2012 38th Euromicro Conference on Software Engineering and Advanced Applications

Testing a Component-based Application for Road Traffic Crossroad Control using the SimCo Simulation Framework

Tomáš Potužák, Richard Lipka, Přemek Brada, Pavel Herout
Department of Computer Sciences and Engineering
University of West Bohemia
Plzen, Czech Republic
e-mail: {tpotuzak, lipka, brada, herout}@kiv.zcu.cz

***Abstract*—Component-based software development is an important trend in software engineering. Using this approach, a system can be constructed from a set of individual components (i.e. pieces of software with defined interfaces and functionality). On deployment, components are usually not tested for their correct functionality since this is considered to be implied. Simulation tests are nevertheless often used to determine extra-functional properties and quality of services. However, only models of the components are usually used for such testing. In this paper, we present the SimCo – a simulation framework for testing of real software components in a simulation environment. This enables thorough tests of components without the need to create their (potentially incorrect) models. The SimCo itself is constructed from components as well in order to enable its modularity and usability. The utilization of the SimCo is demonstrated on a case study involving component-based application for the control of a road traffic crossroad.**

Keywords—software component, simulation framework, simulation testing, crossroad control

I. Introduction

Component-based software development is an important trend in software engineering, which emerged at the turn of the century and become widespread during the last decade. Using the component approach, an application can be constructed from a set of individual software components. These components are pieces of software with clear interfaces and well-defined functionality, but with no externally observable inner states. So, each software component can be considered as a black box [1].

The software components are used in order to maximize reusability and third party composition. This means that a programmer, which creates a component-based application, is often not the author of the individual components. So, the software components are usually not tested by third parties for they correct functionality, since this is considered to be ensured by the manufacturer of the component. Nevertheless, simulation tests are often used to determine extra-functional properties of the components and their quality of services, which can also play a significant role in the designed component-based application. Some examples of such testing can be found in [2] and [3]. The testing of the software components is an important branch in contemporary research. However, in most cases, only the models of software components are used for simulation testing [4].

In this paper, we present the SimCo – a simulation framework, which is designed for testing of real software components in a simulation environment. This enables thorough testing of the software components without necessity to create their models, which can be potentially incorrect. The simulation framework itself is component-based (i.e. constructed from individual software components) in order to maximize its modularity and usability [5].

II. Discrete Simulation

There are two types of simulation, regarding their approach to simulated time – *time step* simulations and *discrete-event* simulations. Time step simulations divide time into equally-sized time intervals (time steps). During each step, states of all simulated objects are recalculated. Discrete event simulations recalculate the state of the system only when an event occurs. Unlike the time step simulation, the time is changed irregularly from one event to another [6].

A. Handling of Events in the Discrete Event Simulations

Using the discrete event simulation, the time progress is ensured by the interpreting of events. Each event contains a time stamp determining when it should occur in the simulation and an action, which shall be executed [6].

All events are handled by a calendar. When simulation is started, the calendar chooses the event with the earliest time stamp, sets simulation time to the time of the event, and performs the action of the event. Then, next event according to the time stamp is chosen 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 [6].

The simulation is stopped when calendar contains no more events or at a specified simulation time.

B. Features of the Discrete Event Simulations

The discrete event simulation allows using a finer division of simulation time, but is advantageous only when events are not abundant. The execution of each event brings overhead, which slows down the calculation. In a discrete event simulation, time between two events can be arbitrary long or short. Events are also useful when a long period of simulated time without any event might occur. In that case,

978-0-7695-4790-9/12 $26.00 © 2012 IEEE
DOI 10.1109/SEAA.2012.18

the simulator does not need to compute simulation steps without any activity and can just skip to the nearest event [6].

III. Component Models and Frameworks

As was stated before, a software component is a well-defined black box entity, which has no externally observable state. It is a part of third party composition (i.e. author of the component can be and often is different from the author of the component-based application, which utilizes the component). This abstract definition is common for all software components as view on software components and understanding of them is very diverse. Therefore, there is no universally accepted more specific definition. The matter is well discussed in [1].

Component models specify how software components look, behave, and interact, which results in diversity of component understanding mentioned above. Component models also specify if there are one or more types of components.

A component framework is an implementation of a specific component model. There are many component models and even more component frameworks used and developed in industry and in research sphere. For this paper, three of them are important (see following sections).

A. OSGi

The OSGi component model is a dynamic component model designed for Java programming language [7]. There are several OSGi frameworks, which implement the OSGi component model. In such an OSGi framework, it is possible to install, start, stop, and uninstall the individual software components and entire component-based applications without necessity to restart the framework itself [8].

A software component in an OSGi framework is known as a *bundle*. An OSGi bundle has the form of a standard Java `.jar` file with a specific *manifest* file describing the features of the bundle, provided services, required bundles, and so on. Each bundle can contain any number of classes and can provide arbitrarily complex functionality.

