修 士 論 文

部品の配置と通信路に着目した
分散システムの形式仕様による文書化

指導教官　落水浩一郎 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

宮田勇人

2001 年 2 月

要 旨

分散システムは、多くのソフトウェア部品で構成され構造が複雑になりやすい。そのため、分散システムの運用 · 管理者には、システムの概要を正確に把握するための文書が必要になる。

本論文では、形式仕様記述言語を用いた分散システムの文書化法を提案する。その方法は、ソフトウェア部品の配置の制約と部品間の通信路に着目して、分散システムの文書化ための定義を用意し、定義を用いた文書の記述指針を示す。また、実際の事例に対して、本論文の手法を適用し曖昧さを持たない正確な文書を作成する。

目 次

図 目 次

第 1 章

はじめに

1.1 背景

近年のネットワークの発達によって、分散システムに必要な技術が整備されてきている。分散システムとは、集中処理を行うシステムと対立する概念で、互いに協調して動作する複数の計算機の間で分散処理を行なうシステムである [1]。情報の分散処理を行うことによって、集中処理を行うシステムに対して次の利点を持つ。まず、負荷の分散による処理能力が拡大すること、複数の計算機が利用可能なことによる信頼性の拡大などがある。また、分散システムを利用するユーザーは、複数の計算機の存在を知ることなくシステムを利用することができる。

現在よく見られる分散システムは、ネットワーク上に分散する多数のソフトウェア部品の組み合わせにより構成されている。ここで言うソフトウェア部品とは、Web サーバや Web ブラウザ、プラグインやデータベースシステムなどを指す。分散システムはネットワーク上に分散して配置される多数のソフトウェア部品を使用し、部品間には多数の関連が存在するため、その構造は複雑になりやすい。そのため、分散システムの運用 · 管理者には、複雑なシステムの構造を把握するための文書が必要になる。

既存の分散システムの文書化の手法としては、自然言語や図のような視覚物や配置記述言語を用いる方法がある。図や自然言語で記述された文書は、読みやすさとアクセシビリティーを重視して作成されている。そのため、複雑なシステムの文書化に既存の方法を適用すると記述が曖昧になり、文書でシステムの詳細を正確に伝えることが難しい [2]。また、XML[3] のアプリケーションである既存の配置記述言語は、ソフトウェアの自動配置を目的として設計されているため、文書としての可読性が低い。また、配置方法が中心の記述体系であるため、その設計理由である部品間の様々な関連の記述が困難である。そこ

で、自然言語のような曖昧さを伴わず、かつ部品間の様々な関連を記述可能な分散システムの記述手法が必要となる。

1.2 目的

本論文では、分散システムの運用・管理者に対してシステムの全体像を正確に把握させるために、形式仕様記述言語である Z 言語 [4] を用いた分散システムの文書化法の確立を目指す。

形式仕様記述言語は、定まった語彙と文法を用いて文書を記述するため、記述は曖昧になりにくく、複雑なシステムに対しても煩雑になることなく記述できる。Z 言語は、初等的な数学の集合論と述語論理の知識を前提とする程度で記述が可能であり可読性に優れている。また、現在 Z 言語をサポートする多くのツールが提供されている。

複雑なシステムの構造を把握するために、文書は分散システムのソフトウェア部品に存在する配置の制約と部品間の通信路に着目して記述する。部品間の関連を部品間の通信路として抽象化し、分散システムの構造をソフトウェア部品の配置と通信路によって表現する。これを正確に記述するために、Z 言語を用いて再利用可能である形式的な定義を用意し、文書の全体構成や部品の配置の制約、部品間の通信路に関する記述指針を定める。部品の配置の制約として、個々の部品の配置と複数の部品の間にある配置関係の記述方法を定める。部品間の通信路に関しては、部品と部品の間のデータの流れを指す。

本論文の手法を用いて記述される文書は、記述指針に従って読むことができ、分散システムの運用・管理者が的確にソフトウェア部品を配置できる文書となることが期待できる。

1.3 本論文の構成

第 2章で、分散システムを文書化するための文書を設計し、文書化を行う際に文書に必要な要素や記述に用いる手法について検討する。

第 3章で、文書の記述者と読者それぞれの視点から文書の全体構造を説明する。その後、文書化のための定義と定義を用いた文書の記述指針について述べる。

第 4章では本論文の定義と記述指針に従い、分散システムの事例を文書化した例を示す。事例として、分散コンポーネント統合環境である VEDICI[5][6] を用いる。

第 5章で本論文の手法、事例の文書化について考察し、最後に第 6章でまとめと今後の課題を述べる。

第 2 章

文書の設計

本章では、分散システムの全体像を記述する上で必要なコンセプトや文書の記述方法を検討する。

2.1 分散システムの文書化

本節では、本論文で取り扱う分散システムについて述べたあとで、文書の目的と文書の対象読者、記述者、文書に必要な詳細度を説明する。

2.1.1 分散システムについて

本論文における記述対象である分散システムは、ネットワーク上に分散した様々なソフトウェア部品の組合わせにより構成されている (図 2.1)。ここで言うソフトウェア部品とは、Web サーバや Web ブラウザ、プラグインやデータベースシステムなどを指す。分散システムでは多数のソフトウェア部品を使用するため、ネットワーク上に分散するソフトウェア部品の配置を分散システムの運用・管理者が把握しやすいようにする必要がある。また、部品の配置を決定するときには、部品間の通信路に存在する関連が影響していることがある。例えばアプレットはそのダウンロード元となるホストに存在する部品としか通信できない。アプレットと通信する部品にはダウンロード元のホスト内に配置されなければならない制約がダウンロードという関連によって生じている。そのため、部品の配置と部品間の通信路に着目して分散システムを文書化する必要性がある。

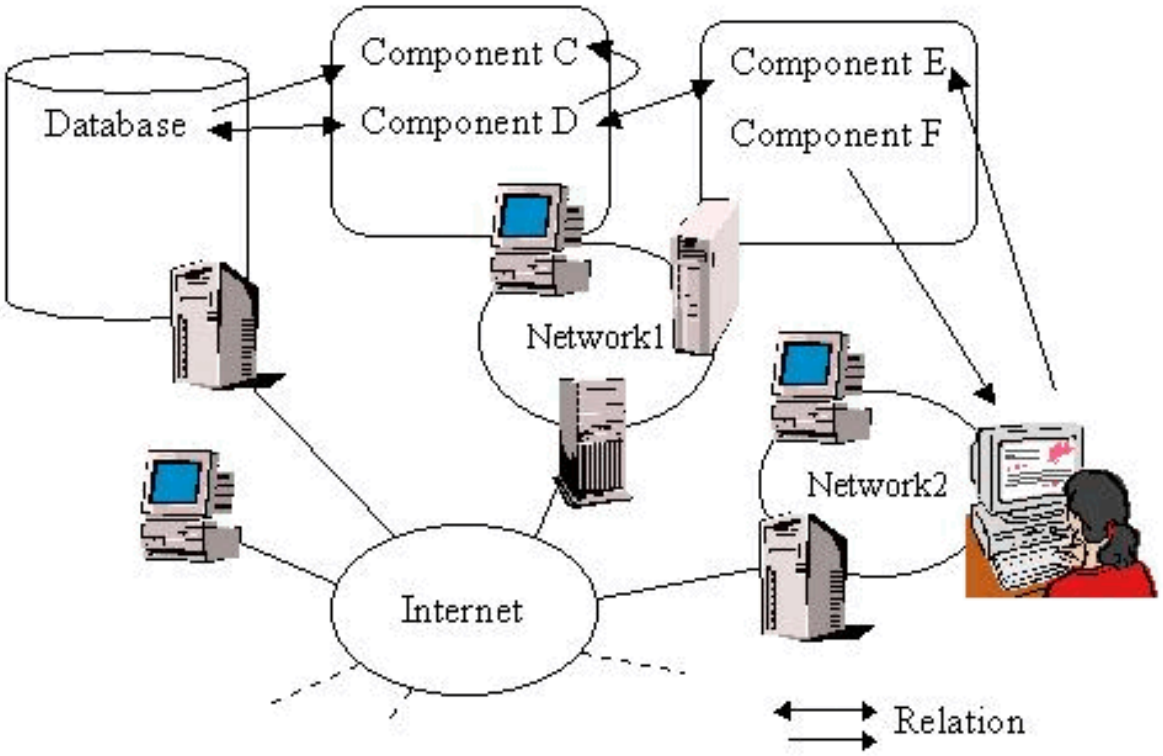

図 2.1: 分散システム

2.1.2 文書の目的

文書は、構造が複雑になりやすい分散システムの部品の配置と部品間の通信路を明らかにするものである。またこの文書によって、なぜ部品がそのように配置されるのかを理解できるようにする。この方針に従い記述した文書によって、分散システムの運用 · 管理者は、誤りなく部品の配置を行うことができる。文書はシステムのセットアップやトラブル時に利用され、ソフトウェア部品の配置の確認のため使われることを目的とする。

2.1.3 文書の対象読者

対象とする読者は、分散システムの運用 · 管理者とする。

2.1.4　文書の記述者

文書の作成は、システムの内部に詳しい開発者が行うものとする。文書を作成するために参考にする情報源としては、仕様書など開発者が残したシステムに関する文書や開発者から直接得られる情報を考える。

2.1.5　文書に必要な詳細度

文書は分散システムを機能ごとに分けて記述する。その機能ごとに記載する内容は、配置の制約と部品間の通信路、その通信路に存在する関連である。関連は、部品の配置に影響するダウンロードのみを記述する。

