オープンSCAを利用したSOAアプリケーション設計
― SCA1.0 の特徴とその応用 ―

澤出 達郎

A SOA Application Design Approach Using Open SCA.
– Features and Applications of SCA 1.0 –

Tatsuroh Sawaide

本稿は, SOA（Service Oriented Architecture）アプリケーションのための開発・実行モデルであるオープン仕様SCA（Service Component Architecture）1.0の特徴とその応用を解説するものである. SCA 1.0は, 再利用性を持つサービスのコンポーネント化テクノロジであり, 言語中立性、再帰的アーキテクチャなどの様々な特徴を持つ. 本稿では, SCAコンポーネントの再利用性を関連する機能について解説し, その応用として上流から下流までのセマンティック・ギャップのないアプリケーション設計や既存のサービスのWeb2.0などの新しい領域での活用について言及する.

This paper explains the features of SCA 1.0 and its applications. SCA 1.0 is a new technology for service componentization which has many unique features such as "Language Neutral", and "Recursive Composition". By leveraging uniqueness, SOA developers can use SCA 1.0 from high level design to low level implementation without a semantic gap and it also enables the reuse of existing assets for next generation applications.

Key Words & Phrases： SOA, SCA, OSOA, コンポーネント, BPEL, XML
SOA, SCA, OSOA, component, BPEL, XML

1. はじめに

SOAにおけるビジネス・プロセスは通常、複数のサービスの組み合わせによって実現される. この組み合わせを一体のプログラムとして表現したものがコンポジット（合成）・アプリケーションである. SCAは, このコンポジット・アプリケーションを作成するための仕様である. SCAにおけるコンポジット・アプリケーションは, 様々な言語で書かれたビジネス・ロジックを包み込む機能の入れ物としての役割を果たす. いったん, ビジネス・ロジックをSCAの中に取り込んでしまうと後はSCAの機能を使用し, それらを自由に組み合わせ, 多様な形式でサービスの提供をすることができる. WebサービスもSCAの中ではサービス提供の一形態という位置付けとなる. 本稿では, 現在, OSOA（Open Service Oriented Architecture）[1] およびOASIS [2]によって策定中のオープンな仕様であるSCA バージョン1.0 [3] を参照する. SCAは, 既にIBMのWebSphere®プロセス・インテグレーション製品[4]のベースとして採用されているが, オープン化されたSCA仕様は一部拡張されており, 違いを持つに至っている. オープンSCAの平易な解説資料としては, OSOAから入手可能なチュートリアル[5]やプログラミングが中心の筆者のもの[6]が存在する. 本稿ではSCAの持つサービス・コンポーネントとしての柔軟性がいかに実現されているのか, また, 再帰的アーキテクチャ, 言語中立性の持つ意味に言及し, その特徴的な仕様を利用することによってもたらされるSOAプログラミングの新たな可能性について解説する.

提出日：2006年8月31日　　再提出日：2007年9月2日

2. SCAの提供する機能とその特徴

最初にSCA仕様の本稿に関連する部分の解説を行う. 本稿で主に取り上げるSCAにおけるコンポジット・アプリケーションの組み立てに関する仕様は, SCA アセンブリ・モデル[7]にて定義されているものである.

2.1 柔軟性を持つサービスの公開と利用の方法

ビジネス・ロジックとは, 必要な機能が実装された

プログラムの単位である. 通常は, その言語プラットフォーム毎の固有の方法でプログラムから呼び出されて利用される. これをコンポーネント（Component）という単位で他のプログラムおよび他のコンポーネントから利用可能な方法で公開することがSCAの機能であり, その方法を仕様として定義している. このコンポーネントをサービスとして公開する機能は,サービス（Service）と呼ばれる. また, 一つのビジネス・ロジックは, サービスの提供を行う一方, そのロジック自身の中で外部のサービスを利用する必要がある場合がある. SCAでは, このサービスの利用に関してもその方法を仕様として定義している. このサービスの利用機能は, リファレンス（Reference）と呼ばれる. サービス, リファレンスともに**図1**のようにSCAにおけるプログラムの構成単位であるコンポジット（Composite）の一部となる. サービスおよびリファレンスにはコンポジット・レベルのもの（図1, ①, ②）とビジネス・ロジックであるコンポーネントのレベルのもの（図1, ①', ②'）が存在する. コンポジット・レベルのものは対外的な接続を示し, コンポーネント・レベルのものはコンポーネント間および対外接続の内部での接続点となる. 図1は, 最も単純な例であるが, 複雑な事例であってもSCAアプリケーションはこのサービス, リファレンスの機能によって外部へのサービス機能を提供することになる.

