Quick Guide to SCA

A quick guide to SCA

Purpose of this guide is to help you understand the high level concepts in SCA as it relates to the specification. For more details on SCA please refer to the various specifications available at www.osoa.org.

What is SCA?

SCA is a standard programming model for abstracting business functions as components and using them as building blocks to assemble business solutions. An SCA component offers services and depends on functions that are called references. It also has an implementation associated it with it which is the business logic that can be implemented in any technology.

SCA provides a declarative way to describe how the services in an assembly interact with one another and what quality of services (security, transaction, etc) is applied to the interaction. Since service interaction and quality of service is declarative, solution developers remain focus on business logic and therefore development cycle is simplified and shortened. This also promotes the development of reusable services that can be used in different contexts. For example, a shopping cart service can be used in a retail application or a travel application without changing. Services can interact with one another synchronously or asynchronously and can be implemented in any technology.

SCA also brings flexibility to deployment. A solution assembled with SCA is deployed as a unit and can be distributed over one or more nodes in the network and can be reconfigured without programming changes.

Applications that adopt SCA programming model can interact with non-SCA applications. Meaning non-SCA application can call into SCA enabled applications and SCA enabled applications can call out into non-SCA enabled applications.

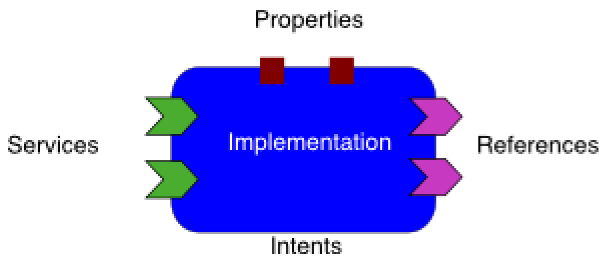
Now let's talk about SCA building blocks and concepts.

SCA Component

The basic building block for SCA is a component. It is the abstraction of a given business function. A component is described with the following attributes:

- **Service:** Describes the functions that this type of component provides. A component can offer one ore more services. A service is an interface.
- **Reference:** This describes the dependencies this type of component has in order to function. A reference is an interface.
- **Property:** This defines configuration parameters that can controls how the business function can behave. For example, what currency to use for an account component.
- **Intent policies:** This describes assumptions on how the component will behave. There are two types of policies.
  - Implementation policy- Impact the behavior of an implementation. For example, transaction, monitor and logging
  - Interaction policy - defines how the components behave with one another. For example, security.
- **Implementation:** Every component has some implementation associated with it. This can be a new business logic or an existing one that is now being used in the assembly. A business logic can handle different operations and some of which are exposed externally as callable services. Component implementation can be in any technology, for example for example BPEL for business processes or XSL-T for transformations or Ruby for scripting or pure Java. How the services, references, properties and intents are defined for an implementation is specific to that particular implementation type.

This is demonstrated below.



The implementation of a component can be in any language that is suitable for the user, for example BPEL for business processes or XSL-T for transformations or Ruby for scripting or pure Java. How the services, references, properties and intents are defined for an implementation is specific to that particular implementation type.

