ULS System Integration via Model Composition

Krishnakumar Balasubramanian*, Douglas C. Schmidt
Vanderbilt University, Nashville

September 7, 2006

1 Some Integration Challenges of ULS Systems

Component middleware technologies are generally a more effective technology base to build ULS systems than the brittle proprietary infrastructure used in legacy systems, which have historically been built in a vertical, stove-piped fashion. With the emergence of commercial-off-the-shelf (COTS) component middleware technologies, such as Enterprise Java Beans (EJB), CORBA Component Model (CCM) and Microsoft's .NET Framework and Web Services, system integrators will increasingly need to integrate *heterogeneous* complex ULS systems built using different COTS technologies, which increase with each new generation of technologies. These technologies differ in many ways, including the protocol level, the data format level, the implementation language level, and/or the deployment environment level.

Although there are well-documented patterns [4] and techniques [10] for system integration using various integration middleware technologies, system integration is still largely a manual process. Such a tedious and error-prone process will will completely break down in ULS system integration due to the scale of the systems being integrated and the differences in the teams, goals, and technology bases when these systems were designed. Despite the benefits of component middleware, therefore, some key unresolved challenges of integrating ULS systems developed using heterogeneous COTS middleware remain, including:

• **Complexity of declarative metadata**. Component middleware technologies use declarative notations (such as XML descriptors, source-code attributes, and annotations) to capture metadata that describes configuration options on interfaces, interconnections between these interfaces, and implementation entities. Examples metadata include EJB deployment descriptors and .NET assembly manifests. System integrators must track and configure these metadata correctly during integration and deployment since the correct functionality of the integrated system depends on the correct configuration of the metadata.

While manually configuring metadata is already a problem in today's large-scale systems [3], it will become completely infeasible in ULS system integration due to the number of components, the differences in the metadata of different technologies, and even the differences between versions of metadata of the same technology due to ULS system longevity. New approaches to ***metadata management*** are therefore needed to achieve integration of ULS systems.

• **Incompatible differences between middleware technologies**. Component middleware technologies generally support applications developed in multiple languages and/or on multiple operating systems, which in-turn run on multiple hardware platforms. This diversity in technology implementation and deployment poses significant challenges during system integration, however, since integrators must reconcile different ways to achieve the same goal in each technology base.

For example, simple activities, such as determining the functionality exposed by a system, become hard due to different ways of describing system interfaces, such as CORBA Interface Definition Language (CCM), Java interfaces (EJB), and Web Services Definition Language (Web Services). Similarly, the on-the-wire protocol format for

*Contact Author

CCM and EJB is Internet Inter-ORB Protocol (IIOP), which is a binary protocol, whereas Web Services uses SOAP/HTTP, which is a text-based protocol. The sheer number of incompatibilities between middleware technologies therefore necessitate automated ***resource adaptation*** techniques in the ULS system integration space.

• **Incompatibilities between different middleware implementations**. The success of COTS middleware technologies have started a trend where multiple implementations of a single middleware technology are available from different providers. Differences between these implementations will likely arise due to non-conformant extensions to standards, different interpretations of the same (often vague) specification, or implementation bugs.

Regardless of the reasons for incompatibility, however, problems arise that often manifest themselves during system integration. Examples of such differences are highlighted by the presence of efforts like the Web Services-Interoperability Basic Profile (WS-I) [2], which is a standard aimed at ensuring compatibility between the Web Services implementations of different providers. It is therefore critical to ***externalize and document*** the implicit design assumptions of implementations of various component middleware technologies, to ensure smooth integration and dynamic replacement of one implementation with another.

1.1 Solution Approach→System Integration using Model Composition

A promising approach to address key challenges with integration of component middleware is to develop Model-Driven Engineering (MDE) technologies [8]. At the core of MDE is the concept of *domain-specific modeling languages* (DSMLs) [6], whose type systems formalize the application structure, behavior, and requirements within particular domains. A DSML is often accompanied by a generators, which analyze the models and synthesize various types of artifacts, such as source code, deployment descriptors, or input to simulators.

