デジタル・ドキュメント 15－1
（1998. 11. 27）

XML技術を利用したソフトウェア文書理解支援システム *CrystalScope* の開発

松島 弘幸　　小田 利彦
リコー ソフトウェア研究所

概要

既存のソフトウェアを再利用したり保守したりするには，ソフトウェアに付随する膨大な量の文書を理解する必要がある．この文書にはCASEツールにより作成された図や，ソースコード等も含まれる．CrystalScopeはこれらの文書を閲覧するためのWWWベースのシステムであり，文書間の関連の表示をユーザの意図によって変化させることで効率的なソフトウェアの理解を支援する．CrystalScopeでは全ての文書をDOMモデルに変換することで統一的に扱い，文書間の関連はXLink/XPointerで定義する．本稿では，このようなXML技術を応用したCrystalScopeの実装について紹介する．

CrystalScope : A Software Understanding Support System Based On XML Technologies

Hiroyuki MATSUSHIMA　　Toshihiko ODA
Software Research Center, RICOH COMPANY

Abstract

CrystalScope is a WWW-based system which helps software developers efficiently browse and understand various kinds of software documents when they reuse or maintain some software. Software documents can be diagrams created using CASE tools or source codes as well as text documents such as specifications. Based on users' intent, CrystalScope can control which links between document elements to be displayed. This paper describes how CrystalScope is implemented utilizing XML technologies, such as DOM and XLink/XPointer.

1　はじめに

ソフトウェアの開発は，一般に要求分析，設計，プログラミング，テストの工程に分けられ，開発中各工程毎に作業結果が電子文書として作成される．この文書にはワープロ等で作成されるテキスト文書だけでなく，CASEツールにより作成された図，ソースコード等も含まれる．本研究ではこれら全てを含めてソフトウェア文書と呼んでいる．

こうして開発されたソフトウェアを再利用したり保守する場合，それを理解することが重要である [1]．しかし，ソフトウェア文書は種類が非常に多く，また各文書間の関連がわかりにくいために，効率よく理解することができない．

文書の理解性を高めるために，HTMLに代表されるハイパーテキスト形式で文書を提供することが考えられるが，文書の量が多くなると，いたずらにリンクを辿ってしまい，不要な情報を探索したり，どの文書を見ているのかわからなくなるという問題がある．またリンク情報を開発文書の中に埋め込む必要があるため保守性が悪く，単方向，固定的，属性(例えばリンクの名前や意味)を持たないというリンクの性質から，ユーザに対して動的に表示内容を変えたり，異なったビューを作るなどの柔軟なユーザインターフェイスを作ることが難しい．更に，CASEツールにより作成した図やソースコードとのリンクを実現するのも難しい．

以上の背景から，ソフトウェア再利用において

表 1: オブジェクト指向開発プロセスにおける成果物と使用ツールの例

開発フェイズ	成果物	使用ツール
要求仕様分析	要求仕様書	MS Word, LaTeX
オブジェクト指向分析	分析仕様書	MS Word, LaTeX
	モデル図	BridgePoint, Rational Rose
オブジェクト指向設計	設計仕様書	MS Word, LaTeX
	クラス API	MS Word, javadoc
	モデル図	Rational Rose
	デザインパターンカタログ	Rational Rose
プログラミング	ソースコード	mule, Visual J++

ソフトウェア文書の理解を効率よく行うための文書閲覧ツール(ブラウザ)を実現することを目的として本稿で紹介する研究を行っている. 本研究で試作中のシステムはCrystalScopeという名前が付けられており, これにはソフトウェア文書をクリスタルガラスのようにクリアに見ることができるようにという思いが込められている.

ハイパーテキストによるソフトウェアの文書化に関しては多くの研究がなされており [2, 3], CrystalScope との類似点も多いが, CrystalScope では多くの問題を XML(Extensible Markup Language) 技術 [4] を採用することで解決している. また本システムは, 運用を考慮して既に普及している Netscape 及び IE で動作する WWW ベースのクライアント/サーバシステムとして開発している.

