社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－SE－150 (5)
2005／11／29

Strutsフレームワークにおける メタモデルを用いた追跡可能性実現手法の提案

大平 直宏† 松下 誠† 岡野 浩三† 楠本 真二† 井上 克郎†
山下 裕介‡ 我妻 智之‡

† 大阪大学 大学院情報科学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3

‡ (株) NTT データ
〒 104-0033 東京都中央区新川 1-21-2

ソフトウェアを高品質かつ短期間で開発・保守したいという要求が高まる中，Web アプリケーションを効率よく開発するためのフレームワークが登場している．このようなフレームワークを用いた開発では，ソフトウェアを新規に開発するときだけでなく，修正や削除などの保守を行う際にもフレームワークの意図する枠組みに従った設計が維持されていることが保証できなければならない．本研究では Struts フレームワークを対象とし，個々の成果物の仕様，およびそれら成果物間の依存関係の仕様となるメタモデルを定義することで，一貫性を保った状態でソフトウェア成果物を管理し，追跡可能性を実現する手法を提案する．

Realization of Model Traceability using Metamodel for the Struts Framework

Naohiro Ohira† Makoto Matsushita† Kozo Okano† Shinji Kusumoto† Katsuro Inoue†
Yusuke Yamashita‡ Tomoyuki Azuma‡

† Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka, 560-8531 Japan

‡ NTT DATA Corporation
1-21-2 Shinkawa, Chuo-ku, Tokyo, 104-0033 Japan

Web application frameworks have been proposed to develop and maintain high quality software in an effective way. In the case of software artifacts developed under these application frameworks, they are required to satisfy the specifications of the framework, not only during the development phase, but also during the maintenance phase. Therefore, it is important to ensure that the software artifacts are kept in a consistent manner according to the frameworks. In this paper, focusing on the Struts framework, we propose a method to realize model traceability and manage software artifacts in a consistent way, by defining metamodels of artifacts and dependency among them.

1 まえがき

Webアプリケーションに代表される中小規模のソフトウェアを高品質かつ短期間で効率よく開発・保守したいという要求は大きく，Javaにおいては Struts[2] や JSF[7] といった Web アプリケーションフレームワークが用いられている．アプリケーションフレームワークを用いたソフトウェア開発では，性質の似たソフトウェアに共通する処理をフレームワークが行ってくれるため，開発者は個々のソフトウェア毎に異なる部分だけを設計・実装すればよく，開発効率を高めることができる．

フレームワークを用いた開発では，フレームワークの仕様を満たす実装が要求されるため，その設計も制限される．しかし，明示的な仕様が定められている実装とは異なり，設計成果物に対する仕様は定められていないことが多い．そのため，開発者が作成・修正した設計成果物がフレームワークの仕様を満たしていることを検証できないという問題が起こっている．また，設計成果物の厳密な仕様が存在しないため，設計・実装間の変更影響を管理できず，一貫性を保証することができない．

そこで，本研究では Struts フレームワークにおける画面遷移を対象に，メタモデリング言語 MOF（Meta Object Facility）[8][9] を用いて，成果物の仕様を形式的に定義し，追跡可能性（トレーサビリティ）を実現する手法を提案する．提案手法は，

(1) 個々の成果物の仕様であるメタモデルを定義し，各成果物が仕様を満たす状態で作成されていることを保証する．

(2) 成果物間の依存関係のメタモデルも定義し，トレース情報自体も成果物と同様に管理し，成果物の変更影響が及ぶ範囲を特定可能にする．

という2つの手順によって追跡可能性を実現し，新規作成のみならず変更が行われた際にも一貫して成果物を管理可能にする．提案手法自体は特定のフレームワークや成果物に依存するものではないが [3]，本研究では Struts フレームワークにおける画面遷移の設計・実装を対象にする．Struts は Java の Web アプリケーションフレームワークとして広く利用されている．一方で，Struts を用いた開発においては，設計成果物である画面遷移図と実装成果物である設定ファイル struts-config.xml の一貫性を保証することが課題となる．

成果物間の関連を定義する研究としては，独自の変換規則を記述してモデル変換を実現する研究 [4] や，UML のサブセットと制約言語 OCL を用いて，互いに変換可能なモデル間の関連を記述する研究 [1] などがある．これらはいずれも関連のあるモデルの変換に重点を置いているが，本研究では成果物とその依存関係を MOF で統一的に記述し，トレーサビリティを実現することを目指している．

