DITA によるソフトウェア関連文書とソースコードの統合管理環境の提案

金谷 祥平　伊藤 恵　大場 みち子　奥野 拓

一般的なソフトウェア開発では，各作業工程毎に成果物となるドキュメントを作成する．これらの多くは，ファイル形式や記述方式が開発プロジェクト毎に異なる場合が多い．その為，前工程のドキュメントの記述内容に対して，後の工程で変更が必要となる場合，各ドキュメントの対応箇所の追跡と，それらに対する整合性確保は大きな負担となる．本研究は，ソフトウェアドキュメントに記述された情報を再利用可能な形で管理することで，ソースコードを含む各成果物間の整合性と追跡性を確保する環境を提案する．方法として，DITA(Darwin Information Typing Architecture) を用い，ソフトウェアドキュメントを再利用可能な情報の単位で管理する．さらに，これらの要素を入力として各要素に対応したソースコードの自動生成，構文解析による変更の追跡を行うことで，各要素の対応付けを行う．

In software development processes, several documents are created in period of each development phases. Most of these have different file type and writing style for every projects. Thus if a documents created in a development phase are modified in later phases, developers spend time and efforts on keeping integrity and tracing to correspondence of all documents. This study proposes an environment for integrity and traceability of deliverables of each phases. As an approach of this proposal, the environment manages documents as serieses of reusable information with DITA (Darwin Information Typing Architecture). For correspondence of documents with source codes, the environment generates source codes from parts of documents which corresponds to program modules and traces changes with parse for source codes.

1　はじめに

1.1　背景

一般的なウォーターフォールモデルに基づくソフトウェア開発では，要件定義，方式設計，詳細設計，実装，テストと複数の工程を持ち，各工程毎に成果物が存在する．このうち，実装工程の成果物はソースコードである．また，要件定義から詳細設計の工程では，ソフトウェアの機能的，非機能的仕様について後の工程ほど具体的なドキュメントが作成される．そして，それらは基本的に人間が読む為に作成され，ファイル形式や文書としての体裁が開発プロジェクト毎に異なる場合が多い．コンピュータでこれらの文書管理を支援するツールとして，ファイル単位でのリポジトリ化を行うものはあるが，文書の内容や構造に踏み込んだ管理環境としては不十分である．また，ソースコードに関しては統合開発環境により独立して管理される傾向にあることから，上流のドキュメントとの間での追跡性や整合性の面で問題が生じ易い．一方，ドキュメントの作成方法に関しては，XML による構造化手法が考案されている．DITA(Darwin Information Typing Architecture) は XML に準拠した技術文書の規格である．DITA は情報をトピックという単位で管理し，特殊化という仕組みを導入する事で，トピックの再利用性を向上している．

Proposal of an Integrated Document Management Environment for Source Codes and Software Documents Using DITA.

Shohei Kanaya　Kei Ito　Michiko Oba　Taku Okuno, 公立はこだて未来大学システム情報科学部情報アーキテクチャ学科, Dept. of Systems Information Science, Future University Hakodate.

1.2 目的

本研究では，DITA が持つ情報の再利用性が，整合性維持におけるドキュメント要素の管理に適していると考え，これを用いてソフトウェアドキュメントとソースコードを統合的に管理する環境を提案する．これにより，ドキュメントとソースコードが持つ情報を関連付けし，それらの整合性・追跡性を確保することを最終的な目標とする．

2 関連研究

関連研究調査にあたり,「整合性維持」と「文書構造化」という二つのキーワードに焦点を当て，既存ツールや研究，そして，それらの課題について調査した．

2.1 整合性維持ツール

2.1.1 UModel

UModel とは，Altova により開発されているソフトウェアモデリングツールである [1]．UModel の主要な機能として，ソースコード自動生成とリバースエンジニアリングがある．UModel のソースコード自動生成は，UML(Unified Modeling Language) で定義されたソフトウェアモデルを入力として，Java, C#，VB.NET のソースコードを出力できる．

