Building, Deploying, and Monitoring Distributed Applications with Eclipse and R-OSGi*

Jan S. Rellermeyer Gustavo Alonso Timothy Roscoe
Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland
{rellermeyer, alonso, troscoe}@inf.ethz.ch

ABSTRACT

Designing and testing distributed applications is still a difficult task that requires in-depth knowledge about networking issues. Eclipse is, among other things, a powerful and widely used IDE for the development of complex applications, in particular modular applications for the OSGi framework. Our R-OSGi middleware supports the seamless distribution of OSGi applications along the boundaries of services. By combining R-OSGi with Eclipse into the R-OSGi Deployment Tool, we give developers a tool that automatically handles distribution in a transparent way and integrates the capabilities of R-OSGi into the Eclipse workflow. With this tool, building, deploying, and monitoring distributed applications is as easy as writing OSGi applications in Eclipse and using a graphical editor to visually create distributed deployments of the modules. The tool can also be used to great effect by researchers to test and benchmark distributed applications and for education purposes.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Modules, Packages*; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous

General Terms

Design, Management

Keywords

R-OSGi, OSGi, Deployment, Eclipse, Concierge

1. INTRODUCTION

The Eclipse platform for building, deploying, and managing software is increasingly based on the OSGi standard [8] for module management in Java: the platform itself is composed of a variety of OSGi modules, and is extensible through plugins which are themselves OSGi bundles. Unsurprisingly, Eclipse also has built-in support for developing modular Java applications based on the OSGi standard. However, Eclipse does not currently provide support for remotely deploying an OSGi-based application to a node, or for building distributed applications using OSGi.

In [10] we presented R-OSGi, an extension to OSGi that allows the user to run modular software in a distributed fashion without changing the OSGi framework implementation or the application binaries: remote invocation proxies are transparently interposed at distribution boundaries between modules, and network or remote node failures are mapped to module unload events. The result is that modular OSGi-based applications can be written using OSGi bundles with little extra effort on the part of programmers.

In this paper, we present an Eclipse plugin that allows distributed R-OSGi applications to be deployed and managed from within Eclipse. When combined with R-OSGi itself, the R-OSGi Deployment Tool (RDT) turns Eclipse into a powerful tool for developing, deploying, and monitoring distributed applications. The result is that developers can build modular software with little concern for distribution, and then use RDT to transform the resulting application into a distributed system. The contributions of RDT described in this paper are as follows:

Firstly, RDT can analyze OSGi applications inside Eclipse and generate graphical representation based on Eclipse's Graphical Editing Framework (GEF) [5]. Users of the tool can turn this initial centralized deployment into a distributed deployment by dragging and dropping bundles from one machine to another. The resulting deployment description is transformed into tasks for R-OSGi.

Secondly, RDT exposes R-OSGi's transparent support for load-balancing and fault-tolerance. Such facilities can be added to existing applications with a few clicks, and the resulting system immediately deployed from Eclipse.

Thirdly, RDT can be used to visualize the structure and status of a running distributed application in real time, including such issues as node or network failures and concurrency issues. As well as its use in a software management context, this facility can be used to explore the impacts of different failure models and decide on appropriate fault tolerance strategies.

Finally, RDT can capture all network messages and make them available to the Eclipse user. These traces can be used by developers to for profiling, testing, debugging, and benchmarking distributed applications throughout the entire development cycle.

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322.

Eclipse Technology Exchange (ETX) Workshop '07, October 21–25, 2007, Montréal, Québec, Canada.
ACM .
ISBN: 978-1-60558-015-9

