2008－SE－159 (27)
2008／3／18

シェルスクリプトのための XML モデル

嶋野 淳子† 新家 博文† 四野見 秀明†

メインフレームシステムを中心としたレガシーシステムが抱えている保守の問題を解決するための手法は成熟しつつある．今後はオープン系システムもレガシーシステムと同様の問題を抱える可能性があると考え，オープン系システムで使用されているプログラムの解析手法の開発を行った．オープン系システムの多種多様なプログラム言語を解析するために，プログラムコードを XML に変換して言語依存を少なくし，解析ツールを一元化することを目標として定めた．シェルスクリプト言語の中から Korn シェルを採用し，XML モデルを設計し，このモデルの有効性が確認し他言語への適用を行うこととした．設計した XML モデルに従って変換ツールを実装し，プログラムから生成された XML に基づく解析ツールを開発した．その結果，設計した XML のモデルを用いた場合の開発の容易性，有効性が確認できた．

XML Model for Shell Scripts

Junko Shimano†, Hirofumi Shinke†, and Hideaki Shinomi†

The solving methods for the issue of maintenance have been mature for the legacy system on mainframe systems. There is a possibility that the open systems, also, may have the issue similar to the legacy systems in the future. Then, we have started developing an analytical technique for open systems. Because we make an analysis of various program languages on the open systems, we are aiming at the unification of analysis tools by converting source code into XML to minimize the language dependence. We chose Korn shell from shell script languages, and, designed a XML model for Korn shell. We will apply it to multiple languages in the future. We have developed a tool based on the XML model for Korn shell, and then we converted some program code into XML document. Next, we have developed analysis tools for the converted XML documents. Finally, we have verified that the converted XML documents are possible to develop tool at easily and useful.

1. はじめに

長年に渡って使われつづけているメインフレーム上のシステムなどのレガシーシステムは，システムの複雑化，メンテナンス性の悪さ，コストの増大など，様々な問題を抱えている．こういったレガシーシステムが抱えている問題を解決するための手段として，プログラム解析技法の検討[1][2][3]，プログラム保守支援技法の検討[4][5]，移行技法が研究されてきた．そして，これらの技術を利用してレガシーシステムのプログラム保守支援，新システム環境へのプログラムコードの移行などが実施され始めて既に数年が経過している．

一般にレガシーシステムという言葉は主にメインフレームを指しているが，ライフサイクルの短いオープン系システムも現状のレガシーシステムと同様の問題を抱える可能性を秘めており，今後レガシーシステムという言葉が指す範囲は拡大していくと予想できる．そこで，オープン系システムに対するプログラム解析技術や新システム環境への移行技術の確立が必要になると考え，オープン系システムで使用されているプログラム言語の解析技法の設計を行った．

オープン系システムのプログラム保守支援および新システム環境への移行支援を行うためには，システム全体の解析が必要となる．システム全体を解析するためにはオープン系システムに用いられている全てのプログラム言語を対象としなければならないが，オープン系システムで使用されているプログラム言語は多種多様である．これまでのようにプログラム言語ごとにプログラム保守支援やプログラムコードの移行の技法とツールを作成するのは非効率であり，ユーザの要望に応えられるようになるまで膨大な期間を要してしまう．そこで，プログラム言語の解析や移行を行う前に，プログラムコードから言語依存を少なくするための変換を行い，変換後の言語依存の少ないプログラムコードの解析技法を確立することで，技法とツールの一元化を図ることにした．

変換後の言語依存をなくした表記には，汎用的なマークアップ言語である XML を以下の理由で採用した．

- 稼動環境やプログラム言語に依存しないで処理を実現できる

†株式会社 日立コンサルティング
Hitachi Consulting Co., Ltd.

・ 実装のための API, ツールが整っている

XML モデルの設計は Java や C 言語でなどではすでに実施されているため, これらの言語とは異なる特性を持つシェルスクリプトから行った. そしてシェルスクリプトの中から, SYSTEM V release 4 の配布に含まれたため商用 UNIX 環境で広く使われており, Bourne シェル(以下, Bシェル)とも互換性のある Korn シェル(以下, K シェル)を選択した.

以降、2 節で関連研究, 3 節で XML モデルの設計, 4 節で実装とその評価, 5 節での XML 変換後のシェルスクリプトの解析ツールへの適用, 6 節でまとめと今後の課題について述べる.

2. 関連研究

XML を用いたプログラムのソースコードの中間形式への変換は, これまで Java, C 言語を対象として研究されている. そして XML 表記に変換する目的は, CASE ツールのプラットフォーム構築[6][7][8], プログラム理解[10], プログラム解析情報のデータベース化[11]が主なものであった.

