モデル駆動型アーキテクチャによるソフトウェア開発の合理化

Automatic Software Development with Model Driven Architecture

矢野　令　　大森　麻理　　細谷　竜一

■YANO Rei　■OMORI Mari　■HOSOYA Ryuichi

“モデル駆動型アーキテクチャ（MDA：Model Driven Architecture）”は，分析や設計の“モデル”を中心としたソフトウェア開発の概念である。東芝ソリューション（株）は，独自の“モデル駆動型部品（MBC：Model-Based Component）”及びMDAの中核をなすモデルコンパイラを試作し，MDAに基づく開発を実現した。

MDAの実システムへの適用実験を行い，MDAを適用した場合と適用しない場合の設計を比較することで，熟練した設計者のノウハウを確実に適用できるという効果が得られた。

Model driven architecture (MDA) is a concept of software development that is centered on analysis and design models. Toshiba Solutions Corp. has developed model-based components (MBCs) and a model compiler that is at the core of MDA, and realized software development based on MDA. By applying MDA to a real system development and comparing the machine-made design with a hand-made design, we have identified the effect of MDA in enabling uniform and automated application of expert knowledge of software design.

一般論文

1　まえがき

“モデル駆動型アーキテクチャ”（MDA：Model Driven Architecture）とは，分析や設計の“モデル”を中心としたソフトウェア開発の概念であり[(1),(2)]，近年注目を浴びている。ソフトウェア開発の標準を定めるOMG（Object Management Group）が2001年3月に提唱し，現在，概念の体系化や規格作りを進めている。

ここでは，東芝ソリューション（株）の独自の“モデル駆動型部品（MBC：Model-Based Component）”を用いたMDAの実現方法について述べる。また，試作したモデルコンパイラを適用した場合と適用しない場合との設計を比較する。

2　MDAに基づく開発

2.1　MDAとは

MDAとは，モデルを中心としたソフトウェア開発の概念である。ビジネスロジックやアプリケーションロジックを，基盤となるプラットフォーム技術から分離することにより，ビジネスやプラットフォーム技術の変化に強いシステムを開発することを目的としている。MDAを利用して構築したプラットフォーム非依存のソフトウェアのモデルは，CORBA（Common Object Request Broker Architecture），J2EE[注1]，.NET[注2]，XML Webサービス，ウェブアプリケーションなど，幅広いプラットフォーム上で動作するシステムを実現できる。

2.2　MDAに基づくソフトウェア開発工程

MDAにおけるソフトウェア開発の流れは，モデルの変遷として定義され，具体的にはCIM（Computation Independent Model），PIM（Platform Independent Model），PSM（Platform Specific Model），実装という四つのモデルの流れで表現する（**図1**）。

CIMは計算処理に非依存なモデルであり，システムの入出力やユーザーからの要求を表現する。PIMはプラットフォームに依存しないモデル，換言すると複数のプラットフォームに対応可能なモデルである。PSMはシステムが利用する特定のプラットフォームを想定したモデルである。実装は，システムを構築して運用に移行するためのすべての情報を提供するに足るモデルである。現時点では，CIMからPIMへの変換は手動で行うが，PIMからPSMへの変換及びPSMから実装への変換は，“モデルコンパイラ”という変換器を利用して行う。

従来，この過程を人手で設計していたため，矛盾や不具合が混入しがちであった。しかし，PIMとPSMの変換をモデルコンパイラが自動化することにより，矛盾や不具合が入り込まない高品質なPSMや実装を生成できる。また，変換にかかる時間は，人間が数日かけて設計を行うよりも確実に短く，かつ予測可能になる。