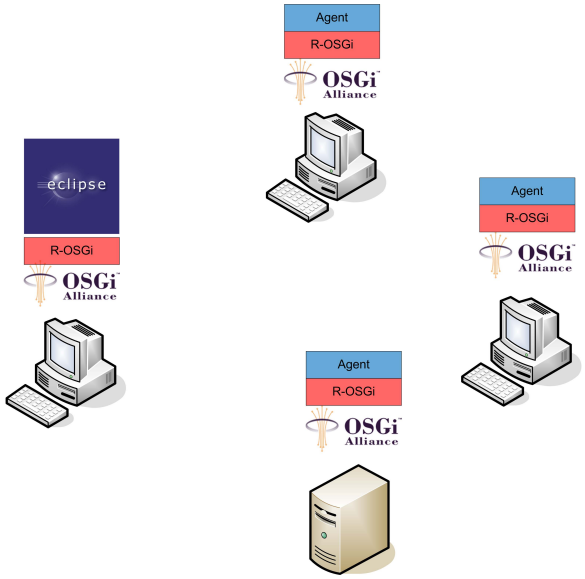

Figure 1: Example Setup

2. THE DEPLOYMENT TOOL

The R-OSGi Deployment Tool for Eclipse consists of two different parts: The actual PLUGIN which extends the Eclipse IDE and the DEPLOYMENT AGENT in charge of distributed deployment. Figure 1 shows an example consisting of the development machine running the Eclipse instance and three nodes running each an OSGi framework with R-OSGi and the Agent. The graphical representation of the deployment is transformed into sets of declarative tasks for each nodes.

The purpose of the PLUGIN is to add support to Eclipse for creating deployment graphs from an existing OSGi application, allowing the user to modify the graph, and to visualize the resulting application once it is deployed. The DEPLOYMENT AGENT is the primary communication channel between the Eclipse instance and the OSGi frameworks running on the nodes. It mainly features remote management commands and processes the tasks disseminated by the PLUGIN during the deploying step. Once the application is running, is also gives feedback about the application state and the network traffic.

2.1 Code Analysis and Graph Generation

The manifest of OSGi bundles explicitly declares package imports and exports but does not give any information about the services provided and consumed by a bundle. Therefore, the service dependency structure can be normally determined only from the running bundles of an application. In the R-OSGi Deployment Tool, we use code analysis to reason about the services used by bundles. Since the source code of the application bundles is usually available when developed in Eclipse, our tool uses the *Abstract Syntax Tree* (AST) maintained by Eclipse for every Java source code resource.

As an initial step, the user specifies the main (root) bundle of the application through a wizard. Starting from this bundle, an AST node visitor traverses the syntax trees of all source files belonging to the bundle project to identify which services are registered (through *context.registerService(...)*) and consumed (*context.getServiceReference(...)* and *context.getService(...)*). For binary resources, the same could be achieved through bytecode analysis (e.g., using the ASM [1] library, also internally used by R-OSGi). The code analysis recurses over all bundle dependencies of the root bundle.

The result of the analysis is the complete set of bundles making up the application, together with information about the registered and consumed services for each bundle. In the next step, the tool creates an initial deployment graph depicting the structure of the application if it was launched on a single OSGi framework.

2.2 Creating a Distributed Deployment

This initial graph can be manipulated by the user to turn the centralized application into a distributed one. New nodes running OSGi frameworks can be added to the graph. The bundles of the application can be dragged and dropped from one node to another, thereby changing the structure of the deployment. Dependencies between the bundles on the level of services are illustrated by connection arrows. The amount of connections between frameworks gives the developer already a coarse estimation on how much network activity a specific distribution setup might involve. Figure 2 shows the Deployment Editor with an opened deployment of an application on three different nodes.

For each service which is accessed by remote frameworks, the R-OSGi middleware transparently creates a service proxy on the consumer peers which calls the original remote service. Dependencies on the level of package imports and exports are automatically resolved by R-OSGi. Events raised through the OSGi EventAdmin service are transparently forwarded to those nodes that have a corresponding EventHandler registered.

2.3 Advanced Features

The Deployment Tool does not only facilitate the creation of distributed applications out of software modules, it also offers to add advanced features such replicating modules for load balancing, or failover redundancy. The user can create redundant copies of bundles through the visual interface by dragging a bundle to a different node with the control key pressed. By default, these copies are unbound. With different GEF connection tools, the user can (graphically) bind the services of this copy to consuming bundles and thereby specify for which purpose the copy should be used. With our tool, the implications of distributed failures can be tested and their impact on the running application evaluated. For education purposes, different failure scenarios can be simulated by disabling nodes of the deployment, or by creating new channel implementations which simulate network failures.