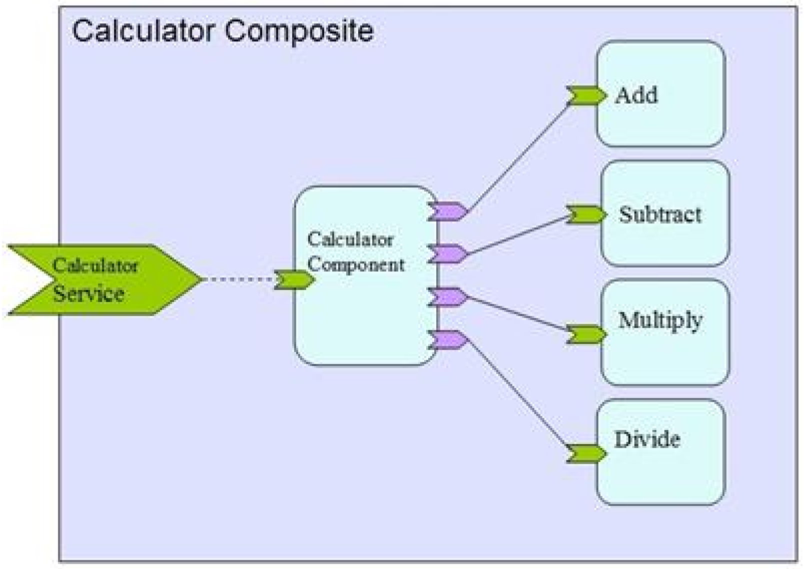
SCA Wire

As mentioned above, an SCA component may have zero or more references. Refrences in SCA define how SCA components invoke services. The relationship between a reference and a service is typically demonstrated through a line in a SCA diagrams and is referred to as a wire.

The term wire can at the begining seem confusing because you may ask how a wire is realized. There is no physical definition for a wire, it is really derived from the relationship between a Service and its refrence(s) at runtime. This is realized through dependency injection in Tuscany.

SCA Composite

An SCA composite consists of components, services, references, and wires that connect them. A composite is the unit of deployment for SCA. A composite can be viewed as a component whose implementation is not code but an aggregation of one or more components co-operating to provide higher level services. Think of composite as a solution, for example a credit check composite may consist of multiple components that together perform the credit checking work. A composite can also be used within a larger solution, for example credit check can be part of a order processing composite. A composite has the same charactersitics as a component. It provides Services, has References to other dependencies, and can be configured using Properties and can have intent policies in just the same way as an individual components can. In thise case, attributes of some of the components that are embedded in the composite get 'promoted' and becom the attribute of the composite. In the example below, you see a calculator composite which consists of 5 components, a calculator service has references to four components:Add, Subtract, Multiply and Divide.



The assembly or wiring is defined in .composite file through Service Component Definition Language (SCDL) that is in XML. For example, calculator.composite would define that calculator component references the other four components.

SCA Contribution

The artifacts that make up a solution get packaged into what is called a contribution. A contribution can take a number of different forms. For example, it could be a jar file, or it could be a directory tree on the file system. A contribution can contain composites, java classes, BPEL processes, XSD files, wsdl files, etc. An SCA application can be divided into multiple contributions with dependencies between them. In general, some services depend closely on other services and it makes sense to package them together. If services are more independent it is best to package them separately so that they can be reused in different contexts. A contribution is a deployable unit. A solution may require multiple contributions that share artifacts and artifacts can be shared between (imported) between contributions.

SCA Domain

Contribution packages get contributed to what is called SCA domain which is the scope of adminstration at runtime. An SCA Domain represents a complete runtime configuration, potentially distributed over a series of interconnected runtime nodes and is a logical view of the running applications or a coherent grouping of components that are working together. An SCA Domain typically represents an area of business functionality controlled by a single organization. For example, an SCA Domain may be the whole of a business, or it may be a department within a business.

