ソフトウェア工学 130−15
（2001．3．23）

文書間の不整合解消に基づくソフトウエアプロセスのモデル化

落合 秀俊, 結縁 祥治, 阿草 清滋
名古屋大学工学研究科情報工学専攻

概要

本稿では，ソフトウエア開発における作業は文書間の不整合解消である，というアプローチから，文書間の整合性記述に基づくソフトウエアプロセスのモデル化手法を提案する．

文書と文書間の関係をハイパーテキストを用いて表現し，リンクに対してリンク先の文書内容を用いた整合条件と不整合解消動作の２つを付加した，文書整合モデルを提案する．本モデルを用いた整合性管理手法を提案する．ソフトウエアプロセスは一連の不整合解消動作として解釈される．

不整合解消に基づくモデル化手法は作業手順を記述するプロセス記述手法に比べ柔軟で簡潔であることを示す．

Modeling Software Processes based on the Document Inconsistency Resolution

Hidetoshi Ochiai, Shoji Yuen, Kiyoshi Agusa
Dept. of Information Engineering, Nagoya University

Abstract

This paper presents a modeling method for software processes based on the consistency management of software documents. In view that each activity in a software process is essentially controlled to maintain the consistency, we propose an explicit consistency description based on the hypertext: an object represents a document and a link represents a relation with a consistency condition and an action to keep the connected documents consistent. A software process is modeled as a sequence of activities to derive the consistency between the documents involved in the development.

We propose an inconsistency resolution algorithm for consistency descriptions to give a concrete software process. Following the algorithm, it is shown by examples that our modeling method gives more flexible and succinct process descriptions than the existing activity-oriented descriptions.

1 はじめに

近年，ソフトウエアの開発プロセスをモデル化しプロセス記述に沿って開発を行うことで開発の効率化と品質向上を目指す，ソフトウエアプロセスの研究がなされている．

従来，作業の状態遷移を記述する手法や手続き型言語を用いる作業指向のプロセス記述手法が多く用いられている [1]．作業指向のプロセス記述手法では起り得る全ての作業手順をプロセスに記述しようとすると記述が複雑になり，可読性を上げるために中心的な戦略のみを記述すると開発がプロセス記述から逸脱する場合が多くなり開発の実状と合わない．

このような問題点に対し，本稿ではソフトウエア開発における作業は文書の修正であるという観

点に注目する．ソフトウエア開発に過程でソースコード，取り扱い説明書，要求仕様書，設計仕様書，テスト報告書，プロジェクトマネージャからの指示メールなどの文書について作成と修正が行われる．これらの文書は互いに独立しているわけではなく文書間に複雑な関係が存在している．開発の過程で文書間の整合性を維持する時に文書修正が発生している．

本稿では，ソフトウエア開発における作業は文書間の不整合解消である，というアプローチから，文書間の整合性記述に基づくソフトウエアプロセスのモデル化手法である文書整合モデルを提案し，整合性管理手法を示す．文書整合モデルによるソフトウエアプロセスの記述を行うことで例外的な状況に対し特別な記述を行うことなく対応できるプロセス記述が可能になる．

2 ハイパーテキストによる文書整合モデル

2.1 ハイパーテキスト

ハイパーテキストとは，関係のある文書の断片をリンクで接続することで構成された文書である．

ソフトウエア開発では，関係のある文書を参照する場合が多い．例えば，ソースコードを読む時に仕様書を参照することは多い．そのため，ソフトウエア開発に用いられる文書をハイパーテキストに対応付けることは自然である [3][4]．

本稿では以下で説明するハイパーテキストモデルを用いることにする．構成要素は以下の 3 つである．

オブジェクト　文書を表す．文書は URI 等の ID で一意に識別される．文書の種類によって必要があれば文書内部の断片を指定する方法を提供する必要がある．

リンク　1 つ以上の文書断片を関連付けをする．文書断片にはリンク内で一意なロール名を付ける．文書断片は (ロール名, 文書への参照, 文書内部の断片) の 3 項組で指定する．

