inputs for tested components. At the same time, the use of actual components increases the confidence in the simulation test results. The approach has been implemented using the OSGi platform in the form of the SimCo framework and toolset, for which the key architectural considerations are discussed together with a short case study illustrating its usage.

**Keywords:** software component, testing, simulation, performance, OSGi.

## 1 Introduction

In computer science and software engineering, component based development is becoming more and more widespread due to maximized reusability of software and also development simplified by using third party components. This tendency is more visible with the expansion of mobile devices. In a mobile device environment, an application obtained from a third party through an application repository like Apple's AppStore can be described as a component which is using the components provided by the device's OS and other applications.

Testing of such applications is very important in order to ensure that the third-party application will not compromise the functionality or properties of the whole system. Such testing often has to be performed by the maintainer of the application repository and not only by the author of the application – the maintainer need not be responsible for the proper functionality of the applications but should ensure that they are safe and without performance problems for his clients. Therefore black box testing is one of the important techniques in this field. Also, for this type of cases, extra-functional properties are more important than the functional ones. However, preparing and executing sufficiently thorough tests for large number of applications or components is a challenging task from technical, process and time perspectives.

© Springer-Verlag Berlin Heidelberg 2013

In this paper, we are presenting a tool for testing real software components in a simulated environment. Because performance of an application is not a simple product of the performance of each component, we have designed our tool to be able to execute tests not only on one component but also on an arbitrary set of components or on the whole application. Using discrete event-based simulation and simulation components, the tests on real components provide useful information about their real performance and, at the same time, might be executed in a shorter time.

The rest of this paper is structured as follows. In Section 2 we review the foundational ideas and research work related to component simulations. Section 3 provides detailed description of the simulation tool and its internal components. Use of the resulting approach is illustrated by two short case studies in Section 4, after which a conclusion closes the paper.

## 2 Background on Component Testing and Simulation

In the component approach, the whole application can be created from a set of individual software parts called components. They are considered to be black box entities, with clear definition of its interface and functionality (sometimes called behaviour) but without observable inner state [1] and knowledge of their inner working. Because all communication within component application has to be performed only through defined interfaces, no hidden dependencies should exist and thus it is easier to substitute one component with another. Component model specifies how software components look, behave and interact, while a component framework is an implementation of a specific component model, providing the required infrastructural functionality.

The designer of a component application usually works with components stored in a repository [11]. Often, the correctness of the functionality of stored components is considered to be implied as they should be tested in a standard way by developers before they are deployed into the repository. For the designer of an application, information about both component's interface features and performance or other extra-functional properties might be useful. The latter information is often obtained by executing simulation tests of the components, as e.g. in the Palladio approach [2–4].

Discrete event simulation is an often used method for simulation based testing or system analysis. Each event contains a time stamp when it should occur in simulation time and an action which shall be executed [8]. All events are handled by a calendar. When the simulation is started, the calendar chooses the event with the earliest time (the smallest value of the time stamp), sets simulation time to the time of the event and performs the action of the event. Then, next event according to its time stamp is selected from the calendar and executed and so on. Each event can cause the creation of one or more new events, which are being added to the calendar [8].

In discrete event simulation, time between two events can be arbitrary long or short, thus it allows using a fine division of time. However, it is advantageous mainly when events are not abundant due to the overhead connected to event execution.

In the area of simulation testing in component oriented development, tools and frameworks based on component simulation are commonly used. Their development began before the year 2000 [9–11] but the research in this field continues till today [12–15].

The frameworks are utilized either for general simulations (such as discrete event simulation [12]) or are focused more specifically (such as simulation of computer networks [14]).

Generally, the research of component-based simulations is focused on the relationship between the simulated components and their composability while using a specific framework [12, 13]. The effort to use components in a distributed computing environment is also very common [16–18]. Besides the focus on specific frameworks, there are also attempts to create formalism for component-based simulations [17]. The variability of such simulations in terms of application reconfiguration is often mentioned as the most advantageous feature of component-based simulations [14].

Simulation testing of components is rarely focused on the functionality of components, since their correct functionality is considered to be implied. Often, the main goal of simulation tests is verification of quality of services (QoS) and extra-functional properties, as is described in [15, 20]. All tests of components are black-box tests with exception of self-testing (see [19] for example). Because distributed use of components is often expected, there is a need to test them for this usage [21].

However, it is necessary to consider that the component's functionality may not be tested properly by its developer, or that black-box component of the third party might be a security risk [21]. We therefore believe it is important to focus on testing of component functionality as well as their suitability for different hardware configurations, as they may be deployed on different devices.