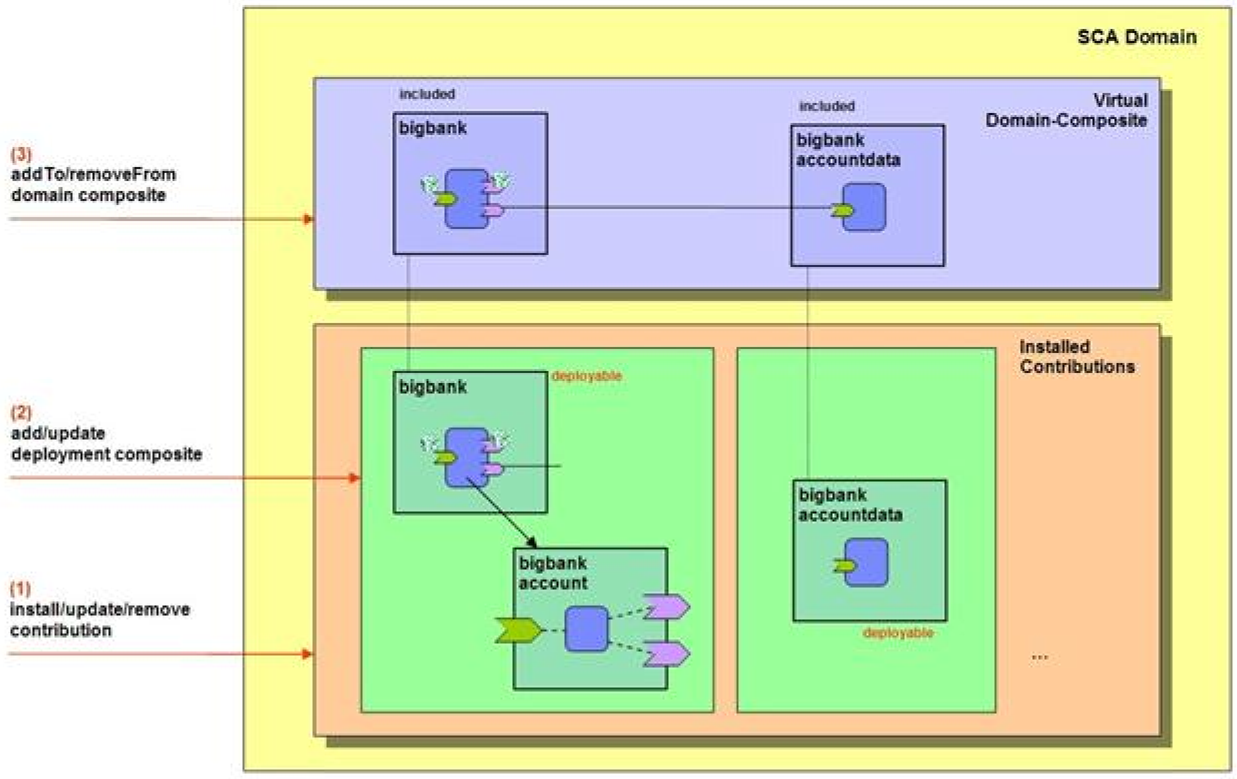
Therefore, an SCA domain consists of the definitions of composites, components, their implementations, **and the nodes** on which they run. Components deployed into a domain can directly wire to other components within the same domain. Communication with services outside of a domain is done through bindings.

SCA Domains can vary in size from the very small to the very large:

- a very small domain could be one within a test environment inside an IDE
- a medium sized domain could be a single server or small cluster supporting a single application
- a large domain could describe all the services within a department or company

In a large domain there may be all sorts of policies about where components can run and how they connect to each other or to external services. However, during development one is not concerned with all this. The code is packaged and made available for deployment. Tuscany SCA Java supports contributions in the form of JAR or filesystem.

Below is an example of domain with two contributions.



SCA Binding

A binding is used as a means of communication between services and handles the protocols. It defines with what communication method a service can be accessed with or with what communication method it can will access other services. There can be different types of bindings depending on technologies used to develop a solution. For example JMS binding, Webservices binding, Atom binding for web20 interaction, etc.

Services can be configured with different bindings and there can be multiple bindings for a service. Bindings for the services and references get defined declaratively in the .composite file. There is a default binding called binding.sca which when used leaves the choice of binding to the underlying infrastructure by default.

The declarative bindings and the abstraction of protocols from business logic brings agility to SCA applications. This allows SCA applications to be purely focused on business logic and not be contaminated with protocol handling information. It also enables the SCA compositions to grow or change without code modification while also working with applications that are not enabled with SCA.

SCA Policy

An enterprise application requires control beyond the business functional capability which can include how security is handled in the enterprise or what transactional capability should be applied to services that are offered. SCA policies define the constraints or capabilities that get applied to services and their interaction. Two types of policies are defined in SCA.

- Interaction Policies - Define the policies that influence interaction of services for example whether authentication is required or not.
- Implementation Policies - Define how the components behave at runtime for example whether it is transactional or not.

SCA mechanisms for defining policies such as intents and policySets can only be used in the context of a single domain. The fact that policies can be defined declaratively make applications adaptive to the environment that they get deployed into or to changes in the business requirements.
For more information about policy check out the sca policy framework specificationversion=1]