R-OSGi supports binding service proxies to a single primary service and adding redundant remote services. If the failover policy is defined for these additional bindings, R-OSGi will switch to the next service whenever the invocation of the primary service fails due to unavailability or network failures. Two different load balancing strategies are selectable. The `ONE` strategy selects the best available service on the first invocation (the one with the least service proxies bound to it). The `ANY` strategy is intended to be used with stateless, idempotent services. With this strategy, the best available service is reselected with every new invocation.

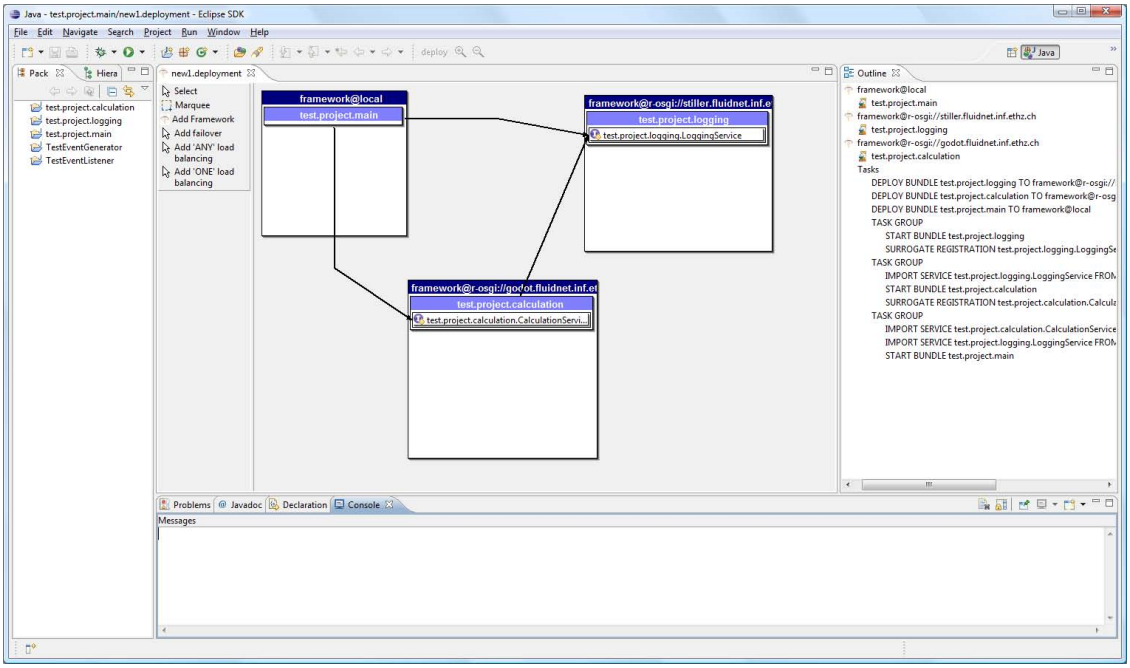

Figure 2: Deployment Editor

3. INTEGRATION INTO ECLIPSE

3.1 Task Structure

In a traditional OSGi application, the start order of bundles does not depend on package imports and exports but is affected by the services that bundles exchange. If bundle B consumes a service provided by bundle A, A has to be started before B in the general case so that B will find the service when it starts. Some bundles are designed to handle missing services (e.g., leaving out certain features or delaying the startup until the service is present) but this behavior is not mandatory and depends on the degree of dynamics that the author of the bundle has taken into account.

In a distributed setting, these constraints cross the boundaries of a single VM. The naive approach to start the local bundles, register every local service for remote access and building a proxy for every remote service on each peer (thereby making the deployment a fully connected graph) does not work. If a bundle requires a service provided by a remote machine, the remote service bundle has to be started prior to the start of the local bundle. In general, it has to be assured that not only the total order of bundle installation is preserved among the peers but also that the remote access of services is established in the right order.

