Addressing Ubiquitous Software Complexity with Mobile Containers

Vasian Cepa
cepa@informatik.tu-darmstadt.de
Darmstadt University of Technology

1 Using Containers to Structure Mobile Applications

We present here an overview of the idea of applying the software containers model to mobile applications using generative programing techniques. We also present an evaluation prototype called MobCon.

A software container is a wrapper component that offers services nearly transparently to other components that '*live*' inside it. The wrapped components contain the functional logic of the application. The services offered by the container are secondary to the logic of the application, but are nevertheless necessary to get an application running. The container services, known as technical concerns [14], include data persistence, logging etc. Components that live inside a container are usually written in a more restricted way that normal components. This restrictions are compensated by automatic usage of container services. The container itself is an abstraction of the surrounding middleware that an application uses. The container concept is made known by its original usage in enterprise application frameworks like EJB [13] and COM+ [6].

A container offers services that are usually cross cutting concerns with the rest of application code. Enterprise containers are specialized for e-commerce business applications. We can reuse the container concept and specialize it to other classes of applications. We are interested to apply this concept to mobile applications. Using a container bring well-known benefits like the natural separation of programmer roles that write functional and technical code. Once the technical concerns are identified, they are coded and debugged and made of the container logic. This reduces the bugs and increases productivity. Apart of this the container idea is especially benefiting for mobile applications. The container hides the middleware related API-s from the rest of the application in a centralized way. Ideally porting a family of mobile applications to a new middleware platform would require only porting the container. This is very important considering the speed the mobile middleware [4] changes today. Even different versions of the same middleware may expose different API-s. The container abstraction can be used to address such mobile software problems in a structured way.

There are several reasons, however, why the enterprise container idea cannot be directly adopted for mobile applications. First, since we are addressing a new domain of application we need to find and parameterize the technical concerns of the mobile applications to be addressed. Second, and more importantly, the design issues involved in building a container have different constrains in the enterprise as compared to mobile applications. For enterprise applications scalability is the main issue. For mobile applications the main issue is performance of the container itself: Every time we introduce a new abstraction, we introduce new layers. Having many layers of abstractions is however unacceptable for mobile applications that run on resource limited devices. Last but not least, current enterprise container technology has several drawbacks [16], which are also unacceptable for the domain of mobile applications, including the inability of such technology to be tailored to application specific needs, its complexity due to lack of tight language integration, as well the implied dependency from a particular component middleware platform, a serious limitation in the mobile domain, where middleware changes rapidly.

Generative techniques in the idea of OMG MDA [5] combined with taking into consideration the cross-cutting nature of technical concerns that a container addresses can be used to implement a mobile container framework. The current research in MDA tries to find ways how to specify such transformations so that they are done automatically. While nowadays we are still far from modeling an entire application instead of writing its code, the MDA ideas are still valid, if we try to organize the source code of an application in a platform independent way that enables automatic transformations. This way we can focus on the main functionality and introduce the technical concerns automatically later. To achieve this we need language support for MDA concepts such as marking with tags and AOP like [1] cross cutting code decoration techniques. Languages used for mobile applications are usually restricted dialects of mainstream languages. Tools that transform mobile source code may not be directly available. This means that we should use techniques that can be easily introduced to any language [1].

[1] For example we cannot use AspectJ [7] with J2ME MIPD, a Java dialect for mobile applications that we use in MobCon prototype.

2 MobCon: A Mobile Container Framework

To demonstrate the feasibility of the proposal, a mobile container prototype called MobCon is being implemented. In the following, we only outline the structure and features of this prototype; more details can be found in [12].

Based on the ides of marking in MDA and presence attributes in several languages like .NET we presented the idea of Generalized and Annotated AST - GAAST languages, that can be used to facilitate marked transformations at the source code level. GAAST idea requires two features to be supported by a language technology (a) support for annotations (marking in MDA) of arbitrary program elements with user-defined tags which are first-class program units with well-defined semantics, and (b) support for explicit meta-representation of programs that is accessible in a programmatic way. A GAAST proposal evaluation in the form of the CT-AST API prototype is developed as part of the MobCon transformer framework.

We selected Java 2 Micro Edition Mobile Information Device Profile (MIDP) 2.0 [3] for our prototype because is relatively simple (we can focus on the container concept not in the technology) and hardware independent and it is well supported. The process of identifying technical concerns for a set of mobile applications is similar to defining software product lines [8]. Various technical concerns have been addressed until now in the MobCon prototype such as data persistence, screen management, logging, image adaptation, encryption, session and context. We use a source code template based approach. Our templates are implemented as Velocity [9] scripts.