本稿では XML 技術を応用した CrystalScope の実装の側面に重点を置いて紹介する. 2 節ではソフトウェア文書について例を挙げて説明し, 3 節では試作中の CrystalScope における理解支援機能について示す. 4 節では XML 技術の応用について詳しく説明し, 5 節でクライアント/サーバの構成について簡単に紹介する. 6 節では XML 技術の適用における問題点を挙げて考察し, 7 節で総括する.

2 ソフトウェア文書

2.1 ソフトウェア文書の例

本研究では, 仕様書などのテキスト文書だけでなく, CASE ツールにより作成された図, ソースコード等も含めて, ソフトウェアの開発工程において作成される全ての文書をソフトウェア文書と呼んでいる. ここでは, オブジェクト指向方法論に基づいたソフトウェア開発プロセスを例にとって, どのような文書が作成されるのか示す. 開発プロセスは以下のフェイズから構成されるものとする.

要求仕様分析 そのソフトウェアに必要とされる機能や制約を調査する.

オブジェクト指向分析 要求仕様よりオブジェクト(クラス)を抽出してその関係や振る舞いを定義する.

オブジェクト指向設計 システム構成や制約, プログラミング言語等を決定し, これらを考慮して分析モデルを詳細化する. またデザインパターンを適用することによって, 適切に具象/抽象クラスを定義し, 再利用性の高い設計にする.

プログラミング 設計仕様に沿って選択した言語 (例えば Java) でプログラミングする.

各フェイズにおける成果物とそれを作成するツールの例を表 1 にまとめる.

2.2 ソフトウェア文書の特性

前節に示したように, ソフトウェア開発では非常に多くの文書が作成される. 要求仕様書, モデル図, ソースコードといったソフトウェア文書のカテゴリのことを, 本研究では文書クラスと呼んでいる. ある文書クラスに属する文書(ファ

表 2: ソフトウェア文書における異文書クラス間の依存関係の例

関係の種類	関係元の文書要素	関係先の文書要素
機能実現関係	要求仕様中の機能仕様項目	設計仕様項目 クラス群 (モデル図, ソースコード)
制約充足関係	要求仕様中の制約仕様項目	設計仕様項目 クラス群 (モデル図, ソースコード)
設計実装関係	設計仕様項目	クラス群 (モデル図, ソースコード)
パターン適用関係	デザインパターン	設計仕様項目 クラス群 (モデル図, ソースコード)

イル) は1つかも知れないし, ソースコードのように非常に多くなることもある. また, 各文書の論理的階層構造におけるノードのことを文書要素と呼ぶ.

さて, ソフトウェア文書には次の 2 つの特性がある.

まず第 1 に, 文書内あるいは文書間に強い関連性が存在することが挙げられる (図 1). ソフトウェア開発のある作業工程において複数のソフトウェア文書が作成されると, これらは次に続く作業工程への入力となるため, 互いに強く関連することになる. 例えば, あるソフトウェアへの要求項目に対して, 設計の構造が決められ設計仕様となり, さらにプログラミングによりソースコードになる. その他の例を表 2 にまとめる. こうした異文書クラス間に存在する関連性だけでなく, 同一文書クラス内の文書要素間にも関連性は存在する. 例えば, ソースコードの場合では Caller-Callee 関係, データ束縛関係等をモジュール間に見つけることができる. このように複雑な依存関係を有するソフトウェア文書を理解するには, 関連の追跡が容易であることが求められる. ところで, これらの関連は, 互いに関連する文書が揃ってはじめて定義できるものであるから, HTML のように特定タグを埋め込むことでしか関連を定義できない場合, 非常に保守性が悪い. また, ワープロや CASE ツールではそもそも関連を定義できない場合が多い.