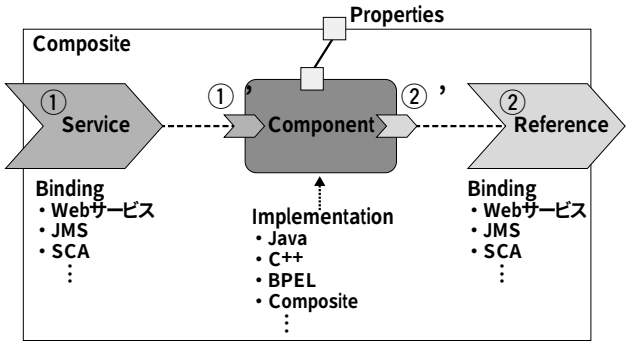

図1.　SCA "コンポジット" の基本構造［7］

2.2　再利用性の高いビジネス・ロジック

外部からの視点でみるとSCAの提供する機能とは, 2.1で述べたサービスの公開および利用である. ただし, これらはWebサービスあるいは他の分散アプリケーションによっても提供可能な機能である. これと比較した際のSCAの特徴の一つは, サービスのビジネス・ロジック部分であるコンポーネントを再利用し易い形態で作成できる点にある. SCAでは, **図2**のように各言語によるビジネス・ロジックの実装（Implementation）をSCAでの定義に基づいて構成されたコンポーネントとしてインスタンス化する.

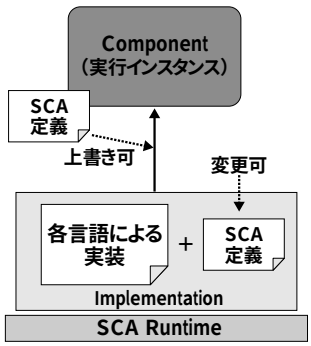

図2.　構成されたインスタンスを生成

この定義は, 実装とは無関係に変更, もしくは上位の定義での上書きが可能である. さらにこのコンポーネントの外側には, 2.1で述べたサービス, リファレンスを定義可能であり, そこでも定義を変更することができる. SCA定義の中には, サービスの参照関係の定義やバインディング・プロトコル指定が含まれる. SCAは, この柔軟なアーキテクチャより, 既存のビジネス・ロジックの再利用性を高めている.

2.3　言語中立性

次にSCA仕様の特徴として挙げられるのは, 言語中立性である. サービスの公開, 利用に当たって必要とされる作成物は, 言語を問わず共通形式のXML文書によって記述される. 実装言語による処理の違いは, XML文書に明記された情報によりSCA実行環境（Runtime）が判断して実行することになる. **図3**は, SCAにおいてコンポーネント実装の定義をしている部分のコード・フラグメントである. 図3の上がJava™用, 下がC++用となる. C++用の実装仕様は, SCA Client and Implementation Model C++［8］にて参照可能である. 両者の違いは, このモジュールのimplementation.xxプロパティがどの言語によって書かれているかを示すのみになっており, 多言語に対応していることがわかる. もし, SCA実行環境が許せば既存のコンポーネントの実装を他言語のものに置き換えることも可能である. SCA仕様においては, SCA実行環境は, 状況に応じて必要な言語をサポートしてよいと定義されており, SCA仕様［7］の"Extension Model"では,

```
*** Javaでのコンポーネント定義 ***
<component name="A-Component">
    <implementation.java
        class="A-ServiceImpl"/>
    <reference name="OtherService" target="OtherComponent"/>
</component>

*** C++でのComponent 定義 ***
<component name="A-Component">
    <implementation.cpp
        library="A-ServiceImpl.dll" header="A-ServiceImpl.h"/>
    <reference name="OtherService" target="OtherComponent"/>
</component>
```

図3.　実装言語の異なるコンポーネント定義

implementation.xxの"xx"部に対して新たな言語の定義を拡張することを可能としている. すなわち, SCA仕様自身は, 特定の言語に依存しないものとなっている.