（注1）　J2EEは，米国Sun Microsystems, Inc.の米国及びその他の国における登録商標又は商標。

（注2）　.NETは，米国Microsoft Corporationの米国及びその他の国における商標。

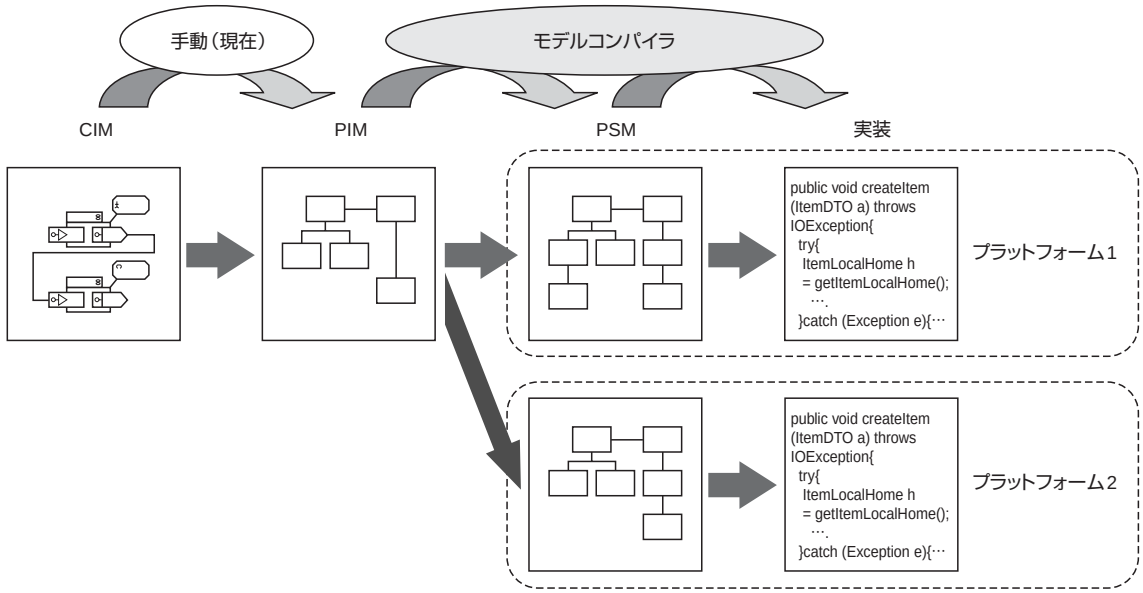


図1．MDAにおけるモデルの変遷 ― MDAにおけるソフトウェア開発の流れは，CIM，PIM，PSM，実装という四つのモデルの流れで表現する。
Transitions between four different types of models: computation independent model (CIM), platform independent model (PIM), platform specific model (PSM), and implementation

2.3 MDAの基盤技術

MDAの基盤を成すOMG標準技術のうち，UML（Unified Modeling Language）[3]とXMI（XML Metadata Interchange）[4]について述べる。

UMLはオブジェクト指向開発のための標準表記法である。ソフトウェアの構造やふるまいを記述する方法として提案され，現在は広く活用されている。MDAでは，PIMやPSMを表現する手段としてUML2.0の利用を推奨している。

XMIは，UMLで記述したモデルをXML（Extensible Markup Language）形式で表現するための共通規約である。これを採用するメリットは，異なるメーカーの開発ツール間で，UMLによって記述した設計モデルの互換性を保証する点である。MDAでは，一般的にはUMLツール間の互換フォーマット，あるいはモデルコンパイラの入出力のデータ形式としてXMIの利用が想定されている。

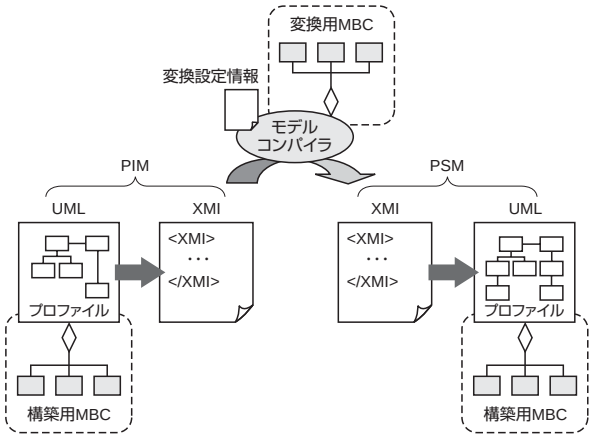


図2．MBCによるMDAの実現 ― 当社では，MBCによってMDAを実現する。
Realization of MDA with MBCs

3 MBCによるMDAの実現

MDAは概念であるため，その実現手段は規定していない。そこで当社は，独自のMBCを用いることでMDAを実現する。当社のMDAの実現方式を**図2**に示す。またMDAで対象とするモデルには，クラス図などで表現する“静的な構造”と，ステートマシン図などで表現する“動的なふるまい”とがある。当社はまず，PIM－PSM間の静的な構造を変換する手法の開発に取り組んだ。以下において，PIM－PSM間の静的な構造の変換に焦点を当てて述べる。