これまでに多く扱われているのは, CASE ツールプラットフォームのための XML モデルである. 吉田らによる XML を用いた汎用的な細粒度ソフトウェアリポジトリの実装[6]では, Java 言語を対象にした XML モデルを設計し, プログラムソースの文, 式といった構文情報の他, 字句情報までも管理できるリポジトリについての研究を行っている. 丸山らによる XML 表現を用いた CASE ツールのプラットフォーム[7]では, 吉田らが設計した XML モデルをベースに, より詳細のモデルを設計し, ソースコード中の様々な情報を共有, 交換する目的のために開発者が表現を拡張できるようになっている[9].

川島らによる XML を用いた ANSI C のためのプラットフォームの検討[8]では, XML を用いた CASE ツールプラットフォームを構築し, 統一的で効率的なソフトウェア開発を検証し, 実証している.

Badros による JavaML の研究[10]では, プログラム理解を目的として Java 言語の XML のモデルを設計している. オブジェクト指向型言語を抽象化した共通モデルへと拡張できる可能性を示唆している.

山中らによるプログラム解析情報の XML データベース化[11]は, 解析効率の向上と解析結果の容易な二次利用を目的とした, XML データベースを作成し, 効果を確認している.

このように, プログラムコードを XML に変換するという技術の有効性は既に実証されている.

3. XML モデル

3.1. XML モデル作成の目的と構想

2 節で述べたように, XML を用いた言語表記とその利用法は様々で, 言語および目的によって様々な XML モデルが出来上がっている. そこで, XML モデルを一つに統一し, 利用目的に応じたツールを開発することができる XML モデルを作成し, 図1のように, 最終的に XML 化したプログラムを用いたプログラム保守支援, 新システム環境への移行支援などのための技法とツール作成を実現することを最終目的として, 各言語に共通して使用できる XML モデルの設計を行った.

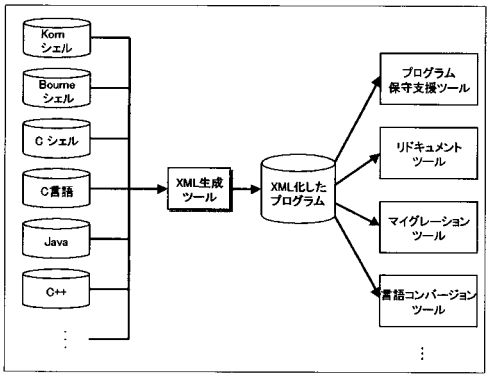

図1 XML 化の目的

プログラム言語に共通する XML モデルは, 最終的にはオープン系システムで広く利用されている手続き型言語とオブジェクト指向型言語を対象とするが, 手続き型言語の XML モデルの設計から行った. オブジェクト指向型言語については, 手続き型言語で設計した XML モデルの有効性を確認した後, 最終的にオブジェクト指向型言語向けに拡張することとした.

手続き型言語を対象とした XML モデル設計の取り掛かりとして, インタプリタ言語であるため Java や C 言語などとは異なる特性をもつシェルスクリプトの中から, B シェルとの互換性のある K シェルを選択した. XML モデルを最終的に各言語に共通して使用できるようにするために, XML モデルの設計時には C 言語の構文も考慮に入れた.

3.2. XML の要素および属性

プログラムは全て一つのプログラムファイルと, ステートメント, エクスプレッションで構成されると考えて, シェルスクリプトの構造とのマッピングを行った. このマッピング結果から, プログラムファイルの要素を File_inf, ステートメントの要素を Stmt, エクスプレッションの要素を Expr と定義した. さらに, その他の要素として, コメント, Expr を構成する変数, 変数宣言, Case 文などのラベルを抽出した. それぞれ, コメントの要素を Comment, 変数の要素を Ident, 変数宣言の要素を Decla(シェルでは変数宣言は行っていないため未使用), リテラルの要素を Literal, ラベルの

要素を Label と定義した．定義した要素を一覧にまとめた結果は表1の通りである．

表1　要素一覧

	要素名	英語名	概要
1	File_inf	File information	ファイル単位のプログラム
2	Func_def	Function definition	関数宣言
3	Stmt	Statement	プログラムを構成する文。(IF Stmt,While Stmt など)
4	Expr	Expression	Stmt中の条件式、演算式など
5	Decla	Declare	データ項目(変数)などの宣言部
6	Comment	Comment	コメント
7	Ident	Identifier	プログラム中で記述されている関数名、コマンド名、ファイル名など
8	Literal	Literal	リテラル値
9	Label	Label declaration	ラベル名(case文のラベルなど)
10	Unknown	unknown	不明要素

加えて，それぞれの要素の属性を表2の通りに定義した．表中のデータ型は，XML Schema[12]で用意されているデータ型で，xs:ID は識別子，xs:string は文字列，xs:integer は整数，xs:boolean はブール型，xs:IDREF は参照 ID の識別子を表す．

