Components and Services: A Marriage of Reason

Philippe Collet, Thierry Coupaye, Hervé Chang, Lionel Seinturier, Guillaume Dufrêne

LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

COMPONENTS AND SERVICES: A MARRIAGE OF REASON

Philippe Collet, Thierry Coupaye, Hervé Chang, Lionel Seinturier, Guillaume Dufrêne

Projet RAINBOW

Rapport de recherche
ISRN I3S/RR–2007-17–FR

Mai 2007

Laboratoire d'Informatique de Signaux et Systèmes de Sophia Antipolis - UNSA-CNRS
2000, rte.des Lucioles – Les Algorithmes – Bât Euclide B – B.P. 121 – 06903 Sophia-Antipolis Cedex – France
Tél.: 33 (0)4 92 94 27 01 – Fax: 33 (0)4 92 94 28 98 – www.i3s.unice.fr
UMR6070

RÉSUMÉ :

MOTS CLÉS :

ABSTRACT:

Component-Based Software Engineering (CBSE) and Service-Oriented Architectures (SOA) are among todayŠs most prominent software architecture approaches. This article investigates the focus and characteristics of these approaches based on works conducted in the context of France Telecom activities, then it argues towards bringing them together, and sets up some recommendations for their integration. The article reports, assesses and compares two experiments which were conducted in this line with the Fractal component model: the first one is based on the Axis and kSOAP Web Services platforms, and the second one is based on the Tuscany SCA platform.

KEY WORDS :

CBSE, SOA, Fractal, Web Service, SCA

Components and Services: A Marriage of Reason*

Philippe Collet[1], Thierry Coupaye[2], Hervé Chang[1], Lionel Seinturier[3], and Guillaume Dufrêne[3]

[1] Université de Nice Sophia-Antipolis, Laboratoire I3S CNRS UMR 6070
Polytech Nice - Sophia, 930 Route des Colles, BP 145, 06903 Sophia-Antipolis, France
{Philippe.Collet,Herve.Chang}@unice.fr
[2] France Telecom R&D
28 chemin du Vieux Chêne, BP98, 38243 Meylan, France
thierry.coupaye@orange-ftgroup.com
[3] INRIA Futurs - LIFL, Projet Jacquard/GOAL
Bâtiment M3, 59655 Villeneuve d'Ascq, France
{seinturi,Guillaume.Dufrene}@lifl.fr

Abstract. Component-Based Software Engineering (CBSE) and Service-Oriented Architectures (SOA) are among today's most prominent software architecture approaches. This article investigates the focus and characteristics of these approaches based on works conducted in the context of France Telecom activities, then it argues towards bringing them together, and sets up some recommendations for their integration. The article reports, assesses and compares two experiments which were conducted in this line with the Fractal component model: the first one is based on the Axis and kSOAP Web Services platforms, and the second one is based on the Tuscany SCA platform.

1 Introduction

Two approaches have raised a huge interest in the last few years in the software architecture and distributed systems area: *Component-Based Software Engineering* (CBSE) and *Service-Oriented Architectures* (SOA). These two approaches essentially appear as competing when reading publications or listening to their respective proponents. Several reasons may be evoked, among which a great variability inside each approach (especially in the component world), a probable lack of maturity (especially in the service world) and altogether a mediatization, not to say a *marketing buzz* organized by the two different communities (e.g. consider today the very few conferences that explicitly discuss and compare both approaches).

This article precisely tackles the comparative study and positioning of components and services with the overall objective of arguing that the two approaches can be seen as much more complementary than competitors. It first describes

* This work was partially supported by France Telecom under the collaboration contracts number 46132097 and 46133723 with CNRS/I3S and 46131097 with INRIA, and by the ANR RNTL FAROS project.

the salient characteristics of CBSE and SOA and draws their respective benefits and disadvantages. Both approaches are then analyzed according to two different viewpoints: software reuse and integration, and functional and technical aspects. We argue that CBSE and SOA should be integrated to enable some form of component-based engineering of services. To demonstrate this, two ongoing experiments based on a hierarchical component model – *Fractal* –, and two service technologies – *Web Services* and *Service Component Architecture* (SCA) are presented, and the benefits of their integration and bridging are assessed.

The remaining of this paper is organized as follows. Section 2 compares synthetically CBSE and SOA. Section 3 introduces some background material on the main technologies used. Sections 4 and 5 present and assess the two ongoing experiments, *Fractal WS* and *Fractal SCA*, to integrate Fractal components with respectively Web Services and SCA. Section 6 draws conclusions on the integration of CBSE and SOA.

