Architectures Orientées Services

Module

L'initiative SCA

Sommaire du module

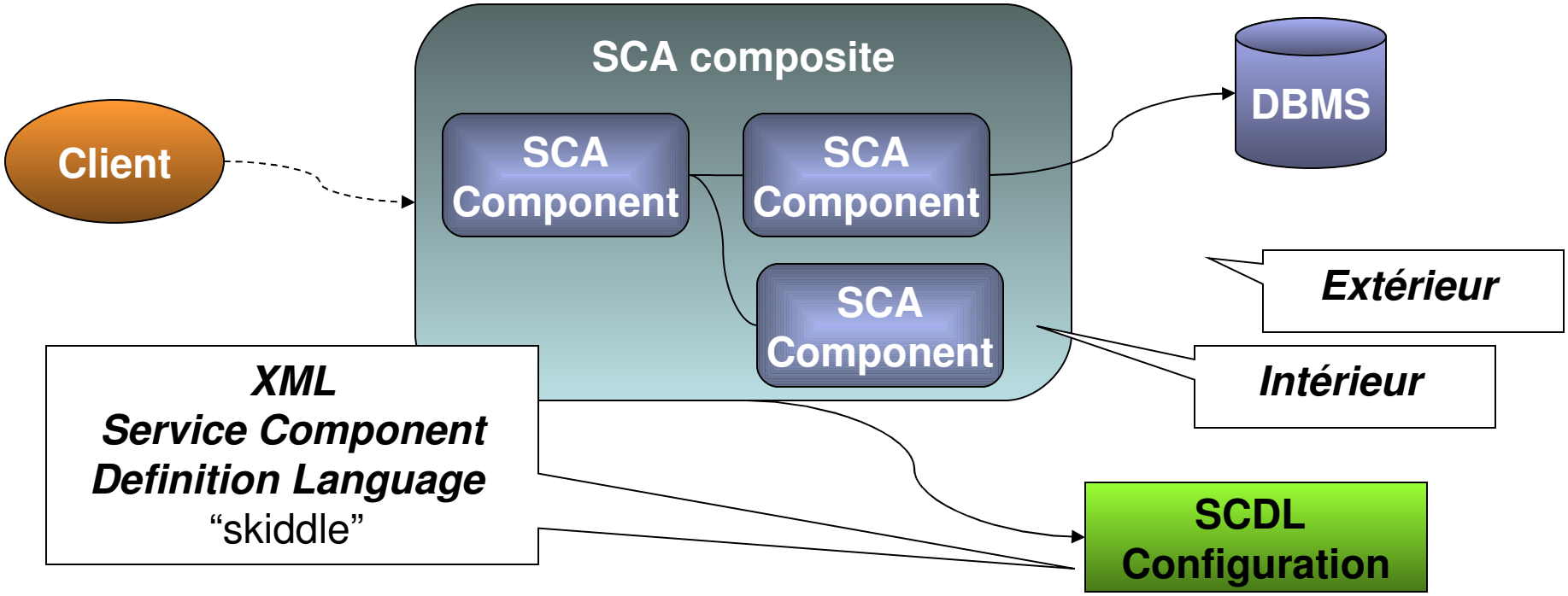
- L'initiative des composants standards d'architecture des services SCA
  - Principes
  - Modèles XML
  - Notion de domaines
  - Propriétés, références et services
  - Exemples d'implémentation
  - Notion de « runtime » SCA

Objectif de l'initiative Service Component Architecture

- L'initiative SCA vise à une redéfinition logique de ce que devrait être l'architecture d'application orientée service
- Qu'est-ce qu'une application
  - un ensemble de composants logiciels travaillant ensemble
  - fondés sur des technologies homogènes ou non
  - tournant sur un ou plusieurs systèmes d'exploitation
- Pour qu'une application s'organise deux choses sont nécessaires :
  - un moyen de créer des composants
  - un moyen de décrire comment ils travaillent ensemble
- L'objet de SCA est une approche générale pour répondre à ces deux nécessités
  - Elle encapsule les technologies existantes (BPEL, UDDI, WSDL, SOAP)
  - Elles étend son modèle à toute technologie capable de l'implémenter

SCA : Composants et Composites

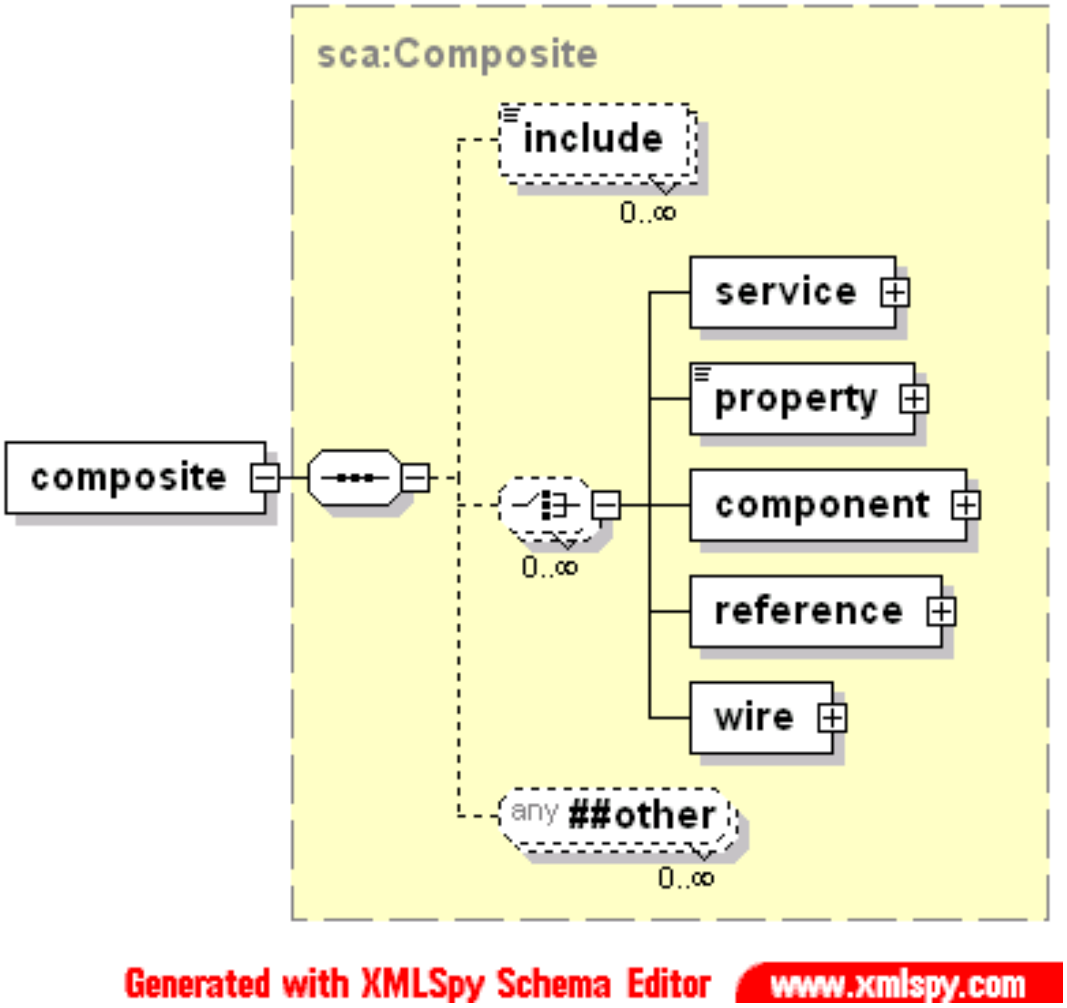
- Concepts de base
  - Intérieur et extérieur de domaine SCA
  - Composites et composants ,
  - Langage de description d'une configuration de composite