By capturing the semantics of a domain, DSMLs can not only be used as effective "metadata management" frameworks but also be used to generate "resource adapters". DSMLs are an effective means to capture implicit assumptions associated with component middleware technologies. Representing elements of a component middleware technology as first-class entities of a DSML makes hidden implicit assumptions explicit at the modeling level. DSMLs can thus be used to highlight–and ultimately help resolve–the complexities associated with incompatible implementations to earlier in a system's lifecycle.

While DSMLs have been used to help software developers create homogeneous systems [5, 9], large-scale enterprise software systems are rarely homogeneous. A single DSML developed for a particular middleware technology, such as EJB or CCM, may therefore not be applicable to model, analyze, and synthesize key concepts of Web Services. To integrate heterogeneous systems successfully, therefore, system integrators need tools that can provide them (1) unified view of the entire enterprise system and (2) fine-grained control over specific subsystems and components.

A promising approach to achieve these capabilities is *model composition* [1], which involves creating a new DSML from multiple existing DSMLs by adding new elements or extending elements of existing DSMLs, defining new relationships between existing elements, and defining relationships between new and existing elements. A key characteristic of the model composition is that it supports the *open-closed* principle [7], which states that a class should be open for extension but closed with respect to its public interface, but at a higher level of abstraction, *i.e.*, at the level of DSMLs. Other

benefits of model composition include its ability to leverage prior investments in existing tool-chains, including domain constraints, generators of the existing DSMLs, while simultaneously adding new capabilities.

We posit that a combination of DSMLs and model composition technologies can help to address the challenges associated with integrating component middleware technologies in ULS system environments, without incurring the drawbacks of conventional approaches. This talk will describe the challenges associated with integration of ULS systems, outline our approach to addressing these challenges in today's large-scale systems context, identify the drawbacks with direct application of these techniques to integrating ULS systems, and suggest possible approaches to rectifying these problems to scale integration capabilities more smoothly to ULS systems.

Krishnakumar Balasubramanian is a graduate student in the Department of Electrical Engineering and Computer Science at Vanderbilt University. His research interests include model-driven development, deployment and configuration of component middleware, and patterns and frameworks for distributed, real-time, and embedded systems development. He received an MS in computer science from Washington University in St. Louis.

Douglas C. Schmidt is a Professor of Computer Science at Vanderbilt University. His research focuses on patterns, optimization techniques, and empirical analyses of software frameworks and domain-specific modeling environments that facilitate the development of distributed real-time and embedded middleware and applications running over high-speed networks and embedded system interconnects. In addition to his academic research, Dr. Schmidt has over fifteen years of experience leading the development of ACE, TAO, CIAO, and CoSMIC, which are widely used, open-source DRE middleware frameworks and model-driven tools that implement patterns and product-line architectures for high-performance DRE systems.

References

[1] Ákos Lédeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti. On Metamodel Composition. In *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA)*, pages 756–760, Mexico City, Mexico, 2001. IEEE.

[2] K. Ballinger, D. Ehnebuske, C. Ferris, M. Gudgin, C. K. Liu, M. Nottingham, and P. Yendluri. WS-I Basic Profile. www.ws-i.org/Profiles/BasicProfile-1.1.html, April 2006.

[3] L. DeMichiel and M. Keith. Enterprise Java Beans 3.0 Specification: Simplified API. jcp.org/aboutJava/communityprocess/final/jsr220/index.html, May 2006.

[4] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, October 2003.

[5] G. Karsai, S. Neema, B. Abbott, and D. Sharp. A Modeling Language and Its Supporting Tools for Avionics Systems. In *Proceedings of 21st Digital Avionics Systems Conf.*, Aug. 2002.

[6] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, Jan. 2003.

[7] B. Meyer. Applying Design By Contract. *Computer (IEEE)*, 25(10):40–51, Oct. 1992.

[8] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2), 2006.

[9] J. A. Stankovic, H. Wang, M. Humphrey, R. Zhu, R. Poornalingam, and C. Lu. VEST: Virginia Embedded Systems Toolkit. In *Proceedings of the IEEE Real-time Embedded Systems Workshop*, London, UK, Dec. 2001. IEEE.

[10] D. TrowBridge, U. Roxburgh, G. Hohpe, D. Manolescu, and E. G. Nadhan. Integration Patterns. msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/intpatt.asp, June 2004.