グラフ　オブジェクトとリンクで構成されるひとまとまりの文書群を表す．

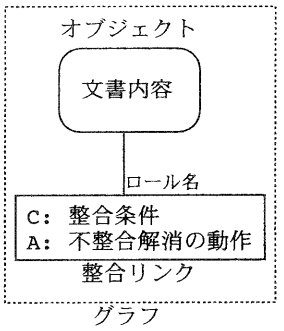

図 1: 文書整合モデルの図表記

1 つのオブジェクトが複数のグラフに所属しても良い．同じようにリンクも複数のグラフに所属しても良い．グラフは処理対象として注目するオブジェクトとリンクを定める．

2.2 文書整合モデル

文書整合モデルとは，ハイパーテキストにおけるリンクに文書間の整合性に関する情報を付加することで文書間の整合性を記述したグラフのことである．

リンクには，整合条件と不整合解消のための動作の 2 つを付加する．整合性の情報を付加したリンクを整合リンクと呼ぶ．

文書整合モデルを図形で表記を行う場合，以下の表記法を用いる．

オブジェクト　角の丸い四角．

リンク　実線の四角と，四角からオブジェクトへ引く実線．四角の内部には整合条件と不整合解消の動作を書く．線にはロール名を付ける．

グラフ　破線の四角．

文書整合モデルの図表記は図 1 のようになる．

2.2.1 整合条件

整合条件とは，リンクが関係づける文書断片の内容が満すべき条件である．整合条件は文書断片を値として用いる論理式として記述する．

グラフにおいて整合条件を評価した結果，真ならば整合，偽ならば不整合と定義する．

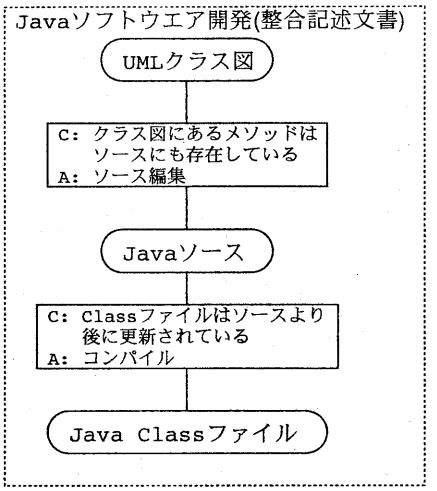

図 2: 整合記述文書の例

2.2.2 不整合解消の動作

整合条件を評価した結果が偽の場合に不整合を解消するための動作をリンクに付加する．不整合解消の動作には，リンクでつながれた文書を修正する作業の手順を記述する．

2.2.3 整合記述文書

文書を表すオブジェクトと整合リンクで構成されるグラフを整合記述文書と呼ぶ．

整合記述文書の整合が取れているとは，整合記述文書に含まれているリンクの整合条件が全て真である状態である．

図 2 は UML クラス図，Java プログラム，Java Class ファイルとその間の整合条件をモデル化した整合記述文書の例である．

3 整合性管理手法

本稿では文書整合モデルの XML 表記を示す．さらに XML 表記に基づきソフトウエア開発における文書の整合性管理手法を示す．

3.1 整合記述文書の XML による表記方法

3.1.1 オブジェクト

オブジェクトは XML 文書とする．XML 文書であれば形式は任意である．

```
<!ELEMENT graph (object*)>

<!ELEMENT object (process*, to*)>
<!ATTLIST object id ID #REQUIRED>
<!ATTLIST object href CDATA #REQUIRED>

<!ELEMENT to EMPTY>
<!ATTLIST to id IDREF #REQUIRED>
<!ATTLIST to role NMTOKEN #IMPLIED>
<!ATTLIST to fragment CDATA #IMPLIED>

<!ELEMENT process (#PCDATA)>
<!ATTLIST process type NMTOKEN #IMPLIED>
```