The R-OSGi Deployment Tool addresses this challenge by maintaining a global task queue which is processed in sequential order. This mirrors the behavior of a local framework where the starting of bundles is also sequential. The total order of bundle installation is not relevant; therefore it is performed in parallel on all frameworks and prior to the application startup phase. For each bundle, the service dependencies are determined and a *BUNDLE_START_TASK* is enqueued in such a way that the dependency structure is properly reflected. If a bundle has any dependencies to services of a remote bundle, the system inserts a *TASK_GROUP* instead that consists of three phases:

1. Importing all remote services that are consumed by the bundle
2. Starting the bundle
3. Exporting all services that are consumed by remote bundles

Through this structure, the resulting startup regime is guaranteed to be identical to that of a centralized OSGi framework. Since the installation is done in parallel, only little communication overhead is incurred for coordinating the startup order. To minimize the overhead of dependency analysis, the Deployment Editor keeps an up-to-date task queue which is initialized with the tasks required for a centralized application (only tasks to install and start each bundle) and modifies the task queue in response to changes on the structure of the deployment.

3.2 Deploying the Bundles and Starting the Application

When the user triggers the deployment of a application, the bundles are generated from the referenced Eclipse projects. The current prototype implementation of the tool interacts with the Concierge [9] Eclipse Plugin to create highly portable OSGi R3 bundles which also run on very resource-constrained devices.

Each framework involved in a deployment is expected to run a Deployment Agent bundle offering a Deployment Service. The services from all frameworks are transparently imported through R-OSGi to the Equinox OSGi framework on which Eclipse and the Deployment Tool run on. As a result, the tool can use the services of the remote frameworks (through R-OSGi service proxies) as if they were local services making the deployment and all further interaction easier. The Deployment Service offers methods to control the

