Extending the UML Profile for Schedulability Performance and Time (SPT) for component-based systems

Dorina C. Petriu and Murray Woodside
Dept of Systems and Computer Engineering, Carleton University, Ottawa Canada
{petriu | cmw} @ sce.carleton.ca

1. Introduction

Component Based Software Engineering (CBSE) is emerging as a paradigm for the development of large complex software systems. CBSE promises to yield cheaper and higher quality assembled systems by reusing configurable generic components that were developed separately [5]. UML 2.0 has extended and improved the representation of components and component-based systems. In UML 2.0, a component is an autonomous unit which has one or more well-defined interfaces (potentially exposed via ports), and its internals are hidden and inaccessible other than as provided by its interfaces [3]. The interfaces *provided* by a component define a formal contract of services offered to the outside world, whereas the interfaces *required* by a component define the services needed from elsewhere. In the performance domain there is a growing interest in modeling component based systems (e.g., [6]). This paper proposes a number of extensions to the STP Profile [2] (more exactly to its Performance Sub-Profile) in regard to component modeling.

2. Component-based case study system

The kinds of UML 2.0 diagrams required for the derivation of a performance model for a component-based system are presented in Fig. 1 to 5 by the means of a e-commerce case study system. In principle, the UML model should describe the software architecture, the deployment of software components to physical resources, the behavioural description of a set of selected key scenarios and the workload applied to each of them. In Fig.1 is shown the component diagram of a e-commerce system, where multiple client browsers are connected through *assembly connectors* (with ball-and-socket symbol) with a *WebServer*, which serves as an intermediary between the clients and the main application server, named *eCommServer*. The *WebServer* parses http requests arrived from the clients and forwards them to the *eCommServer* for processing, then uses its replies to build HTML pages that are send back to the clients. The *eCommServer* component implements the main functionality of the system, represented by use cases such as browse, add/remove products to/from the shopping cart, checkout, create account, etc. (The use case diagram is not give here due to lack of space.)

Fig.2 gives the internal structure of *eCommServer*, showing the classes that realize the services specified by the external provided interface. A delegation connector (drawn with dotted lines) maps requests from an incoming port to internal classes. Similarly, other delegation connectors are used to map the services required by the internal classes to outgoing ports and required interfaces. The ports separate the component's relationship with the outside world from its internal structure. Ports in UML 2.0 are classifiers, and may describe the sequence of behaviours, constraints and overall logic involved in the transition across the port through state machines, if necessary [3].

Fig.3 illustrates the deployment of the *artifacts* that implement the components to physical nodes. This information is necessary for building a performance model of the system, which should capture the contention for all resources, hardware and software. Note that the networks involved (Internet and LAN) are shown as separate nodes.

Figs. 4 and 5 describe the behaviour for the scenario *Checkout* used for performance analysis. In Fig. 4 it is shown the interaction between components, which references the sequence diagram *CreateOrder* given in Fig. 5. Note that *CreateOrder* is described in more detail, as it involves internal classes of the *eCommServer* component. In Fig. 4, *FinancialInstitution* (shown in gray) is an external system and can be represented as an actor in the model. In fact, the operation *checkCreditCard()* performed by it will be modeled as an *external operation* according to the Performance Profile [2]. One of the challenges illustrated in this example is that there are two kinds of levels of abstraction: one based on component boundaries (in order to plug in different components realizing the same interfaces), and another on behaviour boundaries (e.g., in Fig 4 the *CreateOrder* subdiagram is not completely within the *eCommServer* component). This may create problems in structuring the scenarios. The next section addresses the problem of describing component QoS and the role of ports as interfaces between levels of abstractions.