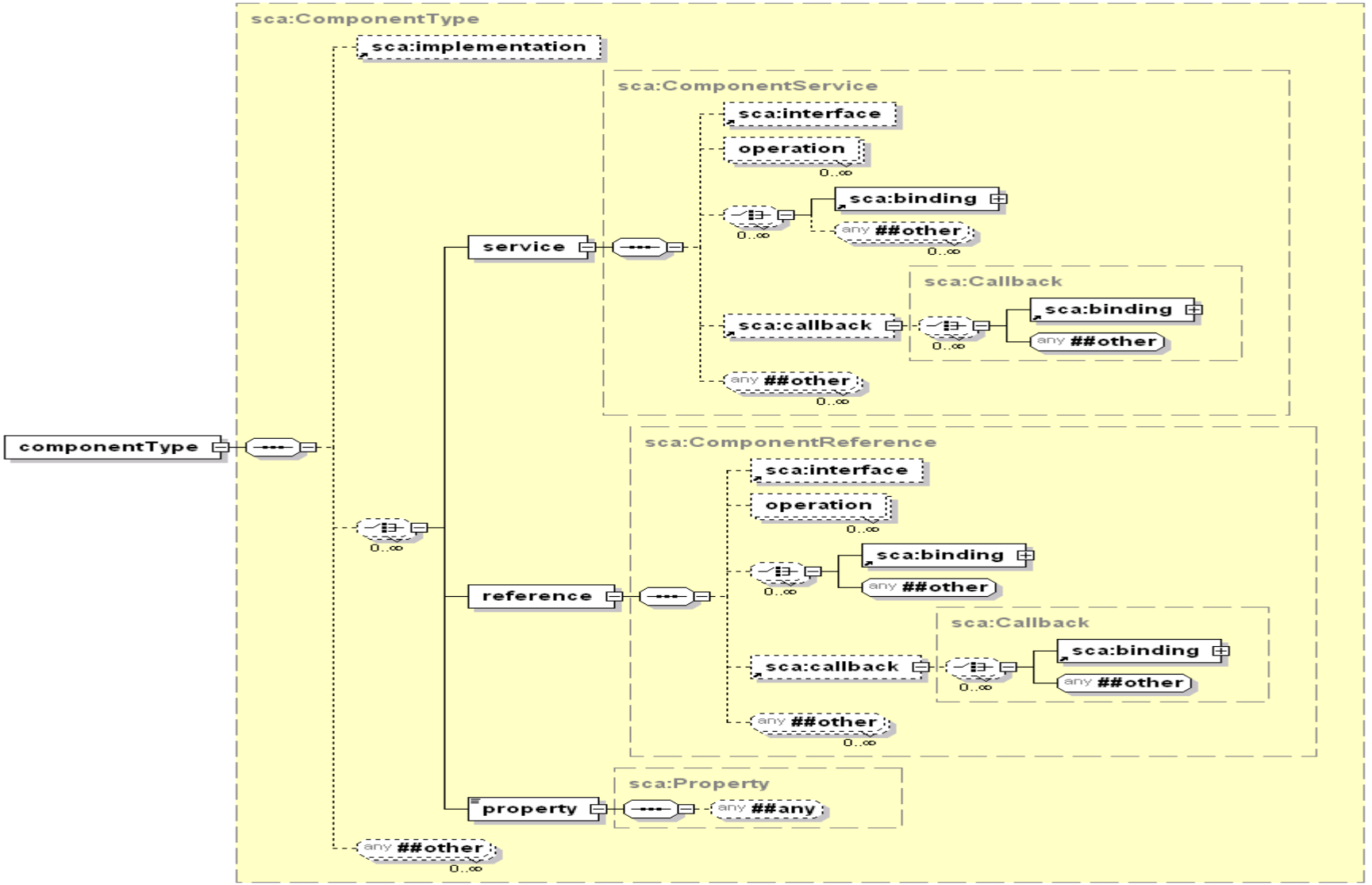
Syntaxe XML de Service Component Description SCDL

```
<composite name="ExempleComposite" ...>
     <component name="Composant1">
        ...
     </component>
     <component name="Composant2">
        ...
     </component>
     <component name="Composant3">
        ...
     </component>
</composite>
```