UML にはソフトウェアの構造を定義する静的モデルと，振る舞いを定義する動的モデルがある．UModel では，静的モデルであるクラス図とコンポーネント図によりプログラム中のクラス構造とその配置を定義し，動的モデルであるシーケンス図により各クラスメソッドの振る舞いを定義する．また，リバースエンジニアリングの機能として，既存のソースコードを UML モデルに変換することが可能である．

2.1.2 XML による整合性検査

XML で記述されたソフトウェアドキュメントとソースコードに対して，整合性を検査する研究が阿草らにより行われている [7]．この研究では，ソフトウェアドキュメント中に記述されたソースコード断片について，実際のソースコードに書かれているプログラム要素との整合性を検査するツール chkSpdDoc について述べられている．chkSpdDoc は CASE ツールプラットフォーム Sapid [8] のリリースに含まれており，ソースコードを取り扱う Sapid と，XML ドキュメントを扱う為の Dapid を用いて整合性を検査している．Dapid は XML 文書を対象とし，文書中のプログラム断片の解析を可能としている．文書中のソースコード断片のマークアップを行い，Dapid を介して関数名や型などの要素を取得する．また，ソースコード側の検査には，Sapid による構文解析を利用し，要素を取得する．Sapid は C，Java，JavaScript，JSP，HTML といった複数言語のソースコードを構文要素に分解し，それら構文要素と構文要素間の関連をリポジトリに格納する．Dapid，Sapid により取得された要素のうち，対応関係にあるものの一致を検査することで，ドキュメントとソースコードの不整合箇所を一覧できる．

2.2 整合性維持の課題

ソフトウェアモデルとソースコードの整合性維持が可能なツールでは，モデリング言語を用いることで視覚的かつ正確なプログラム要素との対応付けが可能である．一方で，ソフトウェア開発で作成されるドキュメント全般を管理する為の仕組みではなく，所定のモデル以外のドキュメントに記述された内容について整合性を確保することはできない．これを解決する為には，作成されるソフトウェアドキュメント全体を構造化する必要がある．

また，2.1.2 節で述べた整合性検査ツールでは，XML 化されたソフトウェア関連文書についての整合性が検査可能であるが，整合性の確保までを行うことはできない．

2.3 XML による文書構造化

ソフトウェアドキュメントには，日本語や英語の文章で記述された部分がある．これらをコンピュータにより処理する場合，XML が有効である．XML を用いる目的は，データ構造を目的に応じて拡張すること，そして，XSL 変換により HTML や PDF といった任意の形式のドキュメントとして出力することにある．XML による技術文書の規格として DITA と DocBook が存在する．

2.3.1 DocBook

DocBook は，OASIS により標準化されている XML 文書の規格である [5]．多くの技術文書に利用されており，表示形式に依存しない文章構造を定義することができるという特徴がある．RTF, man ページ，HTML といった形式で出力できる．

2.3.2 DITA

DocBook 同様，OASIS によって標準化されている XML 文書の規格として DITA(Darwin Information Typing Architecture) が普及しつつある [2]．DITA はトピックとマップという要素で構成されている．トピックとは，DITA における独立した情報の単位である．トピックはコンテキストに依存せず，複数の文書構造に対して情報を提供する．一方で，マップにはトピックの並びや階層を定義する (図 1)．これは，出力ファイルに反映される文書構造を表しており，独立したトピックに対してコンテキストを与える役割を持つ．

DITA を利用するメリットとして，情報の再利用性の向上が挙げられる．従来の文書は，章・節・項といった構造を持ったドキュメントの単位で情報を扱い，文書構造の修正や，新しい文書構造の定義において大きなコストを要している．一方 DITA ではトピックとマップを分離する事で，トピックは文書構造の変更の影響を受けず，マップは情報の変更の影響を受けない．また，特殊化という仕組みを用いることでも再利用性を向上できる．特殊化はオブジェクト指向における継承に似た仕組みであり，既に定義されている要素やトピック，マップに対して，それらをより厳密にした要素，トピック，マップを定義することを言う．特殊化を行った要素やトピックは，継承元と同じように処理することが可能である．これにより，あるドキュメントを記述する為に要素の種類を追加した DITA コンテンツを，他のドキュメントで再利用することが可能である．