Further, it should be noted that real components are rarely tested in simulation tools. Instead, only their models are in the vast majority of approaches used for the simulation purposes as e.g. in Palladio [2]. These models are often created using a static description, such as resource consumption, behaviour description, and so on [15, 20]. Specific descriptive languages have been created for these purposes [4, 15]. Use of models instead of real components brings the issue that the model may not accurately reflect the component's properties and that the results based on its usage cannot be reliably used for reasoning about the components.

## 3 The SimCo Simulation Tool

One idea in using of components is to have them stored in repository as stand-alone applications. This approach is used even in specific domains, e.g. the Openmatics portal [22] where applications form repository are installed into hardware inside vehicles and provide different kinds of functionality, such as position reporting, telemetry and so on. Such components can be created by third party and it is important to test their functionality, behaviour and extra-functional properties (like duration of computations or amount of data flowing through a network interface) before they are offered to customers through the repository, in order to ensure that they will not cause any undesirable effects. Black box test with using of simulation tool may provide convenient way of testing.

Therefore, we designed SimCo simulation framework and toolset for testing of real, implemented components in simulated environment. Its aim is to enable the use of obtained results in comparing different components or applications, as well as for calibrating other simulation models such as Palladio. The remainder of this section describes the overall approach, technical and architectural design of the simulation framework, and its usage.

## 3.1 General Approach

Components in real applications are expected to perform actions (such is an invocation of another's component method) driven by internal component logic, user inputs or external asynchronous events like sensor interrupts. In other words, the time between these communication events is unpredictable, variable and can span long intervals in real time (seconds or even minutes) which makes standard testing of such applications complicated and time demanding or, alternatively, requires changes in tested components or the environment to cause the events to occur more often

Also, we may assume that there will be relatively long periods of time when an application component waits for the result of another component's service. This time slows down the testing process but the results of these services may be important for the application functionality and control flow, especially when handling of extreme or invalid values is being tested.

For both of these reasons we are using discrete event simulation which enables us to (1) speed up the process of testing of components by means of quickly providing pre-processed values through the simulation environment and performing the tests in simulation time, and (2) have a way of creating complex simulation scenarios with a wide range of inputs for components.

The main feature of our approach is however the use of *hybrid simulation* where real components and simulated components are mixed during the testing of the whole application. When the tests are focused only on a part of application or a single component, the rest of the application can be replaced by mock-ups (simulated components) which require a considerably lower amount of time for their execution. The tests will nevertheless be performed on real components themselves without any changes to their implementation. This is consistent with the black box understanding of components and important since the results can be relied on (unlike the model-based approaches).

The simulation tool is designed to test especially extra-functional properties of components, such as duration of computation or amount of data flowing through a network interface. However, it can as well be used to verify standard functionality and state-based behaviour.

## 3.2 Component Technologies Used

One of our aims is that the simulation framework itself be component-based (mainly for extensibility reasons) and that the proposed methods be relevant to the current state of the art component models. There are several industrial component models that were consequently considered as good candidates for the framework implementation, and the following frameworks were chosen since they provide a good balance between the adherence to the principles of component-based programming and industrial relevance.

**The OSGi Framework and Its Implementation.** The OSGi framework [5] describes a dynamic component model and offers service platform for Java programming language where component-based applications and components alone can be remotely installed, started, stopped, updated and uninstalled without requiring application restart [6]. The OSGi framework is commonly used in many different industry areas such as automotive industry, cell phones, portable devices, software development and so on [6]. For SimCo implementation, we use the Equinox OSGi implementation. OSGi components (called bundles) communicate through services which are implementations of specific interfaces. When a bundle is installed into the framework, its exported interfaces (services that are provided by the bundle) are registered, so other bundles may ask framework for bundle providing these services. The registration of a service is done directly in the code of the bundle, or by the Component Service Runtime for services declaratively specified in the manifest.

**Spring.** The Spring framework [6] offers a variety of features to support the development of enterprise-grade applications. The most characteristic features of the Spring framework are inversion of control, aspect-oriented programming, data access, transaction management and remote access. Spring also offers easy configuration of class-based elements called Spring beans as well as their dependencies through an XML configuration file.