2 つ目のソフトウェア文書の特性として, 文書形式の標準化が挙げられる. ソフトウェア開発部門では, 開発プロセスとともに, 仕様書, 計画書, 報告書等についての記述形式や, 使用するツールをある程度定めていることが多い. このように文書クラス毎に記載すべき項目や内容, 項目の順序などをあらかじめ定義して文書を標準化することにより, 文書の作成や品質管理を容易にすることができる.

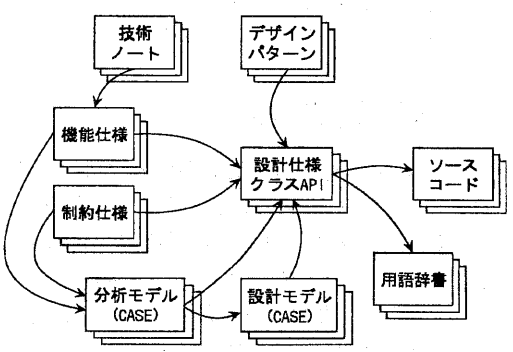

図 1: 文書間の関連

3 CrystalScope の機能概要

複雑に関連し合うソフトウェア文書を効率よく閲覧し理解することを支援するツールとして, 以下に示すような機能を定義した. それぞれユーザの思考過程に対して, それを支援する機能を示している.

文書群の全体像の把握 あるソフトウェアに関係するソフトウェア文書群の全体像を把握できるように, 文書クラス毎に文書要素の見出しのみ表示するパネル (論理構造表示パネル) を提供する. 文書クラスが多くなっ

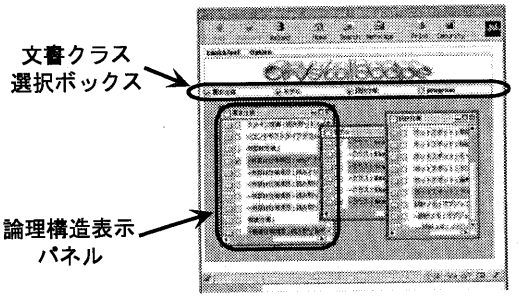

図 2: メインパネルのスナップショット

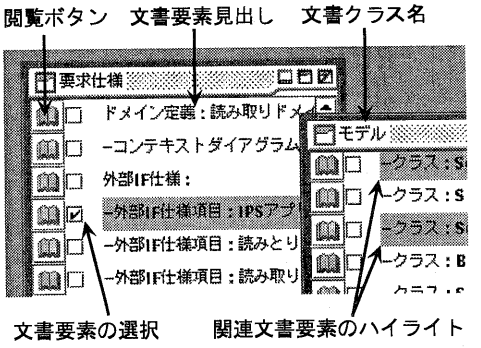

図 3: 論理構造表示パネルの拡大図

た場合を考慮して, 文書クラス毎にパネルの表示/非表示を選択できるものとする.

文書の関連性の把握　文書の探索範囲を限定するために, ユーザは文書の関連性を把握できるのが望ましい. 文書の関連を調べる際に, ユーザが現在注目している文書要素に対して, これと関連のある文書要素を論理構造表示パネル上で区別して表示する. この時, 文書要素間の関連の種類や, 関連をたどる深さなどの設定ができるようにすることで, ユーザの意図に従った誘導を行う.

文書の詳細の把握　文書の詳細は, 論理構造表示パネル上でユーザが指示した時に, 論理構造表示パネルとは異なるパネルに表示する.

こうした機能を満たすシステムとしてプロトタイピングを行っている CrystalScope について, ユーザがソフトウェア文書を閲覧するための手順を示す (図 2, 図 3 参照).