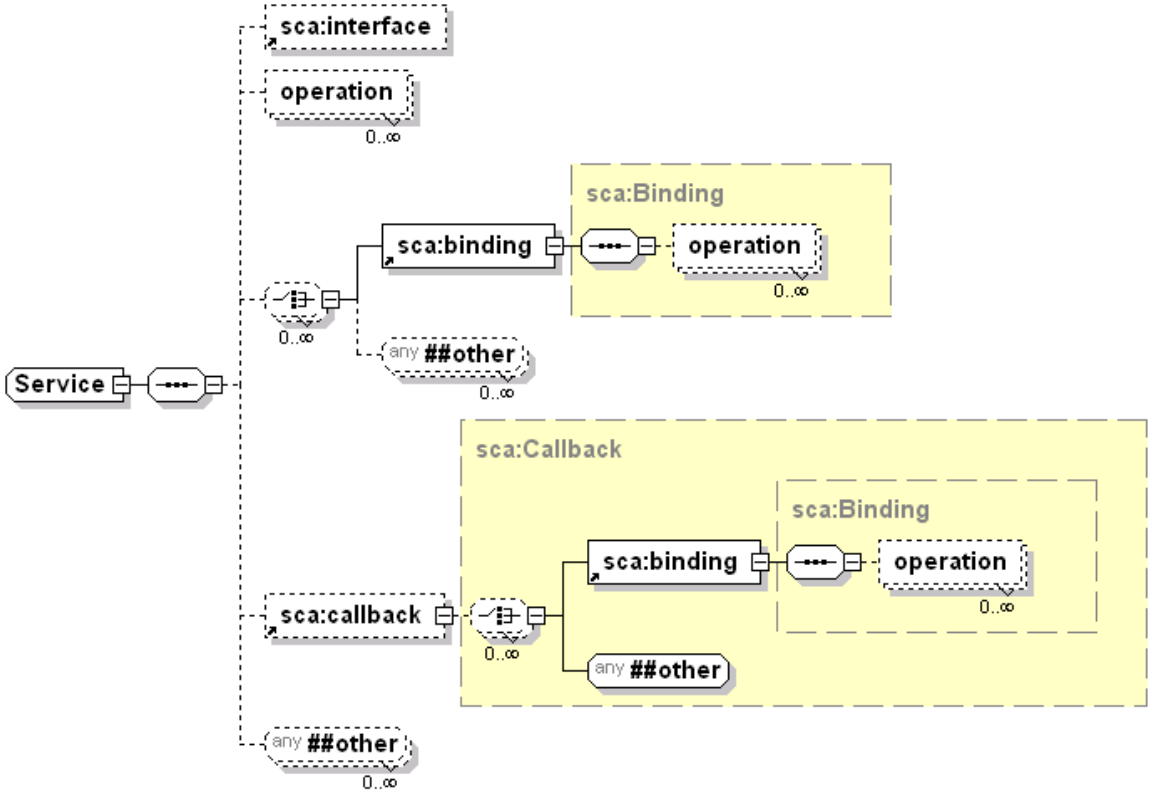
schéma XML de composition SCA

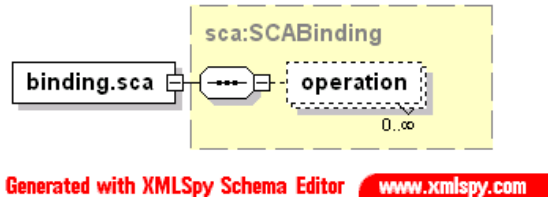


SCA : Component



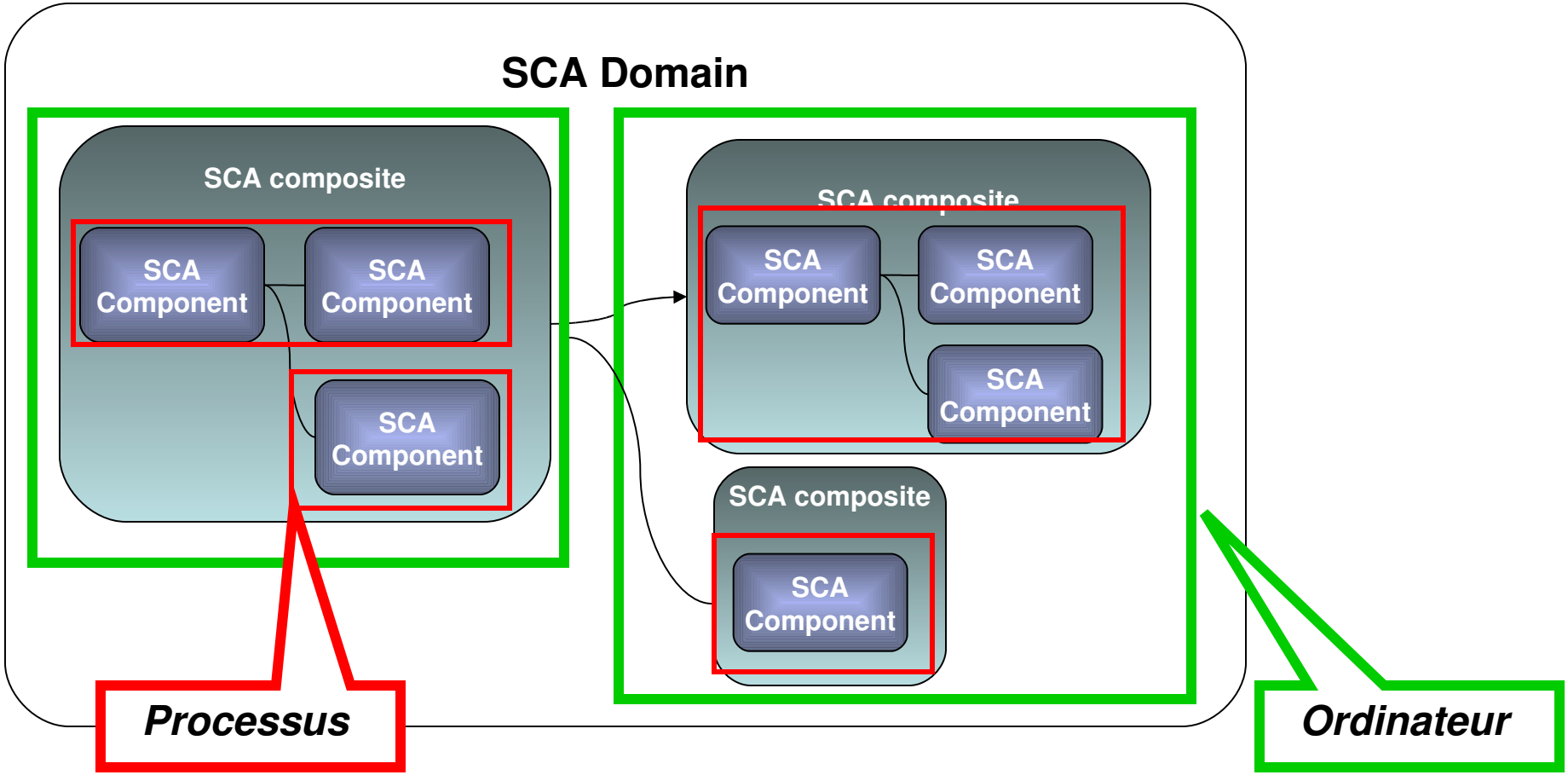