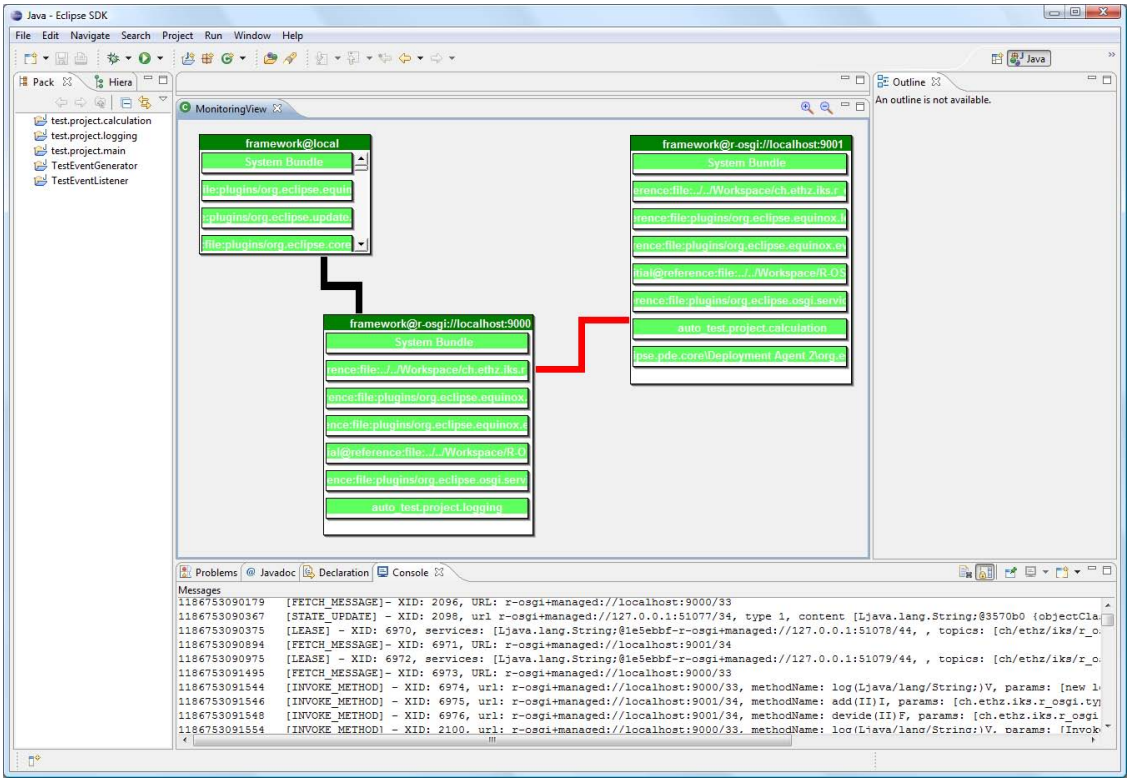

Figure 3: Monitoring View

framework remotely, e.g., to install and start bundles or to establish a connection to another remote framework in order to import a service required for the application. Since the bundles involved in the deployment remain unmodified, the registration of the services for remote access cannot happen by the bundles themselves (since they were not designed to support distribution through R-OSGi). However, R-OSGi allows *surrogate registration* by a third bundle. In the case of the R-OSGi Deployment Tool, the surrogate registrations are performed by the Distribution Services.

3.3 Monitoring the Running Application

As soon as the application is running in the network, the Deployment Tool opens the Monitoring View for this specific application (Figure 3). The view shows the actual state of the OSGi framework on each involved node, i.e., the bundles and services. Once again, the Agents on the frameworks are contacted to get this information. Changes in the state of bundles and services are propagated to the Plugin via EventAdmin events which are transparently forwarded by the R-OSGi middleware.

Furthermore, the system determines which pairs of frameworks communicate through channels. A channel is established if there are remote accesses to services from one framework to the other. R-OSGi offers an extensible model for channels through NetworkChannelFactories. The tool takes advantage of this and registers its own channel type for the protocol "r-osgi+managed". These channels provide feedback about activity and exchanged messages. This information is as well propagated to the Plugin through events and visualized in the Monitoring View. The developer can not only supervise the network activities of the deployment through visual feedback but also inspect the messages in the style of tools like Ethereal/Wireshark[12].

4. RELATED WORK

Several approaches have proposed the partitioning of existing applications in order to create distributed applications. The Coign [6] project uses instrumentation of COM components through binary rewriting, analyzes the dependencies between the components and calculates a graph-cutting based on a cost model for introducing network communication between the subgraphs. Similarly, JOrchestra [11] partitions a Java program by rewriting bytecode to replace local methods with remote invocations, and object references with proxy references. However, JOrchestra requires in-depth knowledge of the source program in order to do the fine-granular distribution of the mobile classes through a distribution plan. The high number of classes in modern Java applications can make this procedure time-consuming and error-prone. In the Eclipse-R-OSGi Deployment Tool, we exploit existing modularization and use the visualization capabilities of Eclipse to allow the developer to create deployments in a more intuitive way. Furthermore, partitioning of existing applications is an iterative process that has to be revised from time to time in order to adapt to the changing environment or to optimize for performance. Hence, we support monitoring of the deployments and advanced features like load-balancing.

Visualization of distributed systems for testing and education purposes has been proposed by projects like Parade [7], PVaniM [2], or ConcurrentMentor [3]. However, these projects describe standalone tools which operate on

existing deployments and do not help to actually develop and deploy distributed application. Since the R-OSGi Deployment Tool is based on Eclipse, visualization can be easily applied to any of the intermediate steps during the design phase. The OverView [4] project has proposed a comparable extension to visualize distributed applications developed in Eclipse. However, they achieve the high level abstractions through explicit modeling in their own Entity Specification Language. In contrast, the R-OSGi Deployment Tool uses existing abstractions of the application based on OSGi bundles and services and does not require any further modeling efforts. Thereby, it provides a more rapid and intuitive way of designing and visualizing distributed applications.

5. CURRENT LIMITATIONS AND FUTURE WORK

The main limitation of the current implementation is the simplifying assumption that all services provided by a bundle will be registered within the startup of the bundle (and not in response to an interaction with the bundle) and unconditionally. Although reasonable for a large amount of bundles, this assumption does not hold for every bundle since the OSGi specifications allow the bundle to register services at any time.