**Spring Dynamic Modules.** The Spring Dynamic Modules extension for OSGi service platform enables the development of OSGi components using Spring framework. Moreover, SpringDM improves manageability of the OSGi services. The main advantage of using SpringDM is a transfer of bundle service dependencies from the code of Java classes into XML configuration files. The code of the bundle is then easier to develop and some changes in registered and used services can be performed only in configuration files, without the need to change the source code of the bundle.

## 3.3 Structure of the Simulation Application in SimCo

As was stated above, in the simulation tool, real and simulated components are used together. In order to provide the needed facilities for component testing and measurements, the whole simulation is composed from four types of components: framework, real, simulated, and intermediate ones.

The first type are *framework components* which constitute the SimCo simulation core and provide supporting features. These components ensure the functionality of the framework and also provide basic services necessary for the simulation. Among these components, a component controlling the calendar is the most important one. It contains all events which occur during the simulation run. Consequently, it controls the progress of the simulation based on the loaded scenario (the scenarios will be described later). Additional components ensuring other functionality, such as logging or measuring performance, belong to this group as well.

Second type is represented by the *real components* of the component-based application under test. There can be one or more real components in one simulation experiment.

They can interact with each other and with the simulation environment. As mentioned above, SimCo requires no changes in the implementation of these components, so they are the same as those which will be deployed in the real application.

If testing of real component's provided or used services is not the goal of the simulation experiment, they can be replaced by their simulation equivalents. Therefore, *simulated components* export the same interface as their real counterparts but the actual computation is replaced by a model according to decision of the designer of the experiment, e.g. a random number generator or a list or pre-processed results, provided as an answer for a service call from the real component.

In order to measure real components' performance and keep the simulation consistent, we need to intercept all events in the simulation even when they are passed directly between a pair of real components. Hence the fourth type of components has to be used, called *intermediate components*, which serve as proxies for the real components. All calls of the component hidden behind this proxy are noted and then passed to the real component, and likewise the answers (returned values) are returned through this proxy. The intermediate component also allows us to model deployment of real components in the distributed environment, as it may be set to cause delays of calls or even to induce errors into the communication.

One of the biggest advantages of using SpringDM is that it is possible to change the components used in the final application only by modifying XML documents that describes composition of the application. This allows us to easily replace real components for their simulated counterparts or to place intermediate components between two real ones.

## 3.4 Considerations Related to the Hybrid Simulation

Depending on the position of the simulated component, it may be required to provide a more complex behaviour than described above. An example is passing events to other components. The settings of each simulated component are stored in its configuration file which however does not contain the description of its behaviour. The behaviour has to be implemented inside the component.

Depending on the topology of the tested application, there are several possible configurations of the inter-component connections, described in Figure 1 below. First, the real components may be connected only to simulated components or to other real components (Figure 1-c). If there is a chain of simulated components attached to a real one (see Figure 1-a), it might be considered to replace the whole chain by implementation of simulation of component A, as it is the only one which interacts with the real component. However, if the simulation models of all components in the chain have been already created, it saves time of the experimenter to use them instead of creating the new implementation.

More complex situation occurs when simulation component is surrounded by two real components (see on Figure 1-b). In this case, it is necessary to consider the possibility that, in reaction to an action invoked by the real component B, simulated component invokes after some computation an action upon the real component A (and even that return value from the component A is passed as a return value for original invocation

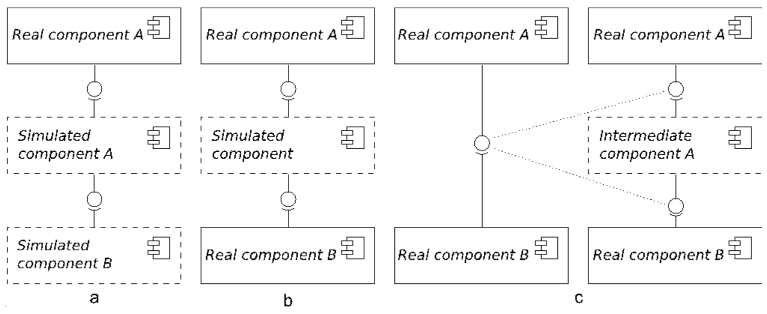

**Fig. 1.** Deployment of simulated and intermediate components

from the component B). In such case, behaviour of simulated component must respect this and it cannot be modelled only by using pre-calculated values.

Lastly but probably most importantly, real components within the simulation should not be aware that they are not the part of a real application but they are deployed in a simulated environment. Hence, from their point of view, simulation environment must act as the environment for which they were originally developed. On the other hand, both the simulation environment and the real components must be under complete control of the simulation tool. Therefore, all interactions among the real as well as simulated components must be performed by using the calendar and events so that we can log all interactions and obtain the desired measurements [23]. For this purpose, all services provided by the real components are wrapped by intermediate components.