記述対象である配置の制約は以下の 2 種類ある。

- 個々の部品の配置
- 他の部品との配置関係

個々の部品の配置とは、例えばソフトウェア部品がどのホストに配置されるという制約である。他の部品との配置関係とは、例えば同一ホストに 2 つの部品が配置されることや、異なるホストに別々に部品を配置するという制約である。これらの配置の制約では以下の情報が必要になる。

- 配置の制約を受ける部品
- 部品が配置されるディレクトリ
- 部品が配置されるホスト
- 部品が配置されるネットワーク

部品間の通信路の制約では、部品間で通信する時、最初に接続の要求を出す部品と要求を受けとる部品を記述する。ここでいう通信とは、部品間のデータのやり取りを示す。具体的な通信路の制約は、どの部品が通信に必要で、どのプロトコルを使用するのかいうこと、通信の待ち受け側のポート番号である。これらの通信路の制約では以下の情報が必要になる。

- その通信に関わる部品

- その通信に関する配置の条件
  部品が同一のネットワーク内に存在しない場合
- 通信の関係 (通信の流れ、ブロードキャスト)
- プロトコル
- ポート番号

部品間の関連には、本論文ではダウンロードを記述する。部品をダウンロードすることで、ホスト内に新しい部品の配置あるいは部品間に新しい通信路が生じる。ダウンロードは、部品の配置に影響するため本論文では記述対象とする。ダウンロードでは以下の情報が必要になる。

- ダウンロードのきっかけとなる通信路
- ダウンロードされるソフトウェア部品
- ダウンロード元のホスト
- ダウンロード先のホスト

2.2 文書化の手段

本節では、文書化の手段を検討する。既存の手法とその問題点を述べた後で、本論文で用いる形式仕様記述言語について説明する。

2.2.1 既存の方法

分散システムの文書化を検討するに辺り、既存の文書化法を検討し本論文で用いた形式仕様記述言語について説明する。

図、自然言語

現在の文書化の方法として、最もよく用いられているのが図や自然言語を用いた文書化の方法である。一般に文書化では、図と自然言語を併用して作成される。文章表現に気を配り、例題を文書に盛り込んだりして、システムの内容を文書の読者に伝えている。

配置を記述するための図の例として Unified Modeling Language (UML) [7] の配置図がある。UML は、ソフトウェア主体のシステムを視覚化、文書化するためのモデリング言語である。UML では、システムの物理的な配置を記述するために配置図を用意している。UML の配置図は、ノードをテキスト文字列である「名前」により区別し、ノード内に含むパッケージを記述する。また、ノード間に存在する関連を記述するためには、ノード間を線で結ぶことで示す。図 2.2は、クライアントがサーバ側にあるデータベースを参照する例が記述してある。

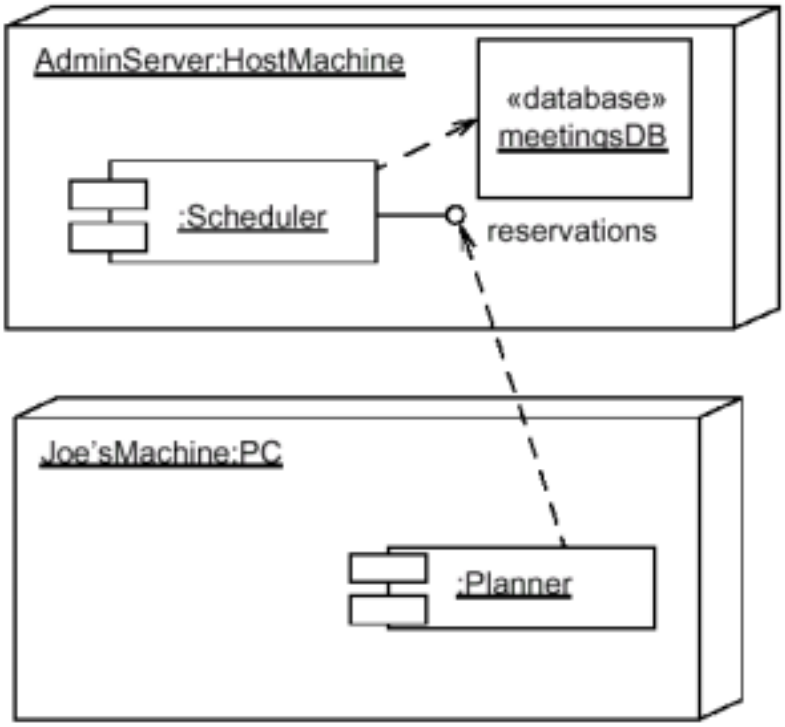

図 2.2: UML の配置図の例

UML などの図や自然言語は、文書を直感的に理解しやすく、さらに可読性に優れていると言える。その反面、文書には曖昧さを伴いやすく、正確に分散システムの文書を作成することが困難である。そのため図や自然言語のみで分散システムの文書化を行うことは難しい。

配置記述言語

配置を記述する言語として XML のアプリケーションである OSD[8] と DSD[9] が存在する。

- OSD

  Open Software Description (OSD) は、Microsoft 社と Marimba 社により共同で開発され、1997 年に W3C で制定された XML のアプリケーションである。OSD は、異機種間クライアントを扱うサーバから配布するソフトウェアパッケージを記述する。

  OSD フォーマットは、ソフトウェアパッケージとそれらの相互依存関係を記述する XML ベースの言語を提供することを目的としている。その Document Type Definition (DTD) のボキャブラリには、大きくソフトウェアパッケージについて記述する要素と、パッケージの実装を記述する要素、およびコンポーネント間の依存関係を記述する要素により構成されている。「プッシュ型」アプリケーションは OSD を利用することで、ソフトウェアのダウンロードを自動的に行うことができる。図 2.3は、OSD を用いて依存関係を表現した例である。

```
<SOFTPKG NAME="com.foobar.www.Solitaire"
   VERSION="1,0,0,0">
   <TITLE>Solitaire</TITLE>
   <ABSTRACT>Solitaire by FooBar Corporation</ABSTRACT>
   <LICENSE HREF=
      "http://www.foobar.com/solitaire/license.html"/>
   <!--FooBar Solitaire is implemented in native code
       for Win32, Java code for other platforms -->

   <IMPLEMENTATION>
      <OS VALUE="WinNT">
         <OSVERSION VALUE="4,0,0,0"/></OS>
      <OS VALUE="Win95"/>
      <PROCESSOR VALUE="x86"/>
      <LANGUAGE VALUE="en"/>
      <CODEBASE HREF=
         "http://www.foobar.org/solitaire.cab"/>
   </IMPLEMENTATION>

   <IMPLEMENTATION>
      <IMPLTYPE VALUE="Java"/>
      <CODEBASE HREF=
         "http://www.foobar.org/solitaire.jar"/>
      <!-- The Java implementation needs the
           DeckOfCards object -->
      <DEPENDENCY>
         <CODEBASE HREF=
            "http://www.foobar.org/cards.osd"/>
      </DEPENDENCY>
   </IMPLEMENTATION>
</SOFTPKG>
```

図 2.3: OSD による記述例

- DSD

  Deloyable Software Description (DSD) は、コロラド大学の Software Dock project において開発された XML のアプリケーションである。DSD は、複雑なシステムの

ソフトウェア配置を自動的に行うことを目的としている。

DSD の DTD は、最も大きい要素である Family がそのソフトウェアに関する情報を DTD の下位要素に持つ。下位要素には、ソフトウェアの名前やライセンス情報などを含む Id や、使用される文字等ソフトウェアの内部を説明する Property、ソフトウェアの配置の概要を示す Artifact などによって構成される。図 2.4は、DSD を用いて依存関係を表現した例である。

```
<Family>
  <Id>
    <Name>Solitaire</Name>
    <Description>The game of Solitaire</Description>
    <Producer>FooBar Corporation</Producer>
    <License>http://www.foobar.com/license.html</License>
    <Logo>http://www.foobar.com/solitairelogo.gif</Logo>
    <Signature>f2a3b8c32091fd46a785c39723e289a9</Signature>
  </Id>
  <ExternalProperties>
    <ExternalProperty>
      <Name>OS</Name>
      <VarType Value="string"></VarType>
      <Description>Operating system</Description>
      <Value></Value> <!-- recorded at deployment -->
    </ExternalProperty>
  </ExternalProperties>
  <Property>
    <Name>Implementation</Name>
    <VarType Value="string"></VarType>
    <Description>
      Selects implementation
    </Description>
    <DefaultValue>Java</DefaultValue>
    <DefaultEnabled>Native</DefaultEnabled>
    <DefaultDisabled>Java</DefaultDisabled>
    <TopLevel Value="true"></TopLevel>
    <Values>
      <Value>Native</Value>
      <Value>Java</Value>
    </Values>
  </Property>
```

図 2.4: DSD による記述例

これらの配置記述言語は、ソフトウェアの自動配置を念頭に作成されており、計算機によって処理される。そのため、これらの配置記述言語を文書として使用とすると可読性に優れているとは言い難い。また、配置方法を中心にタグの定義がなされているため、その設計理由であるソフトウェア部品間に存在する関連の表現方法については、タグの定義が不足している。これらの配置記述言語を、分散システムの文書化のための言語として利用しようとすると、現在のタグの定義ではソフトウェア部品間に存在する多数の関連を表現するためには不十分である。