以降，2節では Web アプリケーションフレームワーク Struts について説明し，3節ではメタモデルを作成する重要性とメタモデリング言語 MOF について説明する．4節では提案手法に必要な各種メタモデルを定義し，5節で MOF メタモデルを用いたトレーサビリティの実現手法を説明する．最後に，6節でまとめと今後の課題を述べる．

2 Struts

ここでは Java を用いた Web アプリケーションフレームワーク Struts[2] と，その開発時に作成される画面遷移図，struts-config.xml について述べる．

2.1 Web アプリケーションフレームワーク

一般に，Web アプリケーションは，ユーザ画面のフォーム情報を入力として受け付け，リクエストに応じた処理を実行し，結果を次の画面として Web ブラウザ上に出力する．ユーザに表示する画面はあらかじめ静的に作成されている場合も実行時に動的に作成される場合もあるが，いずれの場合もその数は開発するアプリケーションの規模に比例して多くなる．そのため，Web アプリケーションを効率的に開発するためのアプリケーションフレームワークが用いられている．Web アプリケーションフレームワークは，画面間の遷移を効率的に管理し，Web アプリケーション全般に共通する処理を受け持つことで，開発者が固有部分の開発のみに専念できるようにする．

2.2 Struts 概略

Struts は Java 言語を用いた Web アプリケーション開発を支援するためのフレームワークであり，アクションサーブレットと呼ばれるコアエンジンが画面遷移を含むアプリケーションの動作全体を制御する．Struts はアプリケーションの機能を次のようなコンポーネントに分離させ，それらをアク

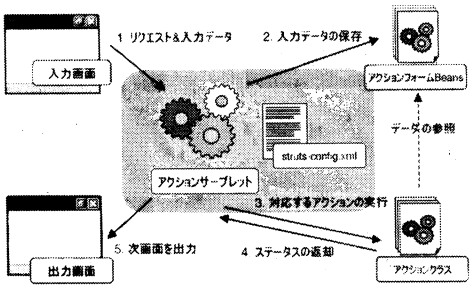

図 1: Struts における処理の流れ

ションサーブレットが統合する形で 1 つの Web アプリケーションを構成する.

- ユーザの入出力画面を表示する JSP ファイルや HTML ファイル
- ユーザがフォームに入力した情報を格納するクラス（アクションフォーム Beans）
- アクションフォーム Beans を参照し，ユーザのリクエストに応じた処理を実行するためのアクションクラス

アクションサーブレットの設定ファイルである struts-config.xml は，これらのコンポーネントの結合方法を定義している.

Struts を用いた Web アプリケーションの大まかな処理の流れは図 1 のようになる．まず，ユーザからのリクエストを受け付けると，アクションサーブレットは，設定ファイル struts-config.xml を参照し，ユーザの入力情報を格納するクラス（アクションフォーム Beans）を決定してデータを保存する．次に，同様に struts-config.xml を参照することで，アクションサーブレットはリクエストを処理するためのアクションクラスを特定し，これを呼び出す．呼び出されたアクションクラスは，アクションフォーム Beans を参照することでユーザの入力情報を取得し，目的の処理を行って，結果をステータスとして返す．最後に，アクションサーブレットはアクションクラスから返されるステータスをもとに，struts-config.xml を参照して次の遷移先を決定し，ユーザ画面として表示する.

つまり，開発者は，表示画面となる JSP ファイル，ユーザの入力情報を保存するアクションフォーム Beans，リクエストを処理するアクションクラスといった個々のコンポーネントを作成し，最後にアクションサーブレットの動作を決める struts-config.xml を適切に設定することで，Web アプリケーションを開発することができる.

2.3 struts-config.xml と画面遷移図

Struts を用いた Web アプリケーションでは struts-config.xml と呼ばれる設定ファイルが個々のコンポーネントを結合する．これは図 2 で表されるような XML ファイルとして記述され，以下の設定項目により画面遷移の基本部分を実現する.

- `<form-bean>`タグ
  アプリケーション中で利用されるアクションフォーム Beans の名前 `name` と実際のクラス名 `type` を対応付ける.
- `<action>`タグ
  リクエストを受けるパス `path` とそのリクエストを処理するアクションクラス `type` を対応付ける．入力フォームのデータが渡される場合，そのデータを格納するアクションフォーム Beans 名 `name` とスコープ `scope` を設定する．また，アクションクラスによる処理を介さず画面を遷移させるには `forward` 属性を設定する.
- `<forward>`タグ
  アクションクラスが処理後に返すステータス `name` と遷移先 `path` を対応付ける.