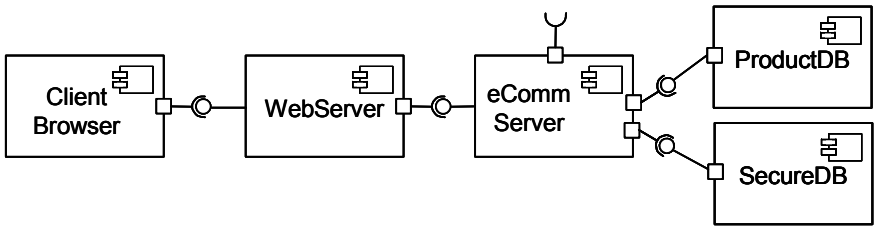

Fig. 1. Software architecture of the e-commerce system

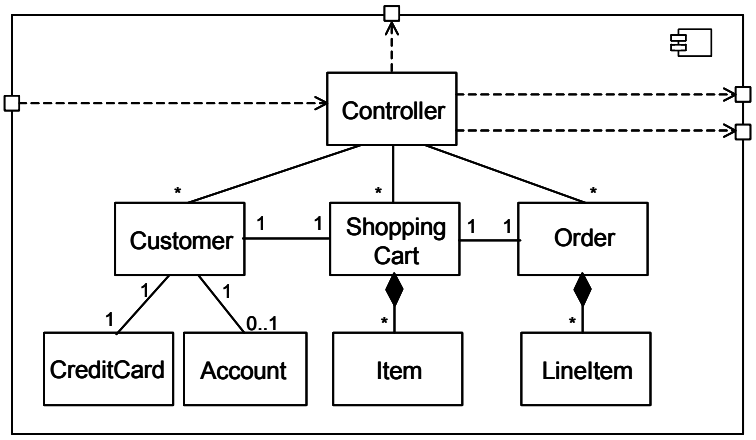

Fig. 2. Structure of the component eCommServer

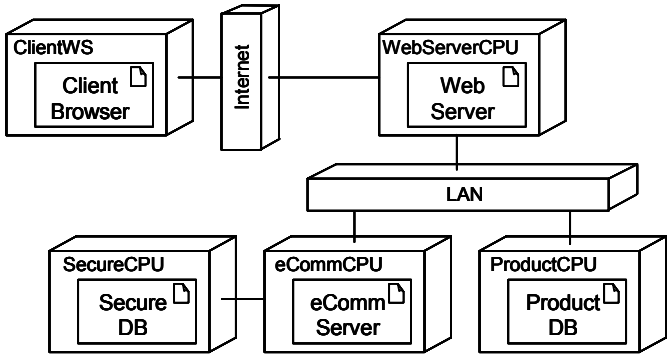

Fig. 3. Deployment of the e-commerce system

3. Performance Profile extensions

3.1. Resources and Services

The first proposed extension is to map the components to resources and their interfaces to services provided (or required) by the resources. So far, the Performance Profile does not use a "Service" stereotype. However, the General Resource Model (GRM), which is a package of the STP Profile, introduces the concept of resources and services. The Performance Sub-profile inherits from GRM, so the extension is not difficult.

In UML 2.0, an interface definition can be mapped to multiple interface realizations, as polymorphism may be involved (i.e., different subclasses may implement the same interface in different ways). This means that the Performance Profile has to allow for multiple service realizations for a given service defined by an interface. In the performance model, each service realization has to be represented separately, as it has a different behaviour, different resource demands, etc. This will have an impact on the way the results will be reported - not per service definition, but per service realization.