PIMは特定のUMLプロファイルを用いて記述する。UMLプロファイルとは，UMLのステレオタイプと制約とタグ付き値の集合であり，PIMの構成要素の性質を示すための情報である。後述する変換用MBCは，UMLプロファイルによって修飾された構成要素のステレオタイプをもとに変換を行う。またPIMは，構築用MBCを組み合わせることで作成する。構築用MBCは，再利用性の高いクラスやその組合せである。信頼性の高い構築用MBCを用いてPIMを構築することで，高品質なPIMを設計できる。UMLで記述したPIMはXMIで表現する。UML－XMI間の変換は，既存のUMLツール

などを利用できる。モデルコンパイラは変換用MBCと変換設定情報に従い，変換を行う。変換用MBCは，PIM－PSM間の変換ルールを部品化したものである。生成するPSMは，PIMと同様，変換ルールで指定したUMLプロファイルで修飾したものであり，構築用MBCで構成する。

このプロジェクトでは，モデルコンパイラ及び変換用MBCの試作を行った。以下において，今回試作したモデルコンパイラ及び，その入出力となるPIMとPSM，変換用MBC，変換設定情報について詳述する。

3.1 PIM

PIMをPSER（Process-Service-Entity-Rule）用UMLプロファイル（以下，PSERプロファイルと略記）を用いて記述した。PSERとは，当社が開発したコンポーネントベースのソフトウェアアーキテクチャである[(5)]。PSERはソフトウェアを構成する業務コンポーネントの分類及び関連を表す。すなわち，アプリケーションの処理の流れを定義し制御する業務プロセス（Process），アプリケーションが必要とする機能を提供する業務サービス（Service），ストレージに格納されるデータをオブジェクト化した業務エンティティ（Entity），そしてシステムや業務上の制約・条件などを表す業務ルール（Rule）である。また，Processコンポーネントは Serviceコンポーネントを利用し，Serviceコンポーネントは Entityコンポーネントを利用する。Ruleコンポーネントはどのコンポーネントでも利用できる。

UMLのクラス図で記述したPIMの構成要素（クラス）は，≪process≫，≪service≫，≪entity≫，≪rule≫のいずれかのステレオタイプで分類し，関連づける。PSERプロファイルで修飾したクラス図の例を**図3**に，そのXMI表現を**図4**に示す。

3.2 PSM

PSMのプラットフォームとしてStruts[(6)]を採用した。Strutsとは，ServletやJSP[(注3)]（JavaServer Pages[(注4)]）を組み合わせてウェブアプリケーションを構築する際に利用するフレームワークである。“プラットフォーム”とは，通常，OS（基本ソフトウェア）やプログラミング言語，ミドルウェアなどを対象としているが，当社は，Strutsのようなアプリケーションフレームワークや実行時に必要となるソフトウェアも含めて，プラットフォームとして扱う。

PSMは，Struts用PSERプロファイル（以下，Strutsプロファイルと略記）に従う。Strutsプロファイルは，PSERプロファイルにStruts特有の≪action≫などのステレオタイプを加えたものである。

3.3 変換用MBC

PIM－PSM間の変換は変換用MBCに基づいて行う。変換用MBCは，変換ルールを再利用可能な部品にしたものである。変換ルールは，「例えばPIMの≪process≫で修飾したクラスに対して，クラス名の末尾に“Action”を付加し，Strutsの Actionクラスを継承する」などのルールである。また，特定のプラットフォーム向けの変換用MBCの集合を，変換ルールセットと呼称する。今回の試作では，Struts用の変換ルールセットを試作し，変換用MBCの実装方式としてXSLT（eXtensible Stylesheet Language Transform）[(7)]を採用した。XSLTは，あるXMLを別のXMLに変換するための仕様である。これにより，PIMのXMIからPSMのXMIへの変換が可能になる。XSLT変換を行うツールにはXalan[(8)]などがある。

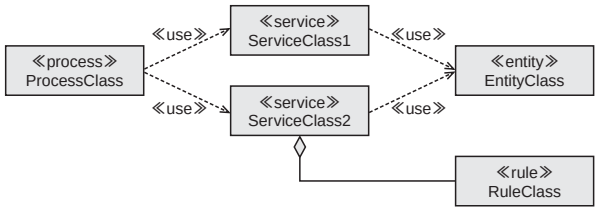


図3．PSER用UMLプロファイル ― PIMを構成するクラスは，≪process≫，≪service≫，≪entity≫，≪rule≫のいづれかのステレオタイプで分類する。

