# Reconfigurable Run-Time Support for Distributed Service Component Architectures

Rémi Mélisson, Philippe Merle, Daniel Romero, Romain Rouvoy, Lionel Seinturier

# Reconfigurable Run-Time Support for Distributed Service Component Architectures

Rémi Mélisson, Philippe Merle, Daniel Romero, Romain Rouvoy, and Lionel Seinturier
INRIA Lille – Nord Europe, ADAM Project-team
University of Lille 1, LIFL CNRS UMR 8022
59650 Villeneuve d'Ascq, France
firstname.lastname@inria.fr

## ABSTRACT

SCA (*Service Component Architecture*) is an OASIS standard for describing service-oriented middleware architectures. In particular, SCA promotes a disciplined way for designing distributed architectures based on a component model and an *Architecture Description Language* (ADL). However, SCA does not cover the deployment and the run-time management of SCA applications. In this paper, we therefore describe the FraSCAti platform, which provides run-time support, deployment capabilities, and run-time management for SCA. Compared to state-of-the-art platforms, FraSCAti brings a dynamic reflective support to SCA and enables both introspecting and reconfiguring service-oriented architectures at run-time. To achieve this capability, the components are completed by a dedicated container, which is automatically generated by the platform. Furthermore, FraSCAti is a highly configurable platform that can be easily customized by finely selecting the features and functionalities which need to be included. In this way, the platform can be adapted to different application needs and middleware environments.

## Keywords

Component-Based Software Engineering (CBSE), Middleware, Service Component Architecture (SCA), Service-Oriented Architecture (SOA)

## 1. INTRODUCTION

*Service-Oriented Architecture* (SOA) promotes a style of distributed software architectures based on high-level characteristics, such as loose-coupling, service encapsulation, and service self-description. On top of these principles, SCA defines a hierarchical component model. SCA software components require and provide services, which are locally or remotely accessible. The SCA standard can be split into three main parts: *i)* Component Implementations define how SCA components should be implemented with programming languages (*e.g.*, Java, C++) or advanced web-oriented technologies like Spring beans or BPEL orchestrations; *ii)* Binding Specifications describe how different communication protocols (*e.g.*, SOAP, Sun JMS) can be bound to SCA components in order to handle their remote communications; *iii)* The Assembly Language provides an XML-based grammar for the description of SCA component assemblies.

A dynamically reconfigurable run-time architecture for SOA has been identified by [6] as a key research challenge for modern SOA systems. This article proposes a solution for this challenge with the FraSCAti middleware platform. The next section describes the architecture of the FraSCAti platform.

## 2. THE FRASCATI PLATFORM

FraSCAti is a reflective platform for deploying, hosting, and managing SCA applications, and its different subsystems are implemented as SCA components. FraSCAti provides an homogeneous view of a middleware software stack where the platform, the non-functional services, and the applications are uniformly designed and implemented with the same component-based and service-oriented paradigm. To achieve this, the architecture of the platform (illustrated in Figure 1) relies on the four following layers:

*Kernel Level.* This level relies on Fractal [3], which is a lightweight and open component framework with basic dependency injection, introspection and reconfiguration capabilities. Fractal does not impose a static execution semantics for components by allowing the programmation of different meta-level activities to customize this semantics.

*Personality Level.* This level customizes the component kernel by providing the execution semantics for components as defined in the SCA specifications [1]. In particular, this level provides an implementation of the SCA API and principles based on the Fractal component model.

*Run-time Level.* This level is in charge of deploying and instantiating assemblies of SCA components. This is achieved in a three-step process: parsing, generation and assembly. The Description Parser component reads XML-based descriptions of SCA assemblies and converts them into an EMF instance of the FraSCAti meta-model. Using generation techniques (both source code and bytecode), the Personnality Factory component is in charge of producing the personality for SCA components. For its part, the Assembly Factory wires application components according to the run-

time model provided by the Description Parser. FraSCAti provides different ways to connect components or expose services out of the application scope. The following communication protocols are supported: SOAP, REST HTTP, JSON-RPC, Java RMI, UPnP, and SLP. Moreover, FraSCAti supports different service description language such as Java, WSDL, and UPnP.

In FraSCAti, the run-time level is dynamically composed according to application needs. A core bootstrap (whose code size is 256KB) is initially launched and then, additional plugins (implemented as SCA architectural fragments) are loaded depending on application needs. For example, the UPnP plugin is only added if a UPnP binding is specified in the SCA application descriptor.