OSGi bundles communicate through *services*. Services are implementations of specific interfaces, which were registered in the OSGi framework by OSGi bundles containing these implementations. Services are one of the most important parts of the OSGi framework, as component-based applications are built from bundles connected with the services. Services are registered directly in the code of the bundle or by the Component Service Runtime for declaratively specified services. To register a service, the OSGi bundle needs the context of the bundle, which is provided by the entry point of the bundle called *bundle activator*. The services can be registered as easy as depicted in Fig. 1.

Various OSGi frameworks are commonly used in many different industry areas such as vehicles, cell phones, PDAs,

```
IMgGen msgGen1 = new MsgGenImpl1();
context.registerService("cz.zcu.kiv.cosi.msgtalk.IMsgGen",
  msgGen1, new Hashtable("type", "MsgGenImpl1"));
```

Figure 1. Example of a service registration in OSGi

```
<beans>
  <bean id="foo" class="x.y.Foo">
    <constructor-arg ref="bar" />
    <property name="baz" ref="baz" />
  </bean>

  <bean id="bar" class="x.y.Bar" />
  <bean id="baz" class="x.y.Baz" />

</beans>
```

Figure 2. Example of bean definitions in a Spring configuration file

and so on. The widespread of the OSGi frameworks is the reason, why it was chosen for the implementation of the SimCo simulation framework.

B. Spring

The Spring framework [1] offers a number of features useful for development of enterprise-grade component-based applications. Among these features are inversion of control, aspect-oriented programming, transaction of management, remote access, and others [8].

Spring enables easy configuration of elements using a XML configuration file. The basic elements of a Spring-based application are so-called *Spring beans* defined in the XML configuration file along with their dependencies (for an example, see Fig. 2).

C. SpringDM

The SpringDM (abbreviation of Spring Dynamic Modules) for OSGi service platform[2] improves the manage-ability of the OSGi services [8]. It enables to develop OSGi bundles with definition of their dependencies and other features in Spring configuration files.

The SpringDM must be installed in the target OSGi framework. Once it is installed and running, it checks every newly installed bundle to the OSGi framework, whether it is a SpringDM bundle (i.e. contains a Spring configuration file) or not. If the newly installed bundle is a SpringDM bundle, its services are registered in the OSGi framework using the Spring configuration file. An example of service registration in SpringDM is depicted in Fig. 3.

IV. Component-based Simulation of Components

Now, as we discussed both simulation and software components issues, we can focus on simulation of software components. The SimCo simulation framework was developed in order to enable simulation of components. Simultaneously, the framework itself is constructed from software components in order to ensure its modularity and simple future development. Hence, both the simulation of

```
<bean id="MsgGenService"
  class="cz.zcu.kiv.cosi.msgtalk.server.MsgGenImpl">
</bean>

<osgi:service ref="MsgGenService"
  interface="cz.zcu.kiv.cosi.msgtalk.IMsgGen />
```

Figure 3. Example of a service registration in SpringDM

[1] http://springframework.org/
[2] http://www.springsource.com/osgi

components and the component-based simulation are described in following sections.

A. Component-based Simulation of Software Components

Component-based simulation frameworks are quite common. They began to emerge in last decade of the 20th century (for example, see [9] or [10]) and the research in this area continues till today (contemporary examples can be found in [11] or [12]). The component-based frameworks can be utilized for a general simulation [13], [14], but can be also focused on a specific area (e.g. simulation of computer networks [12]). The most important feature of the component-based simulations is their variability [12], [14].

The contemporary research of the component-based simulations is focused mainly on the relationship between the individual components using a specific component model and/or framework [11], [13]. Another important area is the utilization of the software components in a distributed computing environment [15], [16].

However, component-based simulations of software components are very rare. Although component-based simulations, which do not simulate software components [11], [12], [13], and monolithic (i.e. not component-based) simulations of software components [17], [18], [19] exist, there are virtually no examples of combination of both approaches.

B. Simulation Testing of Software Components

Simulation testing of software components is often focused on quality of services (QoS) provided by the components and on their extra-functional properties (for example, see [20] or [21]). The software components, which are intended for distributed computing environment, are also tested for this purpose [21], [22].

It should be noted that the functionality of the software components should be theoretically ensured by their manufacturers. However, as stated in Section III, authors of the component-based applications often use components created by other party. This means, these authors cannot be certain that a proper testing of the particular components has been carried out. Because the software components created by an unknown manufacturer can be even a security risk (intentional or unintentional) [22], their correct functionality should not be implicitly assumed without a proper verification.

Another aspect is that the particular software components and the entire component-based application can be running on a wide variety of devices (from a distributed computing cluster to a PDA or a microcontroller). The various computational powers of these devices can significantly influence the quality of services of the software components. Therefore, the target devices and their features should be taken into account during the simulation testing of the components.

