WWWを利用したソフトウェア部品の管理ツール

大月美佳*・吉田紀彦**

WWW-based Management Tool for Software Components

by

Mika OHTSUKI* and Norihiko YOSHIDA**

Cooperative software development in distributed environments has become common because of recent popularization of wide area network. In such a development fashion, reuse of software components is necessary as well as in personal or local development, and object-oriented technology is important as a basis. Reuse of object-oriented software components in distributed environments, however, has not become common yet as has been expected. This originates in not only difficulty in understanding characteristics of object-oriented software components whose structures are complicated, but also absence of support mechanisms for obtaining components in distributed environments. In this paper, first of all, we discuss what are to be handled as reusable object-oriented software components, then propose a management mechanism for promoting reuse of the components in distributed environments.

1．はじめに

我々の最終的な目標は，オブジェクト指向ソフトウエアの分散環境での協同開発を，そのライフサイクル全般に渡って支援することにある．数人以上のチームでライブラリや部品を共有しながら1つのソフトウェアを構築していく場合，分散協同作業であることから従来のソフトウェア開発支援における課題に加えて，次のような課題も解決しなければならない．

- 多様なソフトウェア部品と多様な部品間関係の分散透過的かつ統合的な管理
- 協同作業に係わるアクセス制御，バージョン管理，意見調整支援，理解支援

後者のアクセス制御やバージョン管理については，協同開発支援に限らず一般の開発支援に対してすでに多くの研究があり，商品化もなされている．例えば，ソースコードのバージョン管理としてSCCS[1]，RCS[2]などがある．しかし，それらは概ね一元的な集中管理に基づいていて，分散化には至っていない．また，意見調整支援についてはグループウェア（CSCWとも呼ばれる）[3]の分野で研究が進められており，ソフトウェア協同開発への応用についても幾つかの試みが始まっている[4,5]．

一方，部品管理についてはソースコードのデータベース化の研究[6]などが見られるが，実際にはソフトウェア部品はソースコードだけでなく，仕様ドキュメント，マシンコード，テストデータ，メモやユーザのコメントなど様々なものを含む．さらに，デザインパターン[7,8]やフレームワークなども抽象度の高い部品と考えることができる．そして部品間の関係にも，抽象-具体，継承，全体-部分，参照など，様々なものがある．

また，今後のネットワークのさらなる広域化とそれに伴なう協同開発環境の広域分散化を鑑みると，従来のシステムにみられるような部品の集中管理には限界

平成10年10月1日受理

*九州大学大学院システム情報科学研究科　福岡市東区箱崎（Graduate School of Information Science and Electrical Engineering, Kyushu University, Hakozaki, Fukuoka）

**情報システム工学科（Department of Computer and Information Sciences）

があり，分散管理，しかも実際の所在を意識させない分散透過性が重要となる．

そこで，先の最終目標に向けた第一の取組みとして，本論文では特に上記の前者，部品と関係の分散透過的かつ統合的な管理に焦点を当て，これを実現する分散部品管理システムの構築について述べる．これは協同開発支援の最も重要な基盤の一つとなるものである．

まず統合管理について考えると，先に挙げたような多種多様な部品は書式や関係が定型化されていないため，データベースのような定型的な枠組でそれらを互いに関係づけて統合的に取り扱うのは困難である．このような不定型で多様な形態の部品を相互に関係づけて管理するために，最も有効な枠組の一つと考えられているのが，ハイパーテキストである．

ハイパーテキスト[9]は情報管理のための方式の一つであり，リンクによって結ばれたノードからなるネットワークによってデータを保持，管理する．ノードは情報の任意の一かたまりを表わし，テキストだけでなく音声や画像などのマルチメディア情報も扱い得る．リンクはノードの間の任意の関係を表わし，リンクを順次たどることによって求めるノードを探しだすことになる．従って，ソフトウェア部品をノード，それらの関係をリンクとすることにより，多様で不定型な部品を統合的に保持，管理することができる．

ハイパーテキストシステムの標準モデルであるDexterハイパーテキスト参照モデル[10]では，ハイパーテキストは蓄積層（storage layer），実行時層（run-time layer），部品内部層（within-component layer）の3層から構成される．ハイパーテキストの基本構造，すなわち原子部品（atomic component），複合部品（composite component），およびリンクの振舞いが蓄積層で規定され，アプリケーションから蓄積層へのインターフェースが実行時層で規定される．これに対して，本論文は部品内部層の構築，すなわち，どのようにソフトウェア部品をノードとして蓄積し，関連を分散透過的なリンクとして形成するか，に焦点を当てている．