2.3.3 ソフトウェアドキュメントへの DITA 適用

DITA 適用に向け，ソフトウェアドキュメントの構造化を行う研究が坂井らにより行われている [9]．この研究では，ソフトウェアドキュメントへの DITA 適用により，追跡性の向上とメンテナンスコストの低下が可能であるとし，DITA 適用に向けて適切なトピック粒度について述べている．対象とするソフトウェアドキュメントは，共通フレーム 2007 [6] の主ライフサイクルプロセスにおいて，各プロセスの代表的なドキュメントである．これらを用いることで，一般的なソフトウェア開発で作成されるドキュメントの大部分が対象となっている．

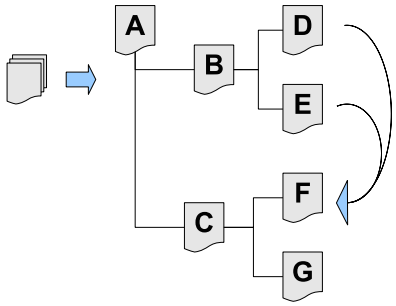

図 1　トピック間の関連

共通フレーム 2007 のドキュメントを項目ごとに細分化する場合，それらの中には再利用の可能な項目と，不可能な項目が存在する．再利用可能な項目とは，他の項目に依存せずに内容が決定できる．一方，再利用が不可能な項目の場合，小さな単位での分割を行うと管理コストの増大する．この為，依存関係にある内容をひとつのトピックに纏めた方が適切である．

2.3.4 DITA Open Toolkit

DITA に準拠して作成されるトピック及びマップは，文書に含まれる情報と，その構造を定義している．これらの情報はレイアウトや体裁とは分離されている．その為，DITA によって作成されたドキュメントを利用するには，HTML や PDF といったファイルに変換する必要がある．DITA Open Toolkit(以下，DITA-OT) [4] はこの変換作業を行うツールである．図 2 では，A, B, C, D, E というトピックに対して，マップ 1, 2 が定義されている．DITA-OT はそれぞれのマップに対して XSL 変換を適用し，複数のファイル形式への変換を行うことができる．

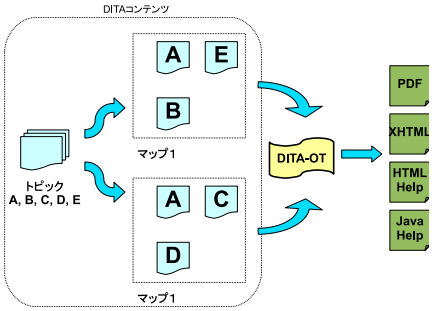

図 2　DITA-OT のイメージ

3　提案

3.1　統合管理環境の全体像

本研究で提案するドキュメント・ソースコードの統合管理環境の全体像を図 3 に示す．提案環境では，DITA で作成されたドキュメントとソースコードを取り扱い，それらの整合性と追跡性の確保に関する機能を提供する．全体として，以下の主要な機能を持つ．

ソースコード自動生成　プログラム要素の情報を含む DITA コンテンツを入力としてソースコードを生成する．この際，プログラム要素には一意な id が付与されていることを前提とし，ソースコード中の対応箇所にも同一の id を埋め込む．

整合性の確保　ソースコード中に埋め込まれた id と，ドキュメントに付与された id を利用し，ドキュメント中の対応箇所を追跡する．それぞれの対応箇所について整合性を検査する．対象となるのは，識別子の名称や型，修飾子など，ドキュメント中に仕様として記述されている情報である．整合性検査の結果，不一致が確認された場合には，古い情報を自動的に修正できるようにする．また，情報の修正にはある要素への参照部分の修正も含まれる．

追跡性の確保　ユーザからどのソースコード要素とドキュメント要素が関連付けられているかを追跡可能とする．

3.2　対象とするドキュメント

共通フレーム 2007 [6] では，ソフトウェアの開発プロセスを定義しており，その中の各工程で作成する成果物についても述べられている．今回管理対象とするドキュメントとしては，ドキュメント中に識別子名や型といったプログラム要素への言及を含むものである必要がある．従って，実装の一つ前の工程であるソフトウェア詳細設計で作成されるドキュメントを対象とする．