As arise from paragraphs above, wide variety of properties of the software components can be and are tested using simulation. However, the real components are rarely used directly. Instead, models of components are created and utilized for simulation testing purposes. Often, only a static description of the software component (e.g. consumption of resources, description of behavior etc.) is used [20]. For this purpose, specific descriptive languages have also been developed [4], [23]. The utilization of the models instead of real components enables to focus on specific features of the components. However, it is possible to miss an important factor during the creation of the model of the software component or even introduce an error into the model. More importantly, it can be difficult to create a model of a component if its source code in unavailable. This is a most common scenario, since components are often considered to be black-boxes (see Section I).

V. SimCo Simulation Framework

Considering the issues of the simulation testing based only on models of the software components mentioned in previous section, we have developed the SimCo simulation framework. The SimCo enables the direct testing of real software components in a simulated environment [5], [24]. It is described in detail in [24]. However, in order to keep this paper self-contained, we will mention all its important features in the following sections.

A. Component Framework Selection

The SimCo simulation framework itself is created from software components in order to ensure its modularity. It utilizes OSGi/SpringDM software components (bundles), because of dynamic nature of the OSGi (see Section III.A) and easy manageability of the services using the SpringDM (see Section III.C).

The software components, which can be tested using the SimCo simulation framework, are also expected to be OSGi bundles. The reason is that the transformation of a component from one component model to another is a difficult task. Since the OSGi component model and its implementations are quite widespread (see Section III.A), this restriction is not a big issue.

B. Simulation Type Selection

As was stated in Section II.A, there are two types of discrete computer simulation, which are commonly used – the event-driven and time-stepped simulations. Both were considered for the SimCo simulation framework, which is designed for running of real software components in a simulated environment, observation, and controlling of the actions performed on and by the components.

The actions of the software components (i.e. invocation of the methods/services) can be performed arbitrary (from the simulation framework point of view) and there can be long time intervals between two successive actions. Hence, an event-driven simulation, where events correspond to performed actions is more suitable than a time-stepped simulation, which is suited for regular calculations.

The event driven approach also allows speedup of the simulation. If the real components are waiting for a complicated calculation, which should be provided by the simulation environment, it is possible to use some pre-calculated values instead of a real calculation. The simulation time can be then advanced by the time interval, which would be required for the calculation.

For the reasons mentioned above, the event-driven approach is used in the SimCo simulation framework [24].

C. Types of Components of the Simulation Framework

There are several types of the software components in the SimCo simulation framework.

First type is represented by the components, from which the simulation framework itself is constructed. These *core* components provide basic services necessary for functioning of the simulation. Among them, the *Calendar* is most important. All events, which occur during the simulation run, are managed by this component. So, it controls the progress of the entire simulation [24]. Of course, there also other components, which provide additional functionality such as logging, visualization, and so on. Newly required functionality can be easily added in the form of software component(s) due to the component-based design of the simulation framework.

Second type is represented by *real* components, which shall be tested using the SimCo simulation framework. In one simulation experiment, there can be one or more real components, which can interact among each other and with the simulation environment [5].

The simulation environment of the real tested components consists of the third type of the components – the *simulated* components. These components provide services, which are required by the real tested components. So, the real tested components (second type) have available all services they need for their functionality and they are not aware that they are running in a simulated environment [5].

On the other hand, the SimCo simulation framework must have a full control of the real tested components. Hence, all interactions between the real (second type) and simulated (third type) components are performed using the calendar and the events. This can be accomplished easily, since the logic necessary for the interaction with the calendar can be added to the simulated components [24]. Nevertheless, the interactions between two real tested components must be treated in the same way, but it is not possible to add any new logic to the real components, since they are considered black boxes. Hence, there is the fourth type of components – so called *intermediate* components. These components wrap the services provided by the real tested components. They provide the same services and functionality, but ensure the interaction with the calendar (see Fig. 4).

For a more detailed description of the particular types of the components and their interactions, see [5] and [24].

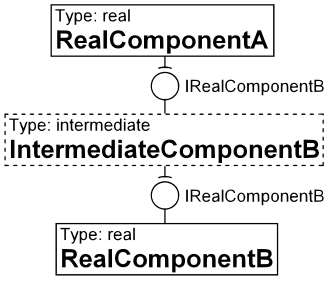

Figure 4. Utilization of the intermediate component

D. Real Components vs. Simulated Environment

There are several issues regarding the running of the real software components in a simulated environment. First of all, the simulation used in the SimCo simulation framework is event-based (see Section V.B), which means that the simulation time advances irregularly from one time stamp of an event to another. On the other hand, the real components are constructed to run using the regularly incremented real time of the computer. This discrepancy of the time flow can be problematic if the real component utilizes some time services from the Java Core API [24].