2.4 高い自由度を持った呼び出し方法

言語選択とともに外部とコンポーネント間の呼び出しに関しても高い自由度が提供されている. 先の図1の一番外側の枠はSCAでの作成物であるコンポジットを現している. コンポジットの中には, 実装に結びついたコンポーネントを複数個包含することが可能であるが, これらのコンポーネントの外部からの呼び出し, およびこのコンポーネントが依存する外部サービス利用のプロトコルは, 必要に応じてWebサービス, JMS (Java Message Service)など, 複数のものから自由に選択することが可能となっている. 定義は, バインディングの設定として2.2でのSCA定義の中や図1のようにコンポジット・レベルでのサービス, リファレンスに双方において設定可能である. 尚, SCA仕様[7] "Extension Model"では, このバインディングに関しても言語実装と同様に拡張を可能としている.

2.5 結線による自由な組み立て

コンポジットには参照関係を持たせた複数のコンポーネントを内包させることができる. 参照関係の種類には, ワイア(Wire)とプロモート(Promote)が存在している.

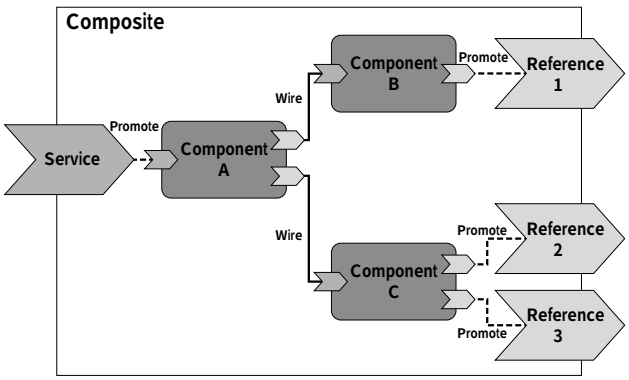

図4. 結線によるコンポジットの組み立て

図4のように参照関係は, 図上においては, 結線によって表現される. コンポーネント間での結線がワイアであり, コンポーネントと外部との結線がプロモートである. この参照の機能は, コンポーネントの実装に変化を加えることなく組み合わせを自由に変更することが可能となっている. 参照関係の定義は, サービス, リファレンスの定義中, または明示的に独立したWireエレメントの定義によって行う. 自動的に適合する参照関係を判断して結線を行う自動ワイアリング機能も提供されている.

2.6. 開発コードへの緩やかな制約

コンポジットの中に含まれるビジネス・ロジックの部分がコンポーネントである. このコンポーネントが実装のインスタンスを作り出すことになる(2.2参照). プログラミングの観点においてこのSCAによるコンポーネント開発の特徴は, 開発すべきソース・コード中へのSCA採用の影響を最小限に留められるように工夫されていることである. これは, 他の分散アプリケーション開発の方法と異なる点である. Java BeanベースのSCAコンポーネントとWASに採用されているJ2EEのビジネス・ロジックを担うEJB 2.1を比較してみると, 双方ともリモートからの呼び出しが可能な分散コンポーネントであるが, その開発の方法において違いを持つ. EJBでは, ビジネス・ロジックの実装クラスにおいてもjavax,.ejb.SessionBeanなどのEJB固有クラスの実装が必要であるが, Java BeanのSCA化では, 特別なインターフェースの実装やクラスの継承は必要なくいわゆる“ POJO" (Plain Old Java Object)のまま分散コンポーネント化できる. これは, ビジネス・ロジックの開発において採用するコンポーネント化テクノロジがソース・コードに対し影響を与えないということになり, 採用のリスクを減らす意味で大きな意味を持つ. またこの特徴は, コンテナのような特定の実行環境を必要としないテストも可能にしている.