3.3　ドキュメントの構造化

ドキュメントの修正の際に，ソースコードとの整合性を維持する．その為，記述されたテキストの内容がソースコードのどの要素に該当するのかというメタ情報を持たせる必要がある．2 章で述べたように，XML に準じた規格が技術文書構造化に利用されている．XML を用いた場合，テキストをタグでマークアップすることで，様々なメタ情報を含ませることができる．また，XML のタグやタグが持つ属性，発生順序や回数などを DTD として定義することで，自由に拡張が可能である．本研究では，XML 文書規格である DITA を用いてドキュメントの構造化を図る．DITA を用いることにより，先に挙げた XML の利点に加え，情報と文書構造がトピック，マップという形で分離されるので，情報の再利用性が高まる．これにより，整合性の検査が必要なソースコードとドキュメントの対応箇所を最小限に出来る．

ドキュメントに DITA を適用する手法としては，2.3.3 節における適切なトピック粒度を参考にし，ソースコード要素に対応する記述を構造化する上で必要となる要素の定義を行う．

3.4　ソースコードとドキュメントの関連付け

ソースコード自動生成と，自動生成された後に編集されたソースコード中の要素の取得を行う．これにより，ソースコード要素のドキュメント要素の対応の判別と，整合性の検査を可能とする．

3.4.1　DITA-OT を用いたソースコード生成

DITA-OT を用い，DITA 文書を入力としたソースコード自動生成を行う．目的として，クラス名やメ

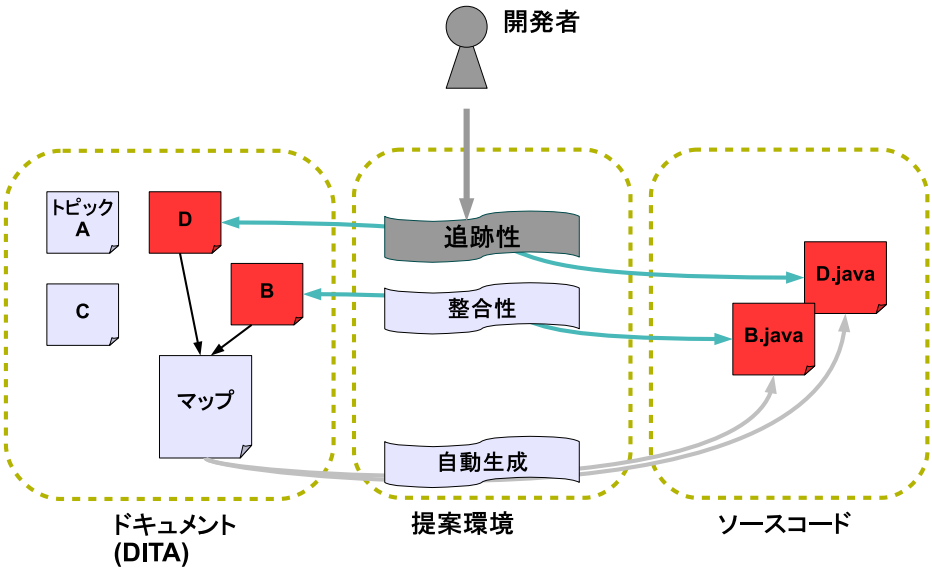

図 3　提案環境の全体像

ソッド名を初めとするプログラム要素の識別子の変更に対応することが挙げられる．ソースコード中に，プログラムの動作や構成に影響しない方法で，入力となったドキュメント要素との対応関係を記述する．具体的には，以下のようにコメントを利用した方法が考えられる．

ソースコード 1　対応関係の埋め込み例

```
// #id=m0001
public static void main(String[] args)
{
...
}
```

上の例では，main メソッドの前にコメントを付加することで,「m0001」という対応関係を表す id を持つことが示されている．この id は，自動生成の入力となるドキュメント要素にも記述されている必要がある．以下に例を示す．