2 CBSE and SOA

This section highlights the most prominent characteristics of CBSE and SOA by notably focusing on the following criteria: encapsulation, granularity, coupling, state and composition. It then discusses these two architectural approaches according to two points that are, according to us, confusing and make their positioning difficult – that is software reuse and integration, and functional and technical aspects. Finally, it discusses current industrial practice and advocates for integration of components and services.

2.1 Characteristics of Component-Based Architectures

The most prominent characteristics of the CBSE approach are:

Black, white, gray boxes! Like SOA, CBSE promotes the principle of separation between interfaces and implementations. It even pushes this principle a step further as components are also free to expose or not their internal structure in terms of sub-components. This has lead to the various characterization of “black boxes”, “white boxes” and even “gray boxes” unit of composition. It is worth noticing that this debate, which is due to some extent, to the well-known definition of component given by C. Szyperski stating that a component is ”a black box without state...”, has lost much of its initial intensity within the CBSE community.

Arbitrary granularity The CBSE approach is agnostic about the granularity of components. They can be of a completely arbitrary size: from the size of a driver or pool in an operating system to a transactional monitor and even a complete DBMS. Actually, one of the greatest strength of the CBSE approach is precisely to enable a uniform control of any software resources at any desired level of abstraction.

State The CBSE approach is also globally agnostic about the fact that components have a state or not, although in many application domains, as this was experimented with the Fractal component model in several technical software infrastructures such as OS, middleware or applications servers, components often do have a state that needs to be handled.

Reflection Most components, at least in the most recent and advanced component models such as OpenCOM [1] and Fractal [2–4] integrate mechanisms to support introspection and more generally reflection capabilities, as they are necessary to control the structural composition of components (component bindings, life-cycle, state...). In a more general way, the needs for such mechanisms have already been motivated in "industrial component models" such as EJB or CCM, to control technical services and non-functional aspects of components.

Composition A component is a software entity which conforms to a component model. It specifies rules about the composition and interaction between components, and it is generally based on a computational or abstract model, a programming model and sometimes an engineering model. There is no frontal opposition here between CBSE and SOA but while SOA insists on orchestration, CBSE insists on composition.

Structural composition Although works in CBSE are interested in various types of interaction (synchronous, asynchronous, by signals, etc.) and composition (of which the behavioral composition which is not without relation with the concept of contract), component models basically propose concepts and mechanisms to control *bindings* between components and in particular the connections between components and their sub-components, that is the hierarchical *structure* of a system. The coupling is thus much stronger between components than between services, but it is weaker than in the preceding approaches, in particular in the object-oriented one. In components, connections are not static nor "hidden in the code". On the contrary, they are externalized and made easily manipulable notably in a programmatic and dynamic way as it is in the most advanced component models.

The main benefits of the CBSE approach are the flexibility, dynamicity (greater manageability and maintainability) and predictability of systems built by assembly of components. Component-based architectures provide *structural abstractions* which make it possible to reason about the *behavior* of software systems with respect to integrity, security, isolation, performances, QoS, real-time... The current major disadvantages of the CBSE approach concern the lack of mature techniques to guarantee predictable properties formally (e.g.real-time). The principal bolt concerns the compromises between the expected flexibility and the trust of the systems built by assembly of components.

2.2 Characteristics of Service-Oriented Architectures

The most prominent characteristics of the SOA approach are:

Black boxes Since services do not have explicit and reified knowledge of the implementation of other services nor of their own internal implementation, services really appear are black boxes. The only visible part of a service is its 'external side' which is made of the interfaces it provides. SOA here aligns with the architectural principle of separation between interfaces and implementations which is now commonly applied in modern programming languages and component-based approach.

Coarse granularity This characteristic is not so much strictly tied and intrinsic to the SOA approach than the previous ones. Nevertheless, the loose coupling and the most frequent usages of the SOA technology (orchestration of web services on the Internet, orchestration of objects/services in domestic environments *à la* UPnP) imply a relatively coarse and fixed granularity for services. In fact, a service, in the SOA approach, is rarely an operation on a hash table or a pool in a DBMS!

Loose coupling Services are *functions* which are externalized and accessible only through interfaces. They represent the services "provided" by service producers to service consumers. Moreover, although a service may have dependencies towards other services, or towards components that are part of its implementation, such dependencies are not made explicit at the service interface level.

Stateless A corollary and even a necessary condition for loose coupling is that services are stateless. Actually, as a service is necessarily implemented by one (or several) program, it does have a state. However, the state of the service does not affect any of its interactions with the service consumers. It is neither accessible, nor manipulable by other services or other entities (like administration consoles).

Discovery As services do not have intrinsic dependencies nor do they have knowledge of the other services with which they have to interact, mechanisms for service discovery (naming, trading, brokering services) are of particular importance.

Orchestration Since dependencies and *a fortiori* interactions betweens services are not visible in service descriptions, it is necessary to specify separately from the services themselves, how and when they interact. This is the classical role of *orchestration* and *choreography* which basically encompass the well-known concepts in software engineering of *workflows* and *processes*.