Unified modeling language (UML) profile for process, service, entity, and rule (PSER)

```
<?xml version="1.0"?>
<XMI xmi.version="1.1" xmlns:UML="http://www.omg.org/UML/1.3">
<XMI.content>
<UML:Model xmi.id="Model_1"
  name="Sample" visibility="public" isSpecification="false"
  isRoot="false" isLeaf="false" isAbstract="false" >

  <UML:Namespace.ownedElement>

   <UML:Class xmi.id="ID_Class_1" name="ProcessClass"
     visibility="public" isSpecification="false"
     isRoot="true" isLeaf="true" isAbstract="false" isActive="false"
     namespace="Model_1" clientDependency="ID_4 ID_5" />
   <UML:Stereotype xmi.id="ID_Stereotype_1" name="process"
     visibility="public" isSpecification="false"
     isRoot="false" isLeaf="false" isAbstract="false"
     icon="" baseClass="Class"
     extendedElement="ID_Class_1" />
...
```

ProcessClassの定義

ProcessClassのステレオタイプの定義

図4．PIMのXMI表現 ― ProcessClassのクラス定義はClass要素で，また，ProcessClassのステレオタイプ《process》はStereotype要素で表現する。

Sample of XML metadata interchange (XMI)

3.4 変換設定情報

変換設定情報の実現方式として，ビルドツールであるAnt[(9)]のビルドファイル（一般的なmakeファイルに相当）を採用した。現時点では，ビルドファイルには，PIMのXMIのファイル名，利用する変換用MBCの名称，出力するPSMのXMIのファイル名を指定している。

3.5 モデルコンパイラ

モデルコンパイラの実現方式として，AntとXalanを採用

（注3），（注4） JSP，Java及びその他のJavaを含む商標は，米国Sun Microsystems, Inc.の米国及びその他の国における登録商標又は商標。

した。Antは変換設定情報を解釈し，Xalanに対して入力XMIのファイル名，出力XMIのファイル名，使用する変換用MBCの名称を与えてXalanを実行する。Xalanは，入力として受け取ったPIMのXMIファイルを変換用MBC（XSLT）に基づいて変換し，PSMのXMIを生成する。

4 モデルコンパイラの適用実験

MDAによる開発の効果を検証するために，試作したモデルコンパイラを実際のシステム開発に適用する実験を行った。適用実験では，システムのPIMからモデルコンパイラを用いてPSMを自動生成した場合と，設計者が自身でPSMを設計した場合との比較を行い，MDAの効果を確認した。以下において，その詳細を述べる。

4.1 システム概要

適用実験の題材としたものは，当社で稼働中の経費購買システムである。申請機能などの一部の機能についてPSERプロファイルに基づいてクラス図を作成し，それをPIMとした。UMLによるPIMのクラス図を**図5**に示す。

4.2 モデルコンパイラによる自動生成PSM

まず，図5のPIMをUMLツールを用いてXMIに変換し，それを試作したモデルコンパイラの入力として，PSMの自動生成を行った。変換用MBCには，試作したStruts用変換ルールセットを用いた。生成したPSMのクラス図を**図6**（a）に示す。