例えば，図 2 は 3 つの画面（スタート画面 `welcome.jsp`，ログイン画面 `logon.jsp`，認証済み画面 `mainMenu.jsp`）の存在を仮定し，これらの画面間の遷移を表現している．`<form-bean>`はアクションフォーム Beans を 1 つ用意し，1 つ目の `<action>`は，パス/Logon からログイン画面への無条件の遷移を表す．また，2 つ目の`<action>`は，パス/SubmitLogon に対する遷移先がユーザの入力情報とアクションクラスの処理結果に依存して決定され，結果が `success` ならば認証済み画面へ，`failure` ならばログイン画面へ振り分けられることを意味する．なお，どの画面がどのパスへリクエストを送るかは画面を表す JSP ファイルへ記述され，struts-config.xml 中には記述されない.

このように，Struts では画面から画面へ直接遷移させるのではなく，一度アクションを処理するノードを経由してから次画面への遷移が行われる．そのため，画面遷移を設計する際には最終的に struts-config.xml として記述されることを意識した設計

```
<struts-config>
...
<form-beans>  <!--アクションフォーム Beans の設定-->
  <form-bean name="LogonForm"
             type="org.example.LogonForm" />
</form-beans>
...
<action-mappings>  <!--アクションクラスの設定-->
  <action path="/Logon" forward="/logon.jsp" />
  <action path="/SubmitLogon"
          type="org.example.LogonAction"
          name="LogonForm"
          scope="request" />
    <forward name="success" path="/mainMenu.jsp"/>
    <forward name="failure" path="/logon.jsp" />
  </action>
  <action path="/Logoff"
          type="org.example.LogoffAction">
    <forward name="success" path="/welcome.jsp"/>
  </action>
</action-mappings>
...
</struts-config>
```

図 2: struts-config.xml の例

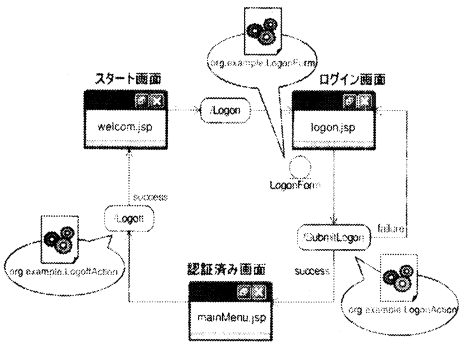

図 3: 図 2 に対応する画面遷移図

が必要となる．例としては，図 3 のような画面遷移図が考えられる．画面遷移図をどのような仕様で作成するかは Struts によって定められているわけではないが，Struts を用いたソフトウェア開発では画面遷移図の設計が struts-config.xml に強く影響することに注意しなければならない．

3 メタモデルと MOF

ここでは，ソフトウェア開発において，成果物とその依存関係のメタモデルを定義する必要性について述べ，本研究で着目したメタモデリング言語 MOF について紹介する．

3.1 メタモデルの重要性

ソフトウェア開発で扱われる情報は多様で，個々の情報はあらかじめ定められた記述仕様に従うモデルとして表現される．しかし，仕様の中には自然言語で定義されたものや，必要以上に条件の緩いものが存在し，本来意図した内容よりも曖昧なモデルの作成が可能になる．そのため，作成したモデルが，管理したい情報を正しくかつ厳密に表現できていることを保証できないという問題がある．

そこで，モデルによって表現される意味論を形式的に定義し，意味解釈の曖昧さを取り除くためのメタモデルが必要となる．メタモデルは，モデルとして記述すべき，あるいは記述すべきでない内容を定義し，作成されたモデルが意図した情報を正しく表現していることを保証する．つまり，管理する情報に対する仕様としてメタモデルを定義することで，実際に作成されたモデルの正当性を検証できる．

3.2 作成すべきメタモデル

厳密なメタモデルを定義した上で管理しなければならない対象に，成果物とその依存関係がある．

成果物は設計，実装に関わらずソフトウェア開発において最も重要な生産物であり，要求される仕様を満たすことが保証されなければならない．特に，フレームワークを用いた開発を行う際には，フレームワークの枠組みに従った成果物の作成が必須となる．そのため，個々の成果物の仕様としてメタモデルを定義し，成果物を管理する必要がある．