The main benefits of the SOA approach concern dynamicity, scalability and availability. Indeed, since a service is basically a function which is accessible from a well-known access point (typically a location on a network), one can easily imagine that several services that provide the same function may be instantiated and executed simultaneously on multiple network nodes. This then provides interesting availability and scalability capabilities of service-based systems. As for its main current weaknesses, they essentially concern aspects such as dependability (security, reliability and availability) and maintainability (manageability and QoS guarantees). It is also worth noticing that loose coupling plays a central role in SOA. Almost all other characteristics (stateless, discovery, orchestration)

of the approach result from it as well as its strengths and weaknesses. Availability is an interesting point in this approach. In an optimistic vision, one can think that a service which is involved in an orchestration will be always available since it can be instantiated and shared several times. However, due to loose coupling, actually nothing can guarantee that, as one may also remove unilaterally a service on a given node.

2.3 Architectural Viewpoints

Software Reuse and Integration When debating about CBSE and SOA, a first point which complexifies their positioning and often results in opposing them frontally relates to the problem of software integration. Software integration constitutes a major industrial bolt, in particular with the emergence of Internet, and in a more general way, with the convergence of computing and telecommunications. In this context, software systems are thus now omnipresent and they are still getting more and more complex and critical (complex software products, development costs, deployment, maintenance and evolution management, public image, time-to-market constraints). This complexity covers several aspects among which the intrinsic algorithmic complexity of such systems (raised for example by distribution, faults or asynchronism) but also, and this is what interests us here, the complexity in terms of systems engineering, as a consequence of the increasing heterogeneity at many various levels (terminals, devices, network protocols, operating systems, middleware, service platforms, services) and also the variability (mobility, nomadism) and openness of nowadays execution environments (dynamic topology). In this context, CBSE and SOA approaches appear different and competing, and from our point of view, although they represent two software integration technologies, they do not relate to, *in fine*, the same type of integration. The CBSE approach rather relates to a *pre facto* integration whereas the SOA approach rather consists in a *post facto* integration.

In CBSE, the integration can be described as *pre facto*, as components are explicitly built in conformity with an architectural model, which then make it possible to compose them gracefully. The CBSE approach has the objective to cover the whole software lifecycle (development, deployment, management, maintenance stages), and this type of integration corresponds to a systematic development and deployment process. Integrated components are homogeneous but can be built with an arbitrary granularity. Moreover, coupling and control on components are strong but dynamic. Software reuse also relates primarily to the "reuse of code as such" (i.e. rather "types" or "libraries" of components which can be instantiated several times), altough instance sharing as in the SOA approach is possible. Finally, some component models like the Fractal one, also allows one to share components within the same system but possibly in various locations of the system.

In SOA, the integration can be described as *post facto* as it does focus on how services are constructed and executed. Services exist and must be integrated almost just as they are. The major constraints relate to the set of underlying protocols notably used for discovery and communication (e.g. SOAP, UPnP).

The SOA approach has the objective to expose and reuse services, and this type of integration corresponds to an opportunistic development in which existing services, managed by multiple administrative entities services (e.g. services on the Web), are integrated within orchestrations. The integrated services are globally heterogeneous, with fixed and rather coarse granularity. Moreover, coupling and control on services are weak to null. A service is (in approaches such as UPnP, a little bit less in approaches such as Web services) frequently a singleton (in object-oriented terminology, a service would be an object rather than a class). Software reuse relates to service orchestrations and corresponds to a form of "reuse of code in execution" through "instance sharing" : a same service (singleton) can be shared and integrated in several orchestrations simultaneously.

Functional and Technical Aspects A second point which scrambles the debate between CBSE and SOA relates to the bare grounds of the two approaches since it relates to the architecture of software systems. Indeed, this might seem obvious but the SOA approach only deals with the functional point of view of software architecture. On the contrary, the CBSE approach covers a broader range of aspects which concern the functional aspect (set of functions/services that form a system) but also and mostly it deals with the structural (the organization of functions/services) and technical aspects (the integration of non-functional aspects).

2.4 Co-existence or Integration

In the current industrial practice, we generally observe the choice of one preferred technology depending on considered applicative contexts. CBSE is typically preferred in middleware and embedded systems which exhibit strong non-functional constraints. While SOA is typically preferred in e-business applications (e.g. organization of travels with online booking of transport tickets, renting cars, hotels, etc.) which necessitate to chain calls to online services provided by distinct companies (booking centrals, airlines companies, car rental companies, etc.). In other applicative contexts, for instance that of enterprise information systems (IS), we observe a *co-existence* of both technologies where they are used in different parts of a system. CBSE is typically used to implement business components into back office application servers (e.g. J2EE), whereas SOA is used to integrate these components by giving an agglomerate view, and interact with other services in the considered information system - and a fortiori with external services typically available on the Internet. This approach matches the one proposed in [5].