図 3: グラフの DTD

3.1.2 グラフ

グラフは図 3 の DTD に基づく XML 文書である．

グラフに所属するオブジェクトにはグラフ内で一意の ID を付ける．グラフ内でオブジェクトを参照する場合はこの ID を用いる．オブジェクトの実体はグラフ内には置かず URI で外部のリソースを指定する．

object エレメント内の to エレメントは，オブジェクトから他のオブジェクトを参照していることを示している．to エレメントの role はオブジェクト内で to を一意に決定する ID である．role が省略された場合は id と role は同じになる．fragment は to の指すオブジェクト内部の一部分を表す XPath[6] である．

オブジェクトに対して適用可能な操作を object エレメント内の process エレメントに記述する．図 4 では操作を行う Java のクラス名を記述している．

図 4 は図 2 のグラフを XML 表記した例である．

3.1.3 整合リンク

リンク先のオブジェクトは to エレメントの role を用いて指定する．

整合リンクの DTD を図 5 に示す．整合条件の変数定義部分は XSLT[7] を用いているため，整合リンクは well-formed XML 文書である．図 6 は UML クラス図と Java ソースの間の整合条件の例である．

整合条件 整合条件は整合リンクの consistency エレメントとして記述する．内部は変数定義と条件

```
<?xml version="1.0" encoding="EUC-jp"?>
<!DOCTYPE graph SYSTEM "Graph.dtd">
<!-- 整合記述文書 -->
<graph>
  <!-- クラス図 -->
  <object id="class_diagram"
   href="file:/objects/doche.xml"/>
    :
  <!-- Javaソース -->
  <object id="java"
   href="file:/objects/Graph.java"/>
  <object id="javaml"
   href="file:/objects/Graph.java.xml"/>
    :
  <!-- クラス図とJavaソースの関係 -->
  <object id="class_diagram-java"
   href="file:/objects/class_diagram-java.xml">
    <process type="execute">jp.ac.nagoya_u.nuie
.agusa.doche.client.CLinkChecker</process>
    <process type="action">jp.ac.nagoya_u.nuie.
agusa.doche.client.CLinkAction</process>
    <to id="class_diagram"
       fragment='/UXF/Package/ClassDiagram/Clas
s[Name="Graph"]'/>
    <to id="javaml"
       fragment='/java-source-program/class[@na
me="Graph"]'/>
    <to id="java"/>
  </object>
    :
    :
</graph>
```

図 4: グラフの例

```
<!ELEMENT clink (consistency,action)>
<!ELEMENT consistency (variable*,condition)>
<!ELEMENT variable ANY>
          <!-- 実際はXSLT断片が入る -->
<!ATTLIST variable name ID #REQUIRED>
<!ELEMENT condition (#PCDATA)>
<!ELEMENT action (unix|execute)*>
<!ELEMENT unix (#PCDATA|arg)*>
<!ELEMENT arg EMPTY>
<!ATTLIST arg role NMTOKEN #REQUIRED>
<!ATTLIST arg type NMTOKEN #REQUIRED>
<!ELEMENT execute EMPTY>
<!ATTLIST execute role NMTOKEN #REQUIRED>
<!ATTLIST execute type NMTOKEN #REQUIRED>
```

図 5: 整合リンクの DTD

部に 2 つで表す.

変数定義の variable エレメントの内部は XSLT の xsl:variable エレメントのコンテントである変数バインドエレメントを用いる. 変数定義の内部ではリンク先の XML 文書の内容が参照できる. 例えば, 図 6 では`$class_diagram` と書くことでクラス図の内容が得られる.

条件部には XPath 式を用いる. リンク先の XML

```
<?xml version="1.0" encoding="EUC-jp"?>
<clink>
 <!-- クラス図とJavaソースとの間の整合性
  クラス図にあるメソッドはソースにも存在している -->
 <consistency>
  <variable name="model_method"
   xmlns:axsl="http://www.w3.org/1999/XSL/Tran
sform">
   <axsl:for-each
     select="$class_diagram/Operation">
    <axsl:variable name="opname" select="Name"
/>
    <axsl:variable name="method"
     select="$javaml/method[@name=$opname]/@n
ame"/>
    <axsl:if test="boolean($method)">
     <method><axsl:value-of select="$method"/>
</method>
    </axsl:if>
   </axsl:for-each>
  </variable>
  <condition>count($model_method/method) =
   count($class_diagram/Operation)</condition>
 </consistency>
 <action>
  <unix>/usr/bin/emacs <arg type="file" role="
java"/></unix>
 </action>
</clink>
```

図 6: 整合リンクの例

文書の内容は XSLT の変数として参照できる. XPath 式の値が boolean でない場合は boolean に変換される. XPath 式の値が整合条件の評価結果である.

