Application Management Plug-ins through Dynamically Pluggable Probes

Kiev Gama, Gabriel Pedraza Ferreira, Thomas Lévêque, Didier Donsez

Application Management Plug-ins through Dynamically Pluggable Probes

Kiev Gama[+], Gabriel Pedraza[+], Thomas Lévêque[*] and Didier Donsez[+]

[+]LIG laboratory, University of Grenoble
Bat. C, 220 rue de la Chimie, Domaine Universitaire
Grenoble, France
{kiev.gama, pedraza, didier.donsez}@imag.fr

[*]Mälardalen University, Department of Computer Science and Electronics, P.O. Box 883, SE-721 23
Västerås, Sweden
thomas.leveque@mdh.se

ABSTRACT

It is widely recognized that applications need to be remotely administrated. In general, application management and monitoring is supported by textual management consoles while graphical user interfaces specialized for their tasks are preferred from average users. Defining what must be monitored and what are the admin actions you want to perform on an application cannot be defined during the application development due to the fact that these needs evolve after the application deployment as we cannot completely predict the execution environment such as available devices. This paper presents an architecture and the corresponding infrastructure that allow administrators to define what they want to monitor and manage and automate the discovery and deployment of corresponding probes and related management console graphical plug-ins. This work has been validated on two different application domains.

1. INTRODUCTION

Nowadays, applications interact heavily with external devices and external services. The unpredictable nature of their surrounding environment such as device availability/unavailability force them to be adapted dynamically. It is widely recognized that applications need to be remotely administrated. With the increasing complexity of systems, good application management tools becomes important to be present in parallel to application execution in order to facilitate application monitoring and to perform reconfigurations at runtime. Administrative tasks including application management and monitoring are usually performed thanks to textual management consoles. Graphical consoles are more convenient to show visual system representations for average users. While generic representations of manageable data are used to build generic console [10][9][5], specialized consoles with dedicated user interface for each manageable data type are more efficient.

Typical management tools provide a predefined set of functionality that does not change over time. However, the dynamic nature of today applications requires on demand probe deployment and their console counterpart: a specialized user interface.

Current tools suffer of lack of high level monitoring and management leading to not efficient administration. In particular, the administrator needs to know which probe provides expected data and the different probe commands to collect data while he is only interested on collected data.

The needed probes cannot be defined at application design and implementation time as new services and components are deployed during the execution. In general, new probes cannot be added unless the system code is patched, the new functionalities added and the application redeployed, usually at the price of an application restart. Administrators need to be able to install probes dynamically according to the resources they want to manage and monitor.

We propose an approach that lets the administrator choose the manageable elements he is interested in and automates discovery and deployment of corresponding probes and console plugins. The proposed approach rely on a plugin infrastructure for both sides: the application and the management console. This approach is validated in two management consoles developed as plug-ins on top of the Oracle Java VisualVM. The first plug-in we present is a general purpose management console for OSGi frameworks, and is part of the open source OW2 chameleon project. The other plug-in targets the open source RFID middleware from the Aspire RFID European project reused by other aspects that provide specific crosscutting concerns, improving modularity for better abstractions and reuse.

Section 2 will list our motivations. Then section 3 will present our approach by describing the plugable probe infrastructure, our modular approach to build specialized consoles and . Then section 4 descibes the two case studies used to validate our approach and the results of our experiments. Section 5 will describe some related works. Finally, section 6 concludes and presents our future works.

2. MOTIVATION

Management consoles are important for observing applications, collecting data and applying metrics to them, as well as allowing to communicate with the target application and to perform reconfigurations if necessary. Dynamic component platforms provide infrastructure for installing and uninstalling components during application execution. The inherent complexity introduced by such dynamism makes even more important to provide management consoles for these types of application. When the application functionality is known at design time it is easier to

Conference'10, Month 1–2, 2010, City, State, Country.

define what must be monitored and what actions should be available to system administrators. However, this is a very hard task doing so in dynamic environments where the set of components may change during application execution. It is more prudent to retard the process of defining management functionality to be configured and decided during application execution instead of trying to predict which functionalities, devices or components will be available or installed in the application.

The OSGi Service Platform[7] is being extensively used for building dynamic Java applications (a major milestone of OSGi usage in industry concerns its adoption in the Eclipse platform [4]). We are particularly interested in monitoring, and in a broader sense, managing applications developed on top OSGi. In that platform components and services may be installed or uninstalled anytime during application execution without requiring it to be restarted.