Another approach put forward by this article consists in pushing further the *integration* of SOA and CBSE to show their complementarity. The proposal is sketched by approaches like SCA (Service Component Architecture) [6] or OSGi. These approaches tends to shuffle concepts and mechanisms coming from SOA and CBSE, resulting in models not always homogeneous, nor very easy to handle. We rather propose to keep intact the two complementary views. The proposal includes a component-based system engineering of services, i.e. an implementation

of services compliant with a component model ; and some mechanisms (a.k.a. *bridges*) to *expose* (or *export*) some components as services allowing both a lifecycle management (deployment, administration, reconfiguration) à la CBSE, while offering a loose coupled communication layer (including discovery) à la SOA between these services/components.

A first experiment in this line was done in the context of the Fractal component model with Fractal JMX [7], which allows for the automatic, non intrusive and declarative (with filtering and parametrization capabilities) exposure of Fractal components as JMX MBeans - providing a *JMX personnality* to Fractal components. Two other experiments are being conducted in the Fractal ecosystem with Web Services and Service Component Architecture (SCA). They are discussed in the sequel of this paper, after the next section which provides background on the context of these two experiments.

3 Background

This section introduces background material on the Fractal component model, Web services and SCA technologies.

3.1 Fractal

Fractal [2–4] is an advanced component model and associated programming and management support devised initially by France Telecom and INRIA since 2001. Most developments are framed by the Fractal project[4] inside the ObjectWeb open source middleware consortium. The Fractal project targets the development of a reflective component technology for the construction of highly adaptable and reconfigurable distributed systems, middleware, operating systems, etc[5].

The Fractal component model itself relies on some classical concepts in CBSE: *components* are runtime entities that conforms to the model, *interfaces* are the only interaction points between components that express dependencies between components in terms of *required/client* and *provided/server* interfaces, *bindings* are communication channels between component interfaces that can be primitive, i.e. local to an address space or composite, i.e. made of components and bindings for distribution or security purposes.

Fractal also exhibits more original concepts. A component is the composition of a *membrane* and a *content*. The membrane exercises an *arbitrary reflexive control* over its content (including interception of messages, modification of message parameters, etc.). A membrane is composed of a set of *controllers* [6] that may or

[4] `http://fractal.objectweb.org/`

[5] Several ObjectWeb softwares (`http://www.objectweb.org/`), such as CLIF, Speedo, JOnAS, GOTM or Petals, embed Fractal technology and are used operationnally e.g. the JOnAS J2EE application server is widely used inside France Telecom for its service platforms, information systems and networks by more than 100 applications.

[6] The way controllers are defined and composed inside membranes is generally what the most distinguishes Fractal implementations one from each others.

may not export control interfaces accessible from outside the considered component. The model is *recursive (hierarchical) with sharing* at arbitrary levels. The recursion stops with base components that have an empty content. Base components encapsulate entities in an underlying programming language. A component can be shared by multiple enclosing components (in this case, its behaviour is controlled by the first super component that encapsulates the components that share the considered shared component). Finally, the model is *open*: everything is optional and extensible in the model, which only defines some common/standard APIs for navigating into and controlling component structures e.g. controlling the bindings between components (BindingController to introspect/bind/unbind components interfaces), the hierarchical structure of a component system or the components life-cycle (creation, start, stop, etc).

3.2 Web Services

Web Services is a widespread technology for integrating, wrapping and exposing existing functionality in a platform- and programming language-independent way, thus moving towards interoperable systems and loosely-coupled applications. On a conceptual level, these software interfaces represent units of functionality provided by *service producers*, each handling a specific functional task. These interfaces require to be properly described and published in *service registries* in such a way that *service consumers* can discover, localize and invoke them dynamically. They can also be combined into higher-level tasks to provide particular business-oriented processes. On a technical level, a Web Service is a software interface which describes a collection of running operations available on the Web that can be accessed over the network through standardized protocols based on the XML language. To achieve interoperable services, Web Services relies on a family of related technologies which include the three core specifications SOAP [8], WSDL [9], and UDDI [10]. These specifications respectively address message exchanges (message format, data encoding...), interface description (operations, messages, concrete format and protocol binding...) and localization of web services (registry and protocols). Other web services protocols (WS-*), at different stages of maturity, are also continuously being proposed and adopted to address higher-level aspects of the Web Services Architecture such as management, security, transactions and business processes, etc. [11].