不整合解消の動作 不整合解消の動作は整合リンクの action エレメントに記述する. action エレメントの子エレメントを順番に実行することで不整合を解消する. action エレメントには UNIX のコマンドとオブジェクトに対する操作の 2 種類が指定可能である.

3.2 整合記述文書に基づく文書管理手法

3.2.1 整合性管理アルゴリズム

整合性管理の処理単位は整合記述文書である. 整合記述文書に所属しない整合リンクは考慮せず, 所属する整合リンクのみ処理の対象とする.

整合記述文書について, 以下の手順で整合性管理をする

Step 1 整合記述文書に所属する全ての整合リンクの整合条件を評価する.

Step 2 評価結果が偽になった整合リンクの一覧

を作成する.

Step 3　もし一覧が空ならば終了.

Step 4　作業者が不整合解消したい整合リンクを1つ選んで実行を開始する. 不整合解消の動作が書いていない場合は人間が不整合解消を行う.

Step 5　不整合解消の動作が終了した, あるいは作業者が終了を整合性管理システムに伝えた時, 変更された文書を検出し変更された文書につながっている整合リンクの整合条件を評価する.

Step 6　Step2 に戻る.

3.2.2　整合記述文書の階層化

整合記述文書の階層化は, 不整合解消の動作に他の整合記述文書を呼び出すことで実現する.

新たに呼び出される整合記述文書には引数として文書を渡すことができる. 呼び出される整合記述文書の引数で指定した文書が置き換えられて実行される.

3.3　実装

本章の整合性管理手法に基づく整合性管理システムを実装した.

実装には Java 言語 (JDK1.3) を用い, XML パーサは Xerces Java Parser 1.2.0 を, XSLT プロセッサは Xalan-Java 1.2 を利用している. XML 文書の動的生成のために ruby による CGI を作成した.

実装規模は Java が 892 行, XSLT スタイルシートが 111 行, 全体で 1196 行である.

整合記述文書の URI を指定してシステムを起動すると, 整合記述文書内の整合リンクを評価して不整合になったリンクの一覧を表として出力する. 利用者が不整合を 1 つ選ぶと不整合解消の動作が起動するようになっている.

3.3.1　整合条件の評価

XSLT を用いて整合条件の評価を行う. 評価対象の整合リンクと所属するグラフを全体の入力として与える. 最終評価結果は `<result value="true"/>` か`<result value="false"/>` のどちらかのエレメント 1 つのみからなる文書である.

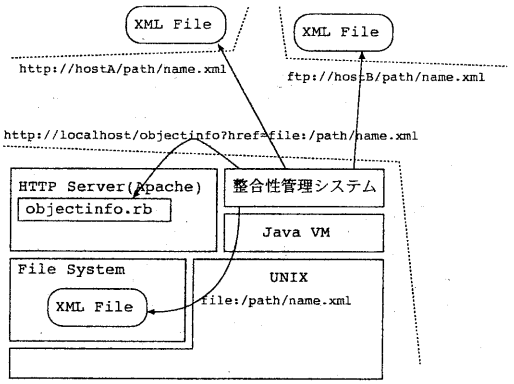

図 7: XML 文書のアクセス方法

3.3.2　文書形式

本システムが対象とする文書へのアクセスは URI で指定する. (図 7).

本稿の中で例に用いている文書には以下の XML 形式を利用している.

UML モデル図　UML モデル図を XML 形式にした文書. UML の XML フォーマットは UXF[8] とする.

Java ソース　Java ソースを JavaML[9] で XML に変換した文書. JavaML は Java プログラムの入れ子構造を XML のエレメントに対応付ける.