表2　属性一覧

	属性名	データ型	概要
1	id	xs:ID	識別ID(データ項目名、関数名など)
2	name	xs:string	ファイル名、データ項目名称(変数名)
3	line	xs:integer	元ファイルの行番号
4	sort	xs:string	elementの種別
5	brace	xs:boolean	brace_group([_ _ _ _])
6	asynchro	xs:boolean	バックグラウンドプロセスの場合はtrue
7	defid	xs:IDREF	変数、関数を宣言している要素のid
8	size	xs:integer	桁数
9	type	xs:string	タイプ
10	quoted	xs:boolean	ExprのAttributeでsort="COMPOUND_STR"の時、""でquoteされている場合はtrue
11	unknown	xs:string	不明属性

全ての要素に共通して使用する属性は識別 ID (属性名：id)，要素の種別(属性名：sort)，元ファイルの行番号(属性名：line)である．識別 ID は，XML を構成するツリーのノードを一意にするために付与する．要素の種別には File_inf, Stmt, Expr などの要素に意味を持たせるための属性値を，要素別に設定した．元ファイルの行番号は，元のファイルの情報を保持するために全ての属性に付与している．共通して使われないその他の属性については，要素に応じて使用する．

このように要素と属性と属性値を定義し，言語非依存を目指した K シェルのスクリプト向けの XML Schema を設計した．そして，この XML Schema に従って実際にプログラムソースを XML に変換し，要素および属性，値の検証と K シェルを対象とした XML の設計を行った．その結果，K シェルの使用を表現するための課題が浮かび上がってきた．その課題の中から，コマンドの表記方法，K シェル特有の構文を持つ let コマンドおよび for 文，シェルスクリプト特有の機構である入出力リダイレクションの表記方法について以下で述べる．

3.3. コマンド表記

K シェルのコマンド表記の設計にあたって，K シェルで使われているコマンドの種別を洗い出した．その種別は以下の通りである．

- K シェル内に組み込まれているビルトインコマンド
- プログラム中で定義する関数
- 外部プログラムまたは外部コマンド起動

外部プログラムまたは外部コマンドの種別は以下の通りである．

- ユーザが作成したプログラムまたはコマンド
- OS が提供しているコマンド
- プログラムプロダクト(以下 PP)やサードパーティが提供している製品などのユーティリティ

C 言語の場合，ビルトインコマンドに相当するのは標準ライブラリ関数，外部プログラムまたは外部コマンド起動に相当するのは標準ライブラリ関数の system()である．

K シェルと C 言語の双方でコマンド呼出が存在し，さらに共通するコマンドの種別があることから，K シェル，C 言語双方で使用できるようコマンドを表3のように分類した．

表3　コマンドの分類

	コマンド種別	概要
1	ユーザ定義アプリケーション呼出	ユーザが作成した外部プログラム，関数の呼出
2	OSコマンド呼出	OSコマンドの呼出
3	組込みコマンド呼出	Kシェル:ビルトインコマンド C言語:標準ライブラリ(ANSI規格)
4	ユーティリティ呼出	PP，サードパーティの製品などのユーティリティ

コマンド呼出の XML 表記の要素には Expr を用い，コマンド呼出を表す sort="FUNC_CALL"を定義した．表3で分類したコマンド種別は sort_func という属性と種別を表現する属性値で定義した．コマンド呼出とその種別の XML 表記は表4の通りである．

表4　コマンド呼出とその種別の XML 表記

	コマンド種別	XML表記(id, lineは省略)
1	ユーザ定義アプリケーション呼出	<Expr sort="FUNC_CALL" sort_func="USER_DEF">
2	OSコマンド呼出	<Expr sort="FUNC_CALL" sort_func="OS_CMD">
3	組込みコマンド呼出	<Expr sort="FUNC_CALL" sort_func="STD_LIB">
4	ユーティリティ呼出	<Expr sort="FUNC_CALL" sort_func="UTILITY">

ユーザ定義アプリケーションの呼出には，ユーザがプログラム中で定義した関数の呼出も含んでいる．同様にユーティリティ呼出にも，ユーティリティから提供されている関数呼出が存在している．しかし，sort と func_sort の属性値だけでは，プログラム中に定義されている関数呼出の表現ができない．そこで，ユーザ定義関数呼出とユーザ定義コマンド呼出，ユーティリティ提供の関数呼出とユーティリティコマンド呼出で違う点を洗い出した．

関数呼出の場合は一つのプロセス上で処理が行われるのに対して，ユーザ定義コマンド，ユーティリティコマンドの呼出の際には，別プロセスで処理が行われる．別プロセスが起動されるか否か，という観点を関数呼出であるか否かの区別に用いることにした．そして，新しく system という xs:boolean 属性を追

加し，別プロセスが起動している場合はsystem="true"と記述することにした．また，別プロセスが起動しない場合はsystem属性を省略する．各コマンドの種別と別プロセス起動の有無をまとめた結果は表5の通りである．