2.2.2 形式仕様記述言語

文書の記述には、図や自然言語に残る曖昧さを排除し、配置記述言語に比べて関連を的確に記述するために形式仕様記述言語を用いる。

形式仕様記述言語は、形式手法の数学的基盤を与えるものである。形式手法とは、ある手法が健全な数学的基盤を持つことを指す [10]。形式手法では、仕様やプログラムを集合、関数、代数といった数学モデルに基づいて作成する。

形式仕様記述言語は自然言語に数学的基盤による強い制限を加えることで、記述方法や、解釈の指針までを定める。形式仕様記述言語を用いることで、意味が明白になり数学的基盤と照らし合わせれば常に曖昧さを取り除くことができる [11]。また、数学の証明技法の応用が可能であり、記述の厳密な証明ができる。現在、形式手法が最も用いられいるのは要求と仕様であるが、本論文では、曖昧さを持たない正確な分散システムの文書を記述が期待出来ることから、文書化の方法に形式仕様記述言語を選択した。

ただ、形式仕様記述言語の欠点として、現実の非形式的な概念を形式手法を用いて記述する場合、形式手法では表現力が数学的基盤によって制限されるため記述が不完全になりやすい。また、形式仕様記述言語の構文規則、意味モデル、意味規則などの形式体系の学習を読者に要求される。

形式仕様記述言語は、大きく分けてモデル指向と性質指向に分けることができる。

モデル指向の手法では、仕様記述者がシステムの振舞いを数学的構造を使ってシステムのモデルを作成することを直接定義している。数学的構造には、組、関係、関数、集合、列などがある。モデル指向形式仕様記述言語には、Z, VDM[12], B[13] などがある。

性質指向の手法は、仕様記述者は、システムの振舞いを直接定義するのではなく、性質の集合を与えることにより行う。多くの場合、性質の集合は、システムの満たすべき公理の集合として与えられる。性質指向形式仕様記述言語には、OBJ[14], CafeOBJ[15], Larch[16], LOTOS[17] などがある。

2.2.3 Z 言語

本論文では、文書化の手法として形式仕様記述言語である Z 言語を選択した。Z 言語は集合論を基盤とした形式仕様記述言語で、文書内の変数は型に従った集合で記述する。文書内には、システムがどのような時にも満たす状態と、それに作用する操作が記述されたスキーマが存在する。スキーマは、箱のような形をしていて、宣言部と述語部から形成される。

Z 言語は、1970 年台後半から 80 年代にかけて、Oxford 大学の PRG (Programming Research Group) と IBM の共同開発により考えられた。そして、最初のリファレンスマニュアルが 1989 年に出版されている。現在は、Z 言語の標準化が進められている。

本論文の手法に Z 言語を選んだ理由は以下の通りである。まず、Z 言語はモデル指向形式仕様記述言語で、静的な分散システムの配置を記述することに適している。また、Z 言語は、初等的な数学の集合論と述語論理の知識を前提とする程度で記述が可能であり可読性に優れている。集合論を含む形式的記述に自然言語などの説明を自由にい取り入れることを許可している。そのため、形式的手法の中では比較的分かりやすく且つ習得しやすい。さらには、可読性の高い文書を作成できることが期待できる。

現在、Z 言語をサポートする多くのツールが提供されており、本論文では Z 言語の型検査を行うために ZTC[18] を用いている。

第 3 章

形式的文書を記述するための定義と記述指針

本章では、文書を記述するための定義と文書の記述指針を示す。以下、3.1節で、作成する文書の構成を説明し、3.2, 3.3節で、文書を記述するための定義と定義を用いた記述指針を説明する。

3.1 全体構成

文書化のための記述方法について概念図を図 3.1にて示す。

文書化を行うために、文書の記述者には文書に関する形式的な定義と記述指針を提供する。定義部分では、節 2.1.5で検討した内容、例えばファイルやホスト、通信路などに関する定義を提供する。記述部分では、その定義を用いた文書の記述指針を記述の例を交えて説明する。

作成する文書は、分散システムが多数の機能によって構成されているため、分散システムの機能ごとに構成する。その機能ごとの文書は、以下の項目で構成される。

- 文書名
- 部品の列挙
- 部品の配置の記述
- 部品間の関連の記述

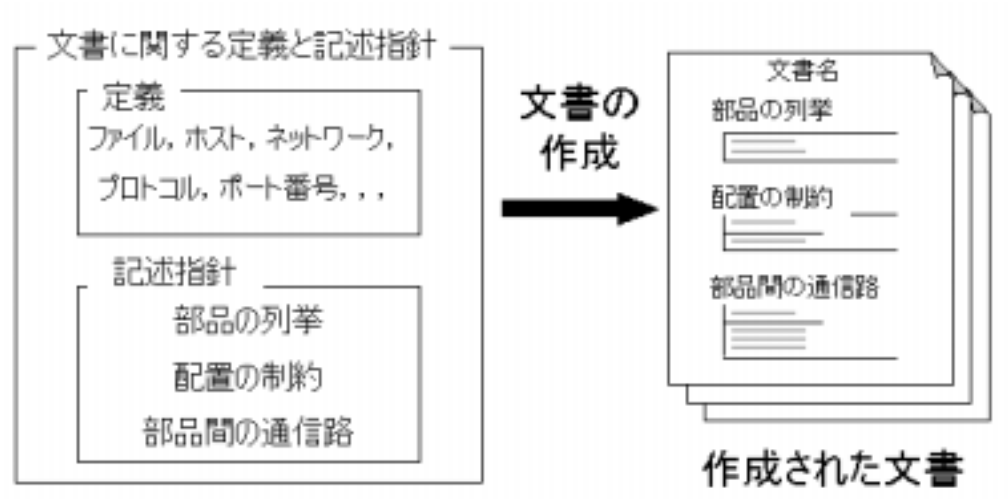

図 3.1: 形式的な文書の記述法

文書名と簡単な文書の説明の後で、文書の導入部として文書名と文書で用いる部品を列挙し、最低限必要な部品の配置の制約を示した後で、部品間に存在する通信路の制約を記述する。記述構成を以下の図 3.1にある作成された文書にて示す。

本論文が提案する手法を用いて記述した文書においては、その機能の名前を文書名とする。読者に文書の内容を感覚的に理解してもらうために、自然言語によって簡単に機能の紹介を文書名の後に記述する。

文書の読者側の指針について

文書化のための定義と記述指針に従い記述した文書の全体構成を、図 3.2にて示す。文書は、文書化の対象となる分散システムに特化しない定義部分と、それぞれの分散システムによって記述される部分から構成される。

まず、定義部分では文書を読むための定義一覧と指針によって構成される。定義一覧では、ファイルやホストそして通信路等の定義を、読者が理解しやすいよう自然言語で説明を加えながら記述する。そして指針の文では、文書がどのように構成されているかなどの文書の構造を説明した後に、部品の配置と部品間の通信路に関する指針を示す。配置や通信路の例を示す時には、例題を交えながら説明し、読者に本手法によって記述された文書の読み方を簡単に学習してもらう。読者側に提供する文書の例を付録 Aに付す。

実際の文書の記述部分では、まず文書化した分散システムの機能一覧を示して、文書化した部分について説明する。さらに、対象システムに特化した定義を示す。システムに特

化した定義とは、例えばそのシステムで使用されるプロトコル、そしてそのプロトコルで使用されるポート番号を指す。文書で記述される機能一覧や、文書に特化した定義を示した後、各機能ごとの文書に移る。読者側の文書は図 3.2の様に構成される。

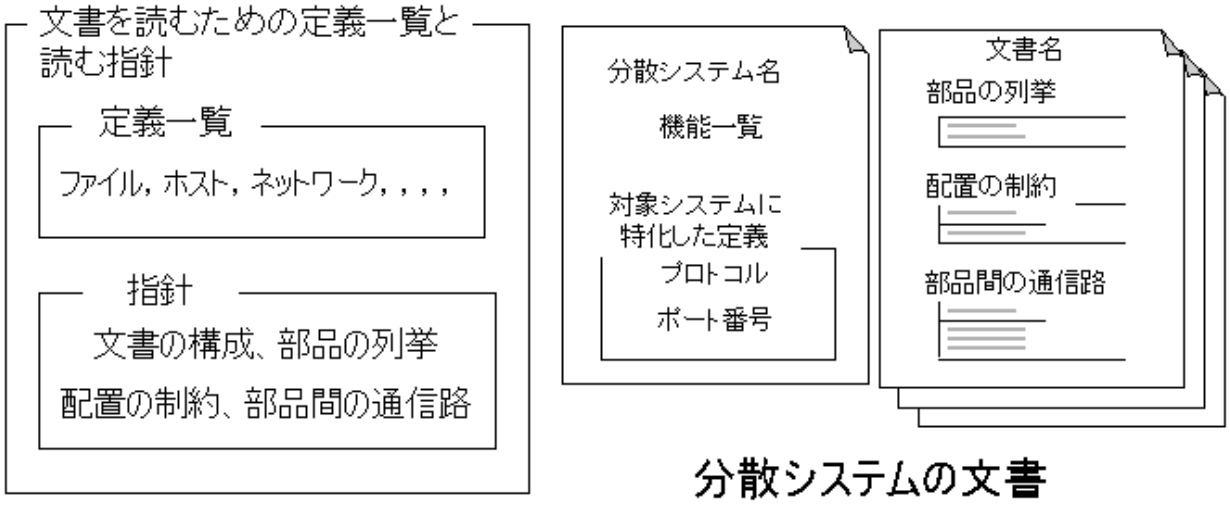

図 3.2: 完成文書の全体構成

3.2 文書化のための定義

3.2.1 基本型

部品やホスト等の名前を表すために、基本型として NAME 型を用意する。

$[NAME]$

3.2.2 ファイル、コンポーネント、ディレクトリ

ファイルやコンポーネントを集合で扱うために、ホスト内のファイルやディレクトリの構造を図 3.5に示したように表現する。ファイルやコンポーネントやディレクトリは、FILE 型で定義する。FILE 型は、その名前を示す NAME 型、ファイルやコンポーネントやディレクトリを区別する TYPE 型、それと FILE 型の巾集合の直積で定義する。ディレクトリには、同一の型を持つファイルやディレクトリが含まれるため、FILE 型を再帰的に定義できる Z 言語の FreeType を用いている。