3.3 SCA

The Service Component Architecture (SCA) is an initiative which aims at providing a component model for SOA. While Web Services mainly focus on system interoperability and loose-coupling, SCA targets the structuring of these systems. The aim is to bring to SOA the benefits of component-based development.

The SCA specifications are being developed by the Open SOA Collaboration[7]. SCA defines a model for the assembling of service components [12] and a

[7] `http://www.osoa.org/`

model for the creation of component services. This model is currently available for the Java [13], C++, Spring and BPEL languages. The idea is to promote the decoupling of the way components are implementing from the way they are assembled. A second important idea is that SCA can accommodate various communication technologies between components: Web Services which are maybe the preferred and most common choice, but also message-oriented middleware such as JMS, or request/response protocols such as IIOP.

The specification of the SCA assembling model [12] is currently subject to some intense work by the Open SOA members. The remainder of this description is based on version 0.96 draft 1 of the specification. The basic artifact of a SCA system is the *component*. A component provides *services* and may depend on services provided by other components. In such a case, these dependencies are called *references*. Services and references are described with an interface definition language which can be Java or WSDL. Several SCA components can be assembled into a *composite*. The top-level composite is called the *system*. The novelty of version 0.96 compared to 0.95 is the ability to have an arbitrate number of inclusion levels for the hierarchy of composites and sub-components. References and services can be connected with so-called *wires*. In addition, the SCA specification defines the notion of a *binding* as the way to specify the communication technology (e.g. Web Services) associated with a service or a reference.

The SCA programming model for Java [13] defines the way SCA components, services, references and properties can be implemented with Java. Two styles may be used: either Java 5 annotations or unannotated POJOs (Plain Old Java Objects) with XML *component type* files.

Several implementations, commercial or open source, of the SCA specification are currently available. In the open source world, the leading platforms are the Apache Tuscany project[8] and the Eclipse STP project[9]. The specification is rapidly changing and is heading towards a fully-fledged component model.

4 Fractal WS

Fractal WS is a toolkit which aims at providing means to make compatible any Fractal component with the technology of Web services and reciprocally. On the one hand, any interface provided by a Fractal component could be transformed into a Web Service, thus making it accessible through web services' protocols. On the other hand, any (external) web service could be accessed inside an assembly of Fractal components and at any level of hierarchy, using a dedicated proxy component. In both cases, appropriate services or components are generated so that the bridge can be made operational. The toolkit is based on the Julia reference implementation in Java of the Fractal component model, and it also relies on the Apache Axis Java-based implementation of the SOAP specification, which provides mapping API and server support for web services. The toolkit is

[8] `http://incubator.apache.org/tuscany/`
[9] `http://www.eclipse.org/stp/`

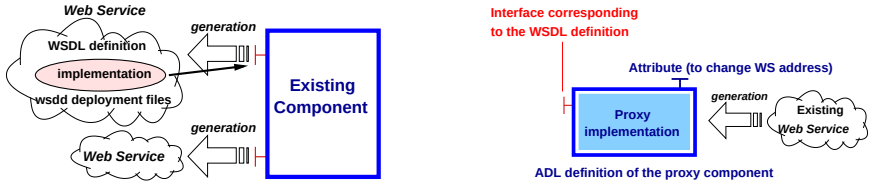
itself implemented with Fractal components. Moreover, deployment can be done in any servlet/jsp-compliant container, like the Apache Tomcat server.

4.1 Bridging Fractal Components and Web Services

Exposing Component as a Service Producing a service consists in deploying the Web Service that corresponds to part or all server interfaces of a Fractal component. Deployment is made on a Tomcat server bundled with Axis. Using the toolkit, one can provide either an ADL definition of a component or an already instantiated component, so that for each of its server interfaces the following files, packaged in an archive, are generated (see Figure 1a):

- the corresponding WSDL file;
- an implementation of this WSDL as a proxy rerouting all messages from the web service to the corresponding component interface;
- deployment descriptor files (deploy.wsdd and undeploy.wsdd).

Besides, the toolkit maintains a cache of all converted interfaces, so that a new WSDL file is generated for a component interface only if its definition is not already in the cache. This then enables several components that provide the same server interface to be substituted as the same implementation of a web service.



(a) From component to services. (b) From service to component.

Fig. 1: Main features of the Fractal WS toolkit.

Using a Service as a Component Making a Web Service accessible as a Fractal component must be done in two times. First, the proxy component and its implementation are generated from the WSDL document of the given Web service. Afterwards, this component can be used in a component architecture. This must be done as the provided interface of the proxy component, which is the component counterpart of the WSDL, is not known before by the rest of the architecture, and thus cannot be specified as a required interface of a component that needs the generated proxy.

More precisely, during the first stage, the toolkit takes the URI of the WSDL, from which it generates (see figure 1b) the following files, packaged in an archive :

- the stubs, data types, as well as component interfaces and type, from the WSDL.