2.7 アノテーションの活用

SCAは, その仕様において各言語に対して本質的に中立であるが, それぞれの言語においては, その特徴を生かす工夫がなされている. 特にJavaにおいてはその先進的な言語仕様であるアノテーションを幅広く採用している. **図5**の例は, アノテーション“ @Service ”を使用してコンポーネントを作成したサンプルである. @Serviceが付加されたクラスがJavaインターフェース定義に基づいてメソッド“ aloha(String message) ”を外部公開することを実現する. SCAが仕様として特別なインターフェースの実装やクラスの使用を必要としないことに合わせ, このアノテーションの使用が可能であることにより, SCAは,“ 簡易な開

```
/* Java Interface */
package test.sca;

public interface AlohaService {
    String aloha(String message);
}

/* Business Logic */
@Service(AlohaService.class)
public class AlohaServiceImpl implements AlohaService {
    public String aloha(String message) {
        return ("Aloha!");
    }
}
```

図5. アノテーション"@Service"の使用例

発 ”と“ コンポーネント・テクノロジ固有のコードを極力排除すること ”の両立に成功している.

2.8 再帰的なアーキテクチャ

コンポーネント化の目的の一つは, 作成したプログラムの再利用である. SCAでは, この再利用を容易するために再帰的なアーキテクチャが採用されている. これは, **図6**のように一度作成したコンポジットを実装の代わりに使用することができるというものである. WebSphere製品に現在, 採用されているオープン化以前のSCA仕様では, 一度生成したSCAアプリケーションは, 外部からサービスとして利用する以外に“ 再利用 ”する方法はなかった. SCA 1.0を利用することにより, 今後はそれらをアプリケーション開発のパーツとして利用可能になる. この機能により, あらゆる粒度を持つSCA コンポジットが実装として再利用可能となった. これはSCAを利用したSOAアプリケーション開発にとって大きな意味を持つ変更である.

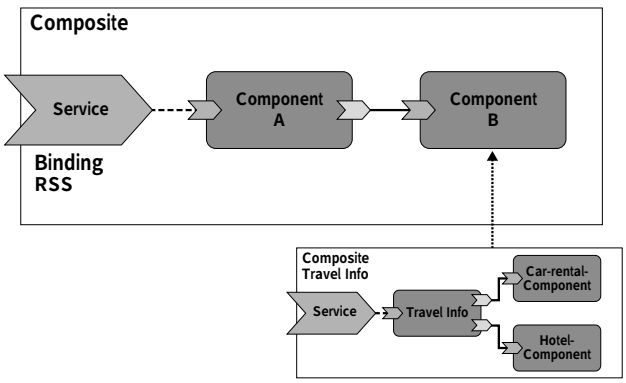

図6. 再帰的なアーキテクチャ

3. SCAによるSOAアプリケーション開発

以上述べてきたSCAの機能および特徴を踏まえ, SOAアプリケーション・デザインとその応用を考察する.

3.1 実装とのギャップのない上流設計

現在, SOAアプリケーションを作るアプローチの一つとしてITを意識しないビジネス・プロセスを定義することから始める方法がある. このアプローチは, ITを意識しないでよいというメリットを持つ. 一方でこの方法は, 概念的な定義の段階でBPMN（Business Process Modeling Notation）などを用い実装をあまり意識せず表記[9]を行うため, 機能を対応付けた表記の変換, および出来上がった実装プログラムから元のビジネス・プロセスへの後戻りの双方が難しいという問題を持つ[10], [11]. これは両者にセマンティック・ギャップが存在することに起因する. この問題を解くにはビジネスに近い高レベルな視点と末端の実装までの双方を同一の言語で表現する方法が考えられる. SCAは, その再帰的なアーキテクチャと言語中立性という特性からこの目的の言語として使用できる可能性がある. **図7**は, SCAで表現できるコンポジット・アプリケーションのレイヤー構造を表したものである. 先に述べたSCAの再帰的なアーキテクチャより, 最終的なアプリケーションのパッケージとなる最上位の抽象度の高いコンポジットから, 末端の実装まで同一のXMLスキーマで表現される. 尚, SCAでの使用するXMLスキーマはSCA仕様[7]の "XML Schemas"にて参照可能である.

