Rapid Composition of Web Services with JOpera for Eclipse

Cesare Pautasso
Department of Computer Science
Swiss Federal Institute of Technology, Zurich (ETHZ)

Submission for the **Research-Industry Exchange** at EclipseCon 2005

4.Feb.2005

Contact Information

pautasso@inf.ethz.ch
http://www.inf.ethz.ch/~pautasso

Dr. Cesare Pautasso
Institute for Pervasive Computing
ETH Zentrum (HRS G7)
CH-8092 Zurich
Switzerland

+41 01 632 0879 (telephone)
+41 01 632 1425 (fax)

Rapid Composition of Web Services with JOpera for Eclipse

Cesare Pautasso, pautasso@inf.ethz.ch

Abstract

Thanks to the interoperability offered by Web services standards it is now possible to build large scale distributed software systems by composing reusable services published on the Web. In this presentation we will demonstrate how we combined Eclipse's user experience (background model checking and incremental recompilation) with a simple visual composition language to develop the JOpera for Eclipse plugins [1]. With them, it is not required to work with XML-based languages when composing many kinds of different services because their interactions can be specified by literally drawing them in a data and control flow graph. The visual representation is then directly compiled to Java code in order to be executed efficiently. A visual monitor and debugger are also included, so that it is possible to interactively watch the progress of the composition as it runs at the same level of abstraction and with the same visual syntax used to define it.

Web services offer a standards-based approach to address many interoperability issues arising when composing distributed software systems out of reusable services. Thanks to the SOAP protocol and WSDL interface description language, an increasingly large number of basic services are being published on the Internet. Clearly, it becomes important to find the right composition abstractions in order to build value added services out of the aggregation of basic ones. Complementing existing approaches based on the XML syntax (e.g., BPML, WSBPEL) we have designed a visual syntax for a service composition language [2]. Thus, the data exchanges between the services (data flow), their order of invocation and the necessary failure handling behavior (control flow) can be all specified with a simple, graph-based, visual language. As opposed to an XML based approach, a visual language can greatly enhance the understandability of the composite system and provide the foundation for building rapid service composition tools. By reusing the Graphical Editing Framework, our effort to build a visual composition environment was greatly reduced. Furthermore, thanks to the flexibility offered by draw2d, we did not need to significantly alter the visual syntax of the composition language.

Among sequential and parallel flows, conditional branches, synchronization points, nesting, recursion, iteration, dynamic late binding, and reflection constructs, a very important feature of the JOpera Visual Composition Language consists of describing how to compose services at the level of abstraction represented by their interfaces. Thus, the description on how to compose the services is kept independent from the actual protocols that are used to perform the actual service invocation. Abstracting the interface of a service from its access mechanism helps to generalize the scope of JOpera for Eclipse beyond the composition of coarse-grained Web services [3]. This way, the developer can freely choose to compose the most appropriate *kind* of services in terms of runtime performance and convenience of use. In other words, if it becomes necessary to integrate local Java modules into the composition, it should be possible to do so without having to publish such code as a Web service (at design-time) and having to invoke it through the SOAP/XML/HTTP stack at runtime. Moreover, with our approach the composition language is not affected when extending the underlying system to support other kinds of services, apparently quite different from Web services. Currently JOpera supports fine-grained Java snippets, "legacy" UNIX (or Windows) shell commands, XPATH/XSL transformations, calls to local Java methods, execution of remote scripts with SSH, or even SQL queries directly sent to a JDBC compliant database.

Thanks to this flexible approach, it becomes possible to unify composition and mediation aspects within the same language. For example, in a typical scenario, Web services published by independent organizations are to be composed in a bottom-up fashion. In general, a mismatch between

these services it is to be expected. With JOpera, in order to solve the problems due to different data representations or incompatible interaction styles, a service fulfilling the role of mediator can be added in a straightforward manner. Furthermore, such mediator can be implemented with the most suitable technology, e.g., an XSLT transformation or a Java snippet.

Unlike most existing service composition languages, the JOpera Visual Composition Language is compiled to Java code to ensure its efficient execution. This code is then once more compiled (into bytecode) by Eclipse's JDT compiler. The result is then automatically deployed for execution by dynamically loading it into JOpera's run-time execution kernel. This is the component responsible for providing a scalable and persistent execution environment for the compiled compositions. It can be deployed in different configurations: as a stand-alone application as well as embedded into other systems (e.g., application servers, or – as we will show as part of this demonstration – plugged into development tools like Eclipse). The kernel can also be replicated across a cluster of computers to handle the concurrent execution of a large number of composite services [4]. As the compilation and deployment steps are completely transparent and do not have to be triggered explicitly by the user, it may a bit difficult to show them happening during the demonstration. However, we will provide sufficient explanation by pointing out the intermediate Java code which under normal circumstances is kept hidden from the developer.

Another important feature aiming towards increasing the efficiency of the developers composing services consists of checking and validating the model of a composition under construction. More specifically, after each modification, the model of the composite service is incrementally checked for consistency with respect to various criteria (e.g., data flow connections linking mismatching services). In case a violation is detected, a specific problem marker is attached to the part of the model that triggered it. During the demonstration we will show that such verification happens in the background, while the developer is composing the services, so that errors and potential problems are reported immediately. As shown also by Eclipse's Java development environment, such immediate feedback is very important to reduce the overhead of the compose-compile-fix development cycle typical of existing service composition tools.

All in all, this presentation will introduce JOpera for Eclipse, describe its architecture and demonstrate concrete examples of several important research contributions. First of all we will show that Web services can be effectively and rapidly composed using a visual language, which leverages XML-based languages (e.g., SOAP, WSDL, WSBPEL) but keeps their machine-oriented syntax hidden at all times from the developer. Second, in order to support the entire lifecycle of a service composition, in addition to advanced composition tools (which support auto-completion and background model checking), we will also present visual tools for the interactive monitoring and debugging of the execution of a composition which seamlessly integrate with the environment used to construct it. Finally, in order to support our claim that the JOpera Visual Composition Language is orthogonal with respect to the types of services to be composed, we will demonstrate the openness of the system by showing how to extend it with a plugin to provide support for a new kind of service without having to modifying the language used to compose it.

We believe that JOpera contributes to the Eclipse community an open and flexible tool for visually composing many different kinds of services and we are very interested in exploring the synergy between our tool for programming compositions and the many existing tools for programming the components, which – thanks to the Eclipse platform – have now the potential to be seamlessly integrated.

References

[1] C. Pautasso. JOpera: Process Support for more than Web services. http://www.iks.ethz.ch/jopera

[2] C. Pautasso and G. Alonso. The JOpera Visual Composition Language, *Journal of Visual Languages and Computing*, in press, Nov. 2004.

[3] C. Pautasso and G. Alonso. From Web Service Composition to Megaprogramming. In *Proceedings of the 5th VLDB Workshop on Technologies for E-Services (TES-04)*, Toronto, Canada, August 2004.

[4] C. Pautasso and G. Alonso. JOpera: a Toolkit for Efficient Visual Composition of Web Services. *International Journal of Electronic Commerce*, 9(2):104–141, Winter 2004/2005.