Tailor-Made Containers: Modeling Non-functional Middleware Service

Ronald Aigner, Christoph Pohl, Martin Pohlack, and Steffen Zschaler

Technische Universität Dresden
Dresden, Germany
{Ronald.Aigner, Christoph.Pohl, Martin.Pohlack,
Steffen.Zschaler}@inf.tu-dresden.de

Abstract. We propose to create tailor-made application servers by composing components providing support for individual non-functional properties. In this position paper we start from a static, asymmetric approach splitting the application server in two parts to support realtime properties and generalize this to a symmetric, componentized architecture. We then analyse how modelling of non-functional properties of systems is influenced by this application server architecture.

1 Introduction

In the COMQUAD[1] project we are developing a container that enables guarantees of non-functional properties of component-based software. In a recent paper [1] we reported on the container's architecture, which has been split into two parts—one which is generic, but cannot give realtime guarantees, and one which is specific to realtime—to enable us to efficiently support realtime properties, as well as confidentiality properties, while reusing large parts of code for component management (namely the open-source JBoss application server [2]). In this position paper we want to explore how this idea of splitting the middleware into two parts can be generalized to other properties, and how this affects the modeling of these properties. In particular, we want to examine how the container and the properties it provides for can be modeled in such a way as to allow a property-specific container to be assembled from various parts.

We propose to modularize containers as aspects that are woven together with the actual component *depending on the set of non-functional properties to be supported*. A container then consists of a core part, which is responsible for generic component management and provides the hooks to which the component itself, as well as the aspects implementing various non-functional properties, can be connected. In order for this to work automatically, either at runtime or at deployment time, both the aspects and their relation to the non-functional property(ies) they realize need to be modeled. In particular it is important to model the interdependencies between different properties or their implementing aspects, respectively.

[1] COMponents with QUantitative properties and Adaptation, a project funded by the German Research Council

The rest of this position paper is structured as follows: Section 2 gives a short overview of our split container architecture as presented in [1]. This architecture is then generalized in Sect. 3 into a componentized, symmetric architecture. The main section of the paper is Sect. 4, which gives a more detailed analysis of different approaches to modeling the relation between non-functional properties and the aspects implementing them. Finally, we briefly review some related work and summarize our position paper.

2 A Split Container Architecture

In the course of the COMQUAD project we built a component container with support for non-functional properties. This includes the ability to fulfill realtime requirements of components. Support for realtime includes the ability of the middleware—the container—and the underlying operating system to provide guarantees for timeliness of execution. It also includes that resources required by a component can be provided to this component when required within a predefined time span. Such guarantees can be given if the component—or some representative of the component—make a reservation with a manager of the resources. The resource manager will then guarantee the timely access to the resource.

Thus, it is essential to extend the container by a resource management providing services for admission and reservation of system resources, such as CPU time, network bandwidth, or memory. The key part of the container is the contract manager, which instantiates and connects the components required to service a client request. In order to select the correct components and component implementations, the contract manager uses the functional and non-functional (including realtime) component specifications.

Typically, applications that give guarantees on realtime properties are split into two parts: *(i)* a small part of code that performs the actions for which guarantees of realtime properties are essential and *(ii)* a usually much larger part for which realtime guarantees are not important. This distinction can be applied to the architecture of a component runtime environment. Most of the complex work is done in the non-realtime part of the component environment. It manages the available component implementations and application specifications, handles requests for creation of component networks, negotiates contracts between components, and maintains a model of the instantiated components and the networks formed by them. The realtime part is essentially restricted to actually instantiating components, reserving resources, and executing the instantiated components in response to user requests.

This concept is closer to the reality of applications with realtime properties than the approach of monolithic realtime applications. Additionally, it also allows us to make use of advanced component technology (namely JBoss [2]) on the non-realtime side while still being able to make full use of the realtime and resource reservation features of the underlying platform on the realtime side. Thus, we can leverage the best of both worlds and move towards a running prototype very efficiently.

3 A Componentized Container Architecture

The split container architecture we introduced in Sect. 2 proved to be a sensible decision for the design of dependable component-oriented systems with support for realtime guarantees. Our current container architecture is now split into two parts: The non-realtime container is based on JBoss, running in a standard Java Virtual Machine (JVM) on L^{4}Linux and providing complex services, such as a component repository. The realtime container is running directly on the L4 microkernel. It contains all the realtime-capable components and the necessary infrastructure.

This motivated us to investigate if other non-functional properties can be handled in a similar way. Consequently, we tried to find similar "splitting lines" in the corresponding middleware support code for those properties and examined possible correlations to our previously introduced architecture split for realtime–non-realtime system parts.