Similar problems can occur when the component requires the connection to a remote server. Such an activity is undesirable, since it is not under control of the SimCo simulation framework [24].

In order to solve the mentioned issues, the Java Core API methods providing input/output and time services, which can be invocated by the real tested components, must be adapted to meet the needs of the SimCo simulation framework. For example, the methods involving time should return values taking into account the simulation time, the remote connection could be only simulated, and so on [24].

There are several possibilities how to achieve this including aspect-oriented programming, manipulation with the Java Core API source code, or manipulating the imports of Java Core API methods in the real tested components' bytecode. For a more detailed description of these approaches, see [5] and [24].

E. Simulation Scenarios

Each simulation testing of real components is described in a simulation scenario. It is a XML file containing the settings of the entire simulation (e.g. the time, at which the simulation should end), description of the particular components (including their types), and description of the events, which shall occur during the simulation (see Fig. 5).

Besides the main scenario file, each component (regardless its type) has a separate XML file with its further description. This file can contain for example the description of the behavior of the simulated component [5]. Nevertheless, the files do not contain the description of the relations of the components. This information is sought directly from the OSGi/SpringDM framework. For a more detailed description of the simulation scenarios, see [24].

```
<simCoScenario>
  <appSettings>
    <stepLength unit="msec">0<stepLength>
  </appSettings>
  <components>
    <component type="REAL">
      <symbolicName>ControlPanel</symbolicName>
      <settingsFile>ControlPanel.xml</settingsFile>
    </component>
    <component type="simulation">
      <symbolicName>TrafficCrossroad</symbolicName>
      <settingsFile>TrafficCrossroad.xml</settingsFile>
    </component>
  </components>
  <events>
  </events>
</simCoScenario>
```

Figure 5. Example of a simulation scenario

VI. CASE STUDY: TRAFFIC CROSSROAD CONTROL

In order to demonstrate how the simulation testing of software components using SimCo simulation framework works, two case studies were created. The first one is the *File manager*. This case study represents a typical component-based application without dependencies on any specific hardware. The File manager is a convenient application for demonstration of simulation testing, since it provides wide variety of services (e.g. viewing of files of different types, various compression algorithms, local and remote access, etc.). These services can be implemented with different quality, which can be easily tested [24]. The File manager case study is described in [24] in detail.

The second case study – the *Traffic crossroad control* – is described in the remainder of this paper. This case study represents a component-based application for the control of road traffic in a crossroad using traffic lights and/or variable traffic signs. Such an application is expected to run on a specific hardware and utilize a variety of hardware sensors. Hence, the simulation testing is very convenient, because the hardware devices can be also simulated and no access to the real devices is required. Moreover, similar to the File manager, there are several services of the application, which can be implemented with various qualities of services (e.g. traffic control algorithm, algorithm for optic detection of vehicles, etc.). These qualities can be easily tested using the SimCo simulation framework.

A. Structure of the Traffic Crossroad Control Application

The components of the Traffic crossroad control application and their relations are depicted in Fig. 6. For the purpose of the testing, the components, which include handling of real hardware devices, were replaced by the simulated components. These components were only simulating the corresponding functionality (see Fig. 6).

There are nine software components in the Traffic crossroad control application. The *TrafficCrossroad* component provides information about the structure of the traffic crossroad (number and orientation of arms, numbers of traffic lanes in these arms, positions of traffic lights and sensors, and so on). Its simulated version also incorporates a simulation of road traffic (see Section VI.B).

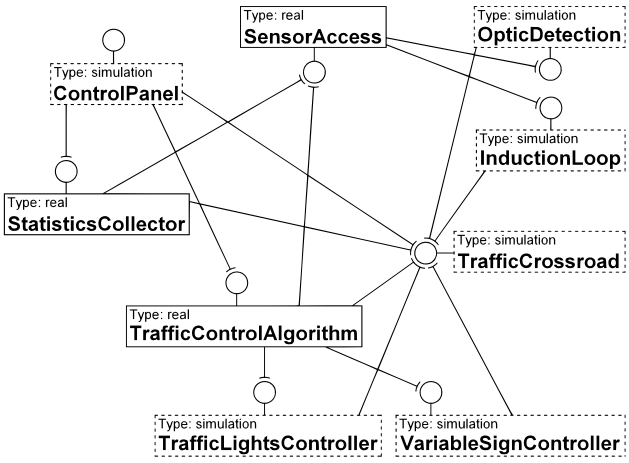

Figure 6. Utilized software components and their relations

The *ControlPanel* component provides the user interface for the entire application. It allows for example to activate or deactivate the traffic lights, set parameters for the traffic control algorithm, display collected traffic statistics, and so on. The *TrafficControlAlgorithm* component incorporates an algorithm for control of traffic lights and variable signs. This algorithm can require information about the situation in particular traffic lanes of the traffic crossroad provided by the sensors accessible via the *SensorAccess* component.