ファイル情報　ファイル属性の XML 文書. 本システムでは所有者, 更新日時を XML 文書化する.

4　不整合解消によるソフトウエアプロセス

4.1　プロセス記述の手法

ソフトウエア開発における作業の流れを手続き的に記述するのではなく, 整合記述文書における不整合解消の連鎖によって間接的に記述する.

以下でプロセス記述に用いられる典型的な文書と整合条件を説明する. これらを組み合わせてプロセス全体の記述を行う.

4.1.1　依存関係

文書間の依存関係から以下のように文書更新プロセスが表現できる. A という文書が B に依存し

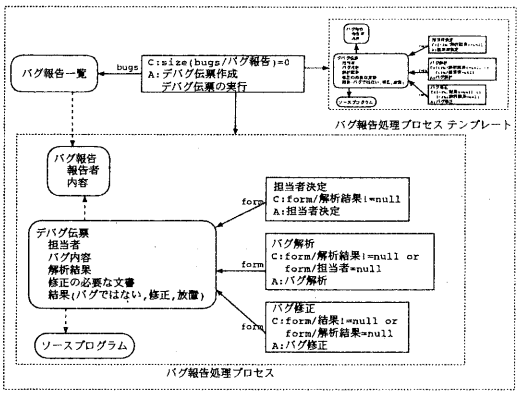

図 8: バグ報告処理プロセスの例

ていて B に C が依存している場合，A が更新になると AB 間が不整合となり，B を修正して不整合を解消する必要が出てくる．B を更新すると BC 間が不整合になるので，C を修正して不整合を解消する．このように不整合解消の連鎖が依存関係に沿って起こることで文書更新プロセスとなる．

文書間の依存関係はソフトウエア開発に多く含まれる．例えば図 2 のような，ソースプログラムとコンパイル結果の機械コード，モデル図とソースプログラムなどがある．

依存関係は整合条件として文書の更新日時を比較することが多い．この場合には文書本体ではなくファイル情報 XML を整合条件に利用する．

4.1.2 伝票を用いた作業手順

処理の進捗状況を記入する伝票の項目に関する整合条件を用いて作業手順を表現する．

図 8 はデバグ伝票を用いたバグ報告処理プロセスの例である．伝票を用いたプロセスは 'バグ報告処理プロセス' グラフである．

'バグ報告一覧' にバグ報告が存在している場合は 'バグ報告処理プロセス テンプレート' をコピー・修正して 'バグ報告処理プロセス' を作成し実行を開始する．

図 9 は 'バグ報告処理プロセス' の動作の 1 つである．

1. 初期状態の 'デバグ伝票'(図 9-a) は伝票作成時

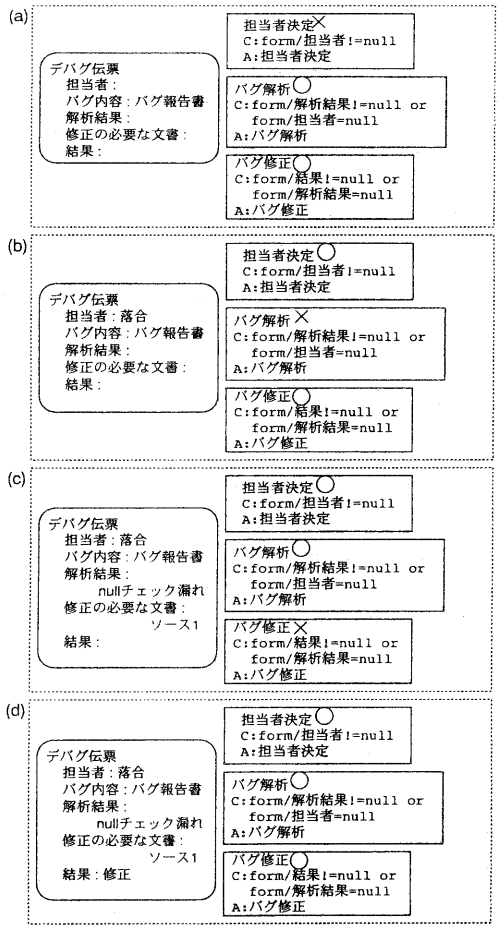

図 9: バグ報告処理プロセスの動作例

に 'バグ内容' が埋められ，'返答の必要性' はデフォルトで偽になっている．この状態で整合リンクの '担当者決定' が偽で他は真であるので，担当者を決定して伝票に書き込む．