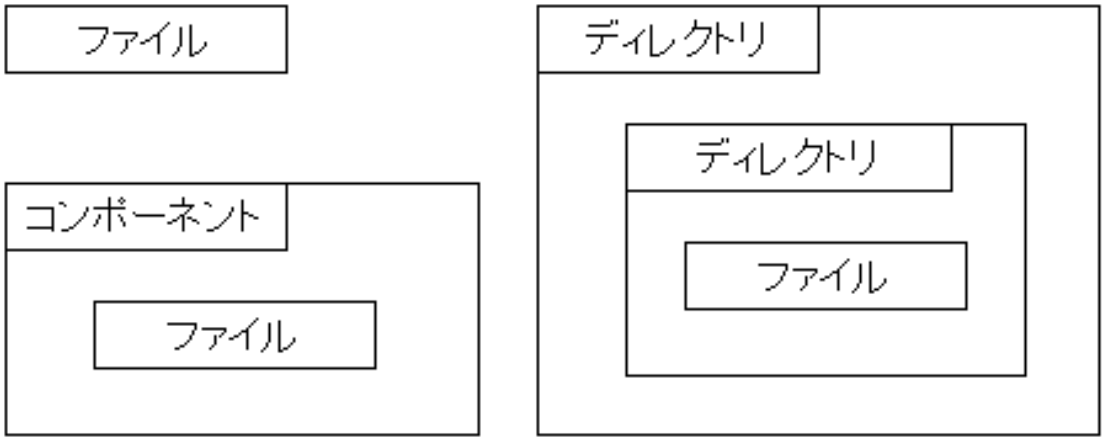

図 3.3: FILE 型に関する概念図

$$TYPE ::= directory \mid file \mid component$$

$$FILE ::= def\langle\langle NAME \times TYPE \times \mathbb{P}\, FILE\rangle\rangle$$

次に、FILE 型の直積の中から一つ一つの要素を取り出すための関数を定義する。関数は FILE 型の名前を示す FILE_name、ファイルやコンポーネントやディレクトリを区別する FILE_type、例えばディレクトリの FILE 型に同一の FILE 型を含む時の FILE 型を示す FILE_file である。今後、直積から一つ一つの要素を取り出す場合

(直積を持つ型名)_(取り出す要素の型名)

で関数名を定義する。

$$\begin{array}{|l}
FILE_name : FILE \pfun NAME \\
FILE_type : FILE \pfun TYPE \\
FILE_file : FILE \pfun \mathbb{P}\, FILE \\
\hline
\forall x : NAME;\ y : TYPE;\ z : \mathbb{P}\, FILE \bullet \\
\qquad FILE_name(def(x, y, z)) = x \\
\qquad \land\ FILE_type(def(x, y, z)) = y \\
\qquad \land\ FILE_file(def(x, y, z)) = z
\end{array}$$

ファイルやコンポーネントやディレクトリの区別が必要な場合、FILE 型を含まない SimpleFile 型と、Component 型、Directory 型を定義する。FILE 型の分類を示した概念図が図 3.4である。

$$SimpleFile == \{f : FILE \mid FILE\_type(f) = file \land FILE\_file(f) = \varnothing\}$$

$$Component == \{f : FILE \mid FILE\_type(f) = component\}$$

$$Directory == \{f : FILE \mid FILE\_type(f) = directory\}$$

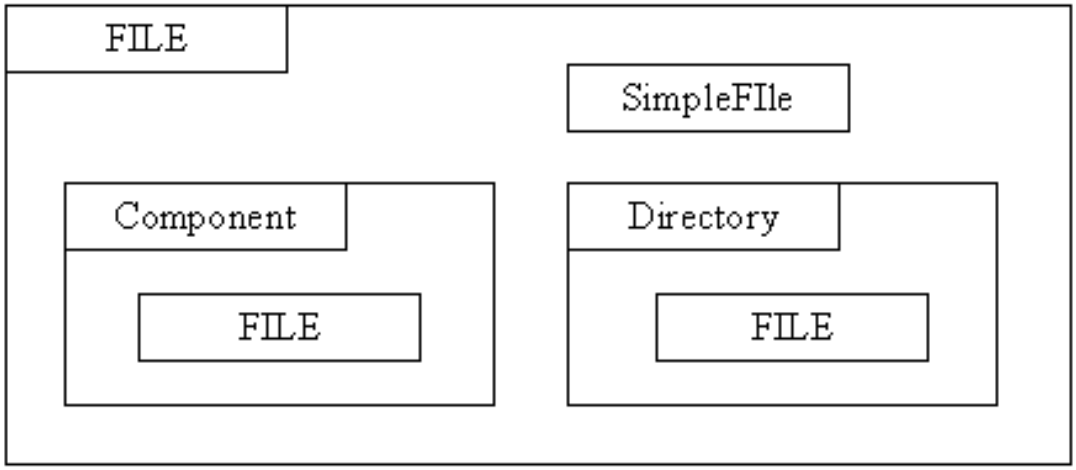

図 3.4: FILE 型の分類

3.2.3 ホスト

ホストは、FILE 型と同様に集合として図 3.5で示したようにホスト内の構造を表現する。ホストは、ホストの名前を表す NAME 型の巾集合とホストに含まれるファイルやディレクトリ、コンポーネントを示す FILE 型の巾集合の直積で定義する。ホストには複数の名前がつけられることがあるため、ホスト名を巾集合で定義する。

$$HOST == \mathbb{P}\, NAME \times \mathbb{P}\, FILE$$

次に、HOST 型から一つ一つの要素を取り出すための関数を定義する。関数はホストの名前を示す HOST_name、ホストの中のファイルを示す HOST_file である。この関数は、ソフトウェア部品の配置に関する制約を記述するために利用する。

$$
\begin{array}{|l}
HOST\_name : HOST \pfun \mathbb{P}\, NAME \\
HOST\_file : HOST \pfun \mathbb{P}\, FILE \\
\hline
\forall\, x : \mathbb{P}\, NAME;\ y : \mathbb{P}\, FILE \bullet \\
\qquad HOST\_name(x, y) = x \\
\qquad \land\ HOST\_file(x, y) = y
\end{array}
$$

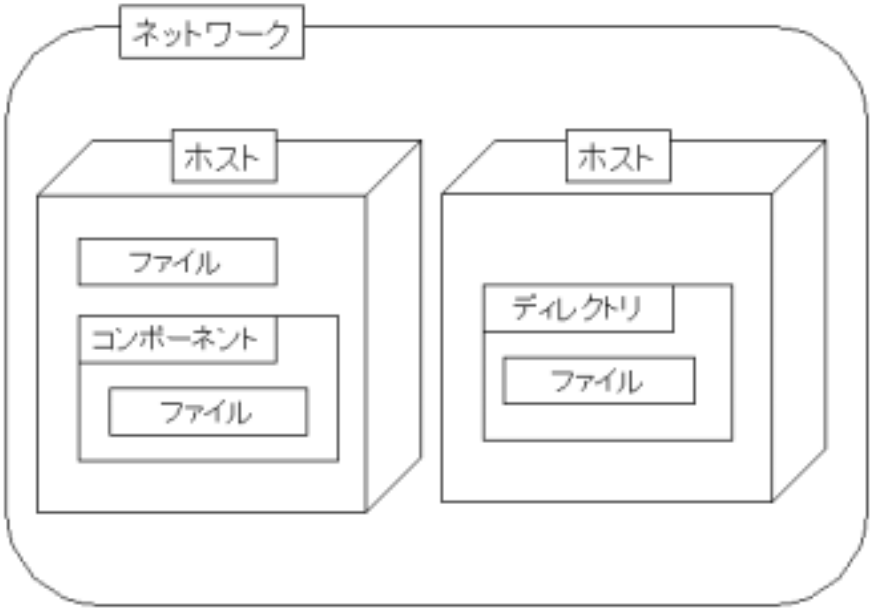

図 3.5: HOST 型、NETWORK 型に関する概念図

3.2.4 ネットワーク

ネットワークは、ネットワークの中に含まれるホストの集合として定義する (図 3.5)。NETWORK 型は、HOST 型の巾集合で定義する。

$$NETWORK == \mathbb{P}\, HOST$$

3.2.5 通信路

通信路を記述する場合、節 2.1.5で述べたように通信に関わる部品とプロトコル、ポート番号を特定する必要がある (図 3.6)。ここでの定義は、通信をする部品の列を示す FILE 型

の列からその通信に利用するプロトコルを示すPROTOCOL型への部分関数である。通信元はFILE型の列の第一番目の要素に記述し、通信先はFILE型の列の最後の要素として記述する。文書内で使用するプロトコルはあらかじめデータ型で定義しておき、comm関数でプロトコルを値として指定する。部品を利用する時のポート番号(PORT)は、port関数を用いて指定しcomm関数と同時に使用する。