ハイパーテキストの分散化は，ノードを分散的に配置してリンクを分散透過的に張ることによって実現する．これが可能であることは，すでに例えばWorld Wide Web（WWW）などによって示されている．

ソフトウェア開発支援に向けた部品管理にハイパーテキストを応用することを試みた関連研究として，これまでに次のような事例がある．

Dynamic Design[11]

米テクトロニクス社が開発したCASE環境であり，C言語を対象としている．ノードとして，要求，仕様，設計ノート，設計仮説，コメント，ソースコード，オブジェクト（マシンコード），シンボルテーブル，ドキュメンテーション，レポートを扱い，リンクとして，前後，言及，参照，手続き呼び出し，実装，定義の関係を扱う．

このDymnamic DesignはHypetext Abstract Machine（HAM）[12]と呼ばれるハイパーテキスト用仮想機械の上に構築されている．HAMが扱うハイパーテキストの構成要素は，グラフ，コンテキスト，ノード，リンクおよびアトリビュートである．グラフはノードとリンクのネットワーク構造を写像したもので，ネットワーク構造の可視化などに用いる．コンテキストはノードの順序を表わし，バージョン管理に用いる．アトリビュートは他の要素に対する注釈である．グラフやコンテキストは興味深い提案であるが，分散性についてみると，HAMの分散性は集中管理であって広域的な分散化や分散透過性は考えていない．

Chimera[13]

Adaの分散統合開発支援環境Archadiaの一部であり，ハイパーテキストの概念を抽象データ型として規定したインターフェースをツールに対して提供することで，分散部品データベースの内容をハイパーテキストとして透過的に扱えるようにするミドルウェア的なシステムである．このシステムは，我々の研究と異なり，ソフトウェア部品の構造と関連をどのように分散環境に配置して透過的に扱うかよりも，異種分散環境でのハイパーテキストのモデル化とインターフェースとしての実現に重点をおいている．

以下，第2章でオブジェクト指向ソフトウェアの分散協同開発支援に向けた分散部品管理システムの概念と基本設計，第3章で詳細設計と実装について述べる．第4章はまとめである．

2．分散部品管理システムの基本設計

オブジェクト指向ソフトウェアでは，部品群はその抽象度に応じて階層構造を成すと考えることができる．すなわち，デザインパターン[7,8]，フレームワーク，OOA/OODにおける図式記述[14,15,16]，クラス定義，といった順に具体化ならびに詳細化がなされる．部品間の関係は同一階層内だけでなく，階層にまたがる形でも抽象-具体関係がある．ここでは最初の取りかかりとして，最も基本的なクラス定義の層をまず考える．より抽象度と集約度の高い部品階層はこの上に重層的に構築される．