The idea was that, provided we could show these splits to be "congruent"[2], we would then be able to generalize our split architecture for use with other properties. However, these splits are not congruent in general; they rather relate to each other in an arbitrary fashion. We therefore propose a more symmetric architecture, which has been depicted in Fig. 1. This architecture provides a set of *core services* such as instantiation, component connection, and so on, and a set of *property dependent services*. We propose a container generation operator $\otimes$ that creates custom-made containers by selecting the appropriate property dependent services and weaving them together. This container generation is performed either dynamically at runtime or statically at deployment time.

Container generation can only be automated if the non-functional properties under consideration as well as the components providing support for them have been appropriately modeled. This will be the subject of the following section.

4 Modeling Aspects

In this section we are going to evaluate various approaches to modeling property dependent services so that they can be automatically selected and merged. We start out by looking at the JMX [3] architecture as a conceptual model and explain why this is not sufficient for our purposes. We then show how aspect-oriented approaches can be applied in our case, but also what the related drawbacks are.

The idea arose to model specific non-functional aspects using a plug-in–extension model such as JMX in JBoss [2]. To illustrate this approach and its limitations we will outline some steps required to provide fuzzy time. One important aspect of security is confidentiality—that is, information must not become available to unauthorized subjects. Covert channels are one way through which information could be transfered secretly and must therefore be considered if confidentiality is one required non-functional property. Covert channels can be classified into storage and timing channels and have been a research subject for many years [4].

In practice covert timing channels cannot be prevented completely. Instead it is common to limit their bandwidth by impeding unprivileged subjects' access to accurate

[2] congruent in terms of mutual non-overlapping parts: a lean, mission-critical part and a non-critical management part, which are identical for different non-functional properties

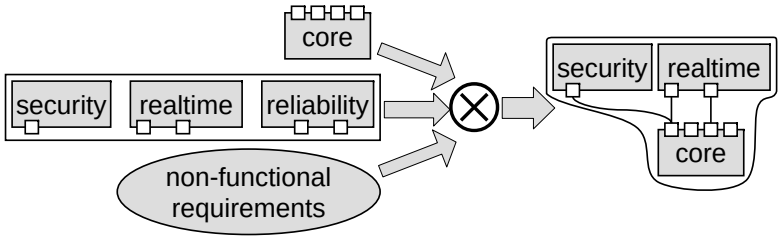

Fig. 1. Composition of a property-specific container from core and property-specific functionality

time sources. For instance, a system could delay the execution of each system call for a random amount of time. In the literature this is called imprecise clocks or fuzzy time [CITE!].

There is an obvious conflict between realtime requirements and the imprecision of time, as timeliness can not be assured for specific services at the same time as the access to accurate timers is restricted. Investigations about the impact of such solutions on realtime systems have already been undertaken years ago. Son et al. [5] describe a very domain specific solution for a realtime database with security requirements. The authors propose to temporarily weaken the security requirements to fulfill realtime requirements, thereby providing a tradeoff between realtime performance and security.

Enforcement of fuzzy time requires pervasive modifications of several underlying services, possibly including program code inspection. All access to time sources must be encapsulated. Possibly the simplest thing to do would be to add a random delay to all services providing timing information, for example a system call to operating system. However, one also needs to prevent access to other time sources, such as NTP servers, which are reachable via network access. As one cannot filter all network traffic for all time sources (network traffic may be encrypted or steganography might be used to hide the data transported), network access must also be delayed for a random amount of time. Unfortunately, many services might be used as time source. Consequently, to provide fuzzy time many components must be considered. This is more than can be modeled with the concepts provided by JMX, which allow only simple non-invasive extensions to the container's functionality.

Invasive modifications are the domain of aspect-orientation [6, 7]. We will therefore analyze how we can use aspects to model property dependent services. Looking more closely at the examples from our experience, we find that we need to extend the standard notion of aspects in at least three ways:

1. Aspects providing for non-functional properties require join points on two levels:
   (a) A runtime level, where they need to be able to intercept method calls, events, or data packets on a stream connection. These join points can be and have been conveniently implemented using interceptors.
   (b) A meta-level, where they need to be able to intercept state changes in a component's life cycle. These join points cannot be implemented by standard interceptor technologies, but require special reflective support at components' meta-level [8].