1. **ソフトウェアの選択**: WWW ブラウザで CrystalScope の URL にアクセスすると, 登録されたソフトウェアの一覧が表示される. ソフトウェアを選択し, 画面サイズ等を設定した上で閲覧ボタンをクリックする.
2. **文書クラスの選択**: 1 で選択したソフトウェアに関して登録された文書クラス名のチェックボックス (文書クラス選択ボックス) を含むメインパネルが表示される. チェックボックスをクリックして選択すると, 論理構造表示パネルが開き, その文書クラスに属する文書群の論理構造が見出しによって表示される. チェックボックスを非選択にすると対応するパネルは閉じる.
3. **ある文書要素に関連する文書要素の導出**: 2 で表示された論理構造表示パネルにおいて, 注目したい文書要素のチェックボックスを選択すると, その文書要素に関連する他の文書要素がハイライトされる.
4. **文書の閲覧**: 2 で表示された論理構造表示パネルにおいて, 本文を閲覧したい文書要素の閲覧ボタンをクリックすると, 別のブラウザウィンドウに本文がハイパーテキスト形式で表示される.
5. **リンクフィルタの条件設定**: 2 で表示されたメインパネルのメニューより『リンクフィルタの条件設定』を選択すると, 設定用パネルが表示される. このパネルでは, 3 で関連する文書要素を計算する際の設定を変更することができる (4.4 節参照).

4 XML 技術の適用

多種多様で開発部門毎に異なる文書標準が存在するソフトウェア文書を扱うのに, XML のタグの拡張性が有効である. 部門毎に独自の DTD を定義することで開発プロセスの違いを吸収することができ, 更に定義された DTD は部門における厳密に定義された文書標準と考えることができる. 文書作成者は XML プロセッサにより文書が DTD に従っているかどうかの検証を行うこ

```
 1: <!ENTITY % common-dtd
 2:             SYSTEM "common.dtd">
 3: %common-dtd;
 4:
 5: <!ELEMENT 設計仕様
 6:   (ホットスポット|クラス仕様|設計メモ)* >
 7: <!ATTLIST 設計仕様
 8:    %文書情報;
 9:    論理要素 (true|false) #FIXED "true"
10: >
11:
12: <!ELEMENT ホットスポット (記述) >
13: <!ATTLIST ホットスポット
14:    名称      CDATA          #REQUIRED
15:    論理要素 (true|false) #FIXED "true"
16: >
```

リスト 1: 文書 DTD の例

```
 1: <!ENTITY % 文書情報
 2:    "名称            CDATA    #REQUIRED
 3:     作成者          CDATA    #REQUIRED
 4:     作成日時        CDATA    #REQUIRED">
 5:
 6: <!ELEMENT 記述
 7:    (#PCDATA|キーワード|図|表)* >
 8: <!ATTLIST 記述
 9:    論理要素 (true|false) #FIXED "false"
10: >
11:
12: <!ELEMENT キーワード EMPTY >
13: <!ATTLIST キーワード
14:    名称      CDATA          #REQUIRED
15:    論理要素 (true|false) #FIXED "false"
16: >
```

リスト 2: 共通 DTD の一部

とができる. こういった理由で, CrystalScope では対象文書として XML をサポートした. CrystalScope で閲覧する各文書の DTD はいくつかの規約に従っている必要がある (4.1 節).

全ての文書を統一的に扱うために, 文書要素の階層構造を表す木構造データにアクセスするためのインターフェイスとオブジェクトモデルを定義した DOM(Document Object Model)[5] のモデルに文書を変換する (4.2 節).

またソフトウェア文書間の複雑な関連を表現するために, 強力なリンク機能を有するという点でも XML が適している. XML のリンク仕様である XLink[6] と XPointer[7] は, HTML における問題点を解消するために柔軟かつ強力なものになっている. CrystalScope では, リンクと文書 (コンテンツ) を分離できるという XLink の特徴を利用している (4.3 節).

XLink/XPointer で定義されたリンク情報をもとに文書要素をつなぐグラフ構造を生成し, ユーザが選択した文書要素を始点としてこのグラフを走査することで関連する文書要素を求める. ユーザの意図に従って走査の方法を変えることにより, 適切な文書にユーザを誘導する (4.4 節).