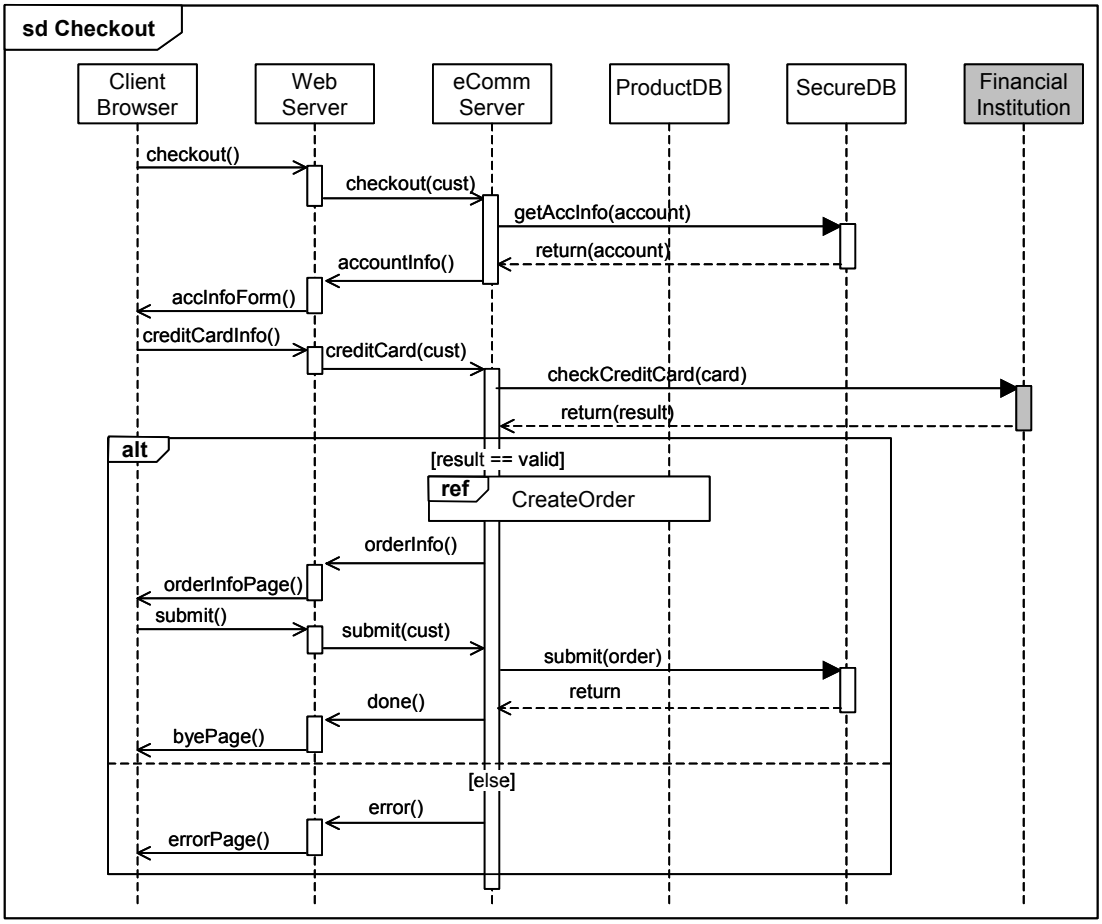

Fig. 4. Sequence diagram for the Checkout scenario

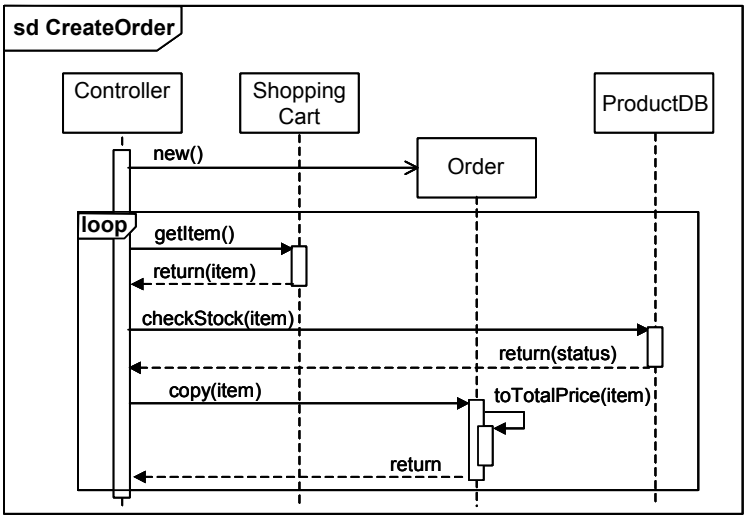

Fig. 4. Sequence diagram CreateOrder referenced by Checkout

3.2. Offered and required QoS

In the present version of the Performance Profile, it is possible to define required and offered performance characteristics (such as response time, throughput, etc.) at the scenario level. (Offered characteristics are obtained as analysis results). Once we introduce provided and required services for resources, the next step would be to attach QoS characteristics to these services.

However, this is not a trivial problem, as "offered QoS" depends not only on the component capacity (which include all underlying resources) but also on the workload offered to the component. For instance, the response time experienced by a request at a component includes not only the service time per se, but also the queueing delay due to waiting for other competing requests. So, the response time depend not only on the components, but on its environment, as well. Therefore, the QoS offered by a component should be expressed as a function of the load. This is complicated when a component offers multiple services, as the service mix plays also a role. More research is needed to find simple yet expressive ways of specifying different offered QoS characteristics. In what regards "required QoS" for required services, there are two arguments for the need to express it as a function of workload, as well: a) it has to be matched with the QoS offered by other resources, which is workload dependent, and b) the Qos required by a certain resource has to be taken into account when determining the QoS offered by the same resource, which is workload dependent.

In the present version of the Performance Profile, the workload level is attached to the first step of a scenario. However, the workload has to be computed for every resource if we intend to express QoS at the resource level as a function of workload.

In the performance evaluation domain was proposed recently a new kind of components that are QoS aware [1]. These components are empowered to perform admission control and to reject request above certain levels in order to insure that they can deliver a certain QoS. The decision whether to accept or reject new requests is taken based on Queueing Network analysis results. However, the majority of today's software components do not have control over their load, and thus cannot guarantee desired QoS levels.

3.3. Ports in the Performance Profile

As mentioned before, ports in UML 2.0 may describe the sequence of behaviours, constraints and overall logic involved in the transition across the port through state machines [3]. Since ports may represent actions or activities, they should be stereotyped as Scenario Steps in the Performance Profile. The performance model should also capture how the ports translate workload levels and QoS values between levels.

4. Conclusions

The paper proposes a number of extensions to the Performance Profile for component-based systems. Some of the extensions are rather straightforward, but others (such as the specification of offered and required QoS as a function of the workload) require more research in the performance domain. Performance analysis will require the flexibility to plug together component performance descriptions as well as system-level behaviour descriptions.

References

[1] D. Mensace, H. Ruan, H. Gommaa, "A Framework for QoS-Aware Software Components", Proc. of 4th Int. Workshop on Software and Performance WOSP'2004, Redwood Shores, California, pp.186-196, Jan. 2004.

[2] Object Management Group, "UML Profile for Schedulability, Performance, and Time Specification," OMG Adopted Specification ptc/02-03-02, July 1, 2002.

[3] Object Management Group, " UML 2.0 Superstructure Specification", OMG Final Adopted Specification ptc/03-08-02, 2003.

[4] D. C. Petriu and C. M. Woodside, "Performance Analysis with UML," in *UML for Real*, B. Selic, L. Lavagno, and G. Martin, Eds. Kluwer, 2003, pp. 221-240.

[5] C. Szypersky, with D. Gruntz and S, Murer, "Component Software: Beyond Object Oriented Programming", 2nd Edition, Addison-Wesley, 2002.

[6] X.Wu, C.M.Woodside, "Performance Modeling from Software Components", Proc. of 4th Int. Workshop on Software and Performance WOSP'2004, Redwood Shores, California, pp.290-301, Jan. 2004.