   These special issues of dynamic insertion and configuration of non-functional properties have also been realized by the developers of OpenORB/OpenCOM [9], Lasagne [10], and OIF [11], among others.
2. Aspects come with pre-conditions. For example, an aspect may require certain other aspects to be present or absent, or certain operations to be available, in order to provide the non-functional property it supports. These pre-conditions are semantic in nature; that is, they do not only require certain names to be available, but certain structures resp. behaviors. This can potentially be combined with techniques from the Model Driven Architecture (MDA, [12]) using UML extension mechanisms for semantic markup and code generation.

3. Aspects need to be parameterized. For instance, because an aspect providing response time properties will be used for different specific response times, there must be a way to pass this in as a parameter, which may already have to be considered at the time of weaving.

An open research issue remains the interference of apparently orthogonal aspects. There are no general solutions to model such influences. The approach of simply using constraint UML dependencies and OCL expressions is clearly limited to effects that are known in advance. However, unanticipated interactions and side-effects are much harder to capture during the software development life cycle. This effect is actually a subset of a problem area called "feature interaction" [13], which has been discussed in the telecommunications community since the early 1990s. An example for this is the treatment of security by means of cryptography, which we introduced in [14]. The container infrastructure has to handle the tradeoff between performance (in terms of timeliness) and security (in terms of confidentiality and integrity). We believe that there are many of these conflicting non-functional requirements and presented these as a prominent and apparent example. We also believe that there are many domain-specific models to cope with the conflicting goals, which should be integrated into a common modeling framework.

5 Related Work

The concepts of Model Driven Architecture (MDA) and Model Driven Middleware [15] include the application of models to middleware configuration and implementation. These include the integration with QoS aware middleware and application. The described tool chain and models also mention the problem of interfering requirements, but do not provide a solution beyond their detection. Nonetheless does MDA provide means to describe QoS requirements on different levels of abstraction.

The PURE operating system family [16] exploits the concept of aspect oriented programming to build tailored operating systems. Similar to our concept of generated containers for specific non-functional properties PURE uses AspectC++ [17] to generate operating systems adapted to the demands of the application running on top of it and the available hardware. Aspects are used to describe and implement the features of the operating system, such as scheduling strategies. As described in this paper, aspects cannot be applied to interfering non-functional properties, such as security and realtime.

Requirements for realtime extensions for Java were defined in the NIST report [18]. The NIST group proposes partitioning the execution environment into a realtime core providing the basic realtime functionality and a traditional JVM, which services normal Java applications. Based on these requirements, the J Consortium defined the Real-Time Core Extensions for Java (RTCE) [19], which follow the idea of a separate core for realtime services. In contrast, in the Real-Time Specification for Java (RTSJ) [20] all services are provided in one JVM, as such containing the realtime and the non-realtime applications. The architectural RTCE approach is similar to the design of our system, in that both run large and complex parts in a classic non-realtime environment and only small, predictable parts in a realtime environment.

Dassault Systèmes introduced a general architecture for software components with support for arbitrary non-functional properties [21]. Their concept for weaving non-functional aspects is very similar to our proposition in that they suggest to generate specialized containers comprising the specific services that are actually needed according to the non-functional requirements of the application. New services can be defined using an aspect definition language; they can be put at use by means of an aspect user language. However, there is no concept so far for supporting composition of aspects. Feature interaction and interference of orthogonal aspects also remain open issues. Also, the non-functional properties shown in [21] are those which can already be handled by modern application servers. It is not clear from the paper whether the authors would be able to model a realtime aspect such as frame rate of a video player.

6 Conclusions

In this position paper we presented the shift from our split container architecture with support for realtime properties towards a more general, componentized approach to dynamically weaving customized runtime environments with support for arbitrary non-functional aspects for component-based software applications. We conclude that the generalized version of the container architecture is a promising step forward, that needs further work and which we would like to suggest as a field for future research. The most interesting research issues remain in the field of modeling the different services required for supporting non-functional aspects. We have shown that even the current state of Aspect-Oriented Modeling still has a number of drawbacks, especially in terms of capturing interferences of aspects, which require more work in that direction.

In [22] and [23] we presented a formal approach to the specification of non-functional properties of component-based systems. The work started in this position paper should enable us to extend the work reported on in those papers to support independent specification of multiple non-functional properties of the same system.