## 3.5 Scenarios and Events

To support complex testing of components, our tool uses scenarios to drive the simulation tests. They allow us to describe the environment and activities of simulated components in a great detail.

The testing scenario is divided into three parts. In the first part, the settings of the entire simulation are described. An example of such setting is the value of simulation time at which the simulation should finish.

In the second part, all the components of the simulation are described. The settings of all components are described separately (one settings XML file per component). For the simulated components, these additional files also contain the description of the components' settings (e.g. seeds and parameters of random number generators). An example can be the return values of the services provided by the simulated components in dependence of their input values.

The third part of the scenario is the description of events which can occur during the simulation. Each event has a source and a target. The source may lie within the simulated system (i.e. simulated and real components) or in outer environment, in order to simulate both behaviour of components and also the actions of users and other outer inputs. The target is always within the simulated system. The event can have parameters relevant to its purpose. For example, an event representing processing of some graphical data would have the path to the data file as its parameter.

Events can be divided in accordance with their occurrence into three classes – regular, casual, and rare. The *regular events* are created periodically during the whole simulation run or during a selected/certain period of time. They can be events based on timers (e.g. periodical checking of changes of a component).

The *casual events* can occur often, but not periodically. However, the time of their occurrence may be described by probability distribution. They could be for example used to represent requests of users.

The *rare events* occurs so infrequently that their probability distribution cannot be identified easily. They even may not occur at all during the simulation run. An example of such event may be an accident, failure, damage of network hardware, and so on. The timestamp of such events can be set differently in different scenarios for one set of tested components. So, it is possible to observe the behaviour of the tested components when the events happen in various points of the simulation time.

Use of scenarios is an important benefit of our simulation tool. It allows controlling the size and length of tests without the necessity to manipulate with tested components. The scenario allows the tester to induce a large number of rare situations in a short time if it is necessary and thus speeding up testing. Scenarios are stored in separate files, so a set of scenarios (and tests) can be stored with the application, to be used when some component of application is changed.

## 4 Examples of SimCo Based Experiments

In this section we illustrate the operation and usage of SimCo on an example component-based application. The experiments were created to represent testing of a typical component-based application without dependencies on any specific hardware.

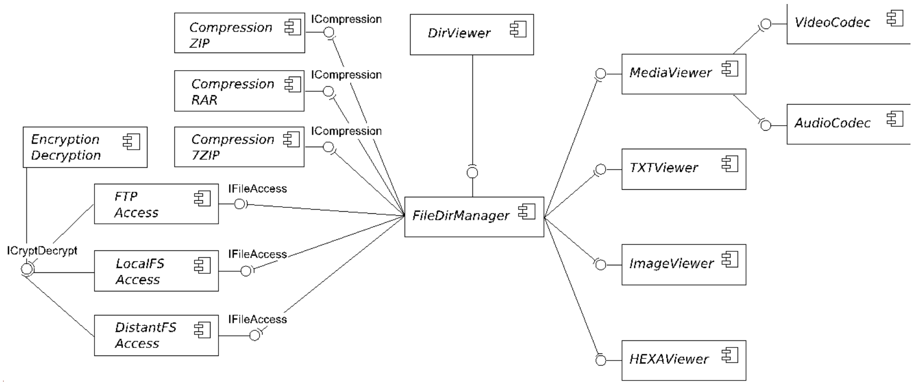

**Fig. 2.** Tested application – File manager

For these purposes we are using a simple File Manager application with several functions. Its architecture is outlined in Figure 2. Due to its purpose, the file manager requires a wide variety of different services, which can be implemented with different quality of services and consequently tested and arbitrarily replaced. It allows us to test performance of several different algorithms or to test properties of network communication.

## 4.1 Performance Experiment

This experiment is designed to demonstrate the use of our simulation tool. The test is focused on the speed of HEXAViewer component. Two different implementations of HEXAViewer were implemented. One preloads the whole file before displaying it. The other one displays only part of the file which will fit to the screen and loads additional data only when required. (This means that time of opening of the first one has a linear dependency on the size of opened file and the second one has a constant time). Three real components are used, HEXAViewer, FileDirManager and LocalFS Access. The DirViewer component is simulated (see Figure 3). As DirViewer component is a GUI of the application, its simulation allows us to simulate the behaviour of the user without the necessity to work directly with the GUI.