- an implementation of the component type, as a proxy which implements the generated server interface and rerouts all the messages to the external Web service. It must be noted that currently the toolkit is able to generate proxy components implemented using either Axis client-side, or the kSOAP[10] lightweight SOAP implementation. The usage of the two implementations is illustrated in the next paragraph.
- an *attribute-controller* interface that allows one to dynamically change the location of a Web service by changing the corresponding attribute of the proxy component.
- some utility classes to be able to instantiate the proxy component through the Fractal API and the ADL definition of the resulting component, so that its integration in any system can be done easier.

In the same way as for service generation, a cache is also maintained so that a WSDL file is only converted if its URI is not already present.

4.2 Application to a Dynamic Communities System

The Fractal WS system has been validated on a client-server distributed system, Amui [11] which is mainly based on a messaging server that automatically and dynamically groups users according to their common interests. Functionally, an user equipped with the Amui client application connects to the Amui server and gives some authentication information (login, password) as well as some keywords that describe its interests. The server then automatically finds the groups whose topics match the user's keywords, and it automatically adds the user into the matched group. Groups are first associated with a chat room and currently once grouped, users are assigned in the same chat room, and they receive various advertisements according to their group topics. However, different plugin-like applications can still be integrated, for example to stream videos or simply to call other applications on all clients. This application which is already operational is based on the JiveSoftware Wildfire[12] server, an enterprise instant messaging server that uses the XMPP protocol, and which open architecture enables us to add new features architectured using components and services.

Architecture We only focus here on some parts of the overall system that are relevant to our purpose. The server, shown on figure 2, is represented by the composite component `AmuiServer`, which is formed out of three subcomponents: `AmuiFacade` pilots the `Core` and a proxy component to an advertisement web service is plugged into the `AmuiServer` component latter. This `AdvertProxy` component is actually generated using the Fractal WS toolkit. The `Core` is piloted through three interfaces: one to match users' keywords to groups' topics, one to manage users and the last one to manage groups. Inside `Core`, each interface

[10] `http://ksoap.objectweb.org/`
[11] *Amui* means *to gather* in Tahitian.
[12] `http://www.jivesoftware.org/wildfire/`

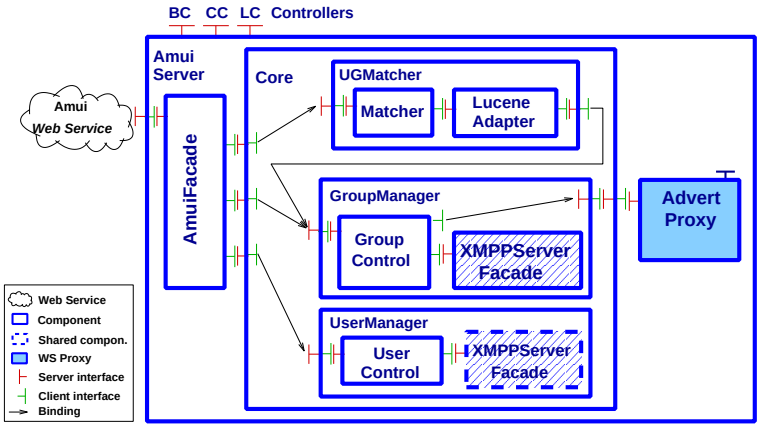


Fig. 2: Architecture of the server.

is bound to a subcomponent. `UGMatcher` implements the main functionality by using the Lucene[13] library encapsulated in a utility component (`LuceneAdapter`). `GroupManager` simply controls group creation and administration, mainly through a facade component to the XMPP underlying server (Wildfire in our case). This enables the architecture to be reused on other XMPP server implementations. `UserManager` controls functionalities related to users in the same way, and the `XMPPServerFacade` component is then *shared* inside this component, using the appropriate mechanisms of the Fractal platform.

Assessment In the server part shown in figure 2, both features of the toolkit are used. The component proxy `AdvertProxy` is generated from a given advertisement Web Service in order to use that Web Service inside the server assembly (feature 1). Moreover, the server itself which is represented by the top-level component `AmuiServer` exposes its unique provided interface as a Web Service in order to be accessed by various clients using the Web Services standard protocols (feature 2).

The web service exposed by `AmuiServer` is actually reused by the client application of the Amui system. Figure 3 shows a very simplified view of the architecture of the client. In particular, the `AmuiClient` component contains the `AmuiServerProxy` component which is generated again by the Fractal WS toolkit from the WSDL describing the web service exposed by `AmuiServer`. `AmuiServerProxy` then encapsulates all the logic to connect to the `AmuiServer` web service, and as the client application may be deployed on different platforms, it is possible to use Fractal WS to generate two different implementations of this proxy (one based on Axis client-side for client application deployed on Desktop PCs, and another one based on the lighter implementation kSOAP for client application deployed on Pocket PCs) and replace the `AmuiServerProxy` component at deployment time.