一方，成果物間の依存関係（トレース情報）も厳密な仕様のもとで管理する必要がある．トレース情報の仕様とは，互いに依存関係を保持した状態で管理しなければならない要素対の定義であり，トレース情報はそのような要素対に実際に引かれた関連である．つまり，依存関係をもつべき要素対をメタモデルとして定義することで，実際に存在すべき依存関係が正しく保持されているかを検証可能にする．

3.3 MOF と MOF リポジトリ

オブジェクト指向設計に基づいて自由にメタモデルを定義するための枠組みを提供するために，MOF（Meta Object Facility）[8][9] と呼ばれるメタモデリング言語が OMG（Object Management Group）標準として提案されている．MOF は UML（Unified Modeling Language）[10] のクラス図に似た意味論をもつが，その表記法は定義されてい

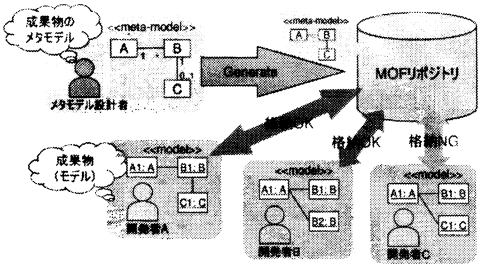

図 4: MOF リポジトリの生成とモデルの管理

ないため UML クラス図を用いて記述されることが多い．UML が同じ性質をもつオブジェクトを UML クラスとして抽象化するのに対し，MOF は同じ性質をもつメタオブジェクトを MOF クラスとして抽象化する．メタオブジェクトとは他のオブジェクトの性質を規定するためのオブジェクトであり，UML クラスに相当すると考えられる．

また，MOF によって記述したメタモデル（以降，MOF メタモデル）に対して，そのインスタンスであるモデルを蓄積・管理するためのリポジトリを MOF リポジトリと呼ぶ．与えられた MOF メタモデルから MOF リポジトリの API を生成する標準規則として MOF to IDL Mapping[8] や MOF to Java Mapping である JMI（Java Metadata Interface）[6] が提唱されており，その実装として Medini[5] などが存在する．

図 4 で表されるように，ある成果物のメタモデルを MOF で定義することで，その MOF メタモデルに従ったモデルのみを管理可能なリポジトリをあらかじめ生成することができる．これにより，開発者がメタモデルに従わない誤ったモデルを成果物として作成した場合や，あるいは修正によってメタモデルを満たせなくなってしまった場合にも検出が可能となり，設計ミスを防止できるようになる．例えば，図 4 中の開発者 C が作成したモデルは，クラス A1，C1 の間に関連が記述されているが，これはメタモデル中でメタクラス A，C 間に関連が定義されないという仕様に矛盾する．

4 Struts に対するメタモデル

5 節で述べるトレーサビリティの実現手法に必要な，画面遷移図，struts-config.xml，およびそのトレース情報のメタモデルを定義する．ここでは，Struts フレームワークにおける画面遷移の基本部

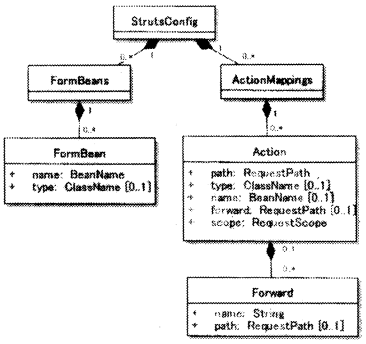

図 5: struts-config.xml の MOF メタモデル

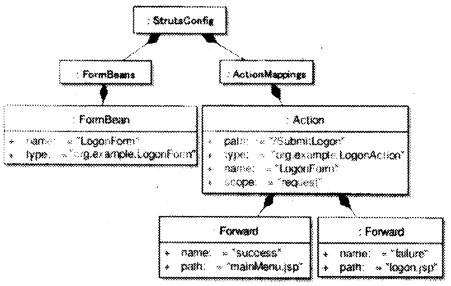

図 6: struts-config.xml のインスタンス例

分にのみ着目し，図 2，3 の例をもとにその概略を説明する．なお，実際の struts-config.xml の仕様は本論文で述べるよりも広く，それに伴って各メタモデルの定義もより詳細なものにする必要がある．しかし，ここで述べるメタモデル定義は Struts の基本部分をカバーしているため，詳細化を進める際にはそのまま利用可能である．

