Power Combination: SCA, OSGi and Spring

Introductory Whitepaper

Table of Contents

1 Introduction

In the enterprise application world, lightweight containers such as Spring and OSGi have emerged in recent times and have gained rapidly in popularity amongst developers. Service Component Architecture (SCA) has gained a high momentum and it seems likely to become a significant standard for the development of enterprise applications using a Service Oriented Architecture (SOA).

This white paper examines:

- Why SCA, OSGi and Spring should be considered as complementary technologies.
- Different possibilities and ways to combine SCA with OSGi and Spring.
- The advantages of the combination of SCA, OSGi and Spring.
- Ongoing activities that are intended to make the combination a reality.

2 Short Overview of SCA, OSGi and Spring

2.1.1 SCA

Service Component Architecture (SCA) [1] is a set of specifications which describe a model for building applications and systems using a Service Oriented Architecture. SCA models solutions as sets of ***service components*** offering services and making references to services supplied by others, which are combined together by ***composites*** which wire references to services and which declaratively apply bindings for communication methods and also apply policies for aspects such as security and transactions. SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as Web services.

2.1.2 OSGi

OSGi technology [2] provides a service-oriented, component-based environment for developers, primarily on the Java platform. It provides a dependency resolution mechanism, with version support and also offers standardized ways to manage the software lifecycle. OSGi is a particular boon when using different components that use different versions of some shared package. These capabilities greatly increase the value of a wide range of computers and devices that use the Java™ platform.

2.1.3 Spring

The Spring Framework [3] is a popular Java/JEE application framework. It offers a model where applications are built as collections of simple Java Beans and reduces the need for the use of complex APIs through the use of dependency injection and aspect technology. It delivers significant benefits for many projects, simplifying development tasks, increasing development productivity and runtime performance while improving test coverage and application quality.

3 Combining technologies

3.1 Combining OSGi with Spring

The Spring-OSGi project [4] makes it easy to build Spring applications that run in an OSGi framework. A Spring application written in this way provides better separation of modules, the ability to dynamically add, remove, and update modules in a running system, the ability to deploy multiple versions of a module simultaneously and have clients automatically bind to the appropriate one, and a dynamic service model.

The Declarative Services Specification (DS) is part of the OSGi R4 Service Compendium [5]. DS prescribes a service component model which uses a declarative model for publishing, finding and binding to OSGi services. This model simplifies the task of authoring OSGi services by performing the work of registering the service and handling service dependencies. This minimizes the amount of code a programmer has to write.

The DS as currently specified has some weaknesses. To make the DS suitable for the enterprise market, the functionality has to be extended, e.g.:

- Injection of object (not only OSGi service) references
- Configuration of interceptors (e.g. registration of intercepted OSGi services) has to be supported
- A stateful OSGi service must not be disposed if a required dependency goes down

- OSGi services are local to a single Java VM, whereas enterprise applications are likely to span multiple machines on a network

An enhanced DS spec that can be implemented with Spring is envisaged. This topic is being discussed in the Enterprise OSGi Expert group [6].

3.2 Combining SCA with Spring

The Spring Framework and SCA share many design principles and SCA views Spring as a partner which can be used as an implementation technology for components and composites. There is a SCA Spring Component Implementation Specification [7] which defines how Spring is used in this way.

Similar to a Spring Bean, a SCA component can contain references to services supplied by other components and it can have configurable properties. In contrast to Spring, SCA is a cross-language, distributed component architecture supporting multiple communication mechanisms between components. SCA may be used to extend the capabilities of Spring components by publishing Spring beans as services accessed by other remote components as well as by providing Spring beans with service references wired to services of other, potentially remote, components. SCA can add useful capabilities to an application implemented using Spring, for example:

- extended support for remote components and multiple protocols
- support for components written in a variety of programming languages, beyond just those supported on the JVM
- extended support for asynchronous programming through the SCA programming model
- support for WS-Policy specified policies for capabilities such as security and transactions

One great feature of Spring is the ease of testing components. The lack of APIs and the injection technique enable testing to be done with simple mock objects. SCA complements this in the services arena since the SCA composition surrounding a service component can be easily switched to a mock configuration for test purposes.

The SCA Spring Component Implementation specification [7] describes two ways in which a component could use Spring for an implementation:

- Use of a complete Spring application context to implement a composite component within an SCA assembly
- Explicit declaration of SCA related beans inside a Spring configuration

In the first approach, a standard Spring configuration file is used, declaring a set of Beans and Bean references. The SCA composite file uses the whole Spring ***application context*** as an implementation of a component, with services and references of the Spring application wired to other (remote) components by the SCA composite. The Spring specific part here is the Spring implementation type element in the SCA composite file. The Spring application context is pure Spring.