[13] Lucene is a full-featured text search engine library (`http://lucene.apache.org/`).

Using the toolkit, it is thus possible to define a web service from a component at any level of their hierarchy. It must be noted that the main differences of CBSE and SOA determined in section 2 are not less marked. The interface of the concerned component must preferably be coarse-grained and stateless to become a *valid* service. Finally, the use of other web services as components also appears as very useful, especially when the other services is not at the same business level. This simply shows that the hierarchical decomposition turns to be also appropriate at the coarse-grained level of services. The Amui system

Fig. 3: Simplified view of the client architecture.

also demonstrates the power of using a hierarchical structure of components along with services: a facade component on the underlying messaging server is used to access it and is also shared at different levels of hierarchy, some APIs are simplified and accessed through a component wrapper, such as the Lucene search library.

5 Fractal SCA

The previous section presented a solution for exposing Fractal components as Web Services and reciprocally. This solution relies on the structural information provided by Fractal components and Web Services to generate the glue code which allows connecting both worlds. In contrast, the second solution presented in this section proposes communication bridges between both world. Yet, we will see that both solutions are close and complementary.

The main goal of the Fractal SCA experiment is to bridge component-based applications written in the Fractal and SCA technologies. The bridge is bidirectional to allow communications to and from Fractal and SCA. The goal is then to interconnect applications written with heterogeneous component technologies. The interoperability is based on the SOAP communication protocol. The remainder of this section describes the extension which has been introduced in Fractal to achieve interoperability with SOAP. This experiment is based on the Tuscany project and on the AOKell [14] implementation of the Fractal component model.

Communications from Fractal to SCA In terms of communication from Fractal to SCA, we set up a solution which relies on the Factory design pattern for creating SOAP bindings. This solution takes advantage of the extensibility of the Fractal component framework. As explained in section 3.1, the control part of a Fractal component can be customized to accommodate different non-functional services. Besides the standard binding controller which creates local bindings for

components located in the same memory address space, we set up a so-called `SOAPBindingController` with the ability to create remote bindings based on the SOAP protocol.

As illustrated in figure 4, this controller generates dynamic proxies to bind a Fractal component to a Web Service entry point exported by a SCA module. Taking the URI of the WSDL as input, the `SOAPBindingController` uses the Java reflection API to generate a proxy instance for the specified Web Service. This proxy instance uses an invocation handler which generates the needed SOAP requests.

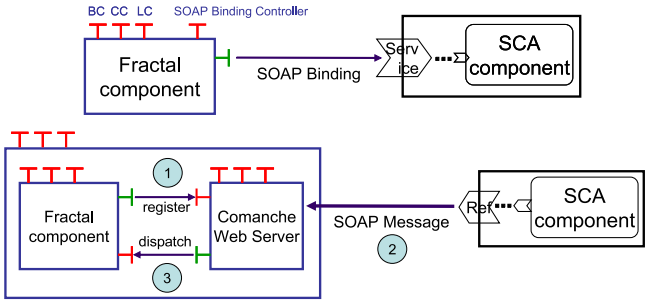


Fig. 4: Main features of the Fractal SCA toolkit.

Communications from SCA to Fractal A SOAP communication service has been defined and implemented for handling communications from SCA to Fractal. This service is generic in the sense that it can receive SOAP requests of any type. Fractal components which will act as servers for SOAP requests must first be registered with the SOAP communication service. The registration takes as input the URI of the SOAP invocation. The communication service performs the dispatching of an incoming request based on the registered URI.

The SOAP communication service is implemented as an assembly of Fractal components. This assembly reuses the existing Comanche web server which is a lightweight and minimal web server written in Fractal. The original version of Comanche serves HTTP GET requests for static documents. A component for handling SOAP requests has been implemented and inserted in the architecture of Comanche.

Assessment While the SCA technology has been used in this experiment, nothing is really specific to the component-oriented functionalities of SCA. We simply rely on the fact that SOAP is the preferred solutions for handling external SCA communications. The solution is then compatible with other Web Service platforms.

Compared to the Fractal WS solution presented in section 4, the Fractal SCA solution differs mainly by the means used to achieve the bridging: Fractal WS uses generative programming techniques to provide statically typed proxy

components, whereas the proxies are dynamically typed with Fractal SCA. A comparison can be drawn with the stub/skeleton mechanism of CORBA middleware platform. Fractal SCA implements a solution which is similar to generic stubs/skeletons, whereas the stubs/skeletons are statically generated with Fractal WS. Static stubs/skeletons are more effective but need to be generated for every new interface, whereas generic stubs/skeletons fit any type of interfaces at the cost of some reduced performances.