クラス定義層のハイパーテキスト構造の概観を図1

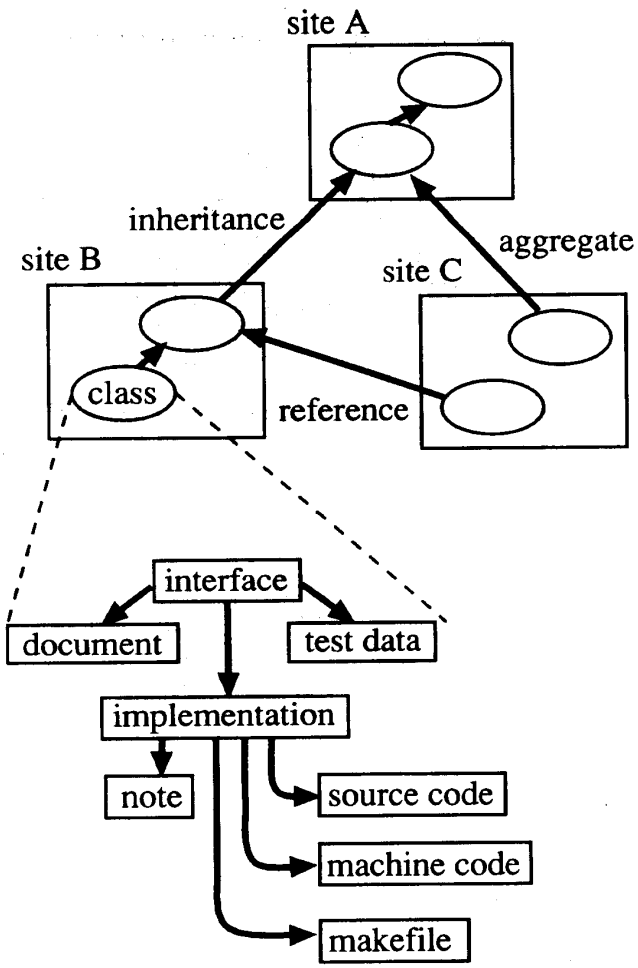

Figure 1. Hypertext structure of class-related information

に示す．ここに含まれるノードの種類は，クラスの公開インターフェースノード，クラスの実装ノード，ソースコードなどの実装レベル部品ノード，メモ他ドキュメントのノードなどであるが，これらに限定されるものではなく，利用者が新しい種類のノードを任意に定義・導入することも可能である．

- 公開インターフェースノード：クラスの公開インターフェース情報を保持する．
- 実装ノード：クラスの非公開インターフェース情報を保持し，実装レベル部品ノードへのリンクを持つ．ここでいうインターフェース情報とは，名前，引数や返り値などの Application Programming Interface（API）としての情報である．
- 実装レベル部品ノード：例えば C++ では，ヘッダファイル，ソースコードファイル，マシンコードファイル，Makefile などが含まれる．他のオブジェクト指向言語でも，クラスのソースコードファイルなどが相当する．

これらのノードは基本的にはクラスを単位としてグループ化され，公開インターフェースノードがそのグループの外部インターフェース窓口となる．そして，クラスの公開インターフェースノード相互の間で，クラス間の関係に応じて，分散的にリンクが張られる．このクラス間関係とは，参照関係，全体-部分関係，継承関係である．

なお，公開インターフェースノードと実装ノードを分けるのは，利用者の利用形態に応じたクラスの見え方の違いを提供するためである．すなわち，エンドユーザ的な参照に対しては，クラスをカプセル化された部品として，内部実装情報をむやみに提供すべきではない．一方で，開発者としての参照に対しては，クラスを拡張可能な部品として，内部実装情報が要求された場合に答えなければならない．このような見え方の違いは，分散協同開発やコンポーネントウェアにおいては特に重要である．

公開インターフェースノードと実装ノードはシステムが自動的に生成と管理を行う．他のノードは利用者が生成や関係付けを行い，システムはその手立てを提供する．クラス間関係に応じた公開インターフェースノード間リンクもシステムが自動的に張る．これらノードやリンクの自動生成は，利用者の与えるクラス定義をシステムが解析して必要な情報を抽出し，それに基づいて行う．

この解析と生成，情報蓄積，要求処理の一連の流れをモデル化して，図 2 に示す．ソースコードのように形式化されたテキストから構造を切り出して内部蓄積形式に変換する入力変換部，蓄積部，ユーザからの閲覧要求や問合せに対してノードとリンクを返答する出力変換部の 3 つの部分から成る．

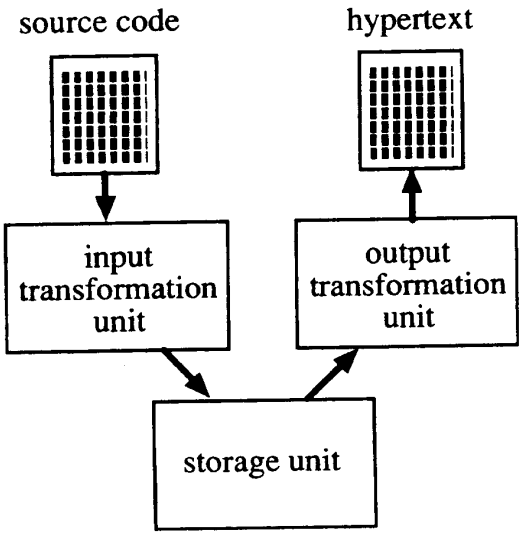

Figure 2. Processing model for hypertext

3．詳細設計と実装

システムの実装は，概念設計と利用者インターフェースがハイパーテキストであれば，具体的内部構造はハイパーテキストそのものでも，より厳密に定型化されたデータベースでも，より平板な単純テキストファイルでも可能である．いずれを採用するかは，情報の詳細化と構造化をどこまで行うか，どこまで複雑な検索要求を受け付けさせるか，などによって決まる．