ブロードキャストに関しては、部品からネットワークに対する部分関数として定義する。部品間の通信路を記述するための定義と関数を以下に示す。例として、プロトコルにHTTPとIIOPを定義した例を示した。

$$PROTOCOL ::= HTTP \mid IIOP$$

$$\begin{array}{l} comm : \mathrm{seq}\, FILE \pfun PROTOCOL \\ port : FILE \pfun PORT \\ broadcast : FILE \pfun NETWORK \end{array}$$

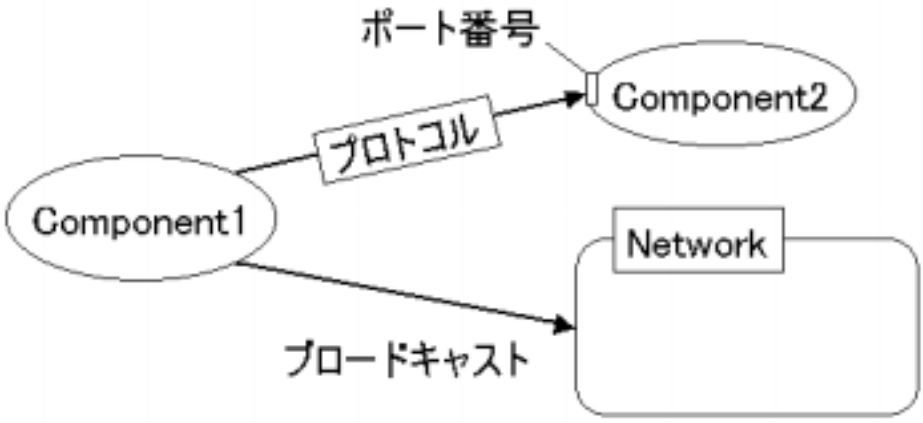

図 3.6: 通信路に関する概念図

3.3 文書の記述指針

3.3.1 部品の列挙

文書は、文書名と文書の簡単な説明の後で、文書に使用するコンポーネントやファイルなどの部品を列挙する。

部品の列挙は、Z 言語のスキーマを用いて表す。ここで列挙する部品は、あとに続く配置の制約や、部品間の通信路の制約で使用する部品を列挙する。スキーマ名の付け方は「文書名 buhin」とする。例えば、document という文書名の文書でコンポーネントの component1, component2, component3 を列挙する時は以下のように記述する。ただし、ここで列挙する部品が同一の場合は明記し、異なる部品である場合は省略可能とする。

$$documentbuhin$$
$$component1, component2, component3 : Component$$

3.3.2 配置の制約

複数の部品の配置関係を記述する指針を示す。配置に関する制約は、節 2.1.5で検討したように、個々の部品の配置と他の部品との配置関係である。以下、個々の部品の配置と他の部品との配置関係に分けてソフトウェア部品の配置に関する定義の記述指針を示す。

個々の部品の配置

部品の配置は、節 3.2.2、節 3.2.3で定義した FILE_file 関数と HOST_file 関数を用いて記述する。個々の部品の配置を記述する時、例えば、ある SimpleFile 型のファイル (f) が、あるホスト (host) 内のディレクトリ (dir) に含まれることを記述するには、以下のように記述する。

$$dir \in HOST\_file(host) \land f \in FILE\_file(dir)$$

他の部品との配置関係

部品間の配置関係を表すためには、ネットワークやホストの定義から集合の要素として記述する。例えば、ある二つの異なる部品 (component1, component2) が同一のホスト (host) に存在することを記述するには以下のように行う。

$$\{component1, component2\} \subseteq HOST\_file(host)$$

また、異なるホストに異なる部品が存在することを示すには、以下のように記述する。

$$host1 \neq host2$$
$$\land\ component1 \in HOST\_file(host1)$$
$$\land\ component2 \in HOST\_file(host2)$$

次に同一のネットワークに二つの異なる部品が存在することを示す。同一のネットワーク (network) 内のホスト (host1, host2) が存在し、ネットワーク内に存在するホストに異なる二つのコンポーネント (component1, component2) が存在するとして記述する。

$$\{host1, host2\} \subseteq network$$
$$\wedge\ component1 \in HOST\_file(host1)$$
$$\wedge\ component2 \in HOST\_file(host2)$$

異なるネットワーク (network1,network2) に、異なる部品が存在することを示すには以下のように記述する。

$$network1 \neq network2 \wedge host1 \neq host2$$
$$\wedge\ host1 \in network1 \wedge host2 \in network2$$
$$\Rightarrow component1 \in HOST\_file(host1)$$
$$\wedge\ component2 \in HOST\_file(host2)$$

部品の配置制約を記述する時には Z 言語のスキーマを用いて表す。スキーマ名は「文書名 Deployment」とする。列挙された「文書名 buhin」をスキーマインクルージョンして、列挙された部品の配置の制約を記述する。スキーマインクルージョンとは、Z 言語のスキーマ計算の一つであり他のスキーマの宣言部と述部を取り込むことである。

3.3.3 部品間の通信路

本節では、部品間の通信路の制約を記述するための記述指針を述べる。

節 3.2.5にある通信路の定義に従って文書を記述する時には、comm 関数を用いて通信に必要なソフトウェア部品を特定し、port 関数を用いて通信の待ち受け側のポート番号を指定する。ただ、ポート番号にはあらかじめウェルノウンポートの集合を定義して利用する。これらのポート番号については、あらかじめ PROTOCOL の型定義で現れたプロトコルに対してスキーマ名「WellKnownPorts」で定義しておく。例えば、

$$PROTOCOL ::= HTTP \mid IIOP$$

の場合には、以下のようにウェルノウンポートを定義しておく。

WellKnownPorts

$$Ports : PROTOCOL \nrightarrow PORT$$

$$Ports = \{(HTTP \mapsto 80),$$
$$(IIOP \mapsto 15000)\}$$

特に、ポート番号を強調したい場合や、ウェルノウンポート以外のポート番号を使用する場合には、port 関数を comm 関数と併用してポート番号を指定する。

次に、部品間の通信路の定義を用いた通信路の記述指針を示す。例えば、図 3.7のような二つの部品 (component1, component2) が通信し、プロトコルに HTTP を用いて、component2 のポート番号が 80 で待ち受ける時には以下のように記述する。この時通信する部品間で利用するプロトコルは必ず記述する。

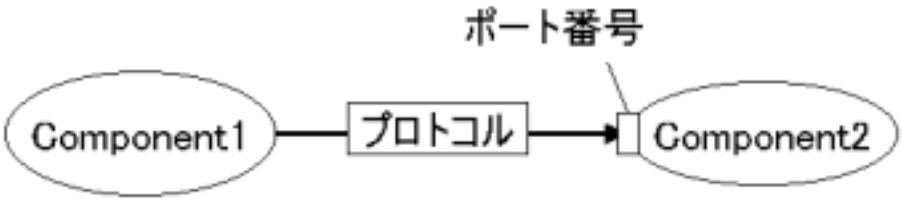

図 3.7: 部品間の通信路

$$comm\langle component1, component2\rangle = HTTP \wedge port(component1) = 80$$

また、図3.8のように三つ以上の部品が通信する時には通信の流れに従って記述する。例えば、異なる二つの部品 (component1, component3) が、別のコンポーネント (component2) を中継して、プロトコル IIOP で通信することを表現するには以下のように記述する。

図 3.8: 通信の中継

$$comm\langle component1, component2, component3\rangle = IIOP$$
$$\wedge\ port(component2) = 15000 \wedge port(component3) = 15000$$

次に、ブロードキャストについては、broadcast 関数を用いる。例えばある部品 (component) が、あるネットワーク (network) に対して、ブロードキャストを行う時は以下のように記述する。

$$broadcast(component) = network$$

文書内で、部品の通信路を記述する時には Z 言語のスキーマを用いて表す。スキーマ名は「文書名 CommunicationPath」とする。通信路を記述するスキーマでは、「文書名 Deployment」をスキーマインクルージョンし、「文書名 buhin」で列挙された部品間の通信路について記述する。記述量が多くなった場合、可読性向上のためスキーマ名を「区別した通信関係名 CommunicationPath」とするスキーマを記述できることとする。ただし、そのスキーマは最終的に「文書名 CommunicationPath」で利用する。

3.3.4 部品間の関連

本節では、部品間の関連を記述するための記述指針を述べる。本論文で取り扱う関連はダウンロードである。

ダウンロード

ダウンロードは、事前 (ダウンロード前) の部品がダウンロード元のホストに存在し、なおかつ事後 (ダウンロード後) の部品がダウンロード先のホストに存在する状態を記述し通信路の制約と合わせて記述する。例えば、図 3.9 のように節 3.3.1の「documentbuhin」で列挙した二つの部品 (component1,component2) がプロトコル HTTP で通信し、それによってある部品 (component3) が component1 側のホストから component2 側のホストへダウンロードされた時には、以下のように記述する。

documentCommunicationPath

$$\Xi documentbuhin$$
$$WellKnownPorts$$

$$comm\langle component1, component2\rangle = HTTP$$
$$\Rightarrow (\exists h1, h2 : HOST \bullet$$
$$\{component1, component3\} \subseteq HOST\_file(h1)$$
$$\wedge \{component2, component3'\} \subseteq HOST\_file(h2))$$