SCA : Service et Binding



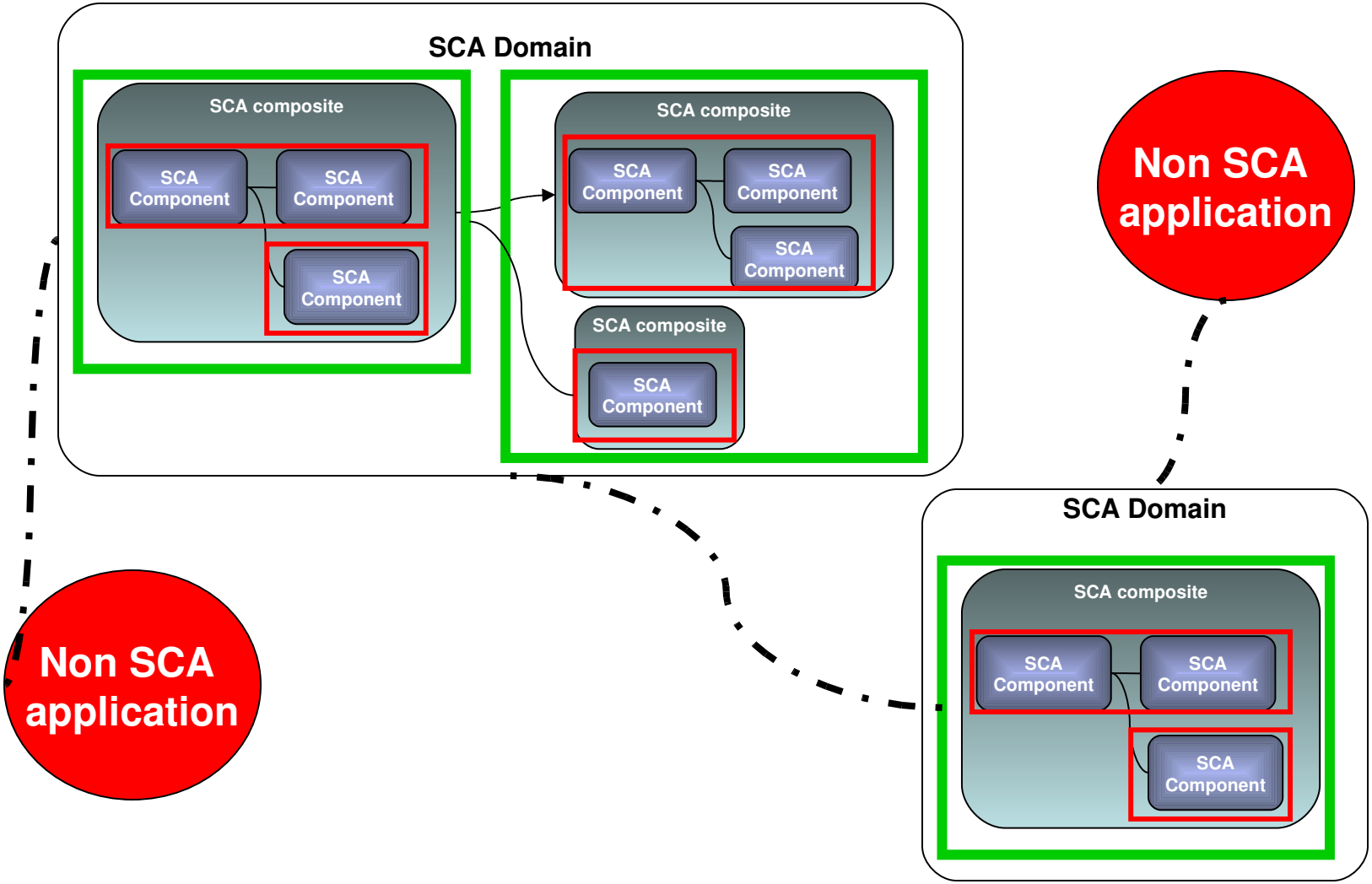


SCA : Notion de domaine,

- Notion de Domaine
  - périmètre d'une offre de solution (i.e. par un vendeur (!))