2. '担当者決定' が真になり，代わりに 'バグ解析' が偽になる (図 9-b)．担当者はバグ報告書を参照しながらバグの原因を調べて '解析結果' を書く．'修正の必要な文書' も必要があれば書き込む (図 9-c)．
3. 'バグ解析' が真になり，'バグ修正' が偽になる．担当者がバグを修正し，修正結果を伝票に書き込む．

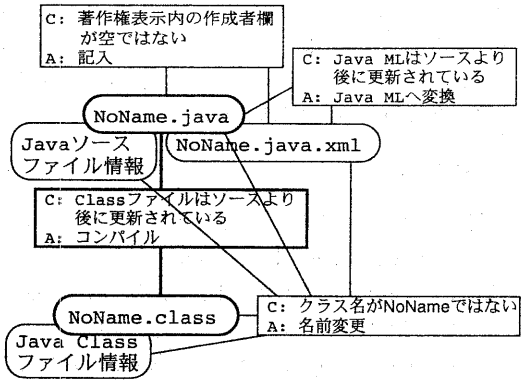

図 10: Java クラステンプレート

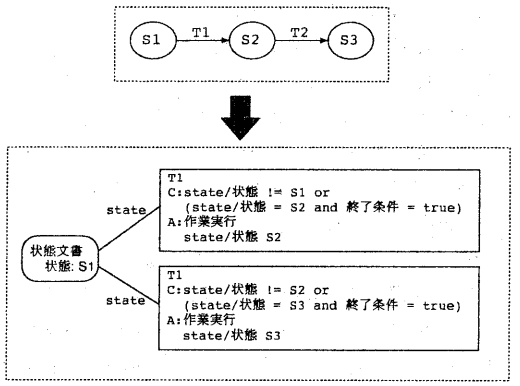

図 11: 状態遷移図を整合記述文書に変換する

4. 全ての整合リンクが真になる (図 9-d).

4.1.3 テンプレート穴埋めプロセス

頻繁に使用する文書や整合リンクをテンプレートとして準備しておき，テンプレートをコピーし一部を書き換えて使用することが良くある．テンプレートの書き換えるべき場所にあらかじめ次のような整合リンクを準備しておく．

- 初期状態で整合条件が偽
- 書き換える，あるいは空欄を埋めると整合条件が真になる

図 10 は Java ソース，クラスファイル，Java ソース–クラスファイル間の整合リンクのテンプレート例である．Java ソースにはプロジェクト共通の基底クラスを継承したクラス，コンストラクタ，著作権表示があらかじめ書いてある．初期状態でクラス名は NoName になっているので，「クラス名が NoName ではない」という整合条件は偽である．名前を付け直すと真になる．同じように，初期状態で作成者の欄が埋められていないので「著作権表示内の作成者欄が空ではない」という整合条件が偽で，記入すると真になる．

4.2 他のプロセス記述方法との比較

本節では文書の不整合解消によるプロセス記述手法と他のプロセス記述手法をソフトウエア開発の観点から比較を行う．

4.2.1 制御フローと制約

プロセス記述には制御フローと制約という 2 種類の観点がある [10].