We want to be able to provide administration tools that adapt to the functionality of the systems components that are available in the managed dynamic application. If we consider that dynamic applications can have new components deployed at any time, we should be able to easily plug the corresponding administration functionality on the administration tool. Since components of the monitored platform may be added or removed at any time, we proposed doing the same on the administration console application. A tool constructed using a plug-in approach would be ideal in that case. Rich Client Platforms (RCP) such as Eclipse [3] and the Netbeans Platform [1], provide extensible environments that facilitate the construction of desktop applications using plug-ins as building blocks. By using plug-ins developers can easily add new functionality to applications, which in fact are a combination of different plugins. The usage of an RCP would facilitate the development of such management consoles.

However, such possibility of dynamically adding manageable components and their counterpart administration plug-in introduces a new challenge: how to identify the administration plug-ins for a given component?

3. PROPOSED APPROACH

We propose a general architecture for decoupling the management consoles (e.g., monitoring, configuration, instrumentation) from the managed application. The approach leverages a plugin-based architecture at two levels: managed application and management consoles. Figure 1 provides a simplified architectural view of our approach. The bottom side of figure shows the managed application and how probes are incorporated as plugins. In the top side of the Figure RCP platforms are extended using a plugin approach to build specific management consoles.

3.1 Managed Application's Probes

The managed application should be able to expose probes that allow external tools to inspect it and possibly reconfigure it. As said previously, in component-based approaches new components could be added to existing applications, in the same way probes can be added to application (even at runtime). Probes could provide information on specific components (leftmost probe in Figure 1) or on the application as a whole (center probe on the Figure 1). Probes can be deployed in a non-intrusive way as separate components (center and rightmost probes), in that case the managed entity (component or application) is not aware of the probe and it can also be easily replaced without affecting the running components. Otherwise, the probes are part of the component, therefore a probe redeployment would mean an update of the whole component, which would be the case in the leftmost probe of Figure 1.

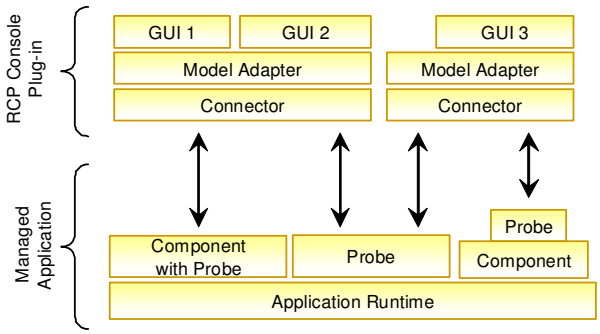

Figure 1. Simplified view of the approach

3.2 Specialized Management Consoles

Management console GUIs can be built on top of RCP platforms. These consoles can offer generic management functionality for the application or specific management functionality for a particular application component. Consequently, the approach allows composing a tailor-made management consoles providing only management functionality for deployed components into the application.

3.3 Automatic Installation Process

Pluggability on both levels imposes a correspondence between managed application's probes and management consoles. When a new probe is deployed into the application runtime, a specific management console can exploit it. On the other hand, when a management console is deployed it requires the suitable application (or component) probe to work properly.

Based on the set of installed probes we should be able to add the corresponding management consoles' plug-ins. A new module the "Console Plugin Resolver" is placed between the two levels, it is responsible for detection of new available probes in the application runtime and then of deployment of the management console plugin into the RCP platform according with the deployed probes.

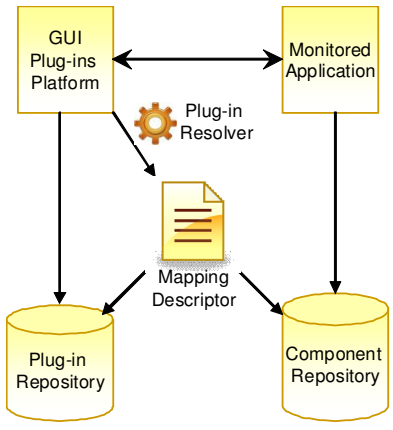

Figure 2. A descriptor to map components to their counterpart management plug-in

Management consoles specify declaratively their required probes, then the Console Plugin Resolver performs a match operation between deployed probes and requires one. When a management console is "able" (all mandatory probes are deployed) to be deployed this task is realized by the resolver.

4. CASE STUDIES

Ideally, the RCP of choice to develop our management plug-ins should provide some basic management functionality. Our case studies concern two sets of plug-ins we have developed for the Oracle Java VisualVM. The VisualVM is a visual tool that instruments Java Virtual Machines (JVM), providing different types of information at the VM level and making available default profilers and data visualizers. The VisualVM supersedes JConsole which was JDK's previous monitoring console generation. It is extensible through a plug-in mechanism since it is built on top of the Netbeans platform. Plug-ins that extend VisualVM functionality easily integrate to the instrumenting infrastructure provided by that platform. No additional components or connection establishment is necessary to access JMX (Java Management Extensions) functionality. A user needs only to connect to a JVM (either local or remote) so the plug-ins are loaded and can have access to the JMX connection to that instrumented JVM.