4.1 文書 DTD の規約

CrystalScope で扱う文書の DTD はいくつかの規約に従っている必要がある. 説明をわかりやすくするために, DTD の例をリスト 1 に示す. これは完全な DTD の一部であるが, 『設計仕様』 要素が文書クラス 『設計仕様』 のルートを表している.

規約 1 共通 DTD のインクルード

全ての文書クラスの DTD は共通 DTD である`common.dtd` をインクルードしなくてはならない (1〜3 行目). 8 行目の 『文書情報』 エンティティと 12 行目の 『記述』 要素はこの共通 DTD において宣言される.

規約 2 『論理要素』 属性

9, 15 行目に現れる 『論理要素』 属性は, このタグで表される文書要素が論理構造表示パネルに表示されるかどうかを示している. 値は`true` または`false` で, `true` の時論理構造表示パネルに表示される. 表示される見出しはタグ名+『名称』 属性の値となるため, 『名称』 属性も定義しておくことが望ましい.

規約 3 『記述』 要素

各文書要素の本文 (内容記述) は 『記述』 要素として宣言しなくてはならない (12 行目).

共通 DTD の一部をリスト 2 に示す. 共通 DTD に宣言される 『記述』 要素は, テキストデータだけでなく, 画像ファイルへのリンクを記述するための 『図』 要素 (XLink 準拠) や, 表を記述するための 『表』 要素が含まれる (6, 7 行

```
 1: <!ELEMENT ソフトウェア
 2:           (記述?, リソース定義*) >
 3: <!ATTLIST ソフトウェア
 4:   名称            CDATA   #REQUIRED
 5: >
 6:
 7: <!ELEMENT リソース定義 EMPTY >
 8: <!ATTLIST リソース定義
 9:   クラス          CDATA   #REQUIRED
10:   フォーマット     CDATA   #REQUIRED
11:   ロケーション     CDATA   #REQUIRED
12: >
```

リスト 3: 文書登録のための DTD

目). また, 文章内に埋め込むリンクを後から定義しやすくするために, あらかじめ重要語句を『キーワード』要素として記述することができるようになっている (12〜16 行目). 『記述』や『キーワード』は論理構造表示パネルに表示しないため, 9, 15 行目で『論理要素』属性を`false`に固定している.

4.2 文書の DOM への変換

CASE ツールで作成した図やソースコードを含む全てのソフトウェア文書は構文解析して DOM に変換する.

さて, CrystalScope ではソフトウェア文書の登録に関しても XML を利用している. その DTD をリスト 3 に示す. 『リソース定義』要素 (7〜12 行目) がひとつの文書 (ファイル) を表していて, 属性『クラス』が文書クラスを, 属性『フォーマット』がファイル形式を, 属性『ロケーション』がファイルの URL を表す. 『クラス』属性の同じ文書は同じ論理構造表示パネルに表示される. 『フォーマット』属性は`"XML"`, `"Rose"`, `"Java"`といった属性値をとり, 『フォーマット』属性値に対応するパーサにより構文解析が実行される. 新しいフォーマットに対するパーサはプラグイン的に追加していくことができる.

4.3 リンクの記述

文書間 (同一文書内も含む) のリンクは XLink /XPointer に準拠した形式で記述する. CrystalScope におけるリンクは, リスト 4 の DTD に従った, リンクのみを定義する文書として記

```
 1: <!ELEMENT リンク定義 (リンク)* >
 2:
 3: <!ELEMENT リンク
 4:  ((リソース|キーワードリソース), リソース+)
 5: >
 6: <!ATTLIST リンク
 7:    xml:link CDATA      #FIXED "extended"
 8:    inline (true|false) #FIXED "false"
 9:    role     CDATA      #REQUIRED
10: >
11:
12: <!ELEMENT リソース EMPTY >
13: <!ATTLIST リソース
14:    xml:link CDATA      #FIXED "locator"
15:    href     CDATA      #REQUIRED
16: >
17:
18: <!ELEMENT キーワードリソース EMPTY >
19: <!ATTLIST キーワードリソース
20:    xml:link CDATA      #FIXED "locator"
21:    href     CDATA      #REQUIRED
22: >
```

リスト 4: リンク定義のための DTD

述する. 従ってリンクは全て拡張リンクである. 3 つ以上の要素を関連づけることができ (3〜5 行目), `role` 属性を利用してリンクの種類を指定する (9 行目). 各ロケータの`href` 属性は XPointer 形式で記述する.