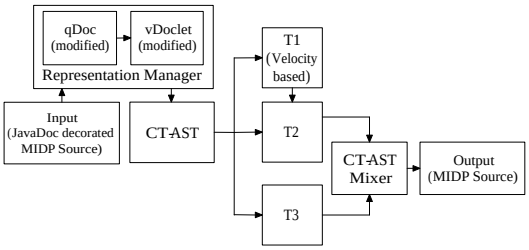

Figure 1: MobCon Transformers

Fig. 1 schematically shows the MobCon transformer framework. As part of this framework, we have implemented a MIDP-based prototype language technology with explicit support for tag-based model-driven development (represented by the Representation Manager (implemented with customized versions of QDox [15] and vDoclet [17]) and the Class Template AST (CT-AST) in Fig. 1). Tags allow us to associate new semantics with language entities. JavaDoc tags are used in our prototype to simulate source code entity annotations, since MIDP lacks such support. The representation manager processes annotated source code, producing a CT-AST representation of it, which serves as the input for the transformers. Only three different such (Velocity-based) transformers (Ti) and their interactions as they process the input introducing the technical concerns directed by the annotations of the input CT-AST-s are shown. Some concerns like image adaptation require code to be placed on the server side. The MobCon framework generates Java code for such concerns, for both mobile side and the server side. The network communication is handled by the framework. Messages are handled to the appropriate application using bookkeeping data, part of session and context concerns. Three types of uses are identified: (a) those that use predefined concerns (tags); (b) those that can append and modify concerns dealing with MobCon transformer framework; (c) users that port the framework to a new set of middleware. The MobCon framework itself is independent of J2ME MIDP and is written in Java. Only transformer scripts written in Velocity are platform dependent.

Generation can be used successfully to implement container-like approaches for small systems. The work of Voelter [11] targets generative component infrastructures for embed systems. Unlike MobCon the idea is that the entire operating system can be customized for the application. PicoContainer [10] is a generic container project based on constructor parameters that tries to define a minimalistic container to be used also in client applications. This project focuses on containers in general, while MobCon is specialized for mobile applications, trying to reduce abstraction layers via generation. The work of Popovici et al [2] investigates the application of so-called *spontaneous containers* in mobile clients. The idea is that the container ideally changes itself to be adapted to environment changes at run-time without stopping the application. MobCon on the other hand addresses ways to structure mobile applications so that they can be maintained easier. Also run-time dynamic AOP requirements of spontaneous containers require more powerful computing devices that those addressed by MobCon framework.

References

[1] A. Rashid A. Colyer, G. Blair. Managing Complexity In Middleware. *AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, 2003.

[2] A. Popovici, et al. Spontaneous Container Services. *ECOOP*, 2003.

[3] C. Bloch and A. Wagner. *MIDP 2.0 Style Guide for the Java 2 Platform, Micro Edition*. Addison-Wesley, 2003.

[4] W. Emmerich C. Mascolo, L. Capra. Middleware for Mobile Computing. *In Advanced Lectures on Networking - Networking 2002 Tutorials, Springer Verlag, LNCS 2497*, pages 20–58, May 2002.

[5] D. S. Frankel. *Model Driven Architecture - Applying MDA to Enterprise Computing*. Wiley, 2003.

[6] T. Ewald. *Transactional COM+: Building Scalable Applications*. Addison-Wesley, 2001.

[7] J. Hugunin M. Kersten J. Palm W. G. Griswold G. Kiczales, E. Hilsdale. An Overview of AspectJ. *In Proc. of ECOOP '01, Springer-Verlag, LNCS 2072*, pages 327–353, 2001.

[8] J. Bosch. *Design and Use of Software Architectures, Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2002.

[9] J. Cole J. D. Gradecki. *Mastering Apache Velocity*. John Wiley & Sons Inc, 2003.

[10] PicoContainer. *http://www.picocontainer.org/*, 2003.

[11] M. Voelter. A Generative Component Infrastructure for Embedded Systems. *Position Paper at Reuse in Constrained Environments Workshop at OOPSLA 03*, 2003.

[12] MobCon: A mobile container framework prototype for J2ME MIDP. *http://www.st.informatik.tu-darmstadt.de/static/pages/projects/mobcon/index.html*, 2003.

[13] R. Monson-Haefel. *Enterprise JavaBeans*. Addison-Wesley, 2000.

[14] D. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 1972.

[15] QDox Java Tag Parser. *http://qdox.codehaus.org/*, 2003.

[16] R. Pichler, K. Ostermann, M. Mezini. On Aspectualizing Component Models. *Software Practice and Experience, Volume 33, Issue 10, pp. 957-974*, 2003.

[17] vDoclet Java Code-Generation Framework. *http://vdoclet.sourceforge.net/*, 2003.