There are two different types of sensors represented by two simulated components. The *InductionLoop* component represents the induction loops places in the particular traffic lanes of the crossroad. Such a loop is able to detect, whether there is a vehicle above it. The *OpticDetection* component represents detection of the number of vehicles in a traffic lane using a camera above the lane and image recognition. These sensors are usually able to distinguish, whether there is one, two, three vehicles or a larger number of vehicles in the traffic lanes. This means that for a larger number of vehicles, the accurate number is not at the disposal. The simulated versions of the sensor components utilize the "perfect" information about the numbers of vehicles in traffic lanes obtained from the simulation of road traffic (provided by the *TrafficCrossroad* component), which is not at the disposal in a real situation. Hence, they reduce this information to a form, which is at disposal in a real situation.

The traffic control algorithm can control traffic lights of the crossroad using the *TrafficLightsController* component and variable signs using the *VariableSignController* component (if the variable signs are present and the traffic control algorithm utilizes them).

The *StatisticsCollector* component collects and summarizes the statistics of the traffic in the crossroad using the data from the sensors acquired via the *SensorAccess* component. The summarized statistics can be than viewed using the *ControlPanel* component.

B. Utilized Simulation of Road Traffic

As mentioned in previous section, the *TrafficCrossroad* simulated component incorporates besides the information about the crossroad also a road traffic simulation. This simulation is necessary for testing of traffic control algorithms and other aspects of the Traffic crossroad control component-based application. It substitutes the real traffic on an actual traffic crossroad.

The road traffic simulation can be visualized as can be seen in Fig. 7. It is a time-stepped simulation based on a car following model [25] adapted for the needs of a single crossroad. The positions of the vehicles are recalculated in each time step, whose length is adjustable, and set to 0.1 second by default. Such as short time step enables very smooth movement of the vehicles.

All main aspects of the road traffic are modeled. There are six types of vehicles [26] with different lengths and widths. The vehicles react on the traffic lights, do not enter the crossroad if they are unable to leave it, and respect all other traffic regulations.

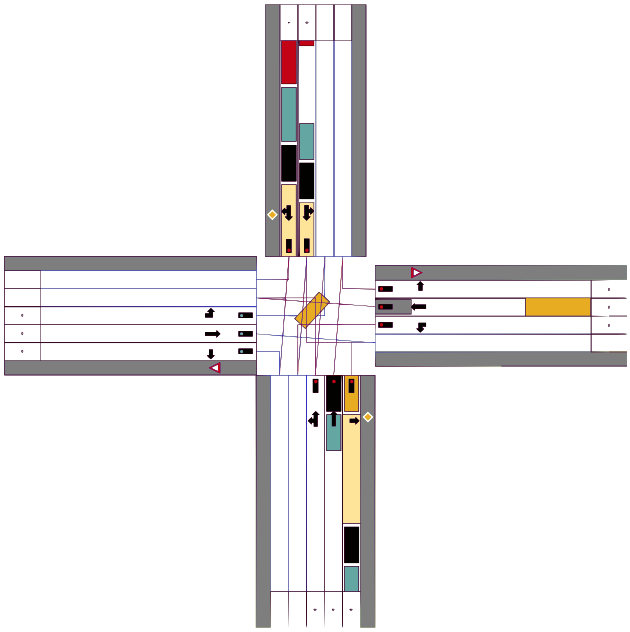

Figure 7. Visualization of the road traffic simulation

VII. Tests and Results

In order to demonstrate the use of the SimCo simulation framework, a testing of the quality of two different traffic control algorithms was performed. The qualities of the algorithms were compared using the mean vehicle queues lengths in particular traffic lanes of the crossroad. Both algorithms are able to control the traffic in a non-conflict way (i.e. they are functionally correct), but the lengths of the vehicle queues determine the fluency of the traffic through the crossroad. The shorter the queues are, the better the control algorithm is.

A. Tested Traffic Crossroad

The testing was performed using model of one traffic crossroad inspired by an actual crossroad of the Pilsen city.

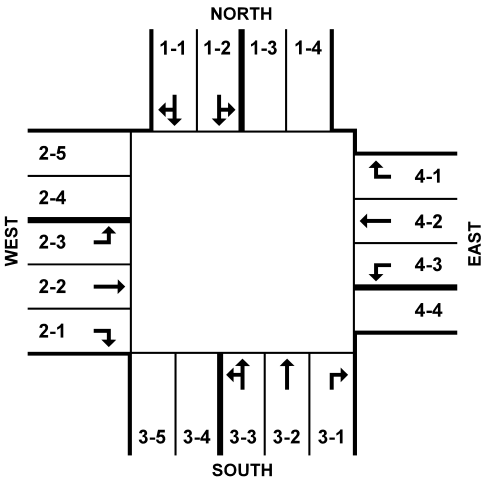

Figure 8. Scheme of the traffic crossorad used for tesing

The crossroad incorporates four arms with several incoming and outgoing traffic lanes in each arm (see Fig. 8). For control of the traffic, standard traffic lights are used.