In the second approach, Spring configuration files are used to define the wiring within the composite and SCA-specific elements are added to the Spring configuration file to declare the services, properties and references of the SCA composite defined by the Spring application.

In both approaches, the Spring application context defines the internal structure of a component implementation. So the existing Spring programming model is largely unchanged, but benefits from being able to play in a wider distributed context, and benefits from the integration that SCA offers.

So, for example, Spring applications written in Java could be linked with components written in BPEL (eg for process definition), in PHP (eg for handling of Web front-ends), C++ and COBOL (legacy applications), using a wide variety of communication methods including Web services and JMS.

3.3 Combining SCA with OSGi

SCA is designed for distributed and heterogeneous systems, whereas OSGi was originally designed for services running in a single JVM on mobile or embedded systems. Now OSGi is also being used for enterprise applications.

Two of the main requirements for enterprise OSGi are distributed computing support (e.g. multi JVM/multi-process) and multi-language compatibility. Although the OSGi specification is not bound to a certain programming language Java is currently the preferred (and most useful) implementation choice, while SCA is targeting a multi-language distributed environment.

OSGi can play several roles with SCA:

- A SCA ***OSGi Binding*** can enable interworking between SCA components and OSGi services
- An SCA ***OSGi implementation type*** allows deploying existing OSGi applications/bundles in an SCA domain and using them as SCA components
- OSGi can be used as underlying technology for an SCA container providing an extension mechanism, dependency resolution and service registry capabilities

The different options will be explained in the following chapters.

3.3.1 SCA OSGi Binding

An SCA component can access an OSGi service running outside of the SCA domain via a reference configured with an OSGi Binding.

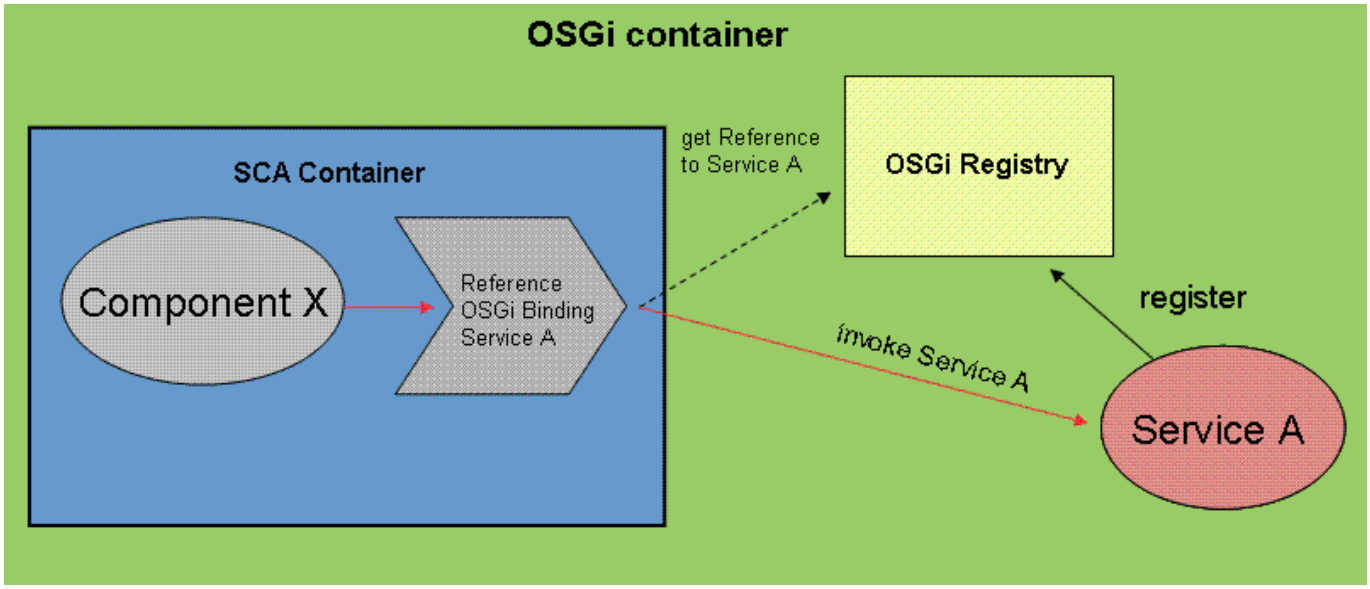

Figure 1: Example OSGi Binding

Service A is an OSGi service registered in the OSGi registry. The SCA container retrieves a reference to Service A from the OSGi registry and injects this reference into the SCA Component X.