References

1. Göbel, S., Pohl, C., Aigner, R., Pohlack, M., Röttger, S., Zschaler, S.: The COMQUAD component container architecture. In Magee, J., Szyperski, C., Bosch, J., eds.: 4th Working IEEE/IFIP Conf. on Software Architecture (WICSA), Oslo, Norway, IEEE (2004) 315–318
2. Fleury, M., Reverbel, F.: The JBoss extensible server. In Endler, M., Schmidt, D., eds.: International Middleware Conference. Volume 2672 of LNCS., Rio de Janeiro, Brazil, ACM / IFIP / USENIX, Springer (2003)
3. Sun Microsystems: Java Management Extensions (JMX) Instrumentation and Agent Specification. v1.2 edn. (2002) http://java.sun.com/products/JavaManagement/.
4. Millen, J.: 20 Years of Covert Channel Modeling and Analysis. Technical report, Computer Science Labroratory, SRI International, Menlo Park, CA (1999)
5. Son, S.H., Mukkamala, R., David, R.: Integrating security and real-time requirements using covert channel capacity. IEEE Transactions on Knowledge and Data Engineering **12** (2000) 865–879

6. Elrad, T., Aldawud, O., Bader, A.: Aspect oriented modeling—bridging the gap between design and implementation. In Batory, D., Consel, C., Taha, W., eds.: Generative Programming and Component Engineering (GPCE 2002). Volume 2487., Pittsburgh, PA, USA, ACM SIGPLAN/SIGSOFT, Springer (2002) 189–201
7. Aldawud, O., Elrad, T., Bader, A.: UML profile for aspect-oriented software development. In Aldawud, O., ed.: 3rd International Workshop on Aspect-Oriented Modeling with UML, Boston, USA, ACM SIGSOFT / SIGPLAN (2003) In conjunction with the International Conference on Aspect-Oriented Software Development (AOSD'03).
8. Kiczales, G., Rivieres, J.D., Bobrow, D.G.: The Art of the Metaobject Protocol. MIT Press (1991)
9. Blair, G.S., Coulson, G., Robin, P., Papathomas, M.: An architecture for next generation middleware. In Davies, N., Raymond, K., Seitz, J., eds.: Middleware 1998, The Lake District, England, IFIP, Springer (1998)
10. Jørgensen, N., Truyen, E., Matthijs, F., Joosen, W.: Customization of object request brokers by application specific policies. In: Distributed Systems Platforms. Volume 1795 of LNCS., New York, IFIP/ACM, Springer (2000) 144–163
11. Filman, R.E., Barrett, S., Lee, D.D., Linden, T.: Inserting ilities by controlling communications. Communications of the ACM **45** (2002) 116–122
12. Miller, J., Mukerji, J.: MDA Guide. The Object Management Group. Version 1.0.1 edn. (2003) OMG document number omg/2003-06-01.
13. Pulvermüller, E., Speck, A., D'Hondt, M., DeMeuter, W., Coplien, J.O.: Feature interaction in composed systems, ECOOP 2001 Workshop Proceedings. Technical Report 2001-14, Universität Karlsruhe (2001)
14. Franz, E., Pohl, C.: Towards unified treatment of security and other non-functional properties. In: Workshop on AOSD Technology for Application-Level Security (AOSDSEC'04), Lancaster, UK (2004)
15. Gokhale, A., Schmidt, D., Natarajan, B., Gray, J., Wang, N. In: Model Driven Middleware. John Wiley & Sons, Ltd. (2004)
16. Beuche, D., Guerrouat, A., Papajewski, H., Schröder-Preikschat, W., Spinczyk, O., Spinczyk, U.: On the development of object-oriented operating systems for deeply embedded systems - the PURE project. In: Proc. 2nd ECOOP Workshop on Object-Orientation and Operating Systems (Lisbon, Portugal). (1999) 27–31
17. Spinczyk, O., Gal, A., Schröder-Preikschat, W.: Aspectc++: An aspect-oriented extension to c++. In: Proc. 40th Int'l Conf. on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002), Sydney, Australia (2002)
18. National Institute of Standards and Technology: Requirements for Real-time Extensions for the Java Platform. (1999) Available at http://www.nist.gov/rt-java/.
19. J Consortium: Real-Time Core Extensions (RTCE). (2000) Available at http://www.j-consortium.org/.
20. The Real-Time for Java Expert Group: The Real-Time Specification for Java. v1.0 edn. (2001) http://www.rtj.org/.
21. Duclos, F., Estublier, J., Morat, P.: Describing and using non functional aspects in component based applications. In: Proc. 1st Int'l Conf. on Aspect-Oriented Software Development, Enschede, The Netherlands, ACM, ACM Press New York, NY, USA (2002) 65–75
22. Zschaler, S.: Towards a semantic framework for non-functional specifications of component-based systems. In: Proc. EUROMICRO conference 2004, track on Component-based Software Engineering, Rennes, France (2004) To appear.
23. Zschaler, S.: Formal specification of non-functional properties of component-based software. In: Proc. Workshop on Models for Non-functional Aspects of Component-Based Systems. (2004) Submitted for Publication.