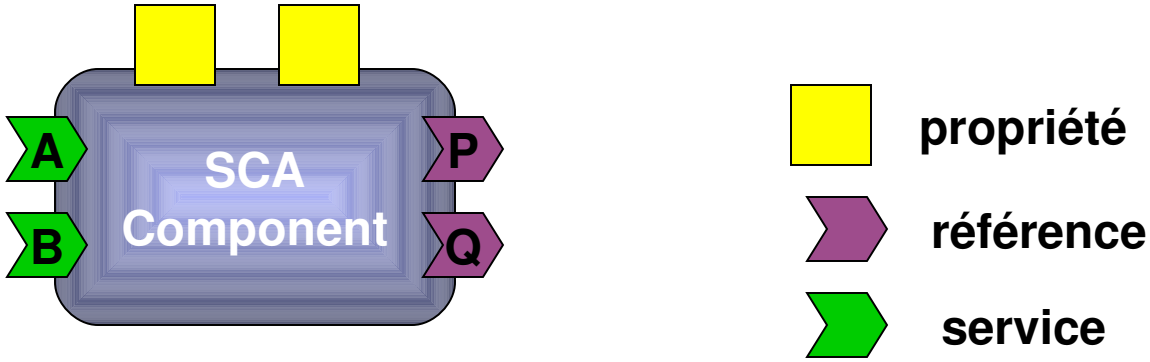


SCA : ouverture multi domaines



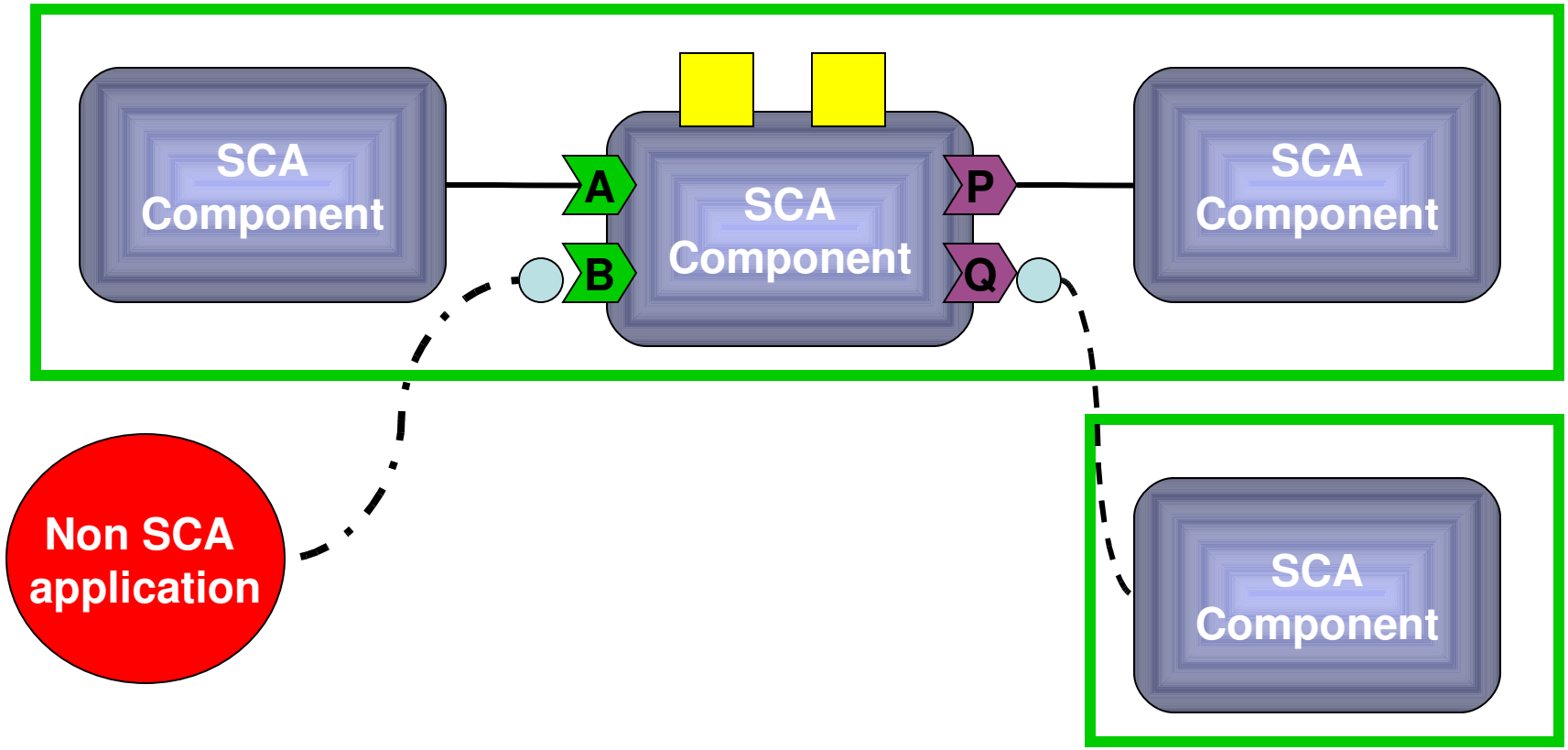
Composant typique SCA

- Un composant SCA est une instance d'implémentation d'une fonction
  - Telles qu'un processus BPEL
  - Telles qu'une classe Java
  - …
- Une configuration SCDL définit comment un composant interagit avec son entourage
- Quelle que soit sa technologie un composant SCA se caractérise par
  - Des *services*, exécutant des *opérations*
  - Des *propriétés* variables valorisées à l'instanciation du composant
  - Des *références* éventuelles à d'autres *services*
- SCA ne préjuge pas de la technologie sous jacente au composant
  - BPEL et WSDL
  - Classe Java
  - …



Communication

- La communication s'effectue par des « bindings » explicites hors des domaines circonscrits



Exemple d'implémentation Java

- Pour autoriser l'appel par des procédures distantes en Java, SCA utilise un mécanisme d'annotation, plutôt qu'un mécanisme d'appel d' API
- Cet exemple commence par l'import d'une annotation d'un package standard SCA. Il utilise ensuite cette annotation @Remotable pour indiquer que le service fourni par l'interface AS peut être accessible à des clients distants.
- Les autres informations nécessaires pour accéder à ce service sont reportées dans la spécification SCDL
- Ici l'interface AS est donc accessible par un processus externe, alors que l'interface MD n'est accessible que par un processus interne comme instance de la classe Calculator

```
import org.osoa.sca.annotations.Remotable;
@Remotable public interface AS {
        int add(int a, int b);
        int subtract(int a, int b);
}
public interface MD {
        int multiply(int a, int b);
        int divide(int a, int b);
}
public class Calculator implements AS, MD{
        public int add(int a, int b) {
                return a + b;
        }
        public int subtract(int a, int b) {
                return a - b;
        }
        public int multiply(int a, int b) {
                return a * b;
        }
        public int divide(int a, int b)
                { if (b == 0) {
                        throw new
IllegalArgumentException();
                } else {
                        return a / b;
}}}
```

SCA pour JAVA : référence

- SCA prévoit qu'un service puisse faire appel à un autre service en le référençant; pour l'implémentation Java, SCA spécifie ce référencement par l'annotation @Reference. Par exemple pour un interface nommé MonitoringService:

```
@Reference protected MonitoringService monitorService;
```

- Ainsi spécifié l'accès à au service `monitorService` ne nécessite pas d'instancier la classe MonitoringService. Il appartient au « runtime »SCA de localiser et de valoriser l'accès aux méthodes du service monitorservice.

```
monitorService.usageCount(x);
```

- La façon dont un runtime trouve l'instance d'un service qui satisfasse une référence est spécifié au niveau du Domaine,
- (cependant cela ne fonctionne pas pour l'invocation de service dans un autre domaine)

SCA pour Java : Propriétés

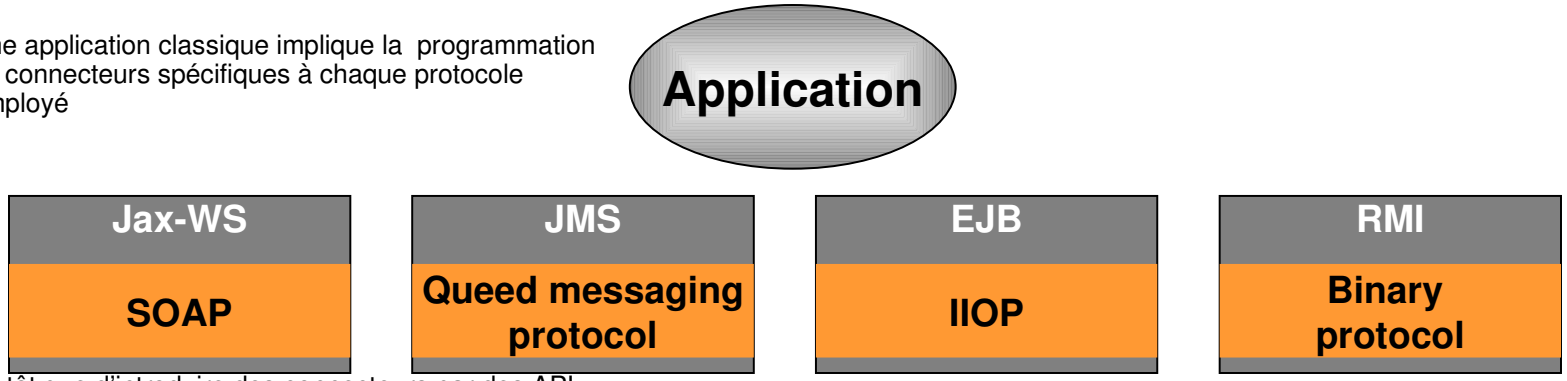
- Comme pour les références, SCA pour Java utilise le même mécanisme pour les propriétés avec l'annotation **@Property**

```
@Property protected String region;
```