Equally, an SCA component which has a service can be exposed as OSGi Service via a service configured with an OSGi Binding.

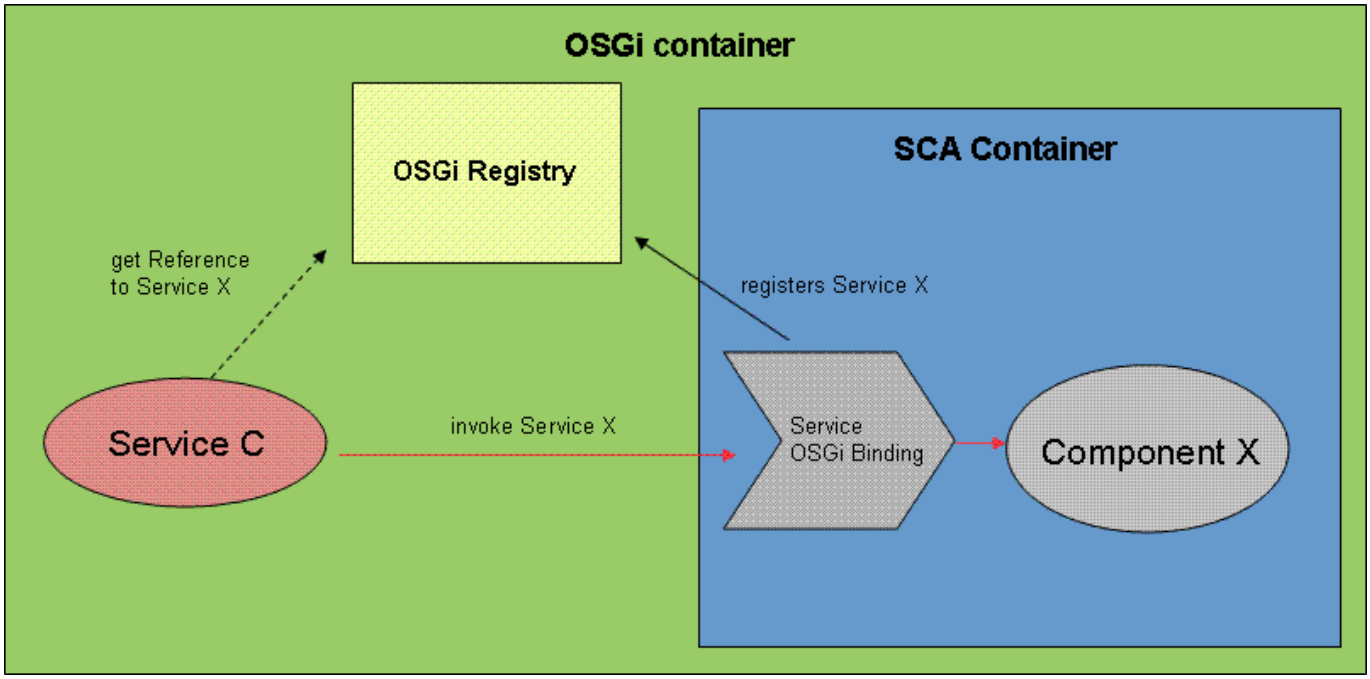

Figure 2: Service OSGi Binding

SCA Service X with an OSGi binding is registered in OSGi by the SCA container. The OSGi Service C is able to find (lookup) Service X in the OSGi registry.

The OSGi binding enables the communication between SCA components and OSGi services inside an OSGi container, but it does not support distributed inter-container communication over VM or runtime node boundaries.

The right hand part of figure 3 shows an OSGi container which hosts a SCA container. SCA component Y uses an OSGi binding for exposing Service Y as well as using the OSGi binding for a reference to Service B. Via the declared service and reference the SCA component Y is able to communicate with the OSGi services A and B both running in the same OSGi container. The left part of the figure shows a remote SCA container hosting components X and Z, perhaps written in a non-Java language such as BPEL. Component X wants to access a pure OSGi service A running in the OSGi container, via a reference and component Z offers a service, in this case using an RMI binding.

Services A and B are native OSGi services – not running within an SCA container. As a result, they cannot profit from the capabilities of SCA. For these components, additional connectors like an OSGi SOAP bundle and an RMI adapter are required to make Service A accessible from outside the container and to allows for the communication between OSGi service B and SCA service Z. A more complete integration of SCA and OSGi would aim to make SCA bindings available to OSGi services in a standard way and eliminate the need for specialized OSGi bundles such as the SOAP bundle and RMI adapter shown here.