表5　コマンド種別と別プロセス起動の関係

	func_sort	説明	system	
			true	false
1	USER_DEF	ユーザ定義アプリケーション	○	○
2	OS_CMD	OSコマンド	○	×
3	STD_LIB	Kshell：Built In C言語：Std Lib (ANSI規格)	×	○
4	UTILITY	PP、Third partyの製品など	○	○
5	UNKNOWN	上記の区別ができないもの	○	○

実装の際は，言語や環境に依存しないよう，外部から種別ごとに作成したコマンド名の一覧情報を読み込み，コマンド名の一覧からから得た情報でコマンドの種別を判断しfunc_sortの属性値を決める．また，コマンド種別情報が与えられない場合のために，func_sortの属性値に"UNKNOWN"も定義した．

プログラム中で定義した関数を呼び出した場合と，ユーザが作成したコマンドを呼び出した場合のXML表記は以下の通りである．(id, lineは省略)

プログラム中で定義したuser_functionという関数を呼び出した場合

```
<Expr sort="FUNC_CALL" func_sort="USERDEF">
  <Ident sort="FUNCTION">user_function</Ident>
</Expr>
```

プログラム中で定義されている関数を呼び出す場合は，別プロセスが起動されないため，system属性は省略される．

ユーザが作成したuser_comという実行ファイルを呼び出した場合

```
<Expr sort="FUNC_CALL" func_sort="USERDEF" system="true">
  <Ident sort="FUNCTION">user_com</Ident>
</Expr>
```

ユーザが作成したコマンドを呼び出した場合は，別プロセスが起動されるので，system属性の属性値を"true"として表記を追加する．このように，関数呼出と外部のコマンドの呼出を区別する．

なお，以上の設計で追加した表6の属性を，表2の属性一覧に追加する．

表6　コマンドのための属性

	属性名	データ型	概要
12	func_sort	xs:string	Exprが sort="FUNC_CALL"の場合の種別
13	system	xs:boolean	Exprのsortが"FUNC_CALL"の時、そのコマンドが別プロセスとして動く場合はtrue

3.4. letコマンド表記

K シェルでは，算術演算式，数値の比較演算式を用いる場合はletコマンドというビルトインコマンドを使用する．C言語など変数の型宣言をする言語であれば，使用する変数が数値であるか否かが明確であるため，演算式を宣言する必要はない．しかし，Kシェルは文字列演算に特化した仕様になっているため，letコマンドで数値の演算を行うことを指示しなければならない．

letコマンドはKシェルのビルトインコマンドであるが，3.3節で述べているコマンドとは異なる単なる数値演算である．また，letコマンドという特殊なコマンドを用いているが，他のプログラム言語の数値演算式と同じであったため，コマンドにはせず，数値演算式として扱うことでKシェルの言語依存をなくした．

letコマンドのプログラム中の表記は，表7の1，2のようにletコマンドを演算式の前に付与する方法と，((演算式))のように二重括弧で演算式を囲む方法の2種類ある．また，letコマンドで表記されていない演算式は文字列代入として扱われる．

表7　letコマンドの有無による変数の値

	プログラム中の表記	a=2,b=3の場合の変数の値	備考
1	let c=a+b	cの値は5	letコマンド
2	((d=b-a))	dの値は1	letコマンド
3	e=a+b	eの値は"a+b"	

このように複数の構文が存在している点，演算式のように見えても letコマンドによる宣言が無い場合の扱いの区別を設計に盛り込み，XMLの表記ルールを以下の通りに定義した．

- letコマンド表記がある場合は数値演算式
- letコマンド表記が無い場合は文字列代入
- letコマンド表記がある比較演算式の場合は数値の比較

letコマンド表記がない比較演算式の場合は，表記の方法によって意味が変化する．letコマンド表記とは別の内容なので詳細は割愛する．

a=2, b=3のように，数値を文字列として代入し，その後演算中で数値として扱われる場合もある．この場合についても代入式がletコマンドで表記されていれば数値代入，無い場合は文字列代入として扱う．

letコマンドがない場合とletコマンドの演算式である場合のXML表記は以下の通りである．(id, lineは省略)

let c=a+b または ((c=a+b))の場合

```
<Stmt sort="EXPR">
  <Expr sort="ASSIGN">
    <Ident sort="DEF_VALUE">c</Ident>
    <Expr sort="ADD">
      <Ident sort="REF_VALUE">a</Ident>
      <Ident sort="REF_VALUE">b</Ident>
    </Expr>
  </Expr>
</Stmt>
```

e=a+b の場合

```
<Stmt sort="EXPR">
  <Expr sort="ASSIGN">
    <Ident sort="DEF_VALUE">e</Ident>
    <Literal>a+b</Literal>
  </Expr>
</Stmt>
```

3.5. for文表記

for 文には以下の2つの構文が定義されている.