4.1 OSGi plug-in

The VisualVM OSGi plugin is a management console for OSGi platforms. It does not address a specific OSGi application, but rather the OSGi platform itself. As illustrated in Figure 3, with this plugin it is possible to perform component lifecycle events (start, stop, uninstall, install, update), list bundles and their respective manifest headers, provided services and list of files. This plugin is part of the official list of VisualVM plugins[1] and is available for download through the OW2 Chameleon project[2].

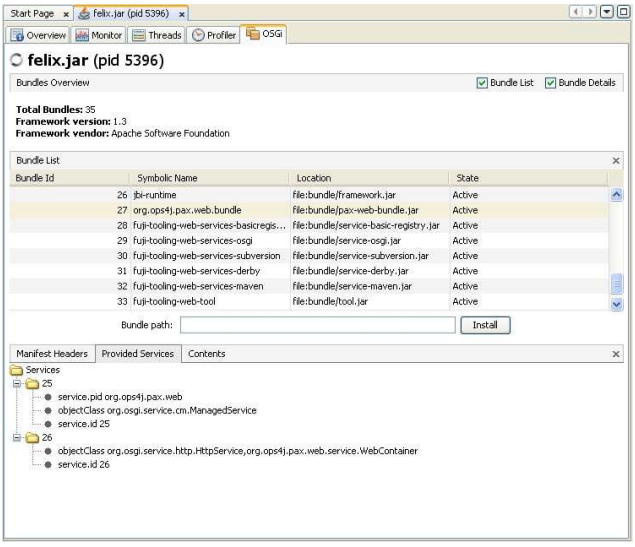

Figure 3. Snapshot of the VisualVM OSGi plug-in

It consisted of an overhauling of an OSGi management console we had done previously as a JConsole plugin that we presented in [2]. Since VisualVM supersedes JConsole as the instrumenting and monitoring console for Java applications, we have migrated the plugin from JConsole to VisualVM, keeping most of the GUI's functionality. Because both JConsole and VisualVM rely on Java's Swing API for building the GUI we could reuse a good part of the GUI code which was not very much affected even though we have changed the MBeans API. This was possible because the adaptation layer of the GUI model hid the changes we had performed on the probe API. However, the information being displayed remained the same.

This plugin has a one to one cardinality with its corresponding probe. In the OSGi application side the probes are deployed in an OSGi bundle that accesses the OSGi API in order to expose management functionality through JMX MBeans. The pluggable probe domain model is shared with the VisualVM plug-in.

4.2 AspireRFID End-to-End Management

AspireRFID middleware[3] proposes an open source infrastructure for creation of applications that uses the RFID technology. This middleware is composed of several components (ALE, BEG, EPCIS, APE). In the context of the AspireRFID European project, we have developed management consoles for some of main components of the AspireRIFD middleware.

ALE server (Application Level Events) is the AspireRFID core component; it is responsible for interaction between client applications and data sources. It allows to clients application utilization of filtered, consolidated RFID tag data and related data from a variety of sources without concerns about data sources heterogeneity.

The ALE component exploits architectural benefits of OSGi platform, consequently different data sources as RFID readers or environmental sensors can be added (as OSGI components) to the middleware without stopping its execution. Data sources' probes are also implemented as OSGi compliant components using the JMX technology.

A plugin on top of VisualVM RCP platform has been developed into the AspireRFID project in order to manage the ALE server and its associated data sources. This plugin has two responsibilities, first the management and monitoring of the ALE server component, but in addition it is able to take into account the apparition of new data sources and it proposes a management interface for these sources. In Figure 4 is presented a screenshot of the ALE Management Plugin, in bottom side is presented how sensors can be monitored when available in the ALE server.