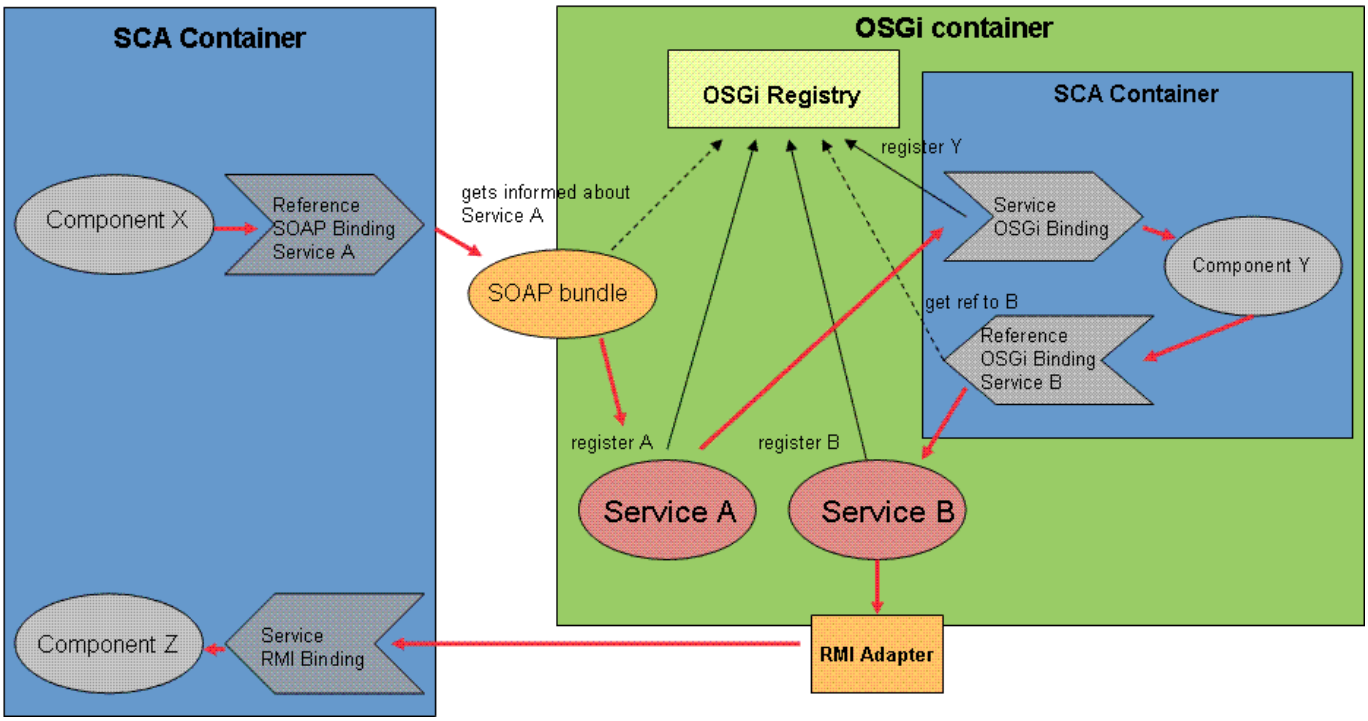

Figure 3: Example Combining SCA Components with OSGi Services

To be able to benefit from the capabilities of SCA the OSGi services (in this case A and B) would have to be provided as SCA components.

3.3.2 OSGi Host

The previous examples show a SCA container running inside an OSGi container - the OSGi container hosts the SCA container. The SCA container itself would consist of a set of OSGi bundles and can provide different implementation types, e.g. for EJBs, BPEL and Java POJO. In other words, OSGi can be used by the SCA container as the mechanism for extending the SCA container and for adding new implementation types and binding types. In addition, Native OSGi bundles can be deployed parallel to the SCA container, and OSGi capabilities such as the Registry can be made available, as shown in the following figure:

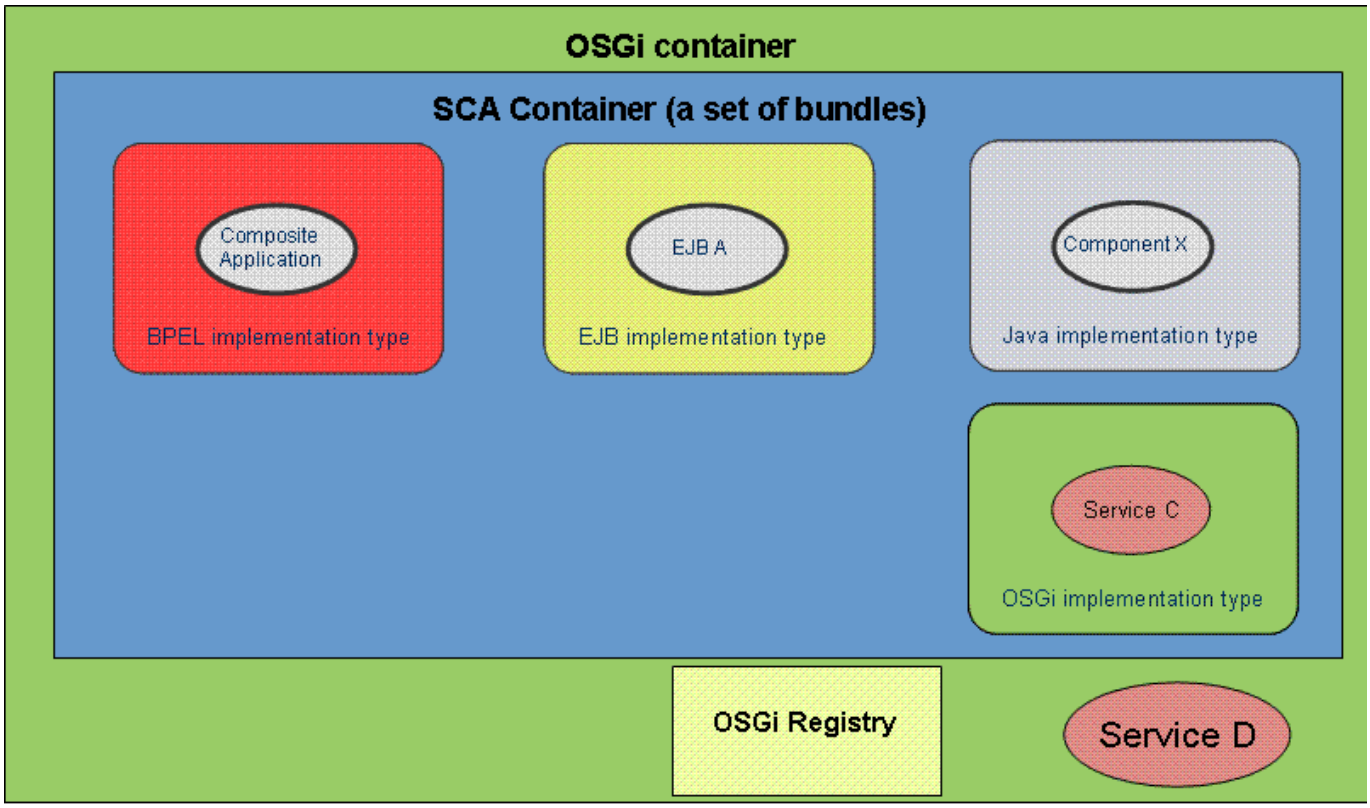

Figure 4: Java based SCA Container hosted in an OSGi Container

3.3.3 OSGi implementation type

The SCA container provides the possibility to deploy existing OSGi applications/bundles as SCA component implementations like the service C shown in figure 4. The service C can benefit from the capabilities of SCA including access to other services via SCA references and it can expose its functionality as a (remote) service via different protocols, with SCA application of policies.

This can be achieved by an SCA ***OSGi implementation type***. Please note that Figure 4 shows only a logical view of the container. Technically it is possible to deploy the service C as separate OSGi bundle which might contain a composite file. The SCA container reads this file and provides the wiring for services and references.

An implementation of an SCA OSGi binding as well as an OSGi Host is under development in the Apache Tuscany SCA Open Source project [8].

3.3.4 Advantages of combining SCA with OSGi

SCA and OSGi are complementary.

SCA provides a definition for components with services and references. It describes a separation between service implementation and service usage and promotes a model where complex detail about communication methods is kept out of the business code within the implementation. SCA describes service integration through composites and provides a declarative way to apply policies for infrastructure capabilities such as security. SCA supports a distributed environment potentially using a heterogeneous mixture of component technologies.

The OSGi container is able to address a wide range of systems – from embed devices up to enterprise servers. Dependency handling, multi-version support and remote management are integral parts of OSGi. Software components and services can be installed, updated and removed on the fly.

The combination of SCA and OSGi provides the best of both worlds including:

- A common Service Component model
- A standardized way to integrate a variety of communication mechanisms in a runtime container
- An OSGi based SCA container can run on a wide range of system and has a clear and simple extension mechanism for supporting new implementation types and new communication bindings
- Integration of different implementation types allows the use of the technology best suited to the job

The following figure shows the integration of a variety of communication mechanisms in an combined OSGi/SCA container. For every supported transport/serialization protocol an SCA binding is provided, e.g. JMS binding, SOAP binding, CORBA binding.