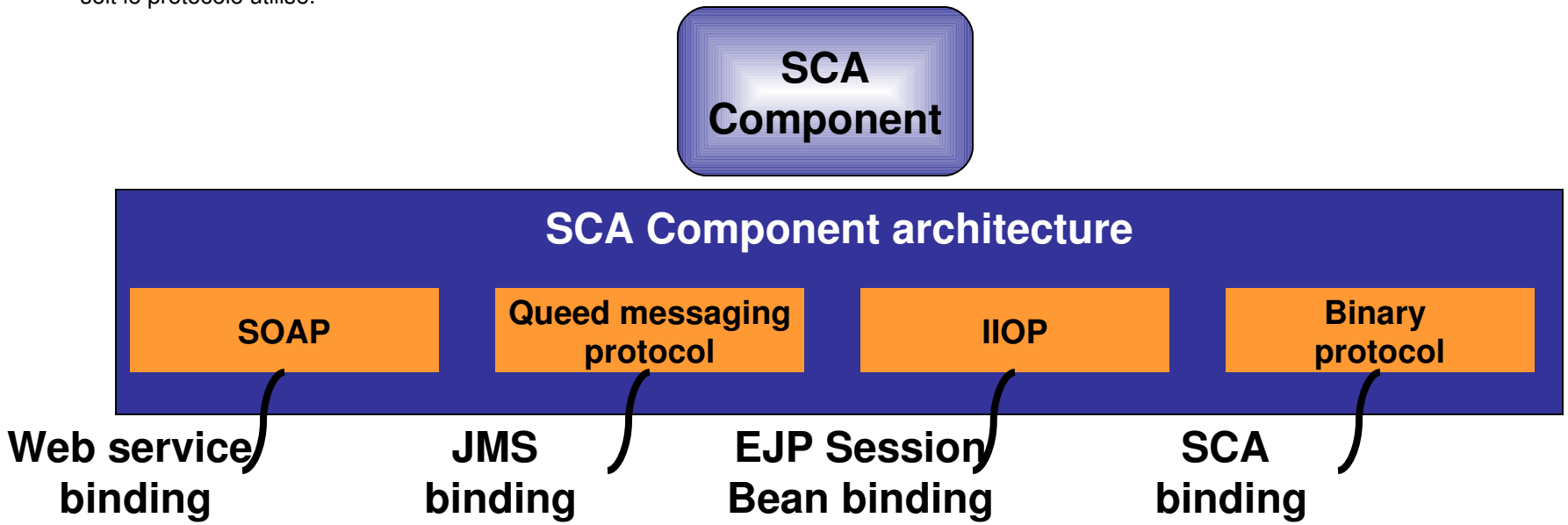
- Cette annotation peut être assignée à un champ d'une classe Java, ou d'une méthode pour assigner une valeur. Elle indique que cette valeur devra être lue selon les prescriptions de la spécification SCDL du composite auquel le composant appartient.
- Les propriétés peuvent être plus complexes, le principe reste le même : fournir le moyen de configurer un composant par des valeurs lues par le runtime

Architecture classique vs. Architecture SCA

- Une application classique implique la programmation de connecteurs spécifiques à chaque protocole employé



- Plutôt que d'introduire des connecteurs par des API vers différents protocoles SCA permet à tout service distant et toute référence de spécifier le protocole qu'il supporte au moyen de bindings
- Le modèle de programmation reste semblable quel que soit le protocole utilisé.



SCA : spécification d'un binding

```
<binding.ws uri="http://www.qwickbank.com/services/serviceA"/>
```

- Indique à la fois le type « web service » de l'association et son adresse
- SCA spécifie d'autres bindings, pour chaque technologie

```
<binding.jms uri="" />
```

Java Message Service binding,

```
<binding.ejb uri="" / >
```

EJB session bean binding.

- SCA permet ainsi d'associer toutes sortes de technologies distantes écrites en C, C++, Cobol etc.

SCA pour Java : autres conventions

- En plus de l'annotation **@Remotable** précédente, le modèle de composant SCA pour Java en définit d'autres :
  - **@OneWay**, indique qu'une opération ne renvoit pas de réponse, et donc ne bloque pas un processus.
  - **@Scope**, contrôle l'existence de l'instance du comosant; il peut par exemple être « *conversational »,* ce qui implique de maintenir son état entre des appels ou « *stateless* », ne maintenant rien au contraire entre deux appels.
  - **@Callback**, permet de définir un interface callback pour supporter une communication bidirectionnelle, ce que SCA nomme *bi-directional* interfaces.
- Tous les attributs ne sont pas utilisables par tous les bindings.
  - Par exemple **@Scope** avec l'option « conversational »ne peut être utilisé que par les protocoles pouvant passer des informations de session tels qu'un binding SOAP utilisant WS-ReliableMessaging.

Configuration SCDL d'un composant

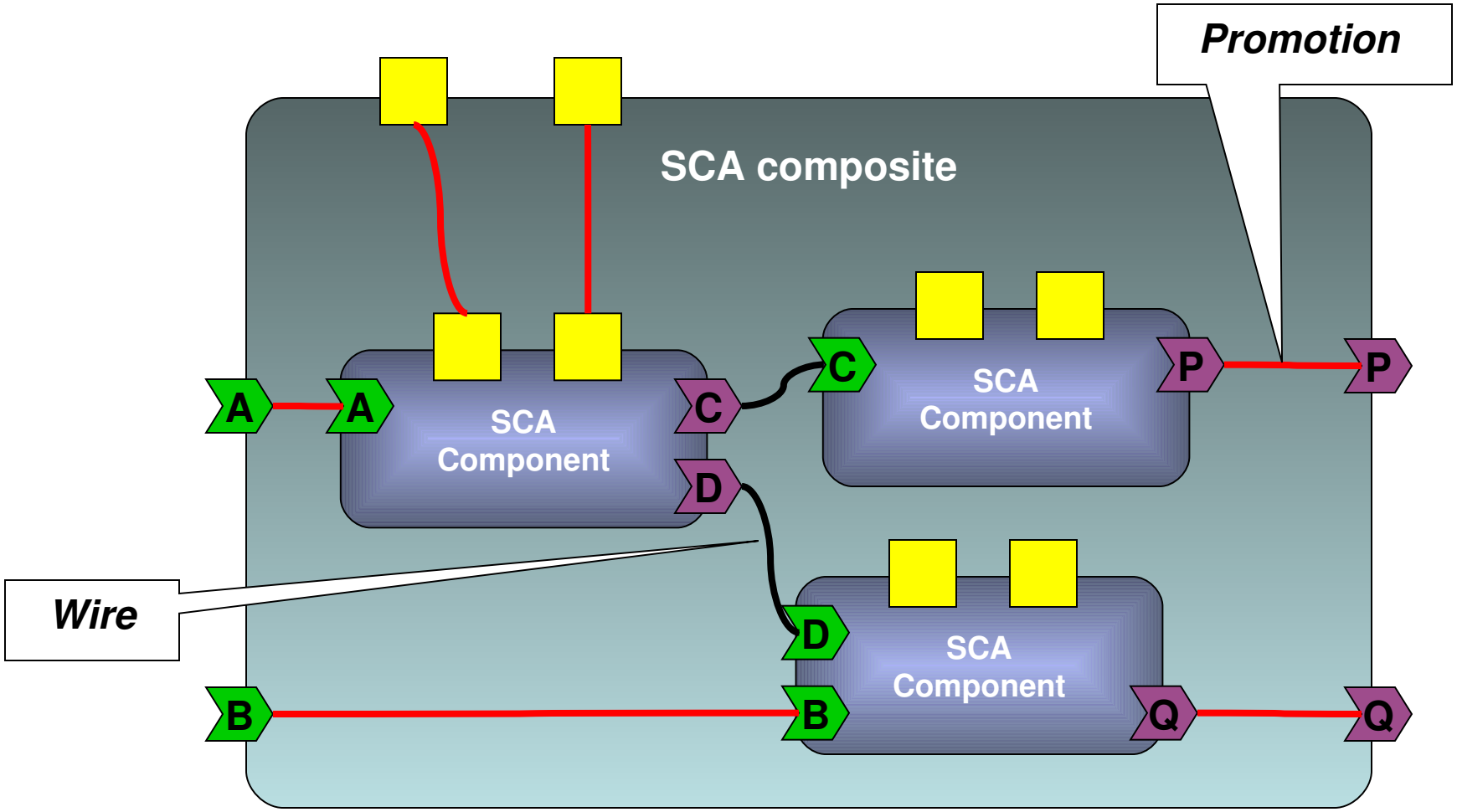
•Exemple de composant exploitant les ressources Java précédentes :

```
<component name="Component1">
 <implementation.java class="services.examples.Calculator"/>
 <service name="AS">
     <binding.ws uri="http://www.qwickbank.com/services/serviceA"/>
 </service>
 <reference name="MonitorService">
     <binding.ws uri="http://www.qwickbank.com/services/serviceM"/>
 </reference>
 <property name="region"> Europe </property>
</component>
```