4.1 成果物のメタモデル

成果物のメタモデルとして，struts-config.xml と画面遷移図に対する MOF メタモデルを定義する．

4.1.1 struts-config.xml のメタモデル

struts-config.xml は XML ファイルであり，そのメタデータ構造は DTD（Document Type Definition，文書定義型）を用いて定義されている．しかし，提案手法では全てのメタモデルを MOF で記述し，成果物を MOF リポジトリの上で統一的に管理することを目的としているため，struts-config.xml のメタモデルも MOF を用いて定義する．

図 5 は struts-config.xml のメタモデルのうち，図 2 で表されるアクションフォーム Beans とアク

ションクラスの記述に対応する部分である．同メタモデルはXML文書における要素をメタクラスとして，属性をメタクラスの属性として定義し，要素間の親子関係をメタクラス間の集約関係として保持する形で定義している．

メタモデルのインスタンスとして，図2相当部分を図6に示す．この例では，図2中の<form-bean>と2番目の<action>の記述に関する部分（図3のlogon.jsp画面からmainMenu.jsp画面への遷移部分）のみを抜粋している．図2と図6では表現形式が異なるが，図2は図6で表されるモデルのテキスト表現に過ぎず，意味として記述しようとしている内容は同一である点に注意されたい．

4.1.2 画面遷移図のメタモデル

画面遷移図のMOFメタモデルはUMLのアクティビティ図を拡張して定義する．図7上部の3つのメタクラスは，アクティビティ図のメタモデルの中心部分を定義している．具体的には，状態遷移を記述するために，ノード（ActivityNode）と辺（ActivityEdge）を定義し，ノードとノードを辺で接続することによって遷移関係を表現する．また，遷移条件として，辺には遷移可能性を判定するためのガード条件（ValueSpecification）が関連付けられている．実際のアクティビティのメタモデルでは，さらに意味論を厳密に定義するためにノードや辺の種類を詳細化したり，イベントのトリガを定義したりということが行われているが，ここでそれらを全て説明することは不可能であり，本質からも外れるため省略する．

2節で述べたように，Strutsでは画面と画面を直接遷移可能な状態で結合するのではなく，アクションサーブレットを介して入力データはアクションフォームBeansへ，処理はアクションクラスへ振り分けられる．そして，その処理結果を受けて次の遷移画面が決定するという構造になっている．つまり，Strutsに適した画面遷移図とは，画面と画面を直接接続するようなモデルではなく，画面から処理ノードへ接続し，処理ノードから次画面へ接続するという，図3のようなモデルが良いと考えられる．そこで，提案するメタモデルでは画面と処理ノードをモデル化するためにDisplayNodeとActionNodeを定義している．DisplayNodeは

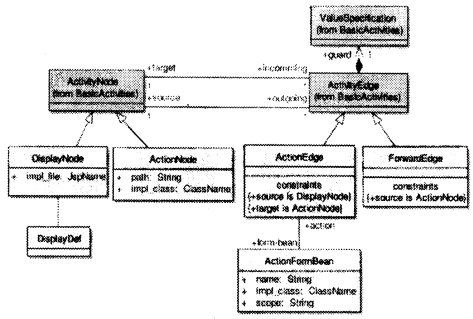

図7: 画面遷移図のMOFメタモデル

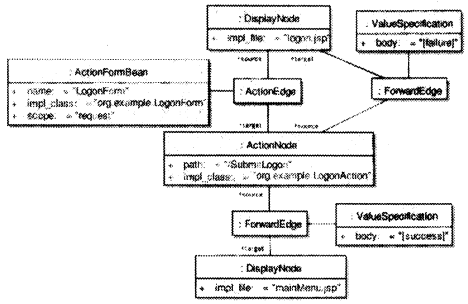

図8: 画面遷移図のインスタンス例

画面を表し，それを実現するJSPファイルの情報を保持する．一方，ActionNodeは処理ノードを表し，このノードへのパスやリクエストを処理するアクションクラスの情報を保持する．また，テキストボックスやボタンの表示レイアウトなど画面に関する詳細な設計をDisplayDefとして遷移モデルから分離させている．