【for構文1 (Ksh88)】

```
for Identifier [in Word ... ];
do
    list ;
done
```

【for構文2 (Ksh93)】

```
for(([expr1];[expr2];[expr3]));
do
    list;
done
```

このように, for 構文1(Ksh88)と for 構文2(Ksh93)は同じ for 文であるが, 別の構文を持っている. for 構文2の構文は C 言語など他の言語の for 文とほぼ同様の構文であるのに対して, for 構文1は K シェル特有の構文である. そこで, Ksh93 で追加された for 構文2は, for 構文1とは別のループ構文として扱い, この2つの for 文を別の XML モデルで表現することとした.

また, for 構文1は他の言語には無い K シェル特有の表記であるが, for 構文2は前述の通り, C 言語など他の言語の for 文とほぼ同様の構文であるため, for 構文2の Ksh93 は C と共通に扱うことを前提として, 表8のように属性値を定義した.

表8　for 構文の属性値

	要素	属性および属性値
for構文1の場合	Stmt	sort="FOR_LIST"
for構文2の場合	Stmt	sort="FOR"

加えて, for 構文全体の XML 表記のルールをそれぞれ以下の通りに決定した. (id, line は省略)

for Identifier [in Word ...];do list ;done (for構文1)の場合

```
<Stmt sort="FOR_LIST">

  <Ident>Identifier</Ident>
 </Expr>
 <Expr sort="LIST">
  <Literal>Word1</Literal>
  <Literal>Word2</Literal>
        :
        :
 </Expr>
 <!-- Wordが空文の場合は<Expr sort="EMPTY" /> -->
 <Stmt sort="BLOCK">
    list のXML表記
 </Stmt>
</Stmt>
```

for(([expr1];[expr2];[expr3]));do list;done (for構文2)の場合

```
<Stmt sort="FOR">
 <Expr sort="***">
    expr1の表記
 </Expr>
 <Expr sort="***">
    expr2の表記
 </Expr>
 <Expr sort="***">
    expr3の表記
 </Expr>
 <!-- exprが空文の場合は<Expr sort="EMPTY" /> -->
 <Stmt sort="BLOCK">
    list のXML表記
 </Stmt>
</Stmt>
```

3.6. 入出力リダイレクション表記

入出力リダイレクションはシェルスクリプト特有の機構である. そのため, 入出力リダイレクションの XML モデルはシェルスクリプト固有のモデルとなり, 完全な言語非依存とすることはできないことが判明した. そこで, 言語特有の構文であってもこれまでの XML モデルを崩さない方針で設計を行った.

K シェルの入出力リダイレクションには, 表9の機能がある.

表9　入出力リダイレクションの機能

	I/Oリダイレクトの表記	機能
1	[n]<word	ファイル word を入力として使用
2	[n]>word	ファイル word を出力として使用
3	[n]>|word	noclobber オプションを無視する 点を除き, 1と同様
4	[n]>>word	ファイル word を追加モードで出力
5	[n]<>word	ファイル word をオープンし, 入力として読取り, 書込みを実施
6	[n]<<[-]word	ヒアドキュメント ハイフン (-) が 付加されている場合は、 wordから先行タブをすべて除去
7	<&digit	標準入力がファイル記述子
8	<&-	標準入力をクローズ
9	>&-	標準出力をクローズ
10	<&p	コプロセスからの入力を標準入力に移動
11	>&p	コプロセスへの出力を標準出力に移動

※ n はファイル記述子. nの指定がない場合は標準入出力になる.

表9の機能に加えて, XML で以下の内容も表現できなければならない.

- 入出力リダイレクションは Expr にも Stmt にも付けることが可能である.
- コマンドの中のどの部分にでも記述可能で, コマンドの前後どちらに置いても良い.
- 記述の順に処理されるため, 順序に意味がある.

XML 表記の定義にあたって, リダイレクションの機能と要件を表10のように分類した.

表10　入出力種別による機能の分類

	入出力の種別		
	INPUT	OUTPUT	INPUT/OUTPUT
標準入出力	◎	◎	◎
ファイル記述子指定	◎	◎	◎
noclobberを無視		○	
追加モード		○	
ヒアドキュメント	◎		

◎:他の機能と同時指定が可能
○:他の機能と同時指定できない場合がある

表10の分類を元にファイルの入出力, ファイル記述子, ヒアドキュメントなどを表現できる XML モデルを設計した.

入出力リダイレクションは Expr にも Stmt にも付けることが可能であるという特徴があるため, XML 表記の要素を Expr, 属性値を sort="REDIRECTION" と定義した. そして, 入出力種別を表現するための属性を mode(xs:string)とし, 表11のように属性値を設定した. sort="REDIRECTION"である場合は, mode 属性を必ず指定しなければならない.