おおまかなプロセスの流れは手続き型言語，状態遷移図を用いた方法では直接記述できる．しかし，制約が守られるようにプロセス内に制約チェックの作業を漏らさず入れることは難しい．

本稿で提案している整合記述文書によるプロセス記述は制約の記述には適しているが制御フローの記述にはあまり向いていない．しかし，伝票を用いた作業手順を用いれば実行順序を意識したプロセス記述が可能である．伝票を一般化すると状態文書になる．図 11 は状態遷移図を整合記述文書に変換する方法である．この方法を用いれば整合記述文書であっても状態遷移を記述することができる．

4.2.2 プロセスの図形表現

ソフトウエアプロセスは人間による作業が多く含まれるため，プロセスには可読性が要求される．図形を用いた表現があると直感的に分かりやすくなるため，プロセスの共通理解やプロジェクトの改善に役立つ．

状態遷移図やペトリネットの場合，図として表現可能することが多い．整合記述文書を用いたプロセス記述では，図 8 のようにオブジェクトとリンクによるグラフ構造を図示することが可能である．

4.2.3 例外的状況への対応

整合記述文書を用いたプロセス記述ではシステムが管理するのは整合条件のみであり，文書変更の順番は問われない．整合記述文書内に記述されていない文書書き換えが発生した場合でも，その時点で偽の整合リンクの不整合を解消することでプロセスを継続できる．

例えば，図 4.1.2 でバグ解析が終わった段階でシステム構造の大幅改訂が行われ，バグを含むモジュール自体が他のモジュールと融合する形で存在しなくなったために伝票の '解析結果' を消すという例外的な書き換えが発生しても，再度 'バグ解析' が偽となり解析をやり直すことが可能となる．手続き的にプロセスを記述していた場合，バグ修正からバグ解析に戻る流れを記述しておく必要があり，記述していない場合は対応できない．

5 おわりに

本稿では，ハイパーテキストのリンクを拡張した文書整合モデルと，文書整合モデルに基づくソフトウエアプロセスのモデル化手法を提案した．文書整合モデルの XML 表記に基づいた整合性管理手法を提案し，Java と XML ライブラリ (XML パーサ，XSLT プロセッサ) を用いて文書の整合性管理システムを作成し，ソフトウエアプロセスの生成方法を示した．

本稿の記述方法は作業を直接記述せず，作業が発生する理由である整合性を記述する．実際の作業は整合性管理アルゴリズムにより導出される．このように，作業の発生原因と作業の順序決定を分離することによって柔軟で簡潔な記述を与えることができる．特に，例外的な文書書き換えに対して特別な記述なしにプロセス記述が可能となり，実際の開発に適用した場合の記述量が手続き型の記述法に比べ減少した．

今後の課題は以下の通りである．現在の文書整合モデルは整合条件をインスタンスレベルで記述することにしているが，文書の種類ごとに整合条件を記述したい場合も多いので，クラスレベルでの記述も導入する必要がある．さらに相互に矛盾する整合条件をチェックする枠組みが必要である．整合性管理システムを実際の開発に適用した際の評価を行う必要がある．

なお，本研究は，科学研究費補助金 基盤 (A)10308008 からの援助を受けた．

参考文献

[1] 井上克朗, 松本健一, 飯田元: ソフトウエアプロセス, 共立出版, March 2000.

[2] Kaj Gronbaek,Randall H. Trigg: Hypermedia, CACM 37:2, February 1994, pp 26-86.

[3] J.Bigelow: Hypertext and CASE, IEEE Software, pp. 23-27, Mar. 1988.

[4] 大橋洋貴, 山本晋一郎, 阿草清滋: ハイパーテキストに基づいたソースプログラム・レビュー支援ツール, 電子情報通信学会ソフトウェアサイエンス研究会, Vol.98, No.28, pp.15–22, 1998.

[5] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler: Extensible Markup Language (XML) 1.0 (Second Edition), The World Wide Web Consortium, W3C Recommendation 6 October 2000.

[6] James Clark, Steve DeRose: XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999.

[7] James Clark: XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999.

[8] Junichi Suzuki, Yoshikazu Yamamoto: Making UML models exchangeable over the Internet with XML, UML'98.

[9] Greg J. Badros: 'JavaML: A Markup Language for Java Source Code', `http://www.cs.washington.edu/homes/gjb/JavaML/` .

[10] Gail E. Kaiser, Steven S. Popovich, Israel Z. Ben-Shaul: A Bi-Level Language for Software Process Modeling, CUCS-016-92, September 1992.