The actual crossroad also incorporates two-way tram lane, but this lane is not incorporated in the model of the crossroad.

The numbers of vehicles arriving to the particular traffic lanes of the crossroad were based on the observation of the actual crossroad. Pseudorandom generators of traffic flow with exponential distribution with accordingly set parameters were used for adding of new vehicles into the particular traffic lanes.

B. Traffic Crossroad Control Application Used for Testing

For the testing, the Traffic crossroad control application described in Section VI was used. There were six simulated components (depicted using dashed line in Fig. 6), which created environment for three real components (depicted using solid line in Fig. 6).

Two implementation of the *TrafficControlAlgorithm* component were created. Each implementation utilizes a different algorithm for control of traffic in the crossroad using traffic lights. Both algorithms are briefly described in following section.

C. Tested Traffic Control Algorithms

First implementation of the *TrafficControlAlgorithm* component utilizes a simple static algorithm for control of the traffic. This algorithm uses preset time intervals for green and red light signals in particular traffic lanes. The optimal lengths of these signals for one isolated crossroad can be determined using mathematical methods [27]. Because the time intervals do not change over time, the algorithm does not require any sensor input with information about the traffic in the crossroad.

The lengths of the time intervals of the green and red light signals were determined by direct observation of the traffic in the actual traffic crossroad, on which the model of the traffic crossroad for testing is based. However, this actual crossroad uses a dynamic algorithm for control of traffic (see next paragraphs). Hence, the lengths of the time intervals are only approximate. The time intervals of green and red signals for particular traffic lanes used in the model of the crossroad are depicted in Fig. 9.

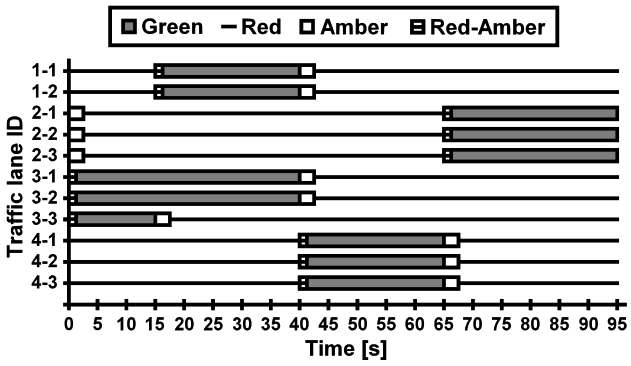

Figure 9. Lenghts of the time intervals of the static algoritm

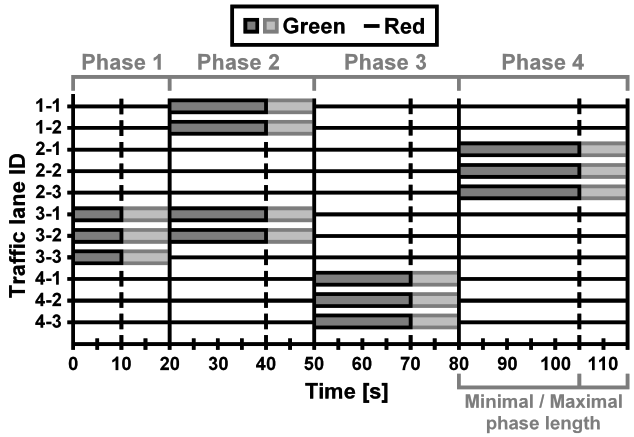

Figure 10. Minimal and maximal lengths of the particular phases

The second implementation of the *TrafficControlAlgoritm* component utilizes a dynamic algorithm for control of the traffic – the Vehicle actuated signal control (VASC).

Using this algorithm, the green and red light signals in particular traffic lanes are grouped into phases with fixed order. In each phase, there is a configuration of the green and red signals. Each phase has a minimal and a maximal length. When a phase is active, it lasts at least till the end of its minimal length. Then, the traffic lanes with green light signal are being observed using sensors. If there is a vehicle detected in one or more of the observed traffic lanes, the length of the phase is prolonged for a small amount of time. This continues until no vehicle is detected or the maximal length of the phase is reached [28].

This algorithm is used in the actual crossroad, on which the model of crossroad for testing is based. The minimal and maximal lengths of the particular phases are loosely based on direct observation (see Fig. 10).