表11　mode の属性値

	種別	mode
1	標準入力	INPUT
2	標準出力	OUTPUT
3	標準入出力	INOUT

ファイル記述子の指定は，全ての入出力種別において可能である．そこで，新たな属性 descriptor (xs:integer)を設定し，descriptor="n"でファイル記述子の指定を表現とした．Digit パラメータが指定されていない場合，description 属性は省略可能である．

">|word"(noclobber オプションによる出力リダイレクトの禁止を無視し強制的に上書き)，">>word"(追加モード)は，出力に関連するオプションであり，同時に指定することができない．そこで，共通の属性とし，属性値で区別することにし，属性を option (xs:string)，属性値を表12のように設定した．出力に関するオプション指定がない場合は，option 属性は省略可能である．

表12　option の属性値

	リダイレクション表記	optionの属性値
1	>|word	CLOBBER
2	>>word	APPEND

ヒアドキュメントは，コマンドへの入力をシェルの標準入力として，label 行まで読み込む間の入力をいう．ヒアドキュメントは，Stmt に相当するもの，Expr に相当するもののどちらでも指定できるため，表9の項番6のwordは項番1～5までのwordと同等の扱いでは表現が難しい．また，その word がヒアドキュメントであることを明確にする必要もある．そこで，ヒアドキュメントの word 場合は，Expr の属性 sort の属性値に"HERE_DOCUMENT_TEXT"を追加し，以下のように表記することとした．(id, line は省略)

```
<Expr sort="REDIRECTION" mode="INPUT">
    <Expr sort="HERE_DOCUMENT_TEXT">
        (ヒアドキュメントの内容)
    </Expr>
</Expr>
```

表9の項番7～11までのリダイレクション表記に関しては，1～6までのリダイレクション表記と揃えるために&以降を全てwordとして考えて扱うことにした．しかし，このwordはヒアドキュメントと同様に項番1～5までの word と意味が違うため，同等の扱いはできない．そこで，このwordに関しても Expr の属性 sort に属性値"FILE_DESCRIPTOR"を追加し，以下の様に表記することとした．(id, line は省略)

例) >&p の場合

```
<Expr sort="REDIRECTION" mode="OUTPUT">
    <Expr sort="FILE_DESCRIPTOR">
        <Literal>p</Literal>
    </Expr>
</Expr>
```

個々のリダイレクションの表記に関する設計を行った後，リダイレクションを実施しているコマンドまたはステートメントを含めたXML表記を設計した．この際に，実行順序の表現も同時に表現する．コマンドまたはステートメントに対してリダイレクションが実施されていることを明確にすることが必要である．そこで，リダイレクションの宣言を行うXML表記を追加することとし，Expr の属性 sort に属性値"REDIRECTION_LIST"を追加した．

以上の結果から作成したリダイレクションの XML 表記は以下の通りである．(id，line は省略，COM1,COM2 はユーザ定義コマンドとする．)

COM1 1>> file1 の場合

```
<Stmt sort="EXPR">
    <Expr sort="FUNC_CALL" func_sort="USER_DEF" system="true">
        <Expr sort="REDIRECTION_LIST">
            <Expr sort="REDIRECTION" mode="OUTPUT"
                  option="APPEND" descriptor="1">
                <Literal>file1</Literal>
            </Expr>
        </Expr>
        <Ident sort="FUNCTION">COM1</Ident>
    </Expr>
</Stmt>
```

COM1 < file1 > file2 の場合

```
<Stmt sort="EXPR">
    <Expr sort="FUNC_CALL" func_sort="USER_DEF" system="true">
        <Expr sort="REDIRECTION_LIST">
            <Expr sort="REDIRECTION" mode="INPUT">
                <Literal>file1</Literal>
            </Expr>
            <Expr sort="REDIRECTION" mode="OUTPUT">
                <Literal>file2</Literal>
            </Expr>
        </Expr>
        <Ident sort="FUNCTION">COM1</Ident>
    </Expr>
</Stmt>
```

以上の設計で追加した表13のリダイレクションのための属性値を，表2の属性一覧に追加する．

表13　リダイレクションのための属性

	属性名	データ型	概要
14	mode	xs:string	ExprのAttributeでsort="REDIRECTION"の時に使用する リダイレクションのI/Oモード
15	option	xs:string	ExprのAttributeでsort="REDIRECTION"の時に使用する ファイルの上書き、追加に関するオプション
16	descriptor	xs:integer	ファイル記述子の指定

4. 実装

K シェルのスクリプトを基に設計した XML モデルを用いて K シェルのスクリプトを XML に変換するツールを作成した．開発言語には，構文解析や XML に関するツールや API が揃っていることから Java を採用した．

ツールの構成は，図2の通りである．

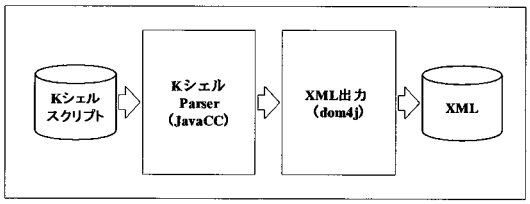

図2　ツールの構成

Kシェルの構文解析部分にはJavaCC[13], XMLフレームワークである dom4j[14]を利用した. K シェルのパーザーは JavaCC で作成した. dom4j ではパーザーが作成した抽象構文木を DOM(Document Object Model)ツリーに変換し, 設計したモデルに従ったXMLを出力する.

