HAL
open science

Reconciling Components and Services: The Apam Component-Service Platform

Jacky Estublier, German Vega

To cite this version:

Jacky Estublier, German Vega. Reconciling Components and Services: The Apam Component-Service Platform. IEEE SCC 2012 - International Conference on Service Computing, Jun 2012, Honolulu, HI, United States. pp.683-684, 10.1109/SCC.2012.14 . hal-00745556

HAL Id: hal-00745556

https://hal.science/hal-00745556

Submitted on 25 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Reconciling Components and Services
The Apam Component-Service Platform

Jacky Estublier, German Vega
Grenoble University. LIG.
F-38041 Grenoble, France
{Jacky.Estublier, German.Vega}@imag.fr

Abstract— For Component Based Software Engineering (CBSE), an application is a strongly structured and rigid assembly of components. Conversely, Service Oriented Computing (SOC) is very flexible and is a good candidate for supporting dynamic applications. Unfortunately dynamic applications are software applications and as such they need to be clearly structured and managed (as with CBSE), and they need flexibility and dynamism as with SOC. No platform today satisfies both needs.
This paper presents the Component-Service model that combines well controlled structure and dynamism, and its implementation into the Apam component-service platform.

Keywords-Service; CBSE, SOC, SOA, service platform, component platform, adaptability .

I. INTRODUCTION

With the advent of context-aware computing (context is changing), ubiquitous computing (devices appear/disappear during run-time) and autonomic applications, the concept of dynamic application appeared. A ***dynamic application*** is defined as an application which behavior and composition depends on "external" run-time factors. A large consensus exists [1] to believe that the best way to address the dynamic application issues is to dynamically change the application architecture [2] at run-time.

The traditional software engineering technology is based on "components". There are many definitions of what a component is, but a rough consensus appeared [3]: a component is a piece of code that makes explicit its functionalities (interface) and its dependencies while hiding its internal structure and content. CBSE supports the software engineering best practices, like version control, levels of abstractions, controlled composition, architecture definition and so on. But the architecture (composition) being defined during design and development, CBSE has a big weakness: the applications architecture is rigid. Hence the paradox: component platforms have all the information needed for managing dynamic applications, but they cannot.

SOC (Service Oriented Computing) has been developed to address many of the component's weaknesses including its lack of flexibility [4]. SOC philosophy is that an application is made of services whose number, availability, location, properties are not completely known during development. The main goal is to support the dynamic apparition and disappearance of services and the substitution of a service by another one. These properties explain why OSGi [5] became the de-facto standard for dynamic applications.

Unfortunately, for flexibility and simplicity, SOC platforms have been designed as "only" a low-level run-time interaction protocol with a very limited set of concepts: instances (service) and interfaces. In substance, the platform does not "know" the applications or their architecture and makes connections dynamically and blindly; a SOC application is very loosely controlled. Hence the paradox: service platforms have the technology for managing dynamic applications, but not the concepts to do it.

To control a SOC application, the developer should manually manage the application dynamism, but the code to write to do so is so complex that is too hard to do in practice [6]. The challenge is to be able to describe "easily" how much control as needed on the dynamic parts of the application, because the current dynamic platforms do not provide any mechanism for that. It means that many actions traditionally performed at development must be delegated at run-time; and therefore, the platform must allow performing these decisions at run-time.

Obviously there is a conflict between, on the one hand, a clear, consistent but rigid architecture, and on the other hand a flexible, dynamic but loosely controlled execution.

In this paper we present how we have extended a SOC platform (namely OSGi), in order to propose concepts and mechanisms allowing dynamic application developers to explicitly associate the kind and level of control they wish.

II. THE COMPONENT-SERVICE MODEL

A "component" is essentially an implementation that provides and requires resources (most often interfaces only). The application architecture is defined connecting client implementations to provider implementations. Therefore the component approach is implementation and architecture centric, and addresses primarily development and composition.

In contrast, a service is essentially a run-time artefact: an instance that publishes an interface and that asks at run-time the services providers it needs. The service approach is instance centric, architecture free and addresses primarily the run-time phase.