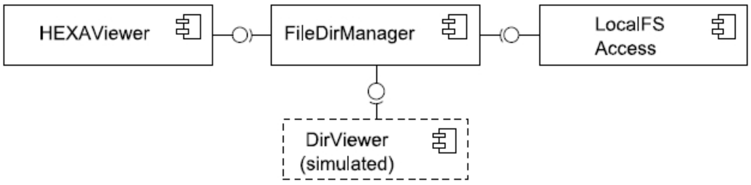

**Fig. 3.** Deployment of components for performance testing of HEXAViewer

The behaviour of the DirViewer is set to open ten times each input file with randomly generated content. We used 14 input files with different lengths and measured time between request for opening of the file and moment when the window is created. The

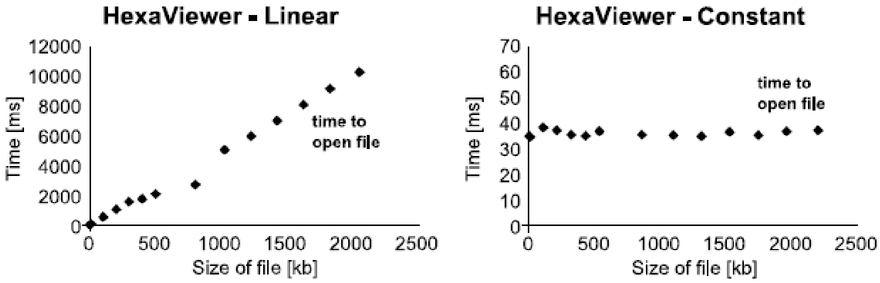

**Fig. 4.** Measured results

simulation tool captures the time of invocation of the file opening operation between DirVewer and FileDirManager and the time of the event created by HEXAViewer informing the application that the window with file is displayed.

The results at Figure 4 show the measured time requirements. An average duration (in milliseconds) of opening of each file is shown for both implementations of HEXAViewer. The results cannot be interpreted only as time required by HEXAViewer to open the file, as duration of other activities in the application influences them as well (e.g. obtaining the file from LocalFS Access), but shows relative performance of two different variations of the application and thus might be used to decide which one is more suitable.

### 4.2 Communication Experiment

This experiment shows the ability of SimCo to measure the amount of communication of a tested component with outside environment. This is particularly important for testing of component based applications for mobile devices, where communication through the cell phone network can lead to cost increase for their users.

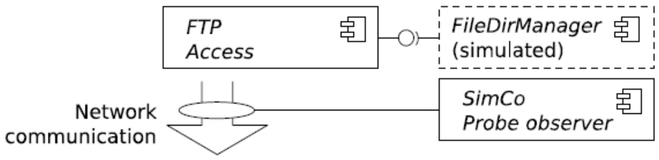

**Fig. 5.** Deployment of components for communication measurement

In the experiment, FTP Access component is used, along with simulated FileDirManager (see Figure 5). FileDirManager provides input for FTP Access and initiates download of ten different files. A probe based on the *libpcap* library is used to measure the amount of transferred data. The probe is capable of analyzing packets of transport and network layer of TCP/IP stack. If analysis of the application protocol is required it has to be added as a plug-in to the probe. We have tested that all required files were transferred and no data were lost. More important, we have tested that we are able to measure amount of such communication with all its overhead.

## 5 Conclusion

In this paper, we have described the design of SimCo, a component-based hybrid simulation tool for testing real components within a simulated environment. In comparison with other simulation testing tools and strategies, we enable the possibility to choose arbitrary part of the tested application and replace the rest by simulation. This enables to make tests of a specific part of the application faster than when the whole application would be tested. At the same time, the important tested components are real, not only their models, so the results describe their real properties. Use of test scenarios which drive the simulation also allows us to store a wide range of tests for each application without the necessity to modify the application itself or to have the application prepared for testing in the way used by unit tests tools like JUnit.

The core of the simulation tool – which is itself component based – is complete now; however there is still remaining work on the automation of repeated tasks and especially on the usability of the tool. So far, the GUI serves only as visualization; we would like to enhance it to allow at least visual editing of the properties of the simulated components and on-line display of the measured results. We are also preparing a second case study to demonstrate the possibilities of the framework.

The remaining issue is handling of API calls problematic from the simulation point of view (time handling, network communication, etc.), so they will be managed completely by the simulation tool. The possibilities of such handling and the hazards of uncontrolled calls are topic of our currently ongoing research.