ツールで K シェルのスクリプトのサンプルからXMLを生成した結果, XML変換前と変換後の総サイズと総ステップ数および処理時間は表14の通りであった. なお, サンプル1およびサンプル2は顧客より借用した実際に運用で使用しているスクリプト, サンプル3はWebから収集したスクリプトである.

表14　ファイルサイズとステップ数

	本数(本)	Kシェル スクリプト		XMLファイル		処理時間(ms)
		サイズ(KB)	ステップ数(step)	サイズ(KB)	ステップ数(step)	
サンプル1	42	207	7,112	1,240	25,081	4,076
サンプル2	13	243	4,797	1,050	21,981	2,814
サンプル3	140	193	8,792	1,590	34,189	4,506

XML ファイルの総ステップ数は K シェルのスクリプトの約 4 倍であった.

表14の処理を実施したマシンのスペックは以下の通りである.

【マシンスペック】
CPU:Intel Pentium M Processor 1600Mhz
メモリ:512MB
OS:Windows 2000

5. 解析ツールへの適用

本研究の目的は, 設計したXMLモデルを使ってXML に変換したプログラムを解析し, 必要な情報を取得することにある. そこで, メインフレームシステム向けのサービスで使用しているプログラムの設計情報を生成するツールに, XML に変換された K シェルのスクリプトを解析する機能を追加し, システムのジョブの実行を確認するためのジョブフロー図を生成する.

ツールの概要はエラー! 参照元が見つかりません。の通りである.

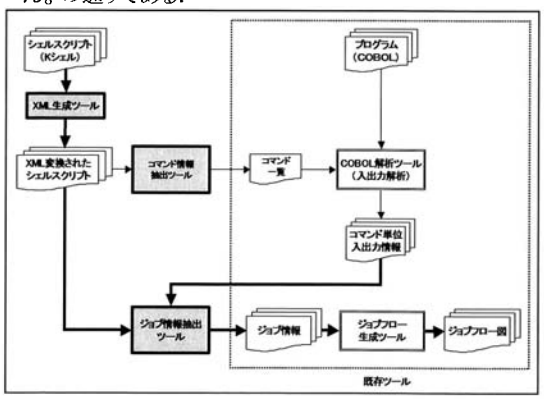

図3

既存ツールとの連携の目的は, オープン COBOL とシェルスクリプトで構成されているシステムの仕様回復サービスを実現し, 既存サービスの適用範囲を拡張することにある. そのため, 既存ツールには修正を加えずにジョブフロー図を生成できるよう, 抽出する情報のフォーマットは既存ツールのフォーマットに従った. 既存ツールとの連携の結果生成されたジョブフロー図は図4の通りである.

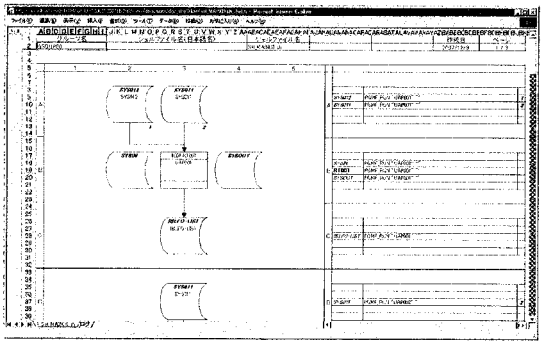

図4　ジョブフロー図

既存ツールとの連携の他に, シェルスクリプトの呼出関連図を作成した. XML 変換後のシェルスクリプトから, シェルスクリプト中で呼び出されている関数と, 呼び出された関数を定義しているスクリプトの情報を抽出してシェルスクリプトの関連情報を生成した. そして, AT&T 研究所が開発したオープンソースのツールパッケージである graphviz[15]を利用して関連情報を視覚化した. 出力したシェルスクリプトの呼出関連図は図5の通りである.