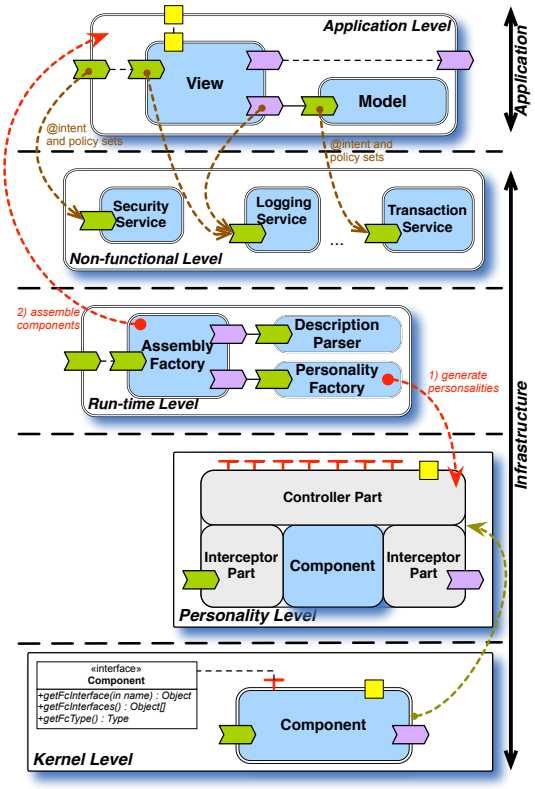


**Figure 1: FraSCAti Platform Architecture.**

*Non-Functional Level.* The SCA Policy Framework specification enables attaching metadata to component assemblies to specify requirements in terms of non-functional properties, such as security or transaction. FraSCAti provides a symmetric mechanism for supporting this specification. In particular, the non-functional services are implemented as regular SCA components. Then, these non-functional services are wired to the business components via an AOP-like interception mechanism.

The FraSCAti platform enables several reconfiguration capabilities, which are distributed into its underlying architecture. At the personality level, SCA elements (*e.g.*, wires, properties, hierarchies) can be modified at run-time. Thus, SCA applications are able to adapt themselves and to fit the context-aware and autonomic application requirements.

In addition to supporting on-demand component instanciation, the run-time level also provides dynamic component bindings. Furthermore, while the application is being executed, components are able to expose services using new communication protocols.

# 3. RELATED WORK

Several implementations of the SCA specifications are already available. For example, IBM, Oracle, Rogue Wave Software develop commercial solutions of SCA while Apache Tuscany [8], Newton , Fabric3, and FraSCAti are distributed under open source licenses. Compared to these solutions, the originality of FraSCAti is to provide dynamic introspection and reconfiguration capabilities for SCA applications and to enable customizing the platform to application needs.

OSGi [7] defines another service-oriented component model for SOA. Compared to OSGi, FraSCAti provides more degrees of reconfiguration, and is therefore not only restricted to bundle deployment and removal.

Many other component frameworks, such as OpenCOM [4], Hadas [2], and K-Component [5], exist for designing and implementing middleware solutions. Compared to FraSCAti, these frameworks are closer to the Fractal model, which is the foundation of the platform.

# Acknowledgments

This work was partially supported by the ARPEGE ITEmIS project, the IST FP7 IP SOA4All project and by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the 'Contrat de Projets Etat Region (CPER) 2007-2013'.

# 4. REFERENCES

[1] Beisiegel, M. et al. Service Component Architecture, Nov. 2007. `www.osoa.org`.

[2] I. Ben-Shaul, O. Holder, and B. Lavva. Dynamic Adaptation and Deployment of Distributed Components in Hadas. *IEEE Trans. Soft. Eng.*, 27(9), 2001.

[3] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal Component Model and its Support in Java. *Soft. Pract. and Exp.*, 36(11-12):1257–1284, 2006.

[4] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Jolla, K. Lee, J. Ueyama, and T. Silvaharan. A Generic Component Model for Building Systems Software. *ACM Trans. Comp. Sys.*, 26(1):1–42, Feb. 2008.

[5] J. Dowling and V. Cahill. The K-Component Architecture Meta-Model for Self-Adaptative Software. In *Proceedings of Reflection'01*, volume 2192 of *LNCS*, pages 81–88. Springer, Sept. 2001.

[6] M. Papazoglou and P. Traverso and S. Dustdar and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):64–71, Nov. 2007.

[7] OSGi Alliance. *OSGi Service Platform Core Specification Release 4*, Aug. 2005. `www.osgi.org`.

[8] The Apache Foundation. *Tuscany SCA*, 2006. `tuscany.apache.org`.

# APPENDIX

# A. PLATFORM INFORMATION

## A.1 Availability

Further information about the FraSCAti platform can be found on the website[1]. In particular, a detailed user guide is available, which provides support for using the platform.

FraSCAti is a *Free/Libre Open Source Software* (FLOSS), distributed under the terms of the GNU *Lesser General Public License*[2](LGPL) as part of the OW2 consortium. Instructions to checkout the source code is available from the website.

## A.2 Tooling Support

The FraSCAti Explorer is a graphical console for managing the deployment and the reconfiguration of SCA applications (cf. Figure 2). This console can be used to interact with SCA applications and change their behavior dynamically. This reconfiguration process is based on a reconfiguration DSL whose interpreter is included in the FraSCAti Explorer.