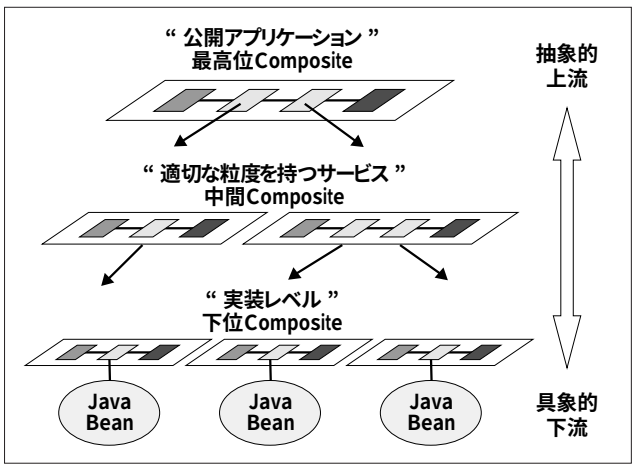

図7. 階層的なコンポジット・アプリケーション

3.2 末端の実装までのサービス・コンポーネント化

最上位, 中間のコンポジットにおいては概念的な実装を行っても最終的に処理を行うのは末端のビジネス・ロジックの実装である. SCAが実装を包み込むコンポーネント仕様として優れている点の一つは, 使用する言語に合わせて, 実装のインスタンス化を細かく制御できる点である（図2）. Javaを実装として使用する場合, 実行の効率やプロセスの配置を考慮したローカル呼び出し, リモート呼び出しといった実装に密接した内容をアノテーションとして記述することができる. この機能により, SCAでは粒度の細かいサービス・コンポーネント化を行っても実行時の効率に影響が少なくなるようにデザイン可能である. 通常, サービスのデザインを行う場合, ある程度の粒度を大きくとってサービス・コンポーネント化を進めることが一般的であると思われるが, SCAではこの特徴を生かし, 実装に近い粒度の細かいものまでコンポジットによるサービス・コンポーネント化を行うことが可能となる. 最終的なサービスとして扱い易さを考えた粒度の調整は, SCAの再帰的アーキテクチャ用いることで後から調整できる. この末端の実装レベルまでコンポジット化を行うことで上位のコンポジットから末端まで同じXMLスキーマに基づくモジュールとして取り扱いが可能になり,

XMLの機械的に読み取り可能な特性を使用した作成物の管理やプログラム構造自身をXML検証の対象とできることのメリットは非常に大きいものと考える.

3.3 フロー機能は後の工程で決定

SCAのコンポジット内における複数コンポーネント間のワイアリングは, 提供側のサービスと利用側のリファレンスに参照関係を定義している. これはUMLに置き換えてみるとサブシステムを多用したクラス図に近いイメージである. コンポーネントを組み合わせたアプリケーションを作成するに当たり, 次に必要と考えられるのはUMLでのシーケンス,ビジネス・プロセスでいうところのフローあるいはプロセス・コレオグラフ(プロセス動作の振り付け)ということになる. このフローの表現についてはSCAの仕様の中では直接定義されてなく, フロー機能を持った何らかの実装がされた一つのコンポーネントを利用するという扱いとなる. このフロー機能を持ったコンポーネントの例としてはWebSphere Process ServerにおけるBPEL(Business Process Execution Language)やBusiness State Machinesが挙げられる. これらは, フローの表現としてそれぞれに特性を持っており, 適用領域も異なる. フロー用のSCAコンポーネントも定義上, 特殊なものではなく他のコンポーネントを参照して動作する一つのコンポーネントという位置付けであり, 自由に選択可能である. SCAのこの特徴を生かすとフローの選択に関しても中立性を生かした手法を取ることができる. 参照関係の定義が終了し, フロー定義が必要になった時点で方法を決定することで事によってリスクも少なく自由度の高い工程が実現可能である.