D. Measured Results

During the testing, the mean lengths of vehicle queues in particular traffic lanes of the crossroad were observed. These values could be obtained easily from the simulation of the traffic crossroad (*TrafficCrossroad* component). The results are depicted in Fig. 11 and summarized in Table I. Each value was calculated as average from ten simulation runs.

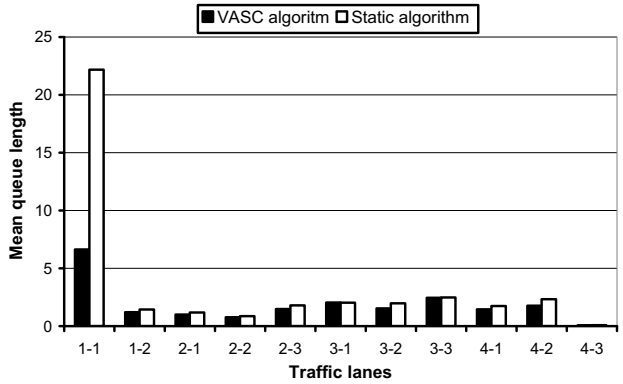

Figure 11. Lenghts of vehicle queues in particular traffic lanes

As can be seen in Fig. 11 and Table I, the control of the traffic crossroad using the *TrafficControl* component with VASC algorithm ensures shorter vehicle queues in nearly every traffic lane than using the *TrafficControl* component with static algorithm. This is an expected behavior, because the VASC algorithms should provide better control of the traffic than the static algorithm.

The results also show the utilization of the SimCo simulation framework and the simulated components for testing of the properties of the real tested components. In this particular case, the simulated components (*TrafficCrossroad*, *TrafficLightsController*, *InductionLoop*, etc.) enable to test the properties of various implementations of the *TrafficControlAlgorithm* component without access to the real hardware of the traffic crossroad.

TABLE I. LENGHTS OF VEHICLE QUEUES IN TRAFFIC LANES

	Mean vehicle queue length [number of vehicles]	
Traffic lane ID	VASC algorithm	Static algorithm
1-1	6.6292	22.1786
1-2	1.2105	1.4416
2-1	1.0041	1.1849
2-2	0.7621	0.8575
2-3	1.4850	1.8035
3-1	2.0473	2.0356
3-2	1.5396	1.9733
3-3	2.4485	2.4745
3-1	1.4551	1.7416
3-2	1.7666	2.4301
3-3	0.0699	0.0892

VIII. FUTURE WORK

In our future work, we will focus on unresolved issues of the SimCo simulation framework and on its testing using the described and also other case studies.

A. Solving the Issues of the SimCo Simulation Framework

The main unresolved issue of the SimCo simulation framework is the handling of Java Core API calls of the real tested components, which involve time or remote connections (see Section V.D). The most promising solution seems to be the replacing of these potentially dangerous API calls of the real tested components using the AOP or manipulation of their bytecode. In our future work, we will focus on its implementation and testing

B. Further Testing using the Presented Case Study

Another direction of our future work is utilization of the described case study and other case studies for further testing of the SimCo simulation framework and extending of its abilities.

In the described case study, it should be for example considered that the component application for crossroad traffic control is likely to be running on a computer with restricted computational power and/or limited memory. This can negatively influence the *TrafficControlAlgorithm* component or other components. Hence, it would be convenient to enable to simulate the restricted hardware conditions to provide the environment for the real tested components, in which they are intended to be running. Then,

measuring of response time of particular services of the components can be performed without necessity to use real crossroad hardware.

IX. Conclusion

In this paper, we described the SimCo simulation framework for testing of the real software components in simulated environment. The simulated environment constitutes from simulated components, which provide required services for the real tested components. The simulated components also enable testing and measuring of the extra-functional properties, quality of services, and also functionality of the real tested components.

The functionality of the SimCo framework was demonstrated on a case study – the component based application for road traffic crossroad control. Two implementation of the *TrafficControlAlgorihtm* component were tested and their qualities of control of traffic were compared using mean lengths of the vehicle queues in particular traffic lanes. As expected, the VASC dynamic algorithm gave betters results than the static algorithm.

In our future work, we fill focus on finishing, enhancing, and further testing of the SimCo simulation framework as discussed in previous section.

Acknowledgment

This work is supported by the Grant Agency of the Czech Republic under grant "Methods of development and verification of component-based applications using natural language specifications," Czech Science Foundation (GACR) 103/11/1489.

References

[1] C. Szyperski, D. Gruntz, and S. Murer, Component Software – Beyond Object-Oriented Programming. ACM Press, New York, 2000.

[2] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," Journal of Systems and Software, vol. 82(1), 2009, pp. 3–22.

[3] P. C. Heam, O. Kouchnarenko, and J. Voinot, "Component Simulation-based Substitutivity Managing QoS Aspects," Electronic Notes in Theoretical Computer Science, vol. 260, 2010, pp. 109–123.