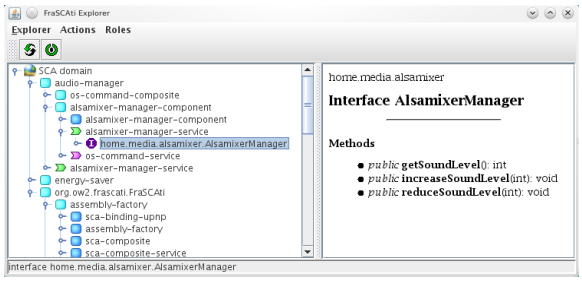


**Figure 2: Screenshot of the FraSCAti Explorer.**

# B. TOOL DEMONSTRATION

Our demonstration illustrates various reconfiguration capacities of FraSCAti in a scenario that puts into practice a distributed feedback control loop for monitoring and reconfiguring a service-oriented architecture.

## B.1 Devices Testbed

Our demonstration includes two laptops and one Android smartphone connected through a WiFi router. The first laptop plays the role of a *multimedia server* with file storage and audio capabilities. Its software layer relies on FraSCAti and the VLC media player[3]. The second laptop plays the role of a regular *personal computer*, running FraSCAti and a web browser. Finally, an Android-based *smartphone* is used to remotely control the media server.

## B.2 The Media Server Scenario

The demonstration is decomposed into three stages, which demonstrate the deployment of FraSCAti with heterogeneous hardware and protocols, highlighting its reconfiguration capabilities. An overview of the architecture for the scenario is depicted in Figure 3.

*Scene 1.* FraSCAti is launched from the FraSCAti Explorer (cf. Figure 2). The platform hosts a media server application, which can be remotely controlled in order to manage sound level or play songs. The audio manager component exposes its services on HTTP by means of the FraSCAti JSON-RPC binding. Additionally, a non-functional component logger, which is able to display logs, is wired as an intent to the audio manager component, in order to registry the remote invocations. At this time, we can use FraSCAti on the personal computer as a client for the media server, with the help of its web browser.

*Scene 2.* Secondly, we extend the remote control capacity by dynamically adding a UPnP binding to the audio manager component. The media server audio service can therefore be discovered at runtime by standard UPnP devices[4] available in the surroundings. In particular, we use the Android smartphone to remotely control the media server without any prior configuration (except the network connection).

*Scene 3.* For the last part of the demonstration, we deploy an energy-saving component into the media server to adjust the screen brightness, or the CPU speed. A feedback control loop deployed on the platform detects the deployment of this new component and automatically launches a client interface on the *personal computer* to control the media server energy saving capabilities.

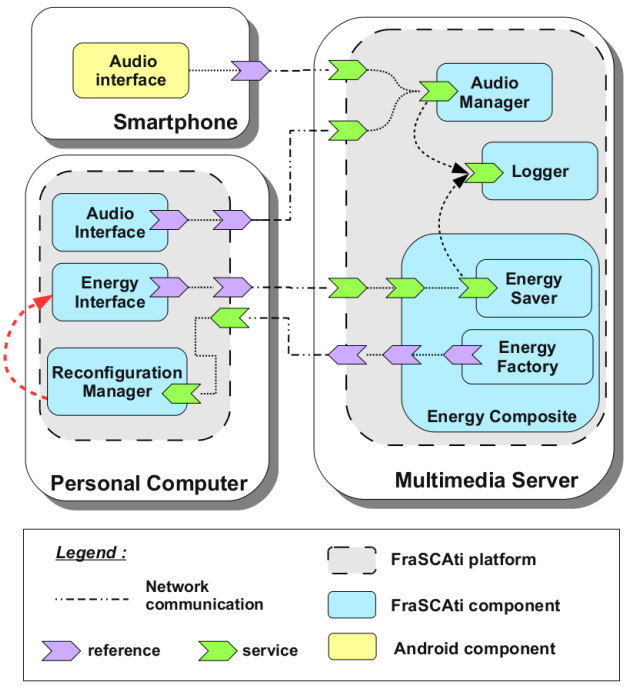


**Figure 3: Media Server scenario.**

[1]FraSCAti website: `http://frascati.ow2.org`
[2]LGPL: `http://www.gnu.org/licenses/lgpl.html`
[3]VLC media player: `http://www.videolan.org/vlc`
[4]The UPnP Forum: `http://www.upnp.org`

## B.3 Scenario Highlights

This scenario highlights key situations where the run-time introspection and reconfiguration capabilities of FraSCAti are illustrated:

- In Scene 1, the JSON-RPC binding demonstrates the automated deployment of a remote service.
- In Scene 1, the logger service describes the seamless integration of non-functional services. Deployed as an independent component, it is clearly separated from the business code and can be used by both the media server and energy saving components.
- In Scene 2, the dynamic reconfiguration to add the UPnP binding can be seen as a technical reconfiguration.
- In Scene 3, the feedback control loop automates a functional reconfiguration, deploying new components to support energy-saving features.