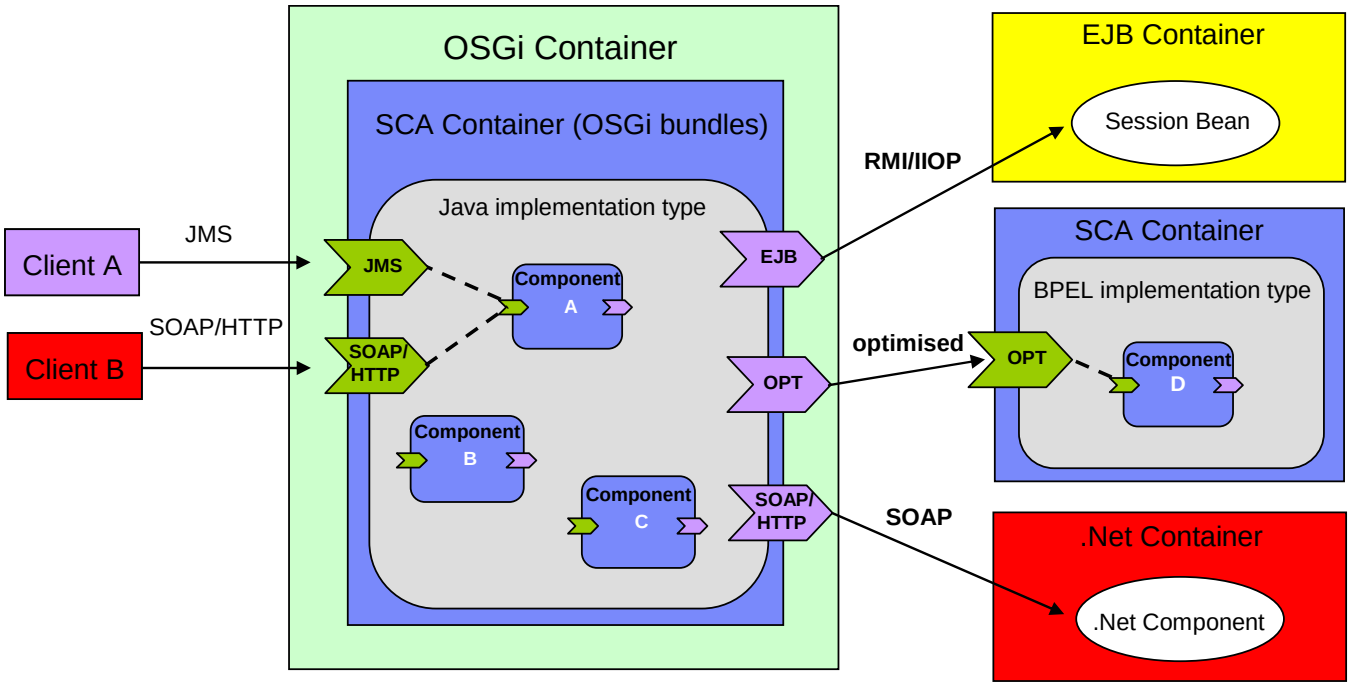

Figure 5: Combined OSGi / SCA Container as part of wider SCA Domain

The Application developer implements application components the most suitable programming language for the business task in hand, e.g. in Java for complex data manipulation or in BPEL for describing a business process. Services and references are configured with bindings and endpoints in the SCA composite file. The SCA container provides the required bindings for the references and services. Via the bindings the SCA components access services running in other (remote) containers, e.g. in an EJB Container or .NET container. A performance optimized protocol might be used for the communication between SCA containers.

SCA brings distributed computing support into OSGi. It allows OSGi applications to integrate in heterogeneous environments, e.g. with .NET and EJB applications.

4 Combining SCA with OSGi and Spring

The preceding section describes the advantages of combining SCA with OSGi. This section explains the advantages of an additional integration of these technologies with Spring.

A way of providing the combination is to take the SCA/OSGi container shown in figure 5 and add in support for the SCA Spring implementation type described in the section Combining SCA with Spring. This combined container allows for the creation of components as Spring application contexts, which is a great way of creating a service implementation from simple Java Beans.