[4] A. Cansado, L. Henrio, E. Madelaine, and P. Valenzuela, "Unifying Architectural and Behavioural Specifications of Distributed Components," Electronic Notes in Theoretical Computer Science, vol. 260, 2010, pp. 25–45.

[5] T. Potuzak, R. Lipka, J. Snajberk, P. Brada, and P. Herout, "Design of a Component-based Simulation Framework for Component Testing using SpringDM", ECBS-EERC 2011 – 2011 Second Eastern European Regional Conference on the Engineering on Computer Based Systems, Bratislava, 2011, pp. 167-168.

[6] R. M. Fujimoto, Parallel and Distributed Simulation Systems. John Wiley & Sons, New York, 2000.

[7] The OSGi Alliance, "OSGi Service Platform Core Specification," release 4, version 4.2, 2009.

[8] D. Rubio, Pro Spring Dynamic Modules for OSGiTM Service Platform. Apress, USA, 2009.

[9] J. A. Miller, Y. Ge, and J. Tao, "Component-Based Simulation Environments: JSIM as a Case Study Using Java Beans," Proceedings of the 1998 Winter Simulation Conference, Washington DC, 1998, pp. 373–381.

[10] C. R. Harrell and D. A. Hicks, "Simulation Software Component Architecture for Simulation-Based Enterprise Applications," Proceedings of the 1998 Winter Simulation Conference, Washington DC, 1998, pp. 1717–1721.

[11] F. Moradi, P. Nordvaller, and R. Ayani, "Simulation Model Composition using BOMs," Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'06), Malaga, 2006.

[12] D. M. Rao and P. A. Wilsey, "Multi-resolution Network Simulations using Dynamic Component Substitution," Proceedings of the 9th Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01), Cincinnati, 2001.

[13] A. Buss and C. Blair, "Composability and Component-Bases Discrete Evenet Simulation," Proceedings of the 2007 Winter Simulation Konference, Washington DC, 2007, pp. 694–702.

[14] A. Buss, "Component Based Simulation Modeling with SIMKIT," Proceedings of the 2002 Winter Simulation Conference, San Diego, 2002, pp. 243–249.

[15] A. Verbraeck, "Component-based Distributed Simulations. The Way Forward?," Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04), Kufstein, 2004.

[16] G. A. Wainer, R. Madhoun, and K. Al-Zoubi, "Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services," Simulation Modelling Practice and Theory, vol. 16, 2008, pp. 1266–1292.

[17] K. Cai, T. Y. Chen, Y. Li, W. Ning, and Y. T. Yu, "Adaptive Testing of Software Components," 2005 ACM Symposium on Applied Computing, Santa Fe, 2005, pp. 1463–1469.

[18] L. Gallagher and J. Offutt, "Automatically Testing Interacting Software Components," Proceedings of the 1st International Workshop on Automation of Software Test, Shanghai, 2006, pp. 57–63.

[19] D. C. Craig and W. M. Zuberek, "Compatibility of Software Components – Modeling and Verification," Proceedings of the International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX'06), Szklarska Poreba, 2006.

[20] S. Becker, H. Koziolek, and R. Reussner, "Model-Based Performance Prediction with the Palladio Component Model," Proceedings of the 6th international workshop on Software and performance, Buenos Aires, 2007.

[21] E. Bondarev, J. Muskens, P. de With, M. Chaudron, and J. Lukkien, "Predicting Real-Time Properties of Component Assemblies: A Scenario-Simulation Approach," EUROMICRO'04, Rennes, 2004.

[22] G. An and J. S. Park, "Cooperative Component Testing Architecture in Collaborating Network Environment," Lecture Notes in Computer Science, vol. 4610, Springer-Verlag, 2007.

[23] J. Wu, L. Yang, and X. Luo, "Jata: A Language for Distributed Component Testing," 15th Asia-Pacific Software Engineering Conference, Beijing 2008.

[24] R. Lipka, T. Potužák, P. Brada, and P. Herout, "SimCo – Hybrid Simulator for Testing of Component Based Applications", SOFSEM 2013, Spindleruv Mlyn, 2013, submitted for publication.

[25] P. G. Gipps: "A behavioural car following model for computer simulation", Transp. Res. Board, vol. 15-B(2), 1981, pp. 403–414.

[26] T. Potuzak, Methods for Reduction of Inter-process communication in Distributed Simulation of Road Traffic, Doctoral thesis, University of West Bohemia, Pilsen, 2009.

[27] S. Guberinić, G. Šenborn, and B. Lazić, Optimal Traffic Control, CRC Press, New York, 2008.

[28] H. Taale, "Comparing Methods to Optimise Vehicle Actuated Signal Control," Procceedings of Eleventh International Conference on Road Transport Information and Control, London, 2002, pp. 114–119.