Since the plugin already does source code analysis, it would be possible to do symbolic processing of the code and reason about the environmental settings that cause the bundle to register a specific required service. This information could then be used to create exactly the environment for a bundle that causes the service to be registered.

Furthermore, it is currently assumed that all bundles are available as Eclipse projects. Binary dependencies are not handled in the current prototype. As discussed before, the same algorithm implemented on the AST of the source code could as well be implemented as bytecode analysis, thereby allowing binary-only bundle resources to participate in the deployment.

In the current state of the system, the visualization operates purely online. If combined with a data lineage facility, the feedback generated by the monitored deployment could be used to replay certain situations and thereby allowing in-depth post-mortem analysis. In addition, the main focus of the prototype implementation is to support the development, testing, and monitoring inside Eclipse. For productive systems, a headless deployment facility would be required. Nothing in the described setup prevents to build such a facility and using the saved state of a deployment graph to deploy applications without user interaction.

As part of future work, we will turn the static deployment tool currently created into a dynamic deployment platform that can autonomously relocate bundles to optimize the performance of the application, depending on cost metrics specified by the developer.

6. CONCLUSIONS

With the R-OSGi Deployment Tool for Eclipse, we present a solution for effectively turning modular into distributed applications. Since this process is inherently difficult for the developer, we needed an environment in which programmers feel familiar and which has support for visualizing the different steps of creating a deployment. With the Eclipse IDE, we found this platform and could use several of its feature to implement our system.

On the other hand, our tool adds the capability to develop, deploy and monitor distributed systems directly in Eclipse and as part of one coherent design process, in which the developer can focus on the application rather than the distribution aspects. It embeds into the typical look-and-feel and the workflows of Eclipse and allows to observe the results of the program even though it is running in a network. The ease of use and the visualization capabilities of the R-OSGi Deployment Tool facilitates simulation, testing, and benchmarking of distributed systems throughout the whole design process. The tool is an ideal choice for exploring the behavior of distributed system in research and education.

7. REFERENCES

[1] E. Bruneton, R. Lenglet, and T. Coupaye. ASM: A Code Manipulation Tool to Implement Adaptable Systems. Technical report, France Telecom R&D, November 2002.

[2] C. D. Carothers, B. Topol, R. M. Fujimoto, J. T. Stasko, and V. Sunderam. Visualizing Parallel Simulations in Network Computing Environments: A Case Study. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 110–117, 1997.

[3] S. Carr, C. Fang, T. Jozwowski, J. Mayo, and C.-K. Shene. ConcurrentMentor: A Visualization System for Distributed Programming Education. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2003.

[4] T. Desell, H. N. Iyer, C. Varela, and A. Stephens. OverView: A Framework for Generic Online Visualization of Distributed Systems. In *Proceedings of the Second Eclipse Technology Exchange: ETX and the Eclipse Phenomenon (ETX 2004)*, pages 87–101, 2004.

[5] Eclipse Graphical Editing Framework. `http://www.eclipse.org/gef`.

[6] G. C. Hunt and M. L. Scott. The Coign Automatic Distributed Partitioning System. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI 1999)*, 1999.

[7] T. L. Naps and E. E. Chan. Using Visualization to Teach Parallel Algorithms. *SIGCSE Bulletin*, 31(1):232–236, 1999.

[8] Open Service Gateway Initiative. `http://www.osgi.org`.

[9] J. S. Rellermeyer and G. Alonso. Concierge: A Service Platform for Resource-Constrained Devices. In *Proceedings of the 2007 ACM EuroSys Conference*, 2007.

[10] J. S. Rellermeyer, G. Alonso, and T. Roscoe. R-OSGi: Distributed Applications Through Software Modularization. In *Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference*, 2007.

[11] E. Tilevich and Y. Smaragdakis. J-Orchestra: Automatic Java Application Partitioning. In *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP 2002)*, pages 178–204, 2002.

[12] The Wireshark Project. `http://www.wireshark.org`.