4.4 リンクの利用

リンク文書は通常の XML 文書と同じく DOM に変換した後, `href` 属性の XPointer の解析を行う.

『リンク』要素の第 1 子要素が『リソース』の場合は『リンク』の全ての子要素 (リソース) 間に双方向リンクを生成する. 一方『リンク』要素の第 1 子要素が『キーワードリソース』の場合はロケータ『キーワードリソース』が指し示す『キーワード』要素 (4.1 節参照) からその他の『リソース』に向かう 1：多のリンクを生成する.

一方, リンク文書以外のソフトウェア文書から生成された DOM については, 階層構造内の全ての親子関係を文書構成関係を表す種類のリンクとして生成する.

以上すべてのリンクの集合を依存グラフと呼び, ユーザが選択した文書要素を始点としてこ

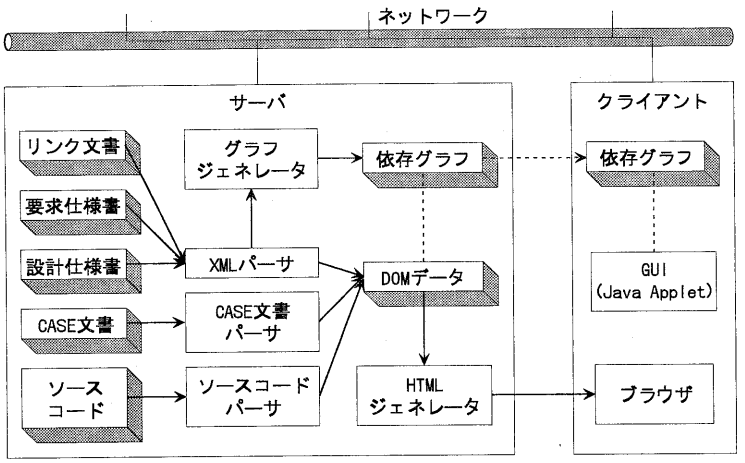

図 4: システム構成

のグラフを走査することで関連する文書要素を求める. 1回の走査を行う実行単位をリンクフィルタと呼んでおり, 複数のリンクフィルタを組み合わせたり, リンクフィルタの設定をユーザが変更することによって結果が変わる仕組みになっている. これにより適切な文書にユーザを誘導することができる.

5 システム構成

システム構成を図 4 に示す.

5.1 サーバ

サーバの機能は Servlet として Java 言語で実装した. WWW サーバとしては Java Web Server 1.1 及び Apache+JServ において動作確認している.

サーバは文書管理, 文書の構文解析, DOM の生成, 依存グラフの生成, HTML の生成を行う. ブラウザの XML への対応が十分でない現状から, 文書の内容はサーバで HTML に変換することにした. この際, 一般にソフトウェア文書は1つの WWW ページとして閲覧するには量が多く, 通信時間も長くなるので, クライアントからの要求に応じて指定された文書要素のみ抽出して動的にページを作成することにした.

サーバ上で生成した依存グラフと見出し情報 (コンテンツを含まない DOM データ) は, ユーザがソフトウェアを選択した時点でクライアントに送信する. このように最初は文書に関する論理情報のみを送信し, コンテンツはユーザが要求したときに送信することでパフォーマンスの向上をはかっている.

文書のフォーマットとしては XML と BridgePoint, Rose(いずれもオブジェクト指向開発 CASE ツール) をサポートしている. XML 文書に関しては, XML 構文解析, DOM 生成, XPointer 解析等の機能を備えた IBM の XML for Java[8] を利用した. BridgePoint, Rose については JavaCC を用いて独自にパーサを開発した. HTML 生成において, CASE 文書に関してはモデル図を表示するための Java Applet を含む HTML を生成する.