6 Conclusion

CBSE and SOA are among today's most prominent software architecture approaches. In this paper, we investigated their graceful integration on the basis of two experiments that were conducted with the Fractal component model. The first integration prototype, Fractal WS, is a toolkit to make compatible any Fractal component with the technology of Web services. Using generative programming techniques, any interface provided by a component can be transformed into a Web service and any Web service can be seen as a Fractal *proxy* component. This toolkit is currently used on a messaging server that automatically put users together into groups according to common interests. The other experiment, Fractal SCA proposes a bidirectional bridge between Fractal and the *Service Component Architecture* (SCA). From Fractal to SCA, components are enhanced so that they are able to create SOAP bindings dynamically and connect them to a Fractal component. On the other hand, a SOAP communication service is provided to handle communications from SCA to Fractal.

The two approaches turn out to be complimentary as they respectively provides static and dynamic forms of bridges. Moreover, the vision that underlies the two experiments is that of large complex distributed systems, possibly spanning multiple administrative domains, engineered in terms of hierarchical components with some of these components *exposed* as services. In this vision, those hierarchical components are preferably reflective components, so as to exhibit sound configuration, deployment and management support, including non-functional contracts. At arbitrary hierarchical levels, some of them are exposed as services, so to provide support for lesser coupled orchestrations with standard workflow languages.

Consequently, integrating CBSE and SOA still need more work and experiments. France Telecom will experiment shortly the Fractal WS bridge in its R&D works in the domain of grid computing. The Fractal WS toolkit itself is going to be extended with orchestration capabilities and used in combination with other Fractal extensions, such as the general contracting system ConFract [15].

Some of these prototypes will also be used in the ongoing research projects the authors are involved in: the RNTL FAROS project, in which a *Model-Driven Engineering* approach is followed to provide a general model for contracting services in platforms mixing SOA and CBSE, and the RNTL SCOrWare project, in which the technical services offered by a service oriented platform such as SCA will be extended with component-based services developed as Fractal assemblies.

Acknowledgements. The authors wish to thank the members of the RNTL FAROS project for discussions on components and services, and Moncef Ghaoui and Ilya Naraghi for their contribution to the implementation of both the Amui system and the Fractal WS toolkit.

References

1. Clarke, M., Blair, G., Coulson, G., Parlavantzas, N.: An Efficient Component Model for the Construction of Adaptive Middleware. In: Proceedings of the IFIP/ACM Middleware Conference. (2001)
2. Bruneton, E., Coupaye, T., Stefani, J.: Recursive and Dynamic Software Composition with Sharing. In: Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP '02). (2002)
3. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The Fractal Component Model and its Support in Java. Software Practice & Experience (SPE) **36** (2006) 1257–1284
4. Coupaye, T., Stefani, J.B.: Fractal Component-Based Software Engineering. In Consel, C., Sudholt, M., eds.: ECOOP'06 WS Reader. Volume 4379 of LNCS., Springer (2007) To appear.
5. Wang, G., Fung, C.K.: Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems. In: 37th Hawaii International Conference on System Sciences (HICSS). (2004)
6. Beisiegel, M., al.: Service Component Architecture - Building Systems using a Service Oriented Architecture. A Joint Whitepaper by BEA, IBM, Interface21, IONA, SAP, Siebel, Sybase (2005)
7. Rivierre, N., Coupaye, T., Bruneton, E., Andrey, L.: Fractal JMX. http://fractal.objectweb.org/fractaljmx/ (2005)
8. W3C: SOAP Version 1.2. W3C Note (2003) http://www.w3.org/TR/soap/.
9. W3C: Web Services Description Language (WSDL) 1.1. W3C Note (2001) http://www.w3.org/TR/wsdl.
10. UDDI.org OASIS TC: Universal Description, Discovery and Integration. OASIS TC Draft (2004) http://www.uddi.org/specification.html.
11. W3C: Web Services Activity. web (2006) http://www.w3.org/2002/ws/arch/.
12. OSOA: SCA Service Component Architecture - Assembly Model Specification. (2006) Version 0.96 draft 1.
13. OSOA: SCA Service Component Architecture - Client and Implementation Model Specification for Java. (2006) Version 0.95.
14. Seinturier, L., Pessemier, N., Duchien, L., Coupaye, T.: A Component Model Engineered with Components and Aspects. In: Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering (CBSE'06). Volume 4063 of Lecture Notes in Computer Science., Springer (2006) 139–153
15. Collet, P., Ozanne, A., Rivierre, N.: Towards a versatile contract model to organize behavioral specifications. In: 33rd International Conference on Current Trends in Theory and Practice of Computer Science SOFSEM 07, Harrachov, Czech Republic. (2007)