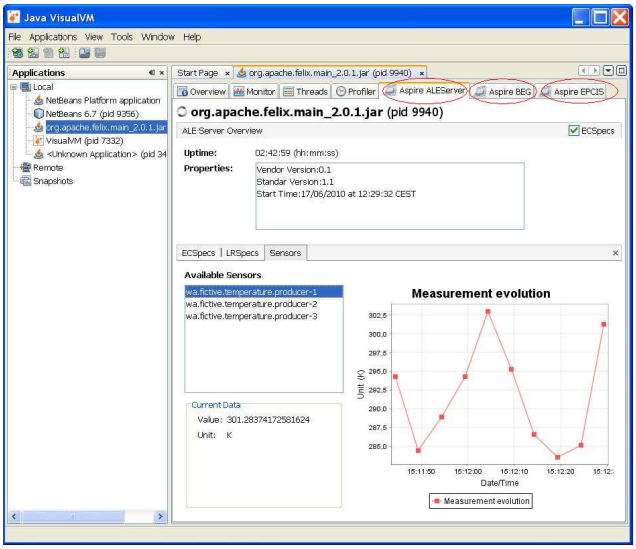

Figure 4. More elaborate functionality broken down into various plug-ins

[1] https://visualvm.dev.java.net/plugins.html

[2] http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/AdminTools

[3] http:// wiki.aspire.ow2.org

For other AspireRFID components as BEG (Business Event Generator), EPCIS (EPC Information System) and APE (Abstract Process Engine) specialized management consoles have been also built. A customized management console is then provided to administrators of the AspireRFID containing only suitable consoles for deployed modules of the middleware. The Figure 4 presents a screenshot with a management environment providing management console for the ALE, BEG and EPCIS AspireRFID components.

5. DISCUSSION

Although it was relatively easy to migrate and adapt the OSGi plugin from the JConsole to the VisualVM, we believe that the possibility of constructing plug-ins that can span multiple tools has a major limitation. That basically related with the GUI because interfaces change from API to API, at least some GUI adaptation code is necessary. Similar problems apply to the usage of the underlying infrastructure (e.g. retrieving the current JMX connection) which also differs among RCP APIs. In general, in order to mimic the same functionality The same management console can be implemented on different RCPs and use the same probes.

By modularizing the code of the probes we are able to keep them separate from the main application. Environments such as OSGi, where it is possible to load components in a pluggable allow introducing new probes on the fly. In the approach we propose, the tooling that perform the management and monitoring resides in an external application built on top of an extensible RCP that should provide facilities to instrument applications. The pluggable probe should have a corresponding plugin on that external application so it can display the non-standard monitored information (i.e. specific to the application domain). or way order to be able to clearly identify/import the reusable parts of the API.

The automatic discovery and deployment of probes and their corresponding management consoles is feasible by providing a central plug-in in the management tool, allowing us able to use the default JMX interfaces for querying the available probes. Based on that info we can then query the descriptor in order to find the corresponding mappings.

6. RELATED WORK

The OSGi Bundle Repository [8] defines the format of an XML file that lists different components and their dependencies. An OSGi framework may also directly install components by using information found in the OBR. It relates to our work especially in the aspect of the representation of dependencies between components, which in our case would be from different platforms.

The closest we have found to our architecture was the Managed OSGi Framework (M-OSGi) [6], which is a management console for OSGi. Instead of using an RCP it provides a simple console based on a plug-in approach where each pluggable tab is resolved based on the MBeans found on the monitored OSGi platform. Disadvantages of this approach concern its underlying platform which is very simplistic, and probe/plug-in cardinality of 1..1.

7. CONCLUSIONS

The approach brings flexibility for dynamically composing customs management consoles adapted to the application components. The automatic discovery and deployment minimizes the burden of administrators when looking for the appropriate management plug-ins for the current application architecture. We think that our contributions are to propose a plugin architecture on application side but also on management console side, the definition of an automatic deployment mechanism for probes and console plugins and the evaluation done on two application domains.

8. ACKNOWLEDGEMENTS

Part of this work has been carried out in the scope of the ASPIRE project (http://www.fp7-aspire.eu), co-funded by the European Commission (FP7 programme, contract 215417). The authors acknowledge help and contributions from all project partners.

9. REFERENCES

[1] Boudreau, T., Tulach, J., Wielenga, G.. Rich Client Programming: Plugging into the NetBeans Platform. Prentice Hall, 1st edition, 2007.

[2] Gama, K., Donsez, D. Service Coroner: A Diagnostic Tool for Locating OSGi Stale References. In Proc. 34th Euromicro Conf. Software Engineering and Advanced Applications (SEAA '08). Washington, DC, USA, 108-115. 2008

[3] Gamma, E., Beck, K. Contributing to Eclipse: Principles, Patterns, and Plug-Ins. Addison-Wesley, 2004

[4] Gruber, O., Hargrave, B. J. , McAffer, J. , Rapicault, P., Watson, T. The Eclipse 3.0 platform: adopting OSGi technology. IBM Syst. J. 44, 2 (January 2005), 289-299.

[5] JConsole. http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html

[6] MOSGi. Managed OSGi Framework. http://felix.apache.org/site/mosgi-managed-osgi-framework.html

[7] OSGi Alliance. OSGi Service Platform. http://www.osgi.org

[8] OSGi Alliance. RFC – 0112 Bundle Repository. http://www.osgi.org/download/rfc-0112_BundleRepository.pdf

[9] VisualVM. https://visualvm.java.net

[10] Web Services Management. http://www.dmtf.org/standards/wsman