さらに，これらのノードを接続する2種類の辺を定義する．1つは，画面ノードDisplayNodeから処理ノードActionNodeへのリクエストを表す辺ActionEdgeである．リクエストが送られる際には，画面からユーザの入力情報が渡されうるので，辺ActionEdgeと関連させる形でActionFormBeanを定義し，実際にデータを保存するためのクラスに関する情報を保持させる．もう1つの辺はActionNodeからの遷移を表す辺ForwardEdgeである．アクションクラスの処理結果によって遷移先は複数存在しうるので，その場合は遷移条件をValueSpecificationとして記述し，複数のForwardEdgeによって遷移先を分岐させる．

メタモデルのインスタンスとして，図3に相当

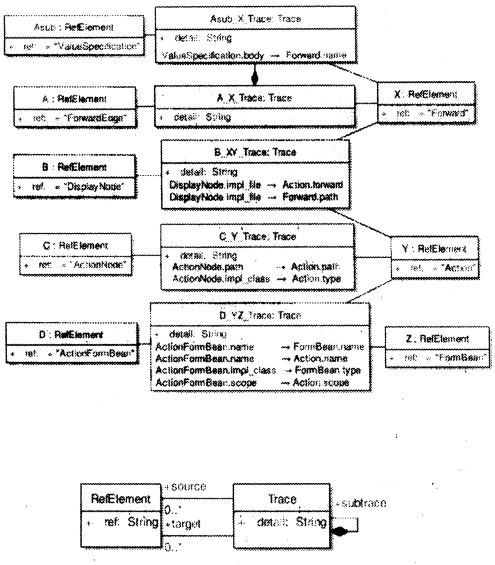

図 10: トレース情報のメタメタモデル

する部分を図 8 に示す．図中では遷移条件として便宜上 2 つの ValueSpecification を定義しているが，ValueSpecification は抽象クラスであるため，厳密にはそのサブクラスのインスタンスが Forward-Edge と関連付けられることを断っておく．

また，ここでも図 3 と図 8 で表現形式が異なるが，struts-config.xml の場合と同様に図 3 は図 8 で表されるモデルの一表現に過ぎない．ただし，直接図 8 のようなモデルを記述するのは困難であるため，開発者はモデリングツール上で図 3 のような画面遷移図を描き，その背後で厳密な意味定義として図 8 が作成されることを想定している．

4.2 トレース情報のメタモデル

次に，成果物間の依存関係（トレース情報）に対しても MOF メタモデルを定義する．ここでは，一方のモデル要素集合の変更が他方のモデル要素集合へ影響を与えるような n 対 n の依存関係を定義することを目指している．画面遷移図と struts-config.xml のトレース情報に対するメタモデルは図 9 のように定義し，このメタモデルをさらに一般化すると図 10 のような構造になっている．なお，ある依存関係がより詳細ないくつかの依存関係で構成される，というような構造的な関係を定義できるようにするため，トレース情報は別のトレース情報を保持しうるようになっている．

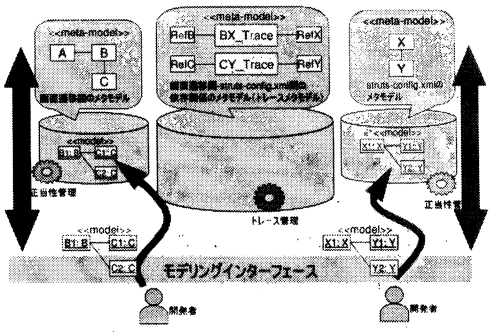

図 11: 成果物の正当性管理

例えば，図 9 の上 2 つの依存関係は，画面遷移図メタモデル中の ForwardEdge が，struts-config.xml メタモデル中の Forward に影響を与えることを意味する．そして，この依存関係をより詳細化する形で，ガード条件 ValueSpecification が Forward に影響を与えることを定義している．なお，ValueSpecification の body 属性が Forward の name 属性に影響する，というような属性レベルでの詳細な依存関係は，detail 属性として記述する．

5 メタモデルに基づくトレーサビリティの実現手法

先に定義した 3 つのメタモデルを用いて，Struts フレームワークにおけるトレーサビリティを実現する手法を述べる．提案手法は，各メタモデルに対する MOF リポジトリをあらかじめ生成し，