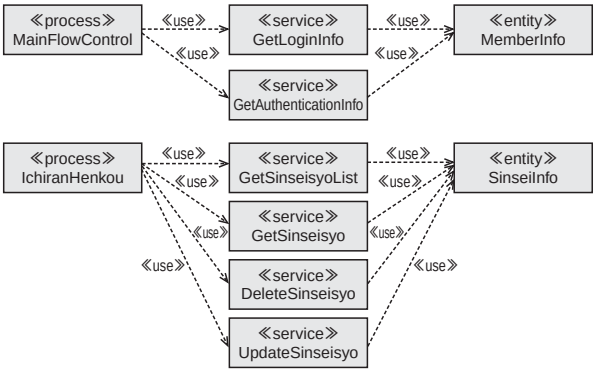
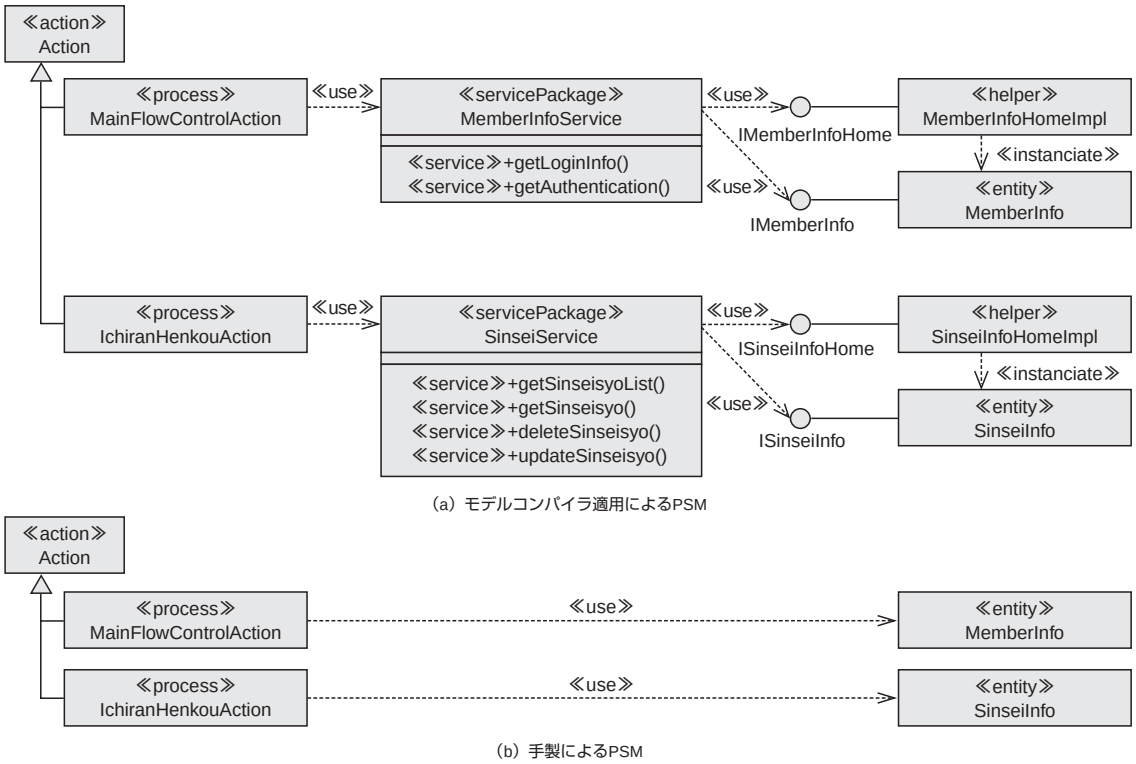


図5．例題システムのPIM ― 二つのProcessコンポーネント，六つのServiceコンポーネント，及び二つのEntityコンポーネントから構成される。
PIM for sample system

4.3 人間の設計による手製PSM

同時に，モデルコンパイラを使わずに，図5のPIMをもとにした手製PSMのクラス図を図6（b）に示す。このPSMは，



（a）モデルコンパイラ適用によるPSM

（b）手製によるPSM

図6．自動生成PSMと手製PSMの比較 ― (a)においては，ProcessコンポーネントはStrutsのActionクラスを継承したり，複数のServiceコンポーネントを一つのクラスに集約したりする。また，(b)においては，ServiceコンポーネントはProcessコンポーネントに取り込まれ，直接Entityコンポーネントを呼び出す。
PSMs generated by model compiler (top) and made by hand (bottom)

一般的なスキルレベルの設計者が，自身の経験と判断に基づき設計を行ったものである。

4.4 PSMの比較と考察

自動生成PSMでは，PIM中でのServiceコンポーネントは一つのクラスに集約し，それぞれをメソッドに変換している。またEntityコンポーネントは生成・消滅に関するクラス及びインタフェースを導入し，Entityを容易に管理できるように変換している。

一方，手製PSMでは，ServiceコンポーネントをProcessコンポーネントの内部に取り込んでいる。機能が一つのクラスに集約しているのでメンテナンス性は向上するが，Serviceコンポーネントを再利用できなくなる。Entityコンポーネントについては特に何も手を加えていない。

以上の比較から二つのことがわかる。一つ目は，変換ルールを自動適用するので，開発者の習熟度によって設計に差が出にくくなることである。手製PSMのServiceコンポーネントの有無に見られるような，設計者の違いによる設計のばらつきを極力抑えることができる。二つ目は，モデルコンパイラにより，変換ルールとして部品化したノウハウを自動適用できるということである。Entityコンポーネントに関する設計はモデルコンパイラによる設計のほうが適切であり，この設計を行わない場合，Entityコンポーネントの生成・消滅に関して実装時に問題が発生することがある。