ここでは，まず最初のプロトタイプであることから，最も直接的な実現として内部蓄積形式もハイパーテキストとし，その物理的配置情報のみを簡単なデータベースで管理する方式を取る．従って，出力変換部は実際には検索・閲覧処理のみを行い，変換処理は必要としない．

システムの対象言語は，開発現場で現在のところ最も多用されているオブジェクト指向言語である C++ を採用した．ハイパーテキストを実装する枠組は World Wide Web（WWW）の機構を一部借用し，その Universal Resource Locator（URL）によって分散環境上での所在を特定するものとした．

ここで，WWW そのものはリンクを辿ることしかできず，データベース的な管理機構を持たないので，名称などのキーと所在をテーブル化して管理し，効率的な検索を可能にする．この管理プログラムがサイトごとのサーバとして機能し，利用者の応対をするクライアント，およびサイトごとの WWW サーバとの連係によって全体的な検索や閲覧を実現する．なお，システム全体の中心的サーバは存在しない．サイトごとのサーバの分散的な協調によって，全体の管理を行う．

システムの全体像を図 3 に示す．図中の httpd が WWW サーバ，cmpd がテーブル管理サーバ，add_class，list_class，query_class がクライアントである．cmpd は Class-URL テーブルを管理する．add_class が先に述べた入力変換部に相当する．このクライアントは与えられた C++ のプログラムを自動解析して型・クラス情報を抽出し，WWW ノード形式である Hyper-Text Markup Language（HTML）ファイルを自動生成して関係ノードへのリンクを張る．他のクライアント，list_class と query_class は出力変換部に相当し，検索要求や閲覧などの処理を行う．

このサーバ・クライアント間では，次のようなプロトコルが授受される．

・ADD，UPDATE，DELETE：型・クラス情報の登録・更新・削除

・LIST，QUERY：型・クラス情報の一覧要求・問合せ

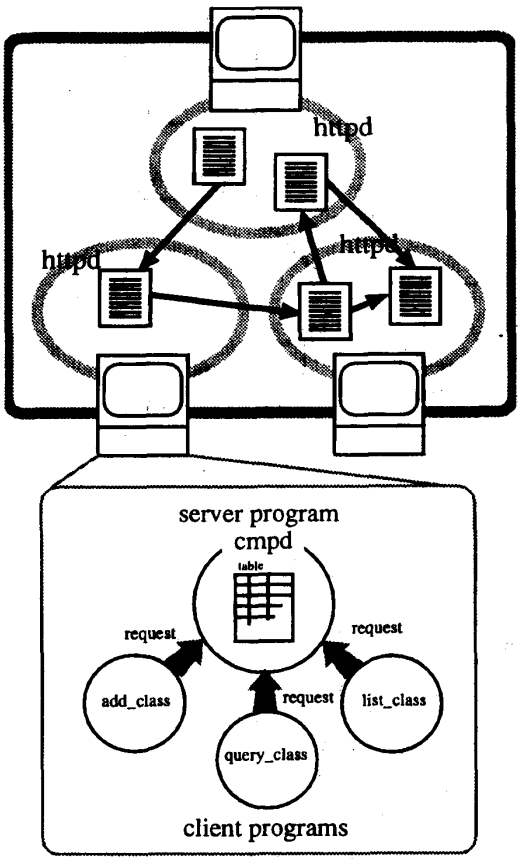

Figure 3. Overview of the system

システムは，サーバ cmpd を C++ で，特に自動入力解析変換サブシステムは YACC と C++ で，またサーバへの問合せ機構は Perl および C++ で実装した．

以下，項目を分けて，システムの詳細について述べる．

3.1 情報抽出とノード生成

情報抽出での中心的処理はノード単位の切り分けとノード間関係の把握である．C++ は純粋なオブジェクト指向言語ではなく，C を基盤としていることから C との互換性を保つためにオブジェクト指向以外の様々な言語要素を内包している．ここではクラスの情報抽出を考え，クラスに属さない大域的な関数やデータは扱わない．

ノード切り分けは，次の 3 種類の情報を抽出することで行う．

・クラス（構造体と共用体を含む）宣言

・その他の型宣言（typedef，enum）

・テンプレートでパラメタ化された宣言

ここで，typedef，enum を扱うのは，C++ ではクラスも型の一種であって，他の型との名前衝突を避けなければならないからである（なお，シグネチャおよび名

前空間の宣言は現段階では扱わない).