- 個々の成果物の正当性管理
- トレース情報を用いたトレース管理

を行うことによってトレーサビリティを実現する．また，トレーサビリティの実現により，成果物間の一貫性保証が可能となる．

5.1 個々の成果物の正当性管理

3 節で述べたように，画面遷移図と struts-config.xml に対して定義した MOF メタモデルからそれぞれ対応する MOF リポジトリを生成可能である．生成されたリポジトリはそのメタモデルを満たすモデルのみを管理可能であるため，開発者が作成・修正したモデルが仕様を満たさない場合には検出が可能となる．つまり，図 11 で表されるように，開発者が編集するモデルと，そのメタモデルとの縦方向の関係を調べることで，成果物

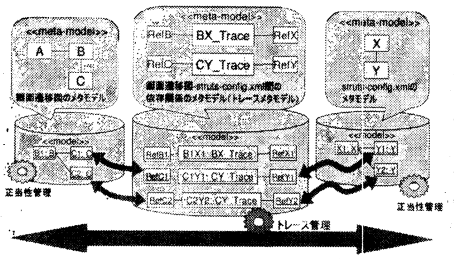

図 12: 成果物間のトレース管理

がその仕様を満たしていることを検証可能である．これにより，リポジトリに蓄積・管理されている画面遷移図や struts-config.xml が，フレームワークの仕様を満たした状態で作成されていることが保証できる．

5.2 トレース情報を用いたトレース管理

4 節で定義したトレース情報のメタモデルに対して，トレース情報を蓄積・管理する MOF リポジトリを生成可能である．このとき，図 12 で表されるように，画面遷移図と struts-config.xml，およびそのトレース情報がメタモデルに基づいて正しく蓄積されていると仮定する．すると，画面遷移図に変更があった場合には，トレース情報を参照することで，struts-config.xml への影響範囲を特定可能である．つまり，開発者が編集する成果物のモデルと，そのトレース情報との横方向の関係を調べることで，個々の成果物の変更影響を管理し，トレーサビリティを実現することができる．

なお，トレース情報は，開発者ではなくツールが自動的に作成・管理することを想定している．実際の開発においては，図 12 のように正しくトレース情報が管理されている状況だけでなく，一貫性が保証できない状況も考えられる．例えば，設計成果物である画面遷移図のみが作成された場合には，対応する struts-config.xml が作成されていないため，メタモデルを満たす形でトレース情報を作成できない．あるいは，画面遷移図に変更や削除があった場合にも，その影響を受ける struts-config.xml に修正が行われていないという状況が考えられる．しかし，このような場合にもツールはメタモデルに矛盾した状態であることを検出可能であるため，開発者に修正を強要し，一貫性を保つことができる．

6 まとめ

本論文では，Struts フレームワークにおける画面遷移図と struts-config.xml に対して MOF メタモデルを定義し，さらにその依存関係もメタモデルのもとで管理することで，トレーサビリティを実現する手法を提案した．提案手法は，Struts フレームワークの全仕様をカバーしているわけではないが，ここで定義した MOF メタモデルは今後詳細化を進める上での基礎となる．

今後は各メタモデルのさらなる詳細化を進め，成果物の一貫性を保証しながら開発を行うためのモデリングツールを実装し，具体的な評価実験と考察を行う予定である．

参考文献

[1] Akehurst, D. H. and Kent, S.: A Relational Approach to Defining Transformations in a Metamodel, *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, London, UK, Springer-Verlag, pp. 243–258 (2002).

[2] Apache Struts Project: `http://struts.apache.org/`.

[3] 我妻智之，神谷慎吾，大平直宏，松下 誠，楠本真二，井上克郎：メタモデルに基づくトレーサビリティ技術の提案，電子情報通信学会信学技報，Vol. 105, No. 270, pp. 25–30 (2005).

[4] Bondé, L., Boulet, P. and Dekeyser, J.-L.: Traceability and Interoperability at Different Levels of Abstraction in Model Transformations, *Forum on Specification and Design Languages(FDL'05)*, Lausanne, Switzerland (2005).

[5] IKV^{++} Technologies AG: `http://www.ikv.de/`.

[6] Java Metadata Interface (JMI): `http://java.sun.com/products/jmi/`.

[7] JavaServer Faces (JSF): `http://java.sun.com/j2ee/javaserverfaces/`.

[8] Object Management Group: *Meta Object Facility (MOF) Specification, Version 1.4* (2002). `http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf`.

[9] Object Management Group: *Meta Object Facility (MOF) 2.0 Core Specification* (2003). `http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf`.

[10] Object Management Group: *UML Superstructure Specification, Version 2.0* (2005). `http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf`.