5.2 クライアント

クライアントは Java Applet として実装した. GUI に Swing を用いているため, ブラウザに Java Plug-in をインストール必要がある.

クライアントは依存グラフを保持しているので, 関連要素の計算はすべてクライアント上で行う. これらの一部をサーバで処理するためには, サーバ上でのセッション管理が必要となるためこのような構成にした.

6 問題点と考察

運用面で大きな問題となるのはデータの作成である．今後 XML が HTML に続く WWW のデファクトとなっていくという予想のもとに XML 技術を採用したわけだが，現状では XML に対応しているツールはまだ少ない．

仕様書などのテキスト文書の多くは，現状では Microsoft Word により作成される．しかし，Word には XML 生成機能がないのでこれを変換する必要がある．いくつか市販のツールが存在するが，使い勝手がよいとは言えない．一方，XML/SGML エディタに関しては調査する必要があるが，使い慣れたツールから乗り換えるには壁が大きいと思われる．

最も解決の難しい問題として DTD の生成が挙げられる．CrystalScope では，プロジェクト毎に異なる任意の種類の文書を扱いたいので，DTD に関しては簡単な規則を定めた以外は自由に定義できるようになっている．逆に言うと，プロジェクト毎に DTD を作る必要があり，これは XML の専門知識を持たない人には難しい作業となる．

最も手間のかかる作業はリンクの生成である．簡単なサポートツールを用意しているが，まだまだ使い勝手はよくない．また，文書の種類や量が多くなるとサポートツールがあったとしても，リンクの保守は容易ではなくなる．従って，可能な限りリンクの自動生成を行う必要がある．CASE ツールの図やソースコードでは，その同一文書クラス内でのリンクについては精度の高い自動生成が可能であろう．一方，テキスト文書のリンク，異文書クラス間リンクの自動生成は，文書の類似性を用いる方法などが考えられるが，精度はよくないだろう．しかし，自動生成した後にツールによって修正していくようにすれば，作業量は減るだろう．

データ作成のもうひとつの側面として，文書表示スタイルの問題が挙げられる．CrystalScope では，サーバ上の Java プログラムが XML 文書を HTML に変換するので，スタイルを変えるには Java のプログラムを書く必要がある．一方，XSL, CSS 等のスタイル記述言語についてはブラウザでのサポート状況とともに検討していく予定である．

7 まとめ

本稿ではソフトウェア文書の理解支援システム CrystalScope を紹介し，CrystalScope における XML 技術の応用について説明して，問題点を考察した．今後，オーサリングツールに関する調査は続行しつつ自作できる部分は自作していき，一方で実際ソフトウェアの開発者に利用してもらうことで評価と改良を行っていく計画である．

参考文献

[1] A. von Mayrhauser and A.M. Vans. Program Comprehension During Software Maintenance and Evolution. *IEEE Software*, Vol. 28, No. 8, 1995.

[2] P.K. Garg and W. Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, Vol. 7, No. 3, 1990.

[3] J.C. French, J.C. Knight and A.L. Powell, Applying Hypertext Structures To Software Documentation. *Information Processing & Management*, Vol. 33, No. 2, 1997.

[4] W3C. Extensible Markup Language (XML). *http://www.w3.org/XML/*

[5] W3C. Document Object Model (DOM) Level 1 Specification Version 1.0. 1998. *http://www.w3.org/TR/REC-DOM-Level-1/*

[6] W3C. Working Draft: XML Linking Language (XLink). 1998. *http://www.w3.org/TR/WD-xlink*

[7] W3C. Working Draft: XML Pointer Language (XPointer). 1998. *http://www.w3.org/TR/WD-xptr*

[8] 田村健人. XML プロセッサとその利用法. 人工知能学会誌, Vol.13, No.4, 1998.