以上の比較は，MDAが熟練した設計者のノウハウをそのまま確実に適用できることを示しており，ソフトウェア開発における品質の向上に大きく貢献することを意味している。従来の開発方法論や規約の適用では，ガイドなどに示された内容を逐一把握しなければならず，実際の開発時に，設計者が人間系においてそれらを一様に適用する必要があった。従来の開発方法論などとMDAとの違いはここにある。

ここで述べた手法は，システムの規模が大きくなり，クラスの数などが増大したり，スキルの異なる複数の開発者が設計を担当したりする場合に，特に大きな強みを発揮する。ここでは簡易なモデルを対象としたが，PSERプロファイルに従う設計であれば，大規模なモデルに対しても適用可能である。

4.5 モデルコンパイラの発展に向けて

今回の実験で，MDAに基づく開発の技術的な実現性と有用性を確認できた。これを踏まえて次の3点を整備することで，実システム開発で大きな効果を上げられる。

⑴ 自動生成範囲の拡大　　試作した変換用MBCでは，PSERプロファイルで記述した部分のみが変換の対象となる。≪process≫クラスと≪service≫クラス間で授受されるオブジェクトなど，そのほかの定型化できる部分に関しても変換の対象を拡大していく。

⑵ 変換用MBCの利便性向上　　現在の変換用MBCはXSLTで記述しているため，変換ルールを人間が理解しにくい。そのため，特別な技術知識がない者でも変換ルールを定義できるような仕組みが必要になる。

⑶ 動的ふるまいの変換とモデル検証　　今回の試作では静的構造を対象としているため，動的ふるまいについても対象とする。また，動的ふるまいの実現と同時に，モデル検証（ソースコードにおける実行とデバッグに相当）についても開発を行う。

5 あとがき

MDAを用いた開発はモデルを中心とした開発であるため，モデルの構造とモデルの変換が重要になる。ここでは，MBC及びモデルコンパイラを活用することで，より品質の高いシステムを開発できることが適用実験から検証できた。

当社は今後，MBC及びモデルコンパイラの更なる開発を行い，実プロジェクトへの展開を行っていく。

文　献

⑴ Joaquin, M., et al.“ MDA Guide Version 1.0.1 ”. OMG.
＜http://www.omg.org/docs/omg/03-05-01.pdf＞,（accessed 2004-01-20）.

⑵ 山城明宏，ほか．プログラムのコーディングからモデリングへ．東芝レビュー．**58**，10，2003，p.65－69.

⑶ Object Management Group.“ UML 2.0 Superstructure Final Adopted specification ”. OMG.
＜http://www.omg.org/cgi-bin/doc?ptc/2003-08-02＞,（accessed 2004-01-20）.

⑷ Object Management Group.“ XML Metadata Interchange（XMI）Specification, v1.2 ”. OMG.
＜http://www.omg.org/cgi-bin/doc?formal/2002-01-01＞,（accessed 2004-01-20）.

⑸ 深谷哲司，ほか．ソフトウェア設計の革新．東芝レビュー．**56**，11，2001，p.40－46.

⑹ The Apache Jakarta Project.“ The Apache Struts Web Application Framework ”. Apache Software Foundation.
＜http://jakarta.apache.org/struts/＞,（accessed 2004-01-20）.

⑺ World Wide Web Consortium. “ XSL Transformations（XSLT）”. W3C.
＜http://www.w3.org/TR/xslt＞,（accessed 2004-01-20）.

⑻ The Apache XML Project.“ Xalan-Java ”. Apache Software Foundation.
＜http://xml.apache.org/xalan-j/＞,（accessed 2004-01-20）.

⑼ The Apache Ant Project.“ Apache Ant ”. Apache Software Foundation.
＜http://ant.apache.org/＞,（accessed 2004-01-20）.

矢野　　令　YANO Rei

東芝ソリューション（株）SI技術開発センター　SI技術担当。オブジェクト指向分析の設計・開発を経て，現在，XML Webサービスの設計・開発業務に従事。

Toshiba Solutions Corp.

大森　麻理　OMORI Mari

東芝ソリューション（株）SI技術開発センター　SI技術担当主任。ソフトウェア設計技術の研究・開発に従事。電子情報通信学会，情報処理学会会員。

Toshiba Solutions Corp.

細谷　竜一　HOSOYA Ryuichi

東芝ソリューション（株）SI技術開発センター　SI技術担当。ソフトウェア設計技術の研究・開発に従事。情報処理学会会員。

Toshiba Solutions Corp.