ソースコード 2　ドキュメントの例

```
<method id=''m0001'' name=''main''>
...
</method>
```

上の例の method 要素にはメソッドの定義が含まれる．id 属性が，前の例のコメントの id と対応している．このように対応関係を埋め込むことで，実装工程でプログラム要素の識別子が変更された場合でも，事後的にドキュメント中の対応箇所を見つけ出し，自動的に古い情報に修正を加えることで整合性維持が実現できる．また，自動生成の手段として DITA-OT を用いることには二つの利点が存在する．一つ目は，このツールが DITA から複数種類のファイルを生成する機能を持つ為，XSL 変換の方法を定義するだけで，ソースコードへの変換が実装できるという点である．二点目は，オープンソースであり，自作のプログラムにライブラリとして組み込むことが期待できる点である．

3.4.2　ソースコード要素の取得

ソースコードへの変更箇所と，変更内容を取得する為に構文解析を行う．解析には Sapid を用いることを検討している．2.1.2 節で述べたように，Sapid はソースコード中に含まれる構成要素及びそれらの関連を細粒度で解析し，リポジトリ化する．Sapid には

C 言語用の API が提供されており，リポジトリへのアクセスが行える．

4 ソースコード自動生成実験

XSL 変換によるソースコード自動生成の実験を行った．この実験の目的は，提案環境の主要機能の一つであるドキュメントからのコード生成について，XSL 変換による方法が有効であるかを検証することである．

4.1 入力ドキュメントと XSL

生成元となる XML ファイルは DITA の基本構造に従い，一つのマップ（ソースコード 4) と一つのトピック（ソースコード 5) から成る．生成されるソースコードは Java 言語で，一つのクラスと，それに関する Javadoc，そして，XML 上で付与された id を含むものとする．その為，ソースコード 5 は一つのクラスに関する記述を想定した DITA トピックとなっている．また，class や method，field 等，DITA の基本要素には無い新しい要素を導入している．これらの要素は，DITA に基本的に含まれる要素を特殊化し，DTD に定義する必要がある．ただし，今回の実験の目的とは直接関係ない為，特殊化の定義は行っておらず，生成元 XML ファイルの DOCTYPE 宣言もコメントアウトしている．また，ソースコード中に付与される id はクラス・メソッド・フィールドにそれぞれ与えられ，XML 中の対応する要素の id 属性と同一のものである．

本実験に使用した XSL スタイルシートをソースコード 3 に示す．また，XSL 変換プロセッサとして，Saxon9.5 [3] を用いた．Saxon は DITA-OT にも使われており，DITA コンテンツの処理に適していると言える．

4.2 結果

自動生成の結果を付録のソースコード 6 に示す．入力ドキュメントに記載されたクラスの情報にある四つのフィールド，三つのメソッドがそれぞれ修飾子や引数，返り値の型も含めて正しく生成されていることが分かる．また，クラス，メソッド，フィールドに対して，それぞれ元ドキュメントの id 属性と同一の値が「// #id=***」という形式のコメントとして付加されていることが確認できる．

5 考察

実験により，XSL 変換によるソースコード生成が可能であることが確認できた．今回生成元として使用した XML は DOCTYPE 宣言を含まず，完全に DITA に準拠しているとは言えないが，特殊化の仕組みを利用して要素の定義を行えば，複数言語でのソースコード生成が可能になると考えられる．また，今回 Java ソースを出力する為に class や method といった独自の要素を利用したが，トピックにおける title 要素を Java クラス名として出力するなど，既存の要素の流用も存在する．既存要素をどう出力し，どういった要素を追加する必要があるのかも検討する必要がある．id 付与の方法に関しても，構文解析の方法と関連して検討する必要がある．

6 まとめと今後の課題

ソフトウェアドキュメントとソースコード間の整合性・追跡性の確保を最終的な目標とし，それらを DITA によるドキュメント構造化を用い，統合的に管理する環境を提案した．統合管理環境構築の為には，ソースコード自動生成が必要となることから，今回 XSL 変換により XML 文書を Java ソースに変換する実験を行った．実験の結果，ソースコード自動生成に XSL 変換の仕組みが有効であることが分かった．今後は，対象とするドキュメントについて具体例を用いて DITA 適用を行い，対象ドキュメントとする．また，提案環境の構築は DITA で用いる要素の決定と自動生成や構文解析，整合性確保の修正プロセスについて考案していく予定である．

参 考 文 献

[1] Altova: UML ツール, http://www.altova.com/ja/umodel.html.