3.4 抽象的なアプリケーション・デザインの例

上流から設計アプローチの例としてSCA1.0の仕様を参考に簡単なアプリケーションのデザインの試行をした. 内容は, 旅行会社が旅程を作成するサービスを提供するアプリケーションである. **図8**は, その最上位のコンポジットである.

```
<composite xmlns=http://www.osoa.org/xmlns/sca/1.0
                targetNamespace="http://travelsvc"
                xmlns:ts="http://travelsvc"
                name="travelsvc">
<component name="itinerary-maker">
    <implementation.composite name ="itinerary-flow"/>
    <reference name="car" target="car-rental-component"/>
    <reference name="hotel" target=hotel-component/>
</component>
<component name="car-rental-component">
    <implementation.composite name ="heats-rental"/>
</component>
<component name="hotel-component">
    <implementation.composite name ="grand-hotel"/>
</component>
</composite>
```

図8. 旅行業アプリケーションの定義ファイル

このアプリケーションでは, 旅程作成アプリケーションがレンタカー, ホテル関連の2つのコンポーネントを参照しつつ動作する. 上位のコンポーネントである "Itinerary-Maker"は, この後, BPELなどのフロー言語によって実装する. この段階ではそれぞれのコンポーネントは, 実装として仮想のコンポジットを指定しているだけで言語も含め未決定であり抽象度の高いレベルのデザインを維持している.(**図9**) 図8のコンポジット定義は, 実装作業の基点となるもので概念的なデザイン・ドキュメントとしてだけ使用するものではなく, その後の開発プログラムの一部となるものである. このようにSCAは, 実アプリケーションの作成を概念的なデザイン・ドキュメントから始められることがわかる.

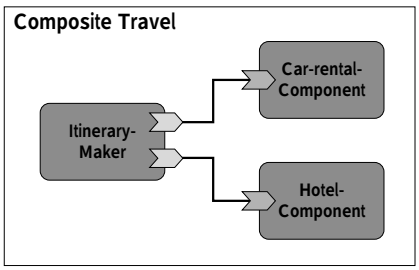

図9. Travel コンポジットの参照関係

3.5 既存アプリケーションへの新機能の追加

長く使用される有用なビジネス・ロジックは, 時間の経過とともに時代に応じた新しいインターフェースを備える必要に迫られる. 例えば, 現在, Webサービスなどのインターフェースによってサービスを提供しているアプリケーションも今後, Web2.0的ニーズへの対応が迫られるケースが発生することが想像できる. SCAで作られたサービス・コンポーネントは, このような環境の変化に容易に対応することができる. **図10**は, 既存のアプリケーションにRSSフィードを付加している例である. RSS [12] は, RSSリーダーに対して情報を発信する時に用いられるプロトコルでWeb2.0的なマッシュ・アップ[13]も容易である. この変更は, 基となるアプリケーションにRSSバインディングを持つサービスと

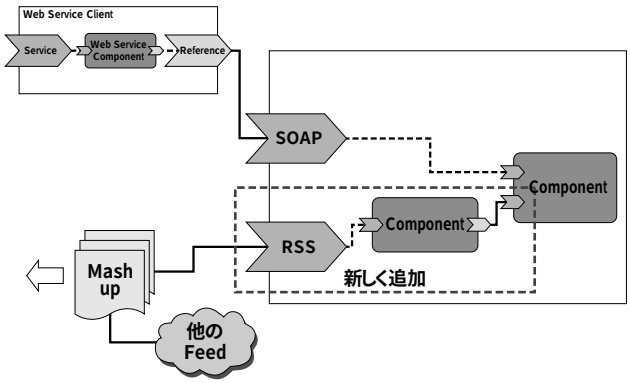

図10. 既存サービスをRSSフィードに対応

RSS用のデータを作るコンポーネントを追加するものである. 元のビジネス・ロジックのコンポーネント自身には変更は加えられていない. 実際にオープンソースのSCA 実行環境であるApache Tuscany［14］では, バインディング（Binding）としてRSSフィードに既に対応しており, このような既存の資産を有効に利用し時代の変化に対応することが可能になっている.