Exemple de composant exploitant une ressource BPEL

```
<component name="Component1">
 <implementation.bpel process="ExampleProcess"/>
 <property name="region"> Europe </property>
</component>
```

SCA : Composite ; le modèle d'assemblage de SCA



SCA : spécification SCDL d'un composite

Signifie au runtime SCA de tenter de connecter automatiquement les services et références entre composants du composite, en comparant leurs intitulés.

```
<composite name="ThreeComponents" autowire="true"...>
 <component name="Component1">
     <implementation.bpel process="Process1"/>
 </component>
 <component name="Component2">
     <implementation.java class="services.examples.class2"/>
</component>
 <component name="Component3">
     <implementation.java class="services.examples.class3"/>
</component>
 <service name="A" promote="Component1/A">
     <binding.ws/>
 <service/>
 <reference name="T" promote="Component2/T"
 <reference name="U" promote="Component3/U"/>
</composite>
```

Il est aussi possible de définir des liens explicites entre composants avec des éléments « wire »

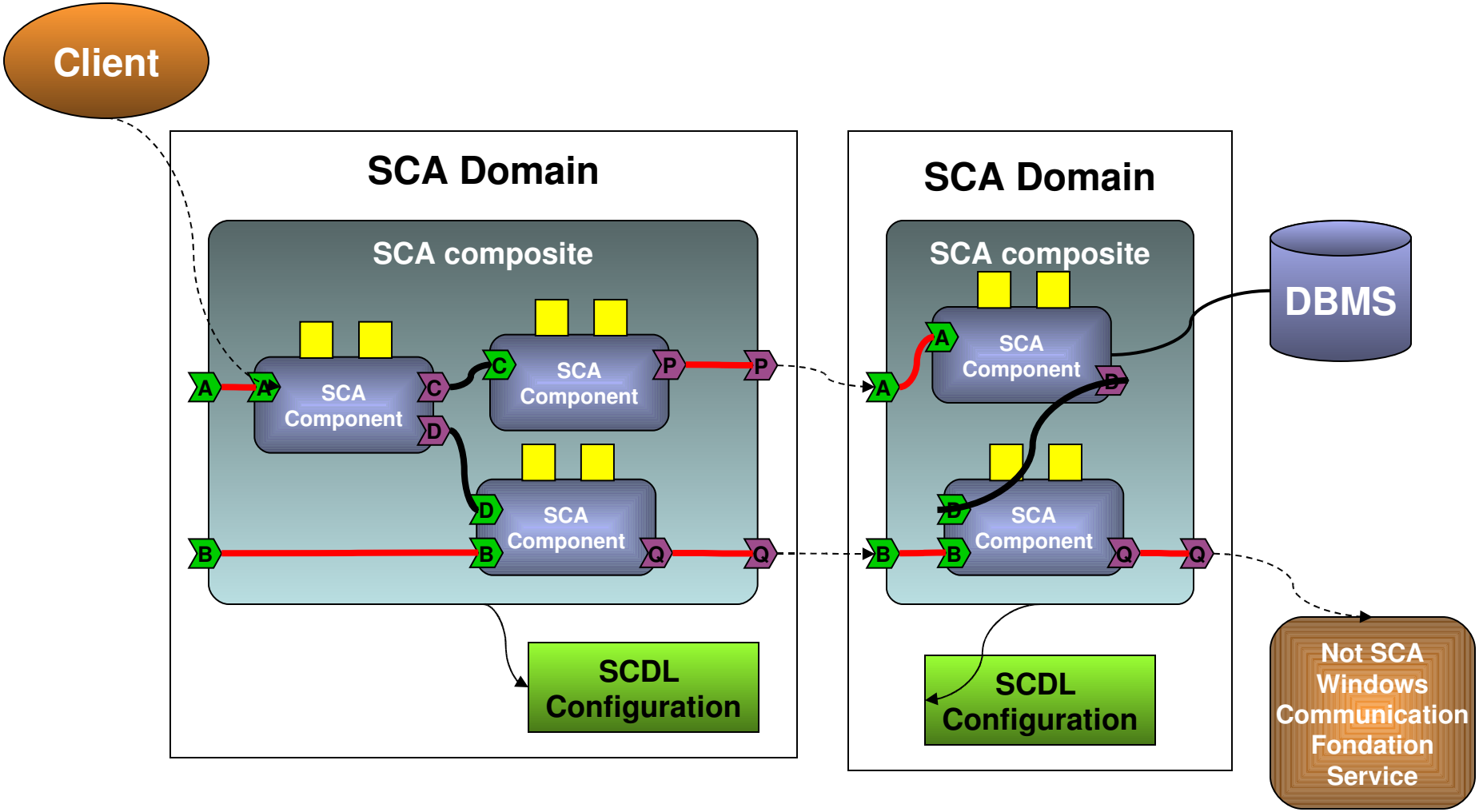
Les services et références promues définissent l'interface entre un composite et l'extérieur