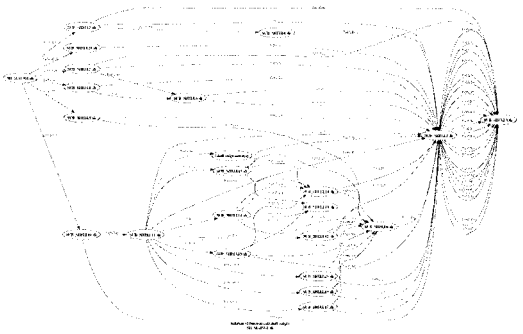

図5　シェルスクリプト呼出関連図

以上のツール開発は, K シェルの知識および構文解析のスキルのない Java の開発経験者が担当したが, 約2ヶ月で開発を終了することができた. Kシェルのスクリプトを XML に変換する際にスクリプトが抽象化されていること, Java の API が充実していることが, ツール開発を容易にしたと考えられる. このこと

から，プログラムから変換されたXMLを用いることで，要望に応じたサービスの提供が期待できる．

6. まとめと今後の課題

3節で述べたKシェルのXMLモデルの設計を通して，リダイレクションのようなシェルスクリプト特有の構文が存在するため，XML 化で全ての言語依存を無くすことは難しいことがわかった．しかし，言語依存の構文がある場合でも，基本の XML モデルは崩さずに言語依存部分が設計できることも同時に確認できた．このことから，今後の他言語における XML モデルの設計において，言語に共通する構文の XML モデルと言語に依存する構文の XML モデルの明確化を行っていく必要がある．

また，5 節で述べたように，XML に変換されたプログラムは複雑な構文解析を行う必要がないため，目的に応じたツールの設計と開発が，プログラムコードを追って解析するこれまでの方法と比べて容易にできた．このことから，XML に変換されたプログラムを利用することの有効性が確認できた．

本稿で述べた XML モデルは K シェルを元に設計したものである．そこで，今後は B シェル，C シェル，さらに C 言語への展開を行う予定である． 以下に各言語のモデルを検討するにあたっての課題を述べる．

B シェルの場合は，K シェルとの互換性があるため，KシェルのXMLモデルを殆どそのまま適用することができる．ただし，数値演算式など，K シェルとは違う構文もあるので差分を調べて対応する必要がある．

C シェルの場合は， K シェルとの差分の洗出しと差分に対する対策を十分に行う必要がある．例えば，K シェルの case 文に該当する switch 文など，K シェルの case 文と同じモデルが適用できるか否かを判断し，適用不可となった場合は XML モデルの追加などを実施する必要がある．

C 言語の場合は，KシェルのXMLモデル設計の際にC言語とKシェルの制御構文を比較しながらC言語にも共通するモデルを意識してきたが，K シェルにはないデータ宣言や構造体，ポインタなど追加が必要である．

参考文献

[1] 四野見秀明，玉井哲雄: プログラム解析を提供する API の実現とその適用，日本ソフトウェア科学会大会論文集, 2003.

[2] 山川敦夫，宮本信夫，上林高治，石本真希，秋庭真一，山村信幸，堀内一: データ中心分析によるプログラム論理の抽出，情報処理学会，Vol.1993 No.4, pp.107-115, 1992-IS-042, 1993

[3] 秋庭真一，戸田淳子，山川敦夫，堀内一: データ抽象化による既存ソフトウェアのリバース手法，情報処理学会，全国大会講演論文集 第49回平成6年後期,pp.335-336, 1994

[4] 四野見秀明，藤井邦和，高橋真由美: データフロー解析に基づくプログラム保守支援，情報処理学会第54回全国大会論文集

[5] 戸田淳子，秋庭真一，山川敦夫，堀内一: ソフトウェアリエンジニアリングを支援するリポジトリブラウザ，情報処理学会，全国大会講演論文集 第49回平成6年後期,pp.337-338, 1994

[6] 吉田一，山本晋一郎，阿草清滋: XML を用いた汎用的な細粒度ソフトウェアリポジトリの実装,情報処理学会誌, Vol. 44, No. 6, pp. 1509–1516, 2003.

[7] Maruyama, K. and Yamamoto, S.: A CASE Tool Platform Using an XML Representation of Java Source Code, 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04), pp. 158–167 , 2004.

[8] 川島勇人，権藤克彦: XMLを用いたANSI CのためのCASEプラットフォーム，日本ソフトウェア科学会，Vol.19, No.6(20021126) pp. 21-34, 2002

[9] Maruyama, K.: Java-XML Tools Project. http://www.jtool.org/.

[10] Badros, G. J.: JavaML: A Markup Language for Java Source Code, Proc. the Ninth International Conference on World Wide Web , 1996.

[11] 山中祐介，大畑文明，井上克郎: プログラム解析情報の XML データベース化－提案と実現－，日本ソフトウェア科学会 Vol.19, No.1(20020115) pp. 39-43, 2002.

[12] C. M. Sperberg-McQueen and Henry Thompson . XML Schema . http://www.w3.org/XML/Schema

[13] CollabNet.: javacc: JavaCC Home. https://javacc.dev.java.net/

[14] dom4j - dom4j: the flexible XML framework for Java. http://www.dom4j.org/

[15] Graphviz - Graph Visualization Software. http://www.graphviz.org/