これらの情報を解析，抽出するための YACC の C++ 文法記述を，最上位について示すと，次のようになる.

```
program:
 /* empty */
 extdefs
extdefs:
 extdefs extdef
extdef:
 fndef
 datadef
 template_def
 .....
```

すなわち，プログラムは定義の並びからなり，定義は関数定義（fndef），データ定義（datadef），テンプレート定義（template_def）などからなる.

型宣言はデータ定義の一部，型つき宣言子（typed_declspecs）の中にある．そして，クラス宣言は型宣言の内の構造体宣言（structspec）に含まれるので，そこからクラスについて以下の情報を抽出する.

・クラス名
・クラスのインターフェース情報：
　メンバ関数とメンバ変数
　アクセス識別子 (private, protected, public)
・クラス間の依存関係：
　継承関係，全体-部分関係，参照関係

これらの文法記述は次のようになっている.

```
structspec:
 .....
 class_head { opt.component_decl_list }
 .....
class_head:
 .....
 aggr identifier
 aggr identifier : base_class_list
 .....
aggr:
 class
 struct
 union
base_class_list:
 base_class
 base_class_list , base_class
base_class:
 typename
 access_list typename
access_list:
 .....
 visspec another_specs
 .....
opt.component_decl_list:
 /* empty */
 component_decl_list
 opt.component_decl_list visspec :
  component_decl_list
 opt.component_decl_list visspec :
visspec:
 private
 protected
 public
```

これ以上の詳細は割愛する．クラス宣言の冒頭部分（class_head）から，クラス名（identifier）と継承の上位クラス（base_class_list）を抽出する．なお，継承にもアクセス権限の区別があるので，それらも識別するインターフェース情報，すなわち関数と変数の宣言は，component_decl_list から抽出する．関数と変数の宣言には他の型への参照が含まれる．変数は全体-部分関係と参照の両者であり得るが，文法上は識別できないので特に区別しない.

以上の解析結果に基づいて，HTML 形式のファイルを生成する．HTML コード生成は抽出情報をファイルに書き出す時点で行なっている．public な上位クラスとメンバのみを公開インターフェースノードファイルに，その他の protected，private な上位クラスとメンバは実装ノードファイルに格納する．プログラムとそれから生成されるノードの関係を図 4 に示す.

生成される HTML ファイルには，クラスの依存関係に従ったリンクが埋め込まれる．これには HTML のアンカータグ

```
<A HREF="URL of target node">any string</A>
```

を用いる．ここで，リンクは URL をそのまま埋め込むとファイルの移動などに伴う一貫性の維持が難しくなるので，Common Gateway Interface（CGI）を用いて cmpd からクラスの公開インターフェースノードの位置情報を得，HTML の Location ヘッダによってその位置のファイルを渡すことにより，cmpd での位置の一括管理を行う．具体的には，cmpd に問い合わせて Location ヘッダを生成する CGI スクリプトとその引数であるクラス ID を，ファイルの URL が入るべき位置に埋め込む.