図11は, さらにSCAの利点を追求し, コンポジットを再帰的に利用してRSSフィードに対応している例である. この例では, 元のコンポーネントだけではなく, 既存のアプリケーションコンポジットにも全く手を加えずにWeb2.0的な利用を可能にしている.

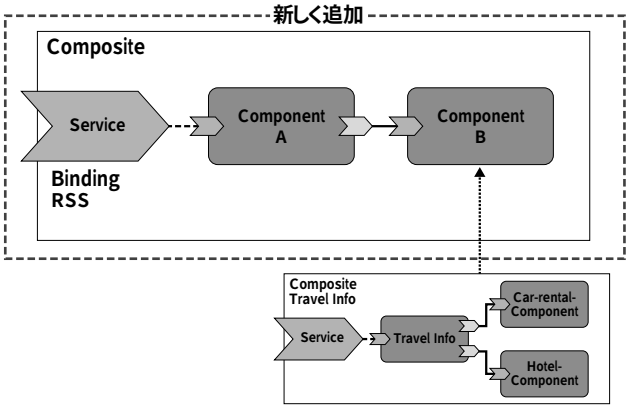

図11.　再帰的な実装の例

4．SCAによるデザイン・アプローチの課題

4.1　統合開発ツールの必要性

本稿の執筆時点では, 再帰的なアーキテクチャ持ったSCA1.0の開発をサポートする統合開発ツールは, 存在していない. ネスト構造を持った複雑なコンポジット開発には, 理解し易いGUIのサポートやそこからのXMLの自動生成, XMLの自動Validationなどを行う開発ツールが必要である. 開発ツールに関しては, Eclipse.orgにてSOA Tools Platform Project［15］がSCAに対応した新しい開発ツールを作成中である. 当プロジェクトが最新のSCA仕様に対応することおよび, 商用製品の早期の出現に期待したい.

4.2　言語中立性に関するSCA実行環境の制約

SCAは, 仕様上の言語中立であるが, その言語を定義・実行できるかどうかは実行をするSCA実行環境に依存している. 一つのSCA実行環境が実行できる実装には制約があり, 言語の選択も現実にはその範囲に限られる. 例えば, コンポジット内にJavaとC++が混在することは許されない（**図12**）.

ただし, SCA 1.0にはドメイン（Domain）という概念が導入され, ドメイン内にネットワークによって結合された複数のSCA実行環境が存在する可能性も語られている［5］. SCAアプリケーションであるコンポジットは, このドメインにデプロイされ実行される. ドメインに展開された複数のコンポジットは, 仮想的なドメイン・レベルのコンポジットを形成することができる. 今後, 一つのドメインに複数の言語をサポートする多種のSCA実行環境が存在する形態が実現すれば, このドメインのレベルで異なる言語の実装が協調してインスタンスを動作させることも可能になるものと思われる.

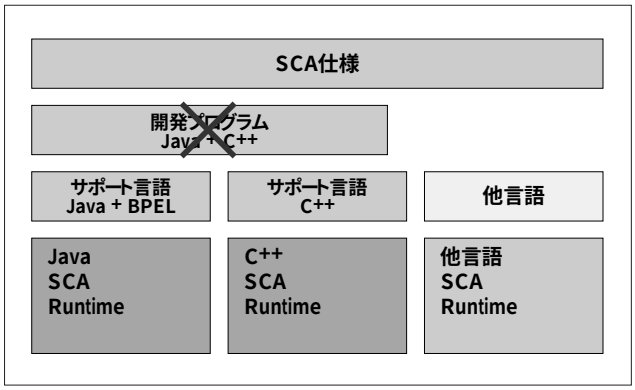

図12.　SCAプログラムと実行環境の関係

5．おわりに

本稿では, 2章においてオープンな仕様であるSCA 1.0の特徴的な機能の解説、3章にてその応用についての考察、4章では, 現状における課題に言及した. このようにSCA1.0は, 再帰的なアーキテクチャを採用することによってサービスの再利用性を高めつつ, アノテーションの積極的な活用によって開発言語としての生産性も向上させた. SCA1.0の登場により, SCAは従来のサービスの実行基盤として役割を超え, SOAデザイン言語としての利用や既存業務アプリケーションの新しい環境の適応などさらに利用の幅が広がったと言える.