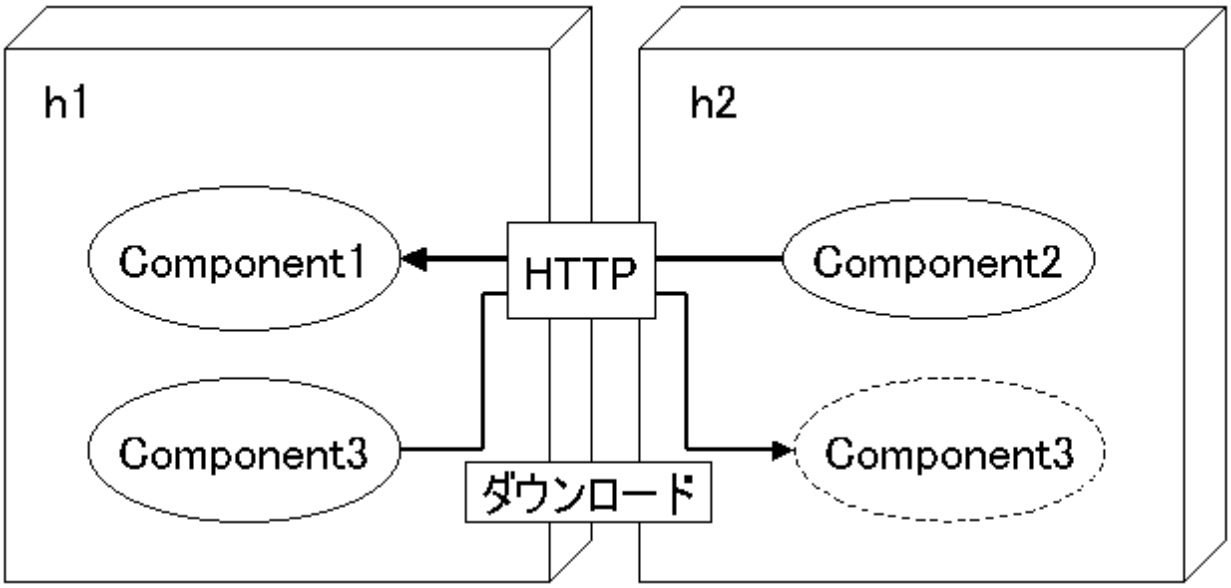
h1
h2
Component1
HTTP
Component2
Component3
Component3
ダウンロード

図 3.9: ダウンロード

第 4 章

事例の文書化

本章では、分散システムの事例として用いた VEDICI の説明と、VEDICI の機能である VediciRepository, OhpDataModelServer に対し本論文で提案する手法を用いて文書化した例について述べる。

4.1 事例 VEDICI

VEDICI は、ネットワーク上に分散するコンポーネントを Web 上で統合する分散コンポーネント統合環境であり、株式会社 PFU と北陸先端科学技術大学院大学との間で共同に開発された分散システムである。VEDICI には、アプリケーションの作成環境とそのアプリケーションの実行環境がある。

アプリケーションの作成環境は、VEDICI の利用者が Web ブラウザから専用のエディタを用いて既存のコンポーネントを関連づけることで、アプリケーションを作成できる。作成されたアプリケーションは VEDICI のサーバ上に保存される。アプリケーションの実行環境では、Web サーバからクライアント側にダウンロードされた VediciPlayer アプレットが、さらに VEDICI サーバの VEDICI アプリケーションをダウンロードし、クライアント側でそれを実行する形態をとる。利用者は VEDICI アプリケーションによって、Web ブラウザからネットワーク上に分散したマルチメディアコンテンツや計算サービスを利用できるようになっている。

VEDICI サーバ側のイントラネットで他のサーバの分散オブジェクトを利用する場合、VEDICI では CORBA の実装の一つである Inprise 社の VisiBroker を利用している。VEDICI アプリケーションは、実行環境でアプレットとして利用されるためダウンロード元となる VEIDCI サーバとしか通信できない制約がある。そこで、CORBA の通信プロトコ

ル (IIOP) の中継を行う GateKeeper を介することで、他のサーバにアクセスすることを可能にしている。

今回対象としたシステムは、VEDICI のフレームワークを用い、遠隔学習用教材として構成されている。教材の利用者は Web ブラウザから、講義ビデオや OHP 等の電子教材を利用し学習できる (図 4.1)。

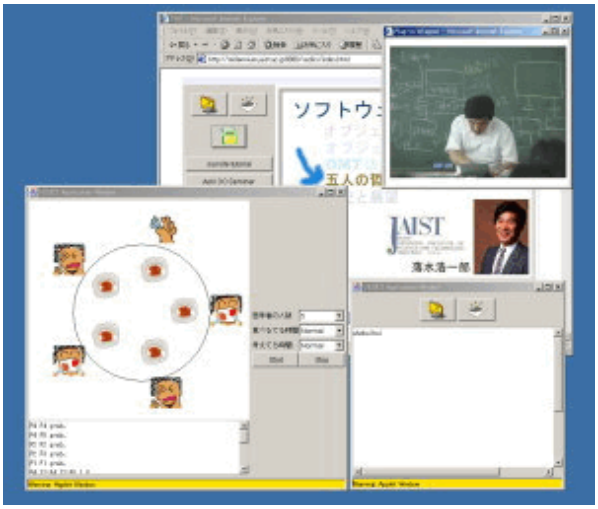

図 4.1: VEDICI を用いた学習アプリケーション

VEDICI では運用 · 管理者に対して、自然言語で記述された文書「システムインストール手引き」と「VEDICI サーバの作り方」を用意している。しかし、二つの文書は曖昧かつ不十分である。VEDICI の文書では、分散システムを構成するソフトウェア部品間の関連が記述されていない。例えば、読者側のネットワーク構成が「システムインストール手引き」が想定するネットワーク構成と異なる場合、システムの構成をどう変えるべきかがわかりにくい曖昧な文書になっており、形式的な文書を用意する必要がある。

以降 VEDICI の機能である VediciRepository、OhpDataModelServer のサーバ間通信、サーバ・クライアント間通信の文書化の例を示す。

4.2 VediciRepository

まず、これらの機能において使用されるプロトコルをあらかじめ定義しておく。なお、VEDICI の運用 · 管理者に対して作成した文書の例は、付録 Bに付す。

$PROTOCOL ::= HTTP \mid IIOP \mid UDP$

上記のプロトコルに存在するウェルノウンポートを以下に示す。

WellKnownPorts

$Ports : PROTOCOL \pfun PORT$

$Ports = \{(HTTP \mapsto 80),$
$(IIOP \mapsto 15000)\}$

以後、4.3でも同じ定義を利用する。

VediciRepository は、Web ブラウザから Web サーバ上におかれたファイルを編集できる機能である。これは Web ブラウザによってダウンロードされたアプレットが、Web サーバ上のスクリプトを実行することにより実現されている (図 4.2)。VediciRepository にある配置の制約と通信路を記述指針に従って記述した文書を付録 B.1に付す。

文書名「VediciRepository」である文書に存在する配置の制約と、部品間の通信路を形式的に記述した例を以下に示す。また、制約を形式的な文書を記述するために用意した自然言語による要約も示す。