[2] DITA コンソーシアムジャパン: DITA コンソーシアムジャパン 公開資料, http://dita-jp.org/?cat=13.

[3] Kay, M. H.: The SAXON XSLT and XQuery Processor, http://saxon.sourceforge.net/.

[4] OASIS: The DITA Open Toolkit — DITA XML.org, http://dita.xml.org/wiki/the-dita-open-toolkit.

[5] OASIS DocBook Technical Committee: DocBook.org, http://www.docbook.org/.
[6] 独立行政法人 情報処理推進機構ソフトウェア・エンジニアリング・センター: 共通フレーム *2007*, 株式会社オーム社, 2007.
[7] 戸板晃一, 山本晋一郎, 阿草清慈: XML を用いたソフトウェア関連文書とソースプログラムの整合性検査ツール, (2001), pp. 129–140.
[8] 阿草清滋: Sapid Home Page (in Japanese), http://www.sapid.org/index-ja.html.
[9] 坂井麻里恵, 奥野拓: DITA 適用に向けたソフトウェアドキュメントの構造化, (2012).

付録: 実験用ソースコードと対象ファイル

ソースコード 3　sample01.xsl

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <xsl:stylesheet version="2.0" xml:space="preserve"
3     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4 <xsl:output method="text" encoding="UTF-8"/>
5     <xsl:variable name="prjName"
6         select="/map/title"/>
7
8     <!-- root -->
9     <xsl:template match="/">
10        <xsl:value-of select="$prjName" />
11        <xsl:apply-templates select="map/topicref"/>
12    </xsl:template>
13
14    <!-- topicref -->
15    <xsl:template match="topicref[@type='concept']">
16        <xsl:variable name="topic"
17            select="document(@href)"/>
18        <!-- [プロジェクト名]/[クラス名].java -->
19        <xsl:result-document
20            href="{$prjName}/{translate(@href,␣'.dita',␣'.java')}"
21            method="text" encoding="UTF-8">
22
23            <xsl:apply-templates select="$topic//concept" />
24        </xsl:result-document>
25    </xsl:template>
26
27    <!-- concept -->
28    <xsl:template match="concept">
29        // #id=<xsl:value-of select="@id" />
30        /**
31         <xsl:value-of select="shortdesc" />
32        */
33        <xsl:value-of select="conbody/class/@modifier"/> class <xsl:value-of
           select="title" /> {
34            <xsl:apply-templates select="conbody/class/field"/>
35            <xsl:apply-templates select="conbody/class/method"/>
36        }
37    </xsl:template>
38
39    <!-- field -->
40    <xsl:template match="field">
41        /**
42        <xsl:value-of select="p" />
43        */
44        // #id=<xsl:value-of select="@id" />
45        <xsl:value-of select="@modifier" /> <xsl:value-of select="@type" /> <
           xsl:value-of select="@name" /> = <xsl:value-of select="@initVal" />;
46    </xsl:template>
47
48    <!-- method -->
```