SCAは, 過去の分散コンポーネント技術の経験を踏まえて開発されている仕様である. そこには, 言語中立性などの従来のCORBA的な発想［16］とともにXML技術, POJOへの指向, DI（Dependency Injection）［17］, アノテーションなどの最新の発想が多く盛り込まれて成長している. 本稿で取り上げたアセンブリ・モデル以外にも非機能要件を記述できるPolicy Frameworkなど［3］, 多くの関連仕様が整備されつつあり, セキュリティやトランザクションといったミッション・クリティカルな業務への対応も今後, 進むものと思われる. 今後の仕様の発展と早期の実行環境の整備に期待したい.

参考文献

[1] OSOA: Open Service Oriented Architecture http://www.osoa.org（2007.9.30）.
[2] OASIS: Open CSA http://www.oasis-opencsa.org/sca（2007.9.30）.
[3] OSOA: *Service Component Architecture Specifications*, http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications（2007.9.30）.
[4] WebSphere Process Server V6解体新書Service Component Architectureガイド WebSphere Developer Domain http://www-06.ibm.com/jp/software/websphere/developer/wbi/diamond/01.html（2007.9.30）.
[5] OSOA, Mike Edwards, Anish Karmarkar, Jim Marino: *Service Component Architecture (SCA) Tutorial : Part 1*, http://www.osoa.org/download/attachments/250/SCA_Tutorial_Part1_Asia-updated.pdf?version=1 (2007.9.30).
[6] 澤出達郎: 入門SCA V1.0, IT Pro連載日経BP社 http://itpro.nikkeibp.co.jp/article/COLUMN/20070515/271071/?ST=develop（2007.9.30）.
[7] OSOA: *SCA Assembly Model Specification V1.00*, http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1 (2007.9.30).
[8] OSOA: *SCA Client and Implementation Model C++ V1.00*, http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModel_CppV100.pdf?version=2（2007.9.30）.
[9] Stephen White: "Introduction of BPMN," http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf（2007.9.30）.
[10] Yi Gao: "BPMN-BPEL Transformation and Round Trip Engineering," http://www.eclarus.com/pdf/BPMN_BPEL_Mapping.pdf（2007.9.30）.
[11] Stephen White:"Using BPMN to Model A BPEL Process," http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf (2007.9.30).
[12] RSS Advisory Board: "*RSS 2.0 Specification*," http://www.rssboard.org/rss-specification (2007.9.30).
[13] Nicholas Chase: "The ultimate mashup -- Web services and the semantic Web, Part 3: Understand RDF and RDFs" IBM developerWorks, http://www.ibm.com/developerworks/edu/x-dw-x-ultimashup3.html（2007.9.30）.
[14] Apache Tuscany Project http://incubator.apache.org/tuscany/（2007.9.30）.
[15] Eclipse.org : SOA Tools Platform Project http://www.eclipse.org/stp/（2007.9.30）.
[16] Object Management Group: "History of CORBA," http://www.omg.org/gettingstarted/history_of_corba.htm（2007.9.30）.
[17] Martin Fowler : "Inversion of Control Containers and the Dependency Injection pattern," http://martinfowler.com/articles/injection.html (2007.9.30).

日本アイ・ビー・エム株式会社
ソフトウェア事業
WebSphere テクニカル・セールス
コンサルティングITスペシャリスト

澤出 達郎 Tatsuroh Sawaide

［プロフィール］

1985年，日本IBM入社．ソフトウェア事業におけるOS，ミドルウェア製品主管部門にてお客様に対する技術支援を経験．1999年より，WebSphere Application Serverを担当し，現在は，WebSphere BPM関連製品を中心にテクニカル・セールスに従事している．

sawaide@jp.ibm.com