まず OhpDataModelServer サーバ・クライアント間通信で使用するコンポーネントは、以下の通りである。

VediciRepositorybuhin

$repository.pl, ActivePerl,$
$HTTP$ サーバ, Web ブラウザ, $Java$ プラグイン : $Component$

配置の制約

VediciRepositoryDeployment

$VediciRepositorybuhin$

$\exists h : HOST \bullet \{Web$ ブラウザ, $Java$ プラグイン $\} \subseteq HOST\_file(h)$

- Web ブラウザと Java プラグインはクライアント側 (同一のホスト) に配置される。

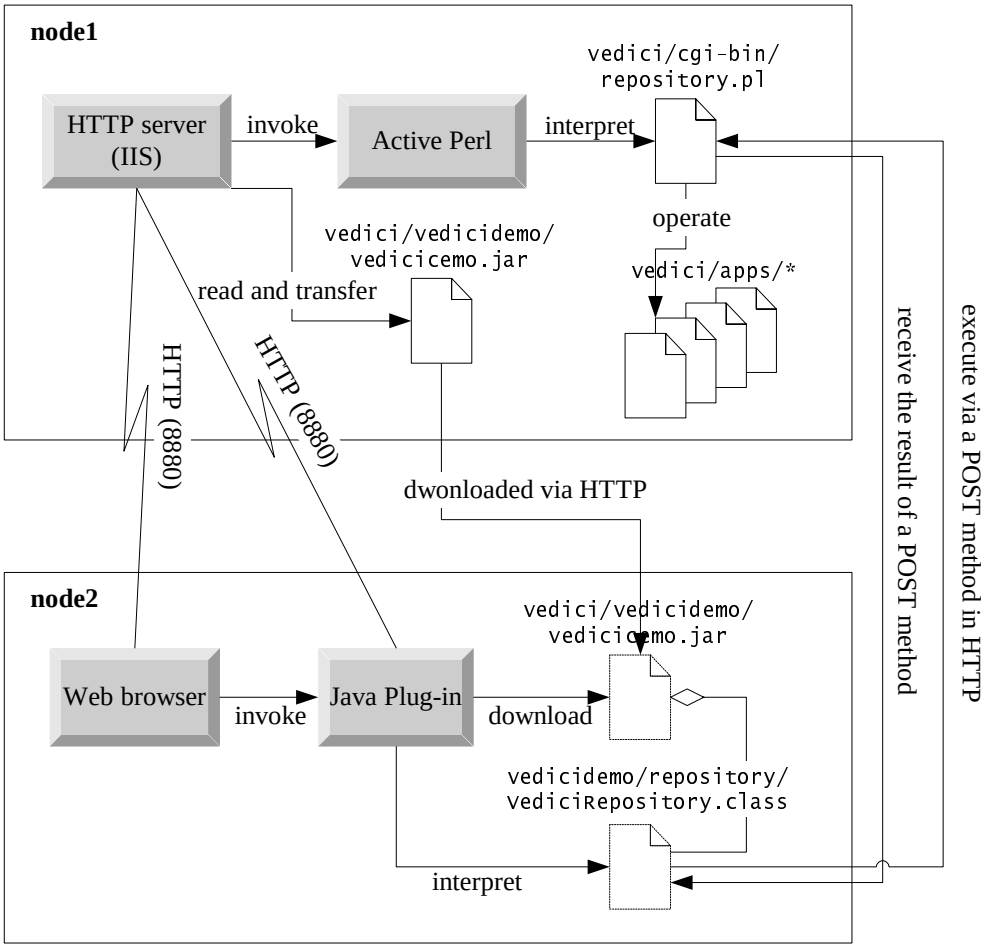

図 4.2: VediciRepository

部品間の通信路

VediciRepositoryCommunicationPath

$$
\begin{aligned}
&\Xi\, VediciRepositorybuhin \\
&VediciRepositoryDeployment \\
&WellKnownPorts \\
\hline
&comm\langle HTTP\ サーバ, ActivePerl\rangle = HTTP \\
&comm\langle ActivePerl, repository.pl\rangle = HTTP \\
&comm\langle Web\ ブラウザ, HTTP\ サーバ\ \rangle = HTTP \wedge \\
&\qquad \wedge\ port(HTTP\ サーバ) = 8880 \\
&comm\langle Java\ プラグイン, HTTP\ サーバ\ \rangle = HTTP \\
&\qquad \wedge\ port(HTTP\ サーバ) = 8880 \\
&\quad \Rightarrow (\exists h1, h2 : HOST \bullet \{HTTP\ サーバ, Vedici.jar\} \subseteq HOST\_file(h1) \\
&\qquad \wedge \{Java\ プラグイン, Vedici.jar'\} \subseteq HOST\_file(h2)) \\
&\exists f : SimpleFile;\ h : HOST \bullet \{HTTP\ サーバ, f\} \subseteq HOST\_file(h) \\
&\quad \wedge\ comm\langle repository.pl, f\rangle = HTTP
\end{aligned}
$$

- HTTP サーバは、ActivePerl と HTTP プロトコルを用いて通信する。
- ActivePerl は、repository.pl を実行する。
- サーバ・クライアント通信は、HTTP サーバと Web ブラウザや Java プラグインの間でのみ行われ、HTTP プロトコルと用いサーバ側のポート番号は 8880 とする。
- vedici.jar は、Java プラグインによりサーバ側からクライアント側にダウンロードされる。
- repository.pl は、vedici.class からの HTTP プロトコルの POST メソッドに基づいて、サーバ上のファイルの処理を行う。

4.3 OhpDataModelServer

VEDICI では、電子教材の OHP を格納するために InterBase というデータベースサーバを用いている。Web ブラウザで OHP の内容を表示する際には、JavaBeans の Bean と

して実装された Lecture1.OhpBean クラスが用いられる。Lecture1.OhpBean クラスは、OhpDataModelServer の持つ CORBA の分散オブジェクトを介して InterBase にアクセスし、OHP の画像データを取得する。続いてその際に関連するコンポーネントの動作について、サーバ間通信とサーバ・クライアント間通信の配置の制約と、部品間の通信路を記述する。

4.3.1 サーバ間通信

OhpDataModelServer が、InterBase のサーバにアクセスする際には、InterBase のための ODBC ドライバと JBuilder に含まれる JDBC-ODBC ブリッジが必要になる。OhpDataModelServer が、InterBase にアクセスする際には、まず JDBC API を用いて JDBC-ODBC ブリッジにアクセスする。JDBC-ODBC ブリッジは、ODBC API を用いて InterBase の ODBC ドライバにアクセスして、最後に ODBC ドライバが InterBase にアクセスする (図 4.3)。OhpDataModelServer のサーバ間通信にある配置の制約と通信路を記述指針に従って記述した文書を付録 B.2.1に付す。

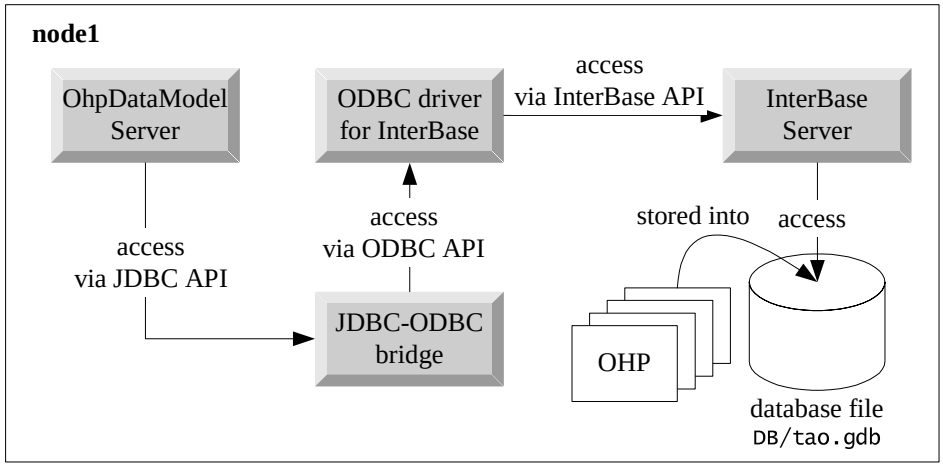

図 4.3: OhpDataModelServer サーバ間通信

文書名「OhpDataModelServer サーバ間通信」である文書に存在する配置の制約を形式的に記述した例を以下に示す。また制約を形式的な文書を記述するために用意した自然言語による要約も示す。

まず、OhpDataModelServer サーバ・クライアント間通信で使用するコンポーネントは、以下の通りである。

OhpDataModelServer サーバ間通信 *buhin*

CORBA サーバ, *JDBC−ODBC* ブリッジ,
InterBase サーバ, *ODBC* ドライバ : *Component*

FILE_name(*CORBA* サーバ) = *OhpDataModelServer*

配置の制約

OhpDataModelServer サーバ間通信 *Deployment*

OhpDataModelServer サーバ間通信 *buhin*

$\exists h : HOST \bullet$
$\{CORBA$ サーバ, $JDBC-ODBC$ ブリッジ, $ODBC$ ドライバ, $InterBase$ サーバ $\}$
$\subseteq HOST\_file(h)$

- OhpDataModelServer,JDBC-ODBC ブリッジ,ODBC ドライバ,InterBase サーバは、全て同一ホストに配置する。

4.3.2 サーバ・クライアント間通信

OhpDataModelServer のサーバ・クライアント間通信では、CORBA のクライアントが、OhpDataModelServer の持つ分散オブジェクトを実行して、その結果 OHP のデータを取得する (図 4.4)。

CORBA のクライアントと分散オブジェクトが通信するためには、クライアントは分散オブジェクトのオブジェクトリファレンスを取得しなければならない。この手続きには GateKeeper と SmartAgent という 2 つのコンポーネントが必要になる。

まず、アプレットとして実装されたクライアントが SmartAgent に対して、オブジェクトリファレンスを問い合わせる。この問い合わせは GateKeeper を介して、GateKeeper と同一のネットワークに存在する SmartAgent に対して行われる。GateKeeper はクライアントと、Web サーバ以外のホストに配置されたサーバとの間で CORBA の通信プロトコル (IIOP) の中継を行う。OhpDataModelServer のサーバクライアント間通信に存在する配置の制約と通信路を記述指針に従って記述した文書を付録 B.2.2に付す。

文書名「OhpDataModelServer サーバ・クライアント間通信」である文書に存在する配置の制約と、部品間の通信路を形式的に記述した例を以下に示す。また、制約を形式的な

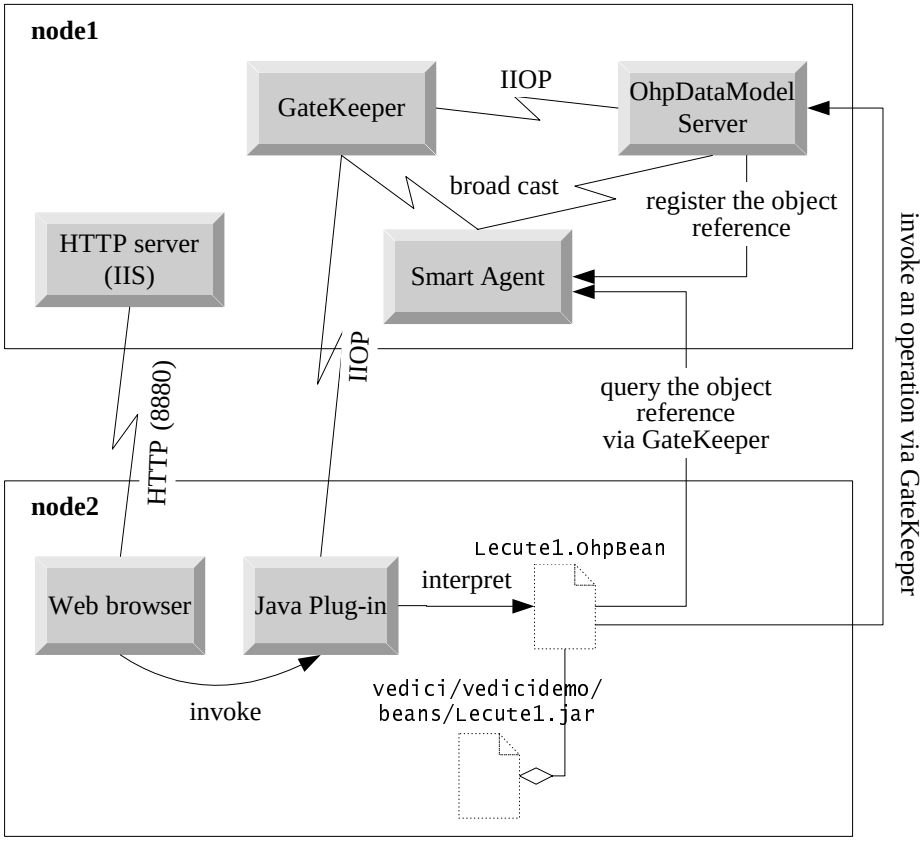

図 4.4: OhpDataModelServer サーバ・クライアント間通信

文書を記述するために用意した自然言語による要約も示す。まず、OhpDataModelServer サーバ・クライアント間通信で使用するコンポーネントは、以下の通りである。