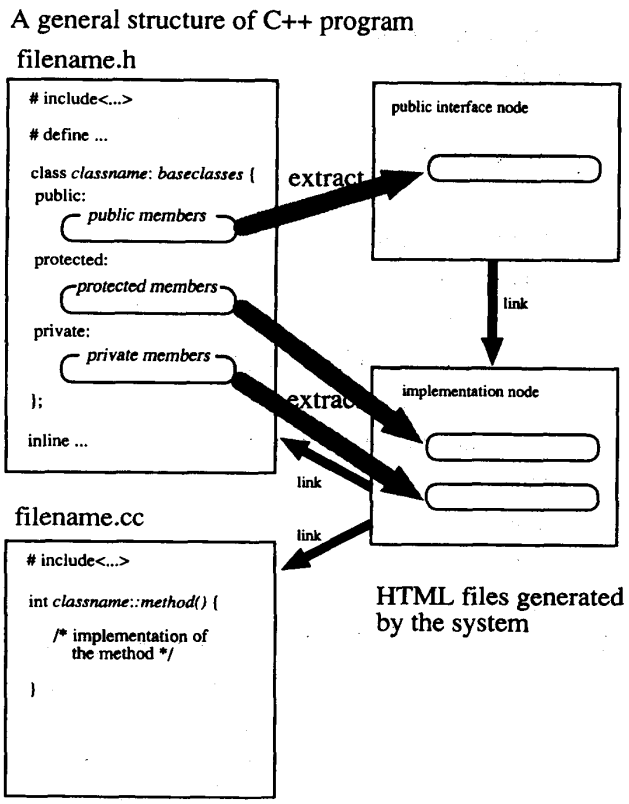

Figure 4. Correspondence between a C++ source code and nodes

3.2 管理テーブルの分散化

先に述べたように，システム全体の中心的サーバは存在せず，サイトごとのサーバの分散的な協調によって全体の管理を行う．広域分散環境では集中管理は得策でないため，これは必然的なアプローチである．この分散協調管理を実現するためには，サイトごとのサーバ，特にClass-URLテーブルの一貫性管理が重要となる．また，リンクの分散透過性はWWWサーバの機能によって達成できるが，この管理テーブルの分散透過性は別に考えなければならない．以下，Class-URLテーブルの分散透過的な一貫性管理について述べる．

一般に，共有資源の分散透過的な一貫性管理を行うためには，次のような幾つかの方式が考えられる．

- あるサイトを集中管理サーバとする（サーバ方式）．
- すべてのサイトに読出し可能なコピーを持たせる（複製方式）．
- 各サイトにキャッシュ的な部分コピーを必要に応じて持たせる．

これらは，その共有資源のアクセス形態，すなわち例えばupdate-manyかwrite-once-read-manyかなどによって使い分けられるべきものである．

ここで，このClass-URLテーブルのアクセス形態は，追加・更新（書込み）が稀で，参照（読出し）が頻繁である．参照は閲覧時だけでなく，クラスの追加時やコンパイル時にも起こる．従って，その分散透過的な一貫性管理には，上の複製方式が適している．頻繁な参照をできるだけ効率化するために，すべてのサイトに管理テーブルの同一コピーを置くものとする．そして，サイト内のサーバ・クライアント間プロトコルに加えて，次のサーバどうしのプロトコルを用意する．

- ADD_FROM_PEER：追加
- UPDATE_FROM_PEER：更新
- DELETE_FROM_PEER：削除

なお，サイト間プロトコルでは通信失敗や要求衝突の可能性があるので，それらに対処しなければならない．要求衝突については，更新と削除については所有者のみが権限を持つので衝突は起こり得ないが，新規登録については特に名称の衝突が起こり得る．その場合には，本システムではとりあえず衝突した要求をいずれも無効化して当該利用者に通知するのみとする．その解決は利用者間の協議調停に委ねるべきであり，それを助けるために例えば簡単な電子会議的なシステムを別に用意するのが有効であろう．

3.3 分散Makefile自動構築

アプリケーション構築の際には，クラスの依存関係情報をコンパイラに知らせてやらなければならない．一般に，アプリケーションのソースコードが複数のファイルに分割されている場合に，それらの間の依存関係情報をコンパイラに知らせるツールとして，make[17]がある．しかし実際には，依存関係情報はプログラムが完全に把握して，Makefileという形で記述してmakeに与えてやらなければならない．

本システムでは，基本的には依存関係情報はリンクで保持されているので，既存ライブラリとの関係など別途必要な情報のみを利用者が補填してやれば，それらの情報に基づいてMakefileを自動的に構築することができる．その際に，1つのアプリケーション内の依存関係情報をすべて一括して1つのMakefileで記述するのでは，部品化と分散管理の妨げとなる[18]．そこで，Makefile自動構築のための補助情報をクラスごとに（専用のノードで）管理しておき，アプリケーション構築時にリンクを順次たどって依存関係情報と補助情報を収集し，Makefileを自動構築する．

この具体的な手順は次のようになる．なお，ここでは分散環境が均質であってサイト間で同じマシンコードを共有できることを前提としている．これは，1つ

のアプリケーションを分散協同開発するのを支援しようとしているのであるから，妥当な条件である．

（1）関連クラスの同定

アプリケーション全体のコンパイルの場合には main ()．クラスの分割コンパイルの場合にはそのクラスを出発点として，リンクをたどってコンパイルに必要な関連クラスを同定する．