SCA : policies

- Les interactions entre les différentes parties d'une architecture peuvent être complexes
- SCA intègre des notions de « policy » pour exprimer les intentions relatives à ces interactions indépendamment de leur réalisation effective
- SCA définit deux grandes sortes de policies :
  - Interaction policies
    - Modifie la façon dont les composants interagissent, par exemple pour prendre en compte des contraintes de sécurité ou de fiabilité.
    - Elles concernent typiquement les bindings
  - Implementation policies
    - Modifie la façon dont les composants se comportent localement.
- Les policies sont des éléments optionnels du langage SCDL
  - Dans son modèle pour Java SCDL prévoit aussi des annotations à associer aux interfaces, méthodes, telles que @Confidentiality , @Authentication,

```
<component name="xs:NCName">
    <implementation.* policySets="listOfQNames«  requires="list of intent xs:QNames">
     …
    <operation name="xs:string" service="xs:string"?
              policySets="listOfQNames"?
              requires="listOfQNames"?/>*
              …
     </implementation>
     …
</component>
```

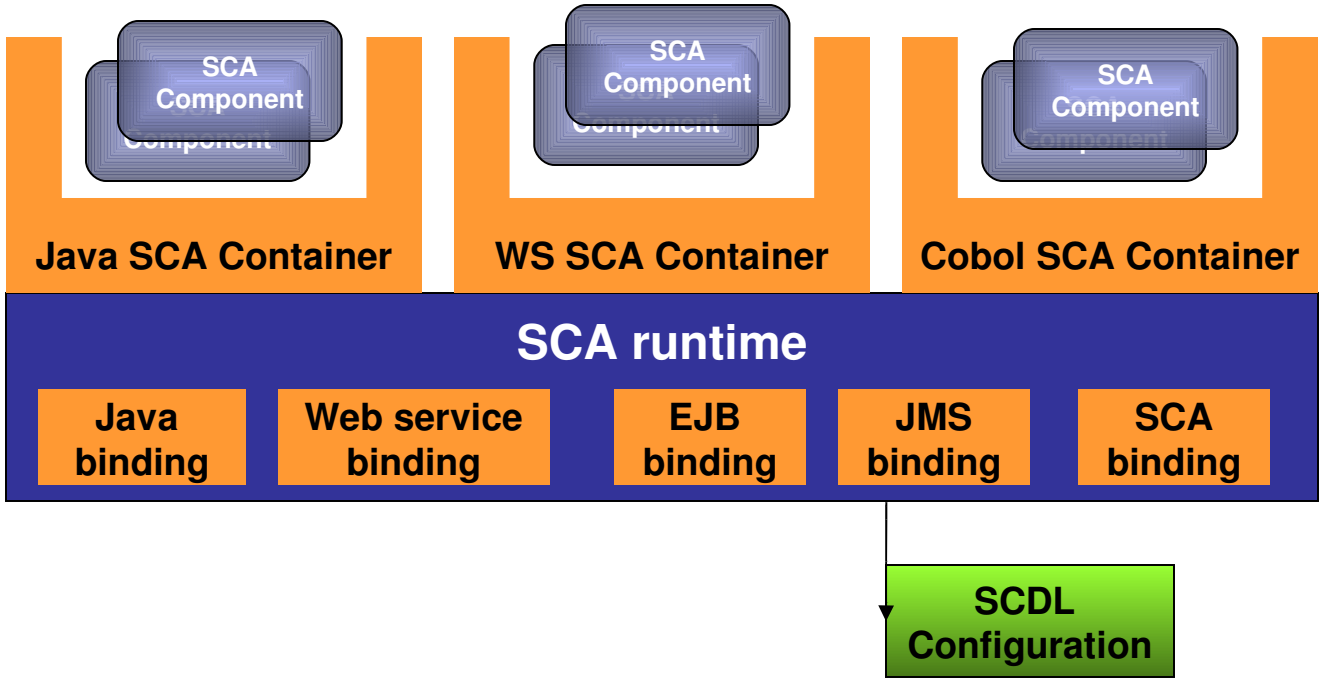
SCA : assemblage des notions



Technologies d'implémentation de SCA

- Solutions Libres :
  - Tuscany
  - Fabric3
  - Eclipse ?

- Architecture typique d'un runtime SCA:
  - Chaque technologie est encapsulée dans un conteneur recevant les composants



SCA : quelques observations en guise de conclusion

- La technologie SCA est trop récente pour pouvoir faire valoir des retours d'expérience
- Elle est structurante des architectures orientés services, indépendamment des technologies employées, en associant la mode nouvelle des web services aux plus anciennes (EJB etc.)
- Elle est destinée à des « vendeurs de solutions logicielles, qui peuvent privilégier tel ou tel aspect de la spécification
- SCA est extensible…
  … les vendeurs ne se priveront pas d'en exploiter la possibilité comme avantage concurrentiel

Un avis sur SCA relevé dans la presse (été 2007)

Burton Cautiously Optimistic about SCA for SOA
Rich Seeley, SearchWebServices.com

The analysts who cover service-oriented architecture (SOA) for BurtonGroup Inc. have some reservations about the Service Component Architecture (SCA) specification, but have concluded the vendor backingis so strong "adoption may be inevitable.“

Touted as "a new programming model for SOA" by its vendor sponsors led by IBM and now making its way through the OASIS standards process, SCA is not yet baked into many products beyond IBM's WebSphere, but Burton analysts expect adoption to pick up in 2008.

Given its apparent inevitability as a vendor supported standard, Anne Thomas Manes, vice president and research director at Burton, spent more than an hour Tuesday in a Web seminar explaining SCA's potential promise and problems to clients.

She said the concerns about SCA at Burton Group include the fact that SCA is made up of more than 14 specifications.

Analysts are skeptical that the various technical committees working on those specifications can reach the goal of creating an overall standard to "simplify" service creation and composition.

This leads to concern that SCA could suffer the same fate as Common Object Request Broker Architecture (CORBA), which failed to achieve its promise in the 1990s because, as one analyst put it, "too many cooks spoiled the broth." "There is some concern that SCA can hide all the complexities," she said.

The good news is that SCA has potential, yet unproven, to be a language-and protocol-independent programming model for SOA.

Languages that will be supported in SCA cover most of those a non-Microsoft coder would be working on today, ranging from COBOL to Ruby. Support is planned for Java, including Plan Old Java Objects (POJO), Spring, Enterprise Java Beans, C, C++, BPEL and PHP.

http://searchsoa.techtarget.com/originalContent/0,289142,sid26_gci1277423,00.html
See also OASIS SCA-TCs: http://xml.coverpages.org/ni2007-07-06-a.html#SCA-TCs

Fin du module