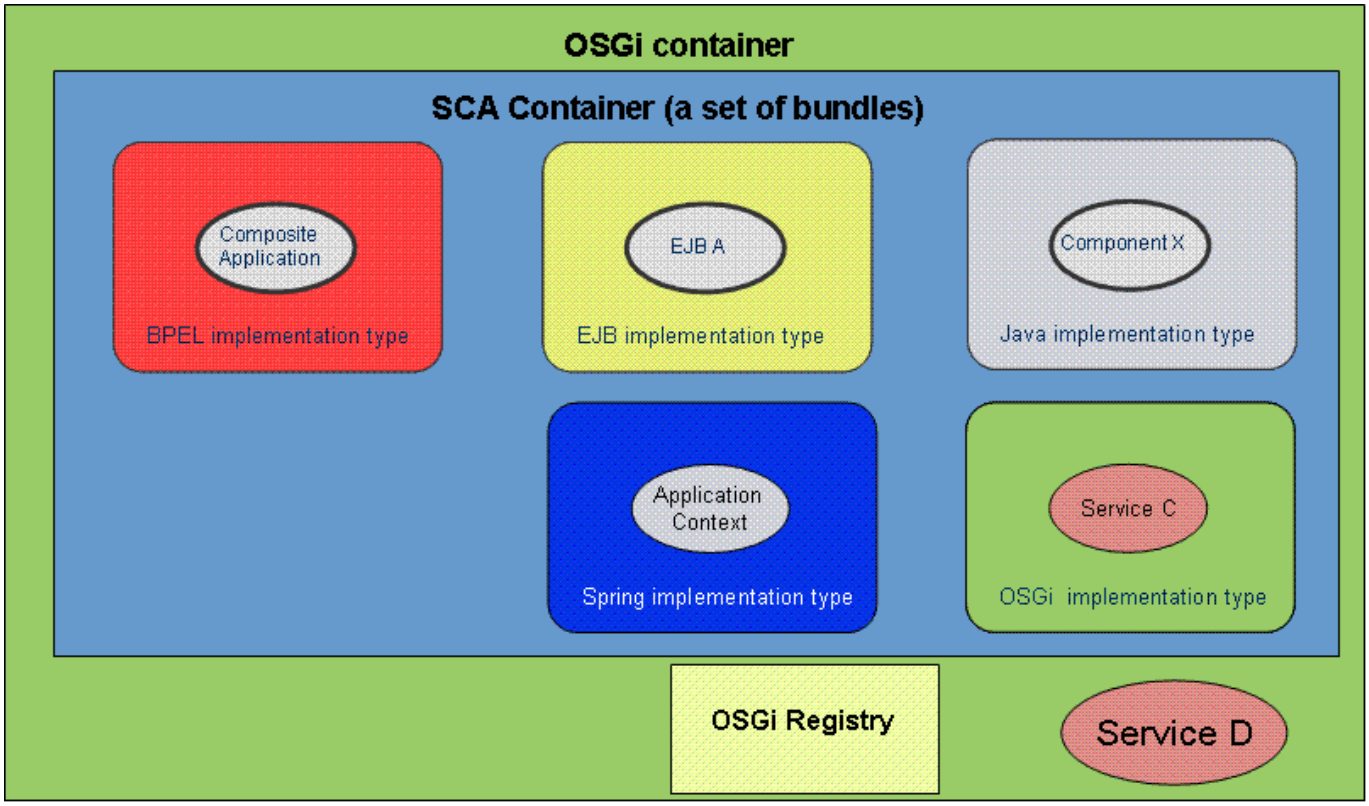

Figure 6: SCA container with implementation type Spring embedded in OSGi container

Figure 7 shows an SCA container with three different implementation types:

Spring implementation type:
Spring is used inside the Spring implementation type and the component is built as a Spring application context. Injection is used to supply beans which represent references and properties of the implementation. SCA is used to provide the wiring and other configuration of the services and references.

OSGi implementation type:
As explained previously, Spring can also be used as the implementation technology for an enhanced OSGi Declarative Services (DS). The DS is responsible for the dependency management inside the OSGi implementation type. For the distributed communication via services and references, SCA is used.

Java implementation type:
This is the pure POJO implementation type described in the SCA Java Component Implementation specification. SCA is used for the dependency management inside of the Java implementation type as well as for the services and references to the outside.

