

Dynamic Updates of Existing, Distributed Applications

Submitted for Doctoral Symposium at ECOOP 2005

Robert Bialek

University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen, Denmark

Abstract. This paper is about our Ph.D. research on Dynamic Updates (DU) of existing, distributed applications. DU are supported by partitioning applications and using the partitions as update units in a component-like, updateable framework. The framework encloses the necessary functionalities for code replacement, state transfer, interface adaptations and update propagation. We demonstrate viability of our ideas by dynamically updating an existing, 3rd party, messaging Java application.

1 Introduction

The growing popularity of the Internet leads to more and more applications being distributed and inter-connected. Nodes of such inter-connected applications require other nodes to be continuously available. At times, such 24/7 applications have to be updated, e.g., to meet new requirements.

1.1 Research problem and hypothesis

The research question is how to allow evolution of continuously executing applications that were not prepared for updates from the outset.

Our hypothesis is that support for Dynamic Updates (DU) can be added to applications after their initial development. By modifying application code at load time or using a pre-processor, the necessary level of indirection and update control mechanisms can be provided. The modifications to the application can be performed (almost) transparently allowing seamless introduction of DU support to existing applications, which were not prepared for DU from the outset.

To prove our hypothesis, we analyse Java applications. We choose Java of two reasons: 1) Popularity, especially for building distributed applications, which makes it easier to find practical examples to illustrate our ideas, and 2) Richness. Java is a rich language with functional properties, side effects, inheritance and others. All these properties need to be considered when building dynamically updateable systems.

1.2 Existing approaches

A number of frameworks address the problem of DU of Java applications. Frameworks such as SOFA [5] and Formaware [6] support dynamic evolution of Java applications by introducing the notion of *updateable components*. In these frameworks, updateable applications need to be built from scratch using the updateable components concept.

Many existing applications, however, are not built using component based approaches. To allow DU of such applications, other solutions have to be found. One approach to solve this problem is to modify the Java Virtual Machine (JVM), as presented in [3] and [4]. In both these approaches, a level of indirection on method calls is introduced so that calls can be redirected at run-time, allowing for dynamically replacing classes. Unfortunately, using a modified virtual machine is not an acceptable solution in all situations as it would require the modified JVM to run on *all* nodes of a distributed application and, therefore, disables the use of a standard JVM. Furthermore, supporting updates of classes in a modified JVM introduces a level of indirection on every method call. This may have a negative impact on performance.

We do not know of any research that addresses the problem of supporting DU of existing applications.

2 Approach

We use the following approach: 1) identify the necessary functionalities supporting DU, and 2) provide a solution for DU of existing applications.

2.1 Identifying the functionalities supporting DU

To support DU of existing applications, we clarify the concept of updates and dynamic updates. *Updates* are changes to applications, where a link between the old and the new version can be provided. *Dynamic Updates* are updates performed on executing applications.

We have identified three kinds of DU and the functionalities supporting them:

1. Local Updates (LU) - are updates of an application's module that are not visible to the rest of the application (e.g., update of a method's body). To support LU the functionalities of *code replacement* and *state transfer* need to be present. State transfer is expressed in a State Transfer Function (STF), which can support both in-process, and out-of-process updates. In-process updates require "advanced" code manipulation to capture the running stack.
2. Global Updates (GU) - are updates of an application's module that influences the rest of an application (e.g. modified method name). To support GU we need adapters that will translate calls from old-version modules to the updates modules and vice versa. Adapters can be expressed in adapter functions.

3. Coordinated Updates (CU) - are updates of an application's module that can not be hidden by adapters (e.g. modified protocol). CU require that all modules involved need to be updated atomically. To support CU, a coordination mechanism, e.g. a Transaction Manager (TM), needs to be present. Formaware framework[6] demonstrates this approach.

The support for LU, GU, and CU together with a support for update propagation in form of update requests, was packed into an updateable framework called DUCS[1].

2.2 Supporting DU of existing applications

Inspired by the J-Orchestra project [8] and the Addistant translator [7] that present solutions to transparently *partition* Java applications to achieve distribution, we propose to group the classes of a Java application into *partitions* and use partitions as update units in the DUCS framework.

The partitioning is performed on Java byte-code by a conversion tool that traces classes and replaces cross partition instantiations, invocations and casts with their equivalents in the DUCS framework. More information on partitioning can be found in [2].

2.3 Evaluation

We evaluate our approach by applying it to a 3rd party, Java, P2P messaging application. We present how the process of progressing from an old to a new version of the application is supported by our solution. Updating the application illustrates how various kinds of updates and the process of propagating the updates in a distributed application are performed. In the evaluation, we measure performance of the application before, during, and after the update process.

3 Conclusion

We have briefly presented our research addressing the support for DU of existing applications. Our solution combines functionalities supporting DU with a partitioning approach that allows existing applications to become dynamically updateable. The solution is evaluated on an existing Java application, a P2P messaging system, that was not prepared for dynamic updates. We demonstrate viability of our ideas by updating the application while running and by propagating the updates in the network.

References

1. R. Bialek and E. Jul. A framework for evolutionary, dynamically updatable, component-based systems. In *The 24th IEEE International Conference on Distributed Computing Systems Workshops*, pages 326–331, Hachioji, Tokyo, Japan, March 23-24 2004.

2. R. Bialek, E. Jul, J.-G. Schneider, and Y. Jin. Partitioning of java applications to support dynamic updates. In *The 11th Asia-Pacific Software Engineering Conference*, Busan, South Korea, November 29-December 3 2004.
3. P. Danielsson and T. Hulten. Jdrums java distributed run-time updateing management system. Master's thesis, MASDA - Department of Mathematics, Statistics and Computer Science, Vaxji University, Sweden, 2001.
4. M. Dmitriev. *Safe Class and Data Evolution in Large and Long-Lived Java Applications*. PhD thesis, University of Glasgow, May 2001.
5. F. Plasil, D. Balek, and R. Janecek. Sofa/dcup: Architecture for component trading and dynamic updating. *Proceedings of ICCDS'98, Annapolis, Maryland, USA, IEEE CS Press, May 1998*, 1998.
6. R. S. M. Rui, G. S. Blair, and E. Carrapatoso. Supporting adaptable distributed systems with formaware. In *The 24th IEEE International Conference on Distributed Computing Systems Workshops*, pages 320–325, Hachioji, Tokyo, Japan, March 23-24 2004.
7. M. Tatsubori, T. Sasaki, S. Chiba, and K. Itano. A bytecode translator for distributed execution of “legacy” Java software. *Lecture Notes in Computer Science*, 2072:236–, 2001.
8. E. Tilevich and Y. Smaragdakis. J-orchestra: Automatic java application partitioning. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 178–204. Springer-Verlag, LNCS 2374, 2002.