Clearly, these visions are complementary; some parts need to be strongly controlled and others can/should be dynamic and opportunistic. To that end we have extended the OSGi definition of what a service is. Instead of being defined by its interface, a service is defined as providing a **specification**, with specification borrowed from components i.e. a set of provided and required resources with constraints. To keep the flexibility missing in components, a specification is implemented by a group of "equivalent" implementations, and dependencies are defined in term of specifications (and optionally some constraints). Therefore, at run-time, when a service asks for a specification, the platform is free to make the choice of the most relevant implementation and instance, with respect to the current context, using the currently available services even if not known during design and development. The platform is also extended by a list of repositories from which implementations can be dynamically deployed if needed.

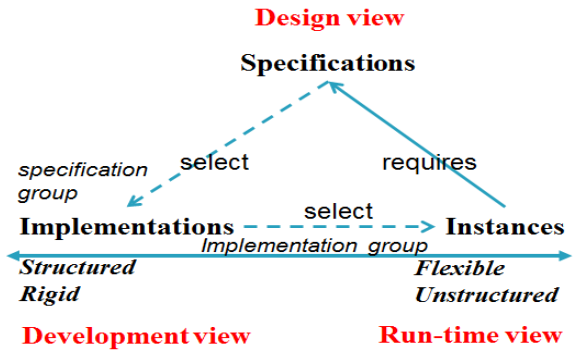

Figure 1: The Component-Service Model

To some extent, the platform acts as a run-time configuration control system, selecting in well-defined repositories the right implementation (in the good revision and variant), and even as an extension of a traditional CMS [7], because it also selects (or creates) the right instances (with the good properties and initialisation parameters), taking into account dependencies, sharing and access control.

We have implemented the Component-Service model sketched above in the APAM (APplication Abstract Machine) platform. Its goal is to extend the OSGi platform in order to define and control applications. From OSGi we retain its dynamism and performances; from component, we retain the strictly defined and controlled architecture. In APAM the application designer is free to develop and execute applications whose dynamic behaviour is anywhere in the range from component-like (rigid) to service-like (loosely controlled).

III. Conclusion

We have tried to solve the conflicting requirements of, on a one side, been dynamic and flexible, including opportunism and non-determinism and on the other side, to be closely controlled, deterministic and repeatable. Fundamentally, our solution is to divide Software Configuration Management in two parts, one performed at development, building well controlled component repositories, and a run-time part, performing composition by selecting implementations and instances into a number of repositories, including the set of services actually running on the platform.

In Apam the concept of specification is central and makes the link between instances (service point of view) and implementations (components point of view); but to be used at run-time it must become a first level entity, designed, packaged, and deployed in the same way as implementations. We believe that the platform recognizes the specification as a first class citizen, and repositories for dynamic deployment are important contributions.

Modern applications will be structured in parts being assembled once for all at development, other parts assembled dynamically depending on the context but using only pre-defined components; others assembled using components discovered dynamically. These future applications will require platforms and models like those provided by Apam.

References

[1] J. Magee and J. Kramer, "Dynamic structure in software architectures", Proceedings of the 4th symposium in Foundations of Software Engineering. 1996

[2] P. Oreizy, N. Medvidovic, R. Taylor, "Architecture-Based Runtime Software Evolution", Proceedings of the 20th International Conference on Software Engineering (ICSE'98).

[3] I. Crnkovic, S. Sentilles, A. Vulgarakis and M.R.V. Chaudron, "A Classification Framework for Software Component Models", IEEE Transactions on Software Engineering, Vol 37, No. 5, September 2011

[4] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", IEEE, November 2007, pp. 38-45.

[5] OSGi Alliance, "OSGi Service Platform Core Specification Release 4", http://www.osgi.org, August 2005.

[6] C. Escoffier, R. S. Hall and P. Lalanda, "iPOJO: an Extensible Service-Oriented Component Framework", IEEE Int. Conference on Services Computing, USA, July 2007

[7] J. Estublier, D. Leblang, A. Van Der Hoek, R. Conradi, G. Clemm, W. Tichy and D. Wiborg-Weber. "Impact of Software Engineering Research on the Practice of Software Configuration Management". Published in IEEE TOSEM. October 2000.