```
49    <xsl:template match="method">
50       /**
51       <xsl:apply-templates select="p" />
52       <xsl:apply-templates select="param" mode="javadoc"/>
53       <xsl:apply-templates select="return" mode="javadoc"/>
54       */
55       // #id=<xsl:value-of select="@id" />
56       <xsl:value-of select="@modifier" /> <xsl:apply-templates select="
          return" mode="code" /> <xsl:value-of select="@name" /> (<xsl:apply-
          templates select="param[1]" mode="code-p" /><xsl:apply-templates
          select="param[2 &lt;= position()]" mode="code-ps" />) {
57       }
58    </xsl:template>
59
60    <!-- method/param 第一引数-->
61    <xsl:template match="param" mode="code-p">
62       <xsl:value-of select="@modifier" /> <xsl:value-of select="@type" /> <
          xsl:value-of select="@name" />
63    </xsl:template>
64    <!-- method/param 第二以降 -->
65    <xsl:template match="param" mode="code-ps">
66       , <xsl:value-of select="@modifier" /> <xsl:value-of select="@type" />
          <xsl:value-of select="@name" />
67    </xsl:template>
68    <xsl:template match="param" mode="javadoc">
69       @param <xsl:value-of select="@name" /> <xsl:value-of select="." />
70    </xsl:template>
71
72    <!-- method/return -->
73    <xsl:template match="return" mode="code">
74       <xsl:value-of select="@type" />
75    </xsl:template>
76    <xsl:template match="return" mode="javadoc">
77       @return <xsl:value-of select="." />
78    </xsl:template>
79 </xsl:stylesheet>
```

ソースコード 4　map01.ditamap

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!--<!DOCTYPE bookmap PUBLIC "-//OASIS//DTD DITA BookMap//EN" "bookmap.dtd">
  -->
3 <map id="map_sample" xml:lang="ja-jp">
4    <title>SampleJavaProject</title>
5
6    <topicref href="Fan.dita" type="concept" />
7
8 </map>
```

ソースコード 5　Fan.dita

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" "concept.dtd">
  -->
3 <concept id="jc01" xml:lang="ja-jp">
```

```
 4 <title>Fan</title>
 5 <shortdesc>扇風機である</shortdesc>
 6 <conbody>
 7    <class modifier="public">
 8      <field modifier="public static final" type="int" id="jc01f01" name="
        MODE_TURN_OFF"
 9         initVal="0" >
10         <p>モード：電源切</p>
11      </field>
12      <field modifier="public static final" type="int" id="jc01f02" name="
        MODE_LOW"
13         initVal="1" >
14         <p>モード：弱</p>
15      </field>
16      <field modifier="public static final" type="int" id="jc01f03" name="
        MODE_HIGH"
17         initVal="2" >
18         <p>モード：強</p>
19      </field>
20      <field modifier="private" type="int" id="jc01f03" name="mode"
21         initVal="MODE_TURN_OFF" >
22         <p>現在のモード</p>
23      </field>
24       <method modifier="public" ret="void" id="jc01m01" name="setMode">
25          <param type="int" name="mode">モード</param>
26          <p>モードをセットする</p>
27       </method>
28       <method modifier="public" id="jc01m02" name="getMode">
29          <return type="int">MODE_TURN_OFF, MODE_LOW, MODE_HIGH</return>
30          <p>現在のモードを返す</p>
31       </method>
32       <method modifier="public" ret="int" id="jc01m03" name="getVelocity">
33          <return type="int">風速[m/s]</return>
34          <p>風速を返す</p>
35       </method>
36    </class>
37 </conbody>
38 </concept>
```

ソースコード 6　Fan.java

```
1
2
3
4       // #id=jc01
5       /**
6        扇風機である
7       */
8       public class Fan {
9
10      /**
11      モード：電源切
12      */
13      // #id=jc01f01
14      public static final int MODE_TURN_OFF = 0;
```

```
15
16          /**
17          モード：弱
18          */
19          // #id=jc01f02
20          public static final int MODE_LOW = 1;
21
22          /**
23          モード：強
24          */
25          // #id=jc01f03
26          public static final int MODE_HIGH = 2;
27
28          /**
29          現在のモード
30          */
31          // #id=jc01f03
32          private int mode = MODE_TURN_OFF;
33
34
35          /**
36          モードをセットする
37
38          @param mode モード
39
40
41          */
42          // #id=jc01m01
43          public  setMode (
44           int mode
45      ) {
46          }
47
48          /**
49          現在のモードを返す
50
51
52          @return MODE_TURN_OFF, MODE_LOW, MODE_HIGH
53
54          */
55          // #id=jc01m02
56          public
57          int
58       getMode () {
59          }
60
61          /**
62          風速を返す
63
64
65          @return 風速[m/s]
66
67          */
68          // #id=jc01m03
69          public
```

```
70        int
71      getVelocity () {
72        }
73
74        }
```