（2）ネットワークに分散した該当クラスの収集

関連クラスのマシンコード，ヘッダ，構築補助情報を，それぞれ当該ノードからファイルとして収集する．そのクラスがまだコンパイルされておらずマシンコードを持たない場合には，（所有者にその旨を通知した上で）ソースコードも収集して自サイト上でコンパイルしてマシンコードを得る．

以上の作業を，関連する各サイトで分散的に，必要なクラスがすべて得られるまで再帰的に繰り返す．

例えば，Class A が Class B，Class C に依存して，Class B が Class D に依存しているとすると，宣言部がファイル Class A.h にあり，実装部がファイル Class A.cc と etc. cc に分かれているといった構築補助情報から，以下のような Makefile が自動生成される．

```
HEADERS=ClassA.h ClassB.h ClassD.h ClassC.h
all: ClassA.o etc.o
ClassA.o: ClassA.cc $(HEADERS)
  $ (CXX) -c ClassA.cc
etc.o: etc.cc $ (HEADERS)
  $ (CXX) -c etc.cc
```

4．おわりに

オブジェクト指向ソフトウェア部品の分散管理システムについて，これまで Sun SPARC station の Solaris 上で主に GNU C++ コンパイラと Perl を用いて実装を進め，本論文で述べた機構を一通り完成させている．但し，分散 Makefile 自動構築は，現在は単一サーバ上での動作まで確認している．

しかし，本システムはまだ最初のプロトタイプであって，さらに次のような課題を解決していかなければならない．

- 現在は分散透過なノード閲覧と問合せを支援しているが，ソースやマシンコードなどのノードのサイト間転送も可能にする．
- ノードについてのオーナー管理とアクセス制御，バージョン管理を，これまでのソフトウェア工学の成果も踏まえて確立する．
- デザインパターンやフレームワークなど，さらに高度な部品をも扱えるようにする．
- 実際のソフトウェア開発保守に適用して，実地の検証を行う．

参 考 文 献

1）Rochind, M. F. ; The Source Code Control System, IEEE Trans. Software Eng., SE-1, 4 (1975), 364-370

2）Tichy, W. F. ; RCS—A System for Version Control, Software—Practice and Experience, 15, 7 (1985), 637-654

3）Marca, D. and Bock, G. eds. ; GROUPWARE : Software for Computer-Supported Cooperative Work (1992), IEEE Computer Society Press

4）Vessey, I. and Sravanapudi, A. P. ; CASE Tools as Collaborative Support Technologies, Comm. ACM, 88, 1 (1995), 83-95

5）Kraul, R. E. and Streeler, L. A. ; Coordination in Software Development, Comm. ACM, 88, 1 (1995), 69-81

6）安田 他；C++ プログラム・データベース構築，情報処理学会ソフトウェア工学研究会報告，94-SE-96 (1994)，145-152

7）Coad, P. ; Object-Oriented Pattern, Comm. ACM, 35, 9 (1992), 152-159

8）Gamma, E., Helm, R., Johnson, R. and Vlissides, J. ; Design Patterns (1995), Addison-Wesley

9）Smith, J. B. and Weiss, S. F. ; Hypertext, Comm. ACM, 31, 7 (1988), 816-819

10）Halasz, F. and Schwartz, M. ; The Dexter Hypertext Reference Model, Comm. ACM, 37, 2 (1994), 30-39

11）Bigelow, J. ; Hypertext and CASE, IEEE Software (1988) 23-27

12）Campbell, B. and Goodman, J. M. ; HAM : A General Purpose Hypertext Abstract Machine, Comm. ACM, 31, 7, (1988) 856-861

13）Anderson, K. M., Taylor, R. N. and Whitehead, Jr., E. J. ; Chimera: Hypertext for Heterogeneous Software Environments, Proc. European Conf. on Hypermedia (1994) 94-107

14）Rumbaugh, J. ; Object-Oriented Modeling and Design, Printice Hall (1991)

15）Booch, G. ; Object-Oriented Analysis and Design with Applications, Benjamin/Cummings (1994)

16）Colman, D., et al. ; Object-Oriented Development — the Fusion Method, Prentice Hall (1994)

17）Feldman, S. I. ; MAKE — A Program for Maintaining Conputer Programs, Software — Practice and

Experience, 9 (1979) 255-265
18) 杉山；Object Make によるコンフィギュレーション管理，情報処理学会ソフトウェア工学研究報告 94-SE-97（1994）9-15