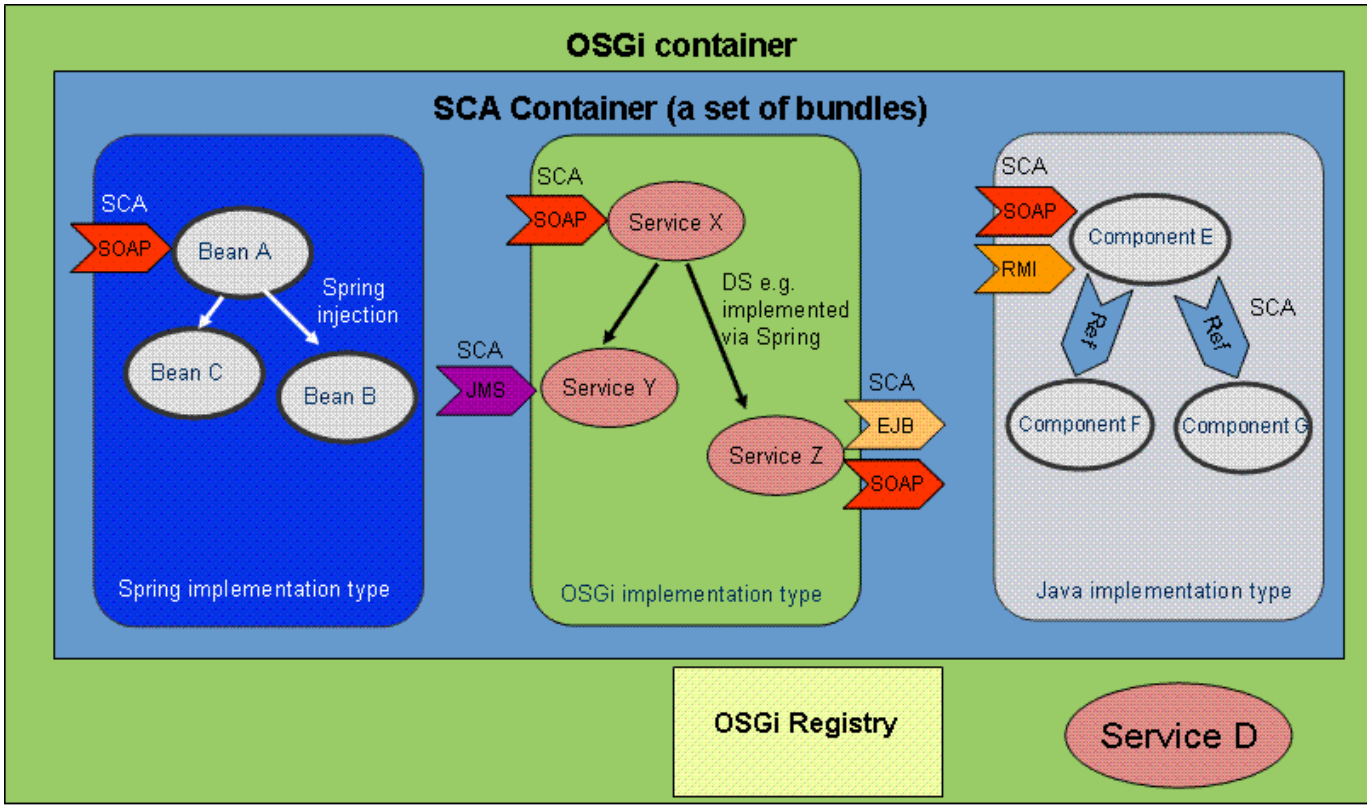

Figure 7: Usage of Spring for dependency management and SCA for distributed communication

Inside the OSGi implementation type the OSGi binding as described previously is not required, because the DS can be used for dependencies inside the OSGi implementation type as well as for dependencies to Services running natively in the OSGi container (e.g. Service D), while SCA references can be used for services accessed through any other communication methods and SCA services used to expose services offered by OSGi implementations. In the Java implementation type the OSGi Binding would still be required to get access to any native OSGi services such as Service D.

5 Summary

SCA, OSGi and Spring are all useful and powerful facilities for the Java programmer to use. In the new service-oriented world that we are entering, using SCA, OSGi and Spring ***together*** provide powerful capabilities for building service implementations from sets of simple Java Beans using few APIs, with managed dependencies, version control and dynamic update capabilities, allied to the capability to compose those implementations with other service components written in Java or in other languages and existing in a distributed network of systems using a range of communication methods.

Simplicity, flexibility, manageability, testability, reusability. A key combination for enterprise developers.

6 References

[1] OSOA Home Page
http://www.osoa.org/display/Main/Home

[2] OSGi Alliance
http://www.osgi.org/

[3] Spring Framework
http://www.springframework.org/

[4] Spring OSGi Project
http://www.springframework.org/osgi

[5] OSGi Specifications
http://osgi.org/osgi_technology/download_specs.asp?section=2

[6] OSGi EEG Charter
http://www.osgi.org/about/charter_eeg.asp?section=1

[7] Spring Client & Implementation Specification
http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

[8] Tuscany Project
http://incubator.apache.org/tuscany/