**Acknowledgements.** This work was supported by the Czech Science Foundation under the grant number 103/11/1489 "Methods of development and verification of component-based applications using natural language specifications".

## References

1. Szyperski, C., Gruntz, D., Murer, S.: Component Software – Beyond Object-Oriented Programming. ACM Press, New York (2000)
2. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1), 3–22 (2009)
3. Heam, P.C., Kouchnarenko, O., Voinot, J.: Component Simulation-based Substitutivity Managing QoS Aspects. Electronic Notes in Theoretical Computer Science 260, 109–123 (2010)
4. Cansado, A., Henrio, L., Madelaine, E., Valenzuela, P.: Unifying Architectural and Behavioural Specifications of Distributed Components. Electronic Notes in Theoretical Computer Science 260, 25–45 (2010)
5. The OSGi Alliance: OSGi Service Platform Core Specification, release 4, version 4.2 (2009)
6. Rubio, D.: Pro Spring Dynamic Modules for OSGiTM Service Platform. Apress, USA (2009)
7. Brada, P., Jezek, K.: Ensuring Component Application Consistency on Small Devices: A Repository-Based Approach. In: Proceedings of the 38th Euromicro SEAA Conference. IEEE Computer Society Press (accepted for publication, 2012)
8. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. John Wiley & Sons, New York (2000)
9. Miller, J.A., Ge, Y., Tao, J.: Component-Based Simulation Environments: JSIM as a Case Study Using Java Beans. In: Proceedings of the 1998 Winter Simulation Conference, Washington, DC, pp. 373–381 (1998)
10. Pidd, M., Oses, N., Brooks, R.J.: Component-Based Simulation on the Web. In: Proceedings of the 1999 Winter Simulation Conference, Phoenix, pp. 1438–1444 (1999)
11. Harrell, C.R., Hicks, D.A.: Simulation Software Component Architecture for Simulation-Based Enterprise Applications. In: Proceedings of the 1998 Winter Simulation Conference, Washington, DC, pp. 1717–1721 (1998)
12. Buss, A., Blair, C.: Composability and Component-Bases Discrete Event Simulation. In: Proceedings of the 2007 Winter Simulation Conference, Washington, DC, pp. 694–702 (2007)

13. Moradi, F., Nordvaller, P., Ayani, R.: Simulation Model Composition using BOMs. In: Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications, Malaga (2006)
14. Rao, D.M., Wilsey, P.A.: Multi-resolution Network Simulations using Dynamic Component Substitution. In: Proceedings of the 9th Int'l Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, Cincinnati (2001)
15. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. The Journal of Systems and Software 82, 3–22 (2009)
16. Verbraeck, A.: Component-based Distributed Simulations. The Way Forward? In: Proceedings of the 18th Workshop on Parallel and Distributed Simulation, Kufstein (2004)
17. de Lara, J.: Distributed Event Graphs: Formalizing Component-based Modelling and Simulation. Electronic Notes in Theoretical Computer Science 127, 145–162 (2004, 2005)
18. Wainer, G.A., Madhoun, R., Al-Zoubi, K.: Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services. Simulation Modelling Practice and Theory 16, 1266–1292 (2008)
19. Yao, Y., Wang, Y.: A Framework for Testing Distributed Software Components. In: Annual Canadian Conference on Electrical and Computer Engineering, Saskatoon, pp. 1566–1569 (2005)
20. Becker, S., Koziolek, H., Reussner, R.: Model-Based Performance Prediction with the Palladio Component Model. In: Proceedings of the 6th International Workshop on Software and Performance, Buenos Aires (2007)
21. An, G., Park, J.S.: Cooperative Component Testing Architecture in Collaborating Network Environment. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 179–190. Springer, Heidelberg (2007)
22. Openmatics. Applications (2012), `http://www.zf.com/brands/content/en/openmatics/-products_services/apps/apps_openmatics.html` (cited June 28, 2012)
23. Potuzak, T., Snajberk, J., Lipka, R., Brada, P.: Component-based Simulation Framework for Component Testing using SpringDM. In: Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Zadar, vol. 20(1) (2010)
24. Šimko, V., Hnětynka, P., Bureš, T.: From Textual Use-Cases to Component-Based Applications. In: Lee, R., Ma, J., Bacon, L., Du, W., Petridis, M. (eds.) SNPD 2010. SCI, vol. 295, pp. 23–37. Springer, Heidelberg (2010)