OhpDataModelServer サーバ・クライアント間通信 *buhin*

CORBA サーバ, *CORBA* クライアント,
GateKeeper, *SmartAgent*,
Web サーバ, *Web* ブラウザ, *Java* プラグイン : *Component*

$FILE\_name(CORBA\ サーバ) = OhpDataModelServer$

$FILE\_name(CORBA\ クライアント) = Lecture1.OhpBean$

配置の制約

OhpDataModelServer サーバ・クライアント間通信 *Deployment*

OhpDataModelServer サーバ・クライアント間通信 *buhin*

$$\exists h : HOST \bullet \{GateKeeper, Web\ サーバ\} \subset HOST\_file(h)$$

$$\begin{aligned}
&\exists h1, h2 : HOST;\ n1, n2 : NETWORK \bullet \\
&\quad n1 \neq n2 \land h1 \neq h2 \land h1 \in n1 \land h2 \in n2 \\
&\quad \land\ CORBA\ サーバ \in HOST_file(h1) \\
&\quad \land\ CORBA\ クライアント \in HOST_file(h2) \\
&\qquad \Rightarrow (\exists h3, h4 : HOST \bullet h3 \neq h4 \land h3 \in n1 \land h4 \in n2 \\
&\qquad\quad \land\ SmartAgent \in HOST_file(h3) \\
&\qquad\quad \land\ SmartAgent \in HOST_file(h4))
\end{aligned}$$

- GateKeeper は Web サーバと同一のホストに配置される。
- CORBA サーバとクライアントが存在するネットワークが異なる場合には、各ネットワークに SmartAgent を起動する。

部品間の通信路

GateKeeper の通信路

GateKeeperCommunicationPath

OhpDataModelServer サーバ・クライアント間通信 *Deployment*

$$
\begin{aligned}
&\exists h : HOST \bullet \neg \{ Web\ サーバ, CORBA\ サーバ \} \subseteq HOST\_file(h) \\
&\qquad \Rightarrow comm\langle Java\ プラグイン, GateKeeper, CORBA\ サーバ \rangle = IIOP \\
&\vee \\
&\{ Web\ サーバ, CORBA\ サーバ \} \subseteq HOST\_file(h) \\
&\qquad \Rightarrow comm\langle Java\ プラグイン, CORBA\ サーバ \rangle = IIOP
\end{aligned}
$$

- GateKeeper は CORBA クライアントと、Web サーバ以外のホストに配置されたサーバとの間で IIOP の中継を行う。

SmartAgent の通信路

SmartAgentCommunicationPath

OhpDataModelServer サーバ・クライアント間通信 *Deployment*

$$
\begin{aligned}
&\exists n : NETWORK;\ h1, h2 : HOST \bullet \{h1, h2\} \subseteq n \\
&\qquad \wedge\ CORBAServer \in HOST\_file(h1) \\
&\qquad \wedge\ SmartAgent \in HOST\_file(h2) \\
&\qquad\qquad \Rightarrow broadcast(CORBA\ サーバ) = n \\
&\qquad\qquad\qquad \wedge\ comm\langle CORBA\ サーバ, SmartAgent \rangle = UDP \\
&\qquad\qquad\qquad \wedge\ port(SmartAgent) = 14000 \\
&\exists n : NETWORK \bullet broadcast(CORBA\ クライアント) = n \\
&\qquad \wedge\ comm\langle CORBA\ クライアント, SmartAgent \rangle = UDP \\
&\qquad\qquad\qquad \wedge\ port(SmartAgent) = 14000
\end{aligned}
$$

- CORBA サーバは、同一のネットワーク内に存在する Smart Agent にブロードキャストを用いて通信する。
- CORBA クライアントは、ブロードキャストを用いて Smart Agent に問い合わせて、オブジェクトリファレンスを取得する。

サーバ・クライアント間の通信路

OhpDataModelServer サーバ・クライアント間通信 *CommunicationPath*

GateKeeperCommunicationPath

SmartAgentCommunicationPath

$comm\langle Web\ \text{ブラウザ}, Web\ \text{サーバ}\rangle = HTTP$

$\wedge\ port(Web\ \text{サーバ}) = 8880$

- Web サーバと Web ブラウザは HTTP プロトコルを用いて通信する。その時サーバ側のポート番号は 8880 番とする。

第 5 章

考察

本章では、文書化のための定義と記述指針、そして事例の文書化に対して考察する。

5.1 文書化のための定義と記述指針に関する考察

形式仕様記述言語を用いた定義と記述指針について

本論文では、形式仕様記述言語である Z 言語を用いて文書化のための定義と記述指針を提供した。形式仕様記述言語を用いることで、自然言語では簡潔に記述することが困難な条件分岐や包括関係などを論理記号の “∨” や “∧” などを用いて簡単に記述できる。例えば部品 A,B,C,D のホスト h に対する配置関係に関して、本論文の手法と自然言語による記述例を以下に示す。

$$\{A, B, C\} \subseteq HOST\_file(h) \vee \{A, B, D\} \subseteq HOST\_file(h)$$

- A,B の部品は必ずホスト h に配置され、C,D の部品はどちらか一つがホスト h に配置されていればよい。

上記の例から分かるように、自然言語による文書は部品間の関係が分かりにくく、本論文の手法を用いて記述された文書は自然言語による記述に比べて可読性に優れていると言える。記述された文書についても、“∨” や “∧” などの関係が読み取りやすい文書となった。

また、Z 言語のツールであるスキーマや FreeType 等を用いたことで読みやすい文書になった。なぜなら、スキーマ表記は文書の区切りを判断しやすく、スキーマインクルージョンでどの文書を参照してるかが把握できる。さらに、Z 言語は非形式的な記述を文書に取り入れること許しているため、機能ごとの概要を自然言語の文書の先頭に記述した。

数学的な記号を羅列した文書に比べ、文書の内容を簡単に把握してから文書を読むためはいりやすい文書になったと考えられる。

本論文で用いた形式手法の欠点として、2章の文書化の手段を検討したときに現実の世界を形式的な記述にするのは困難であることを述べた。しかし本論文の手法では、現実の世界から形式的な記述にするための定義を用い、記述指針に従って文書を記述することでこの問題を解決出来る。また文書の読者は、記述指針にしたがって順々に文書を読むことができ、分散システムの部品の配置や通信路を把握できる文書となった。

自然言語で制約を加えた記述指針について

本論文では、分散システムのソフトウェア部品に関する配置の制約と通信路の文書化のための定義を形式仕様記述言語を用いて用意した。しかし本論文で提案した手法では、一部の記述指針を自然言語を用いて与えている。例えば部品間の通信路についての記述指針では、以下のように通信路に必要なソフトウェア部品とプロトコルを併記する指針を、自然言語を用いて与えている。

$$comm\langle component1, component2\rangle = HTTP$$

しかし Z 言語では、以下に示す通信プロトコルに関する制約を含まない記述に関しても記述上誤りではない。

$$\mathrm{dom}\ comm = \{\langle component1, component2\rangle\}$$

この例のように、その指針に従って文書を記述する場合、記述者が必要な情報を欠落する危険があり、その解決策を考える必要がある。

その方法の一つとして、曖昧さの残る記述指針について形式的な記述指針を用意する方法が考えられる。それには記述指針に従って文書が記述されないと、型検査ツールによってエラーとなる定義を改めて用意する必要がある。もう一つの方法としては、計算機による支援で記述の意味検査を行う方法が考えられる。本論文で提案した手法に沿って文書が記述されているかを検査することにより、文書の開発者の記述ミスを防ぐ方法である。本論文で使用している型検査ツールなどの機能を追加することにより検査を行う方法が考えられる。

関連の内容について

本論文では、分散システムの部品間に存在する通信を通信路として抽象化した。関連に関しては、分散システムの部品の配置に特に影響するダウンロードのみ記述指針を与え

た。本論文の手法によって、ソフトウェア部品間の通信路やそれに起因する配置の制約を記述できる。しかし、文書は対象読者から見える全ての詳細部分を記述する必要があり、ダウンロード以外の関連に関しても記述できることが望ましい。

関連を記述する方法を以下に挙げる。

- 記述指針で関連を記述する指針を与える方法
- 関連を記述するための関数を既存の定義を用いて定義する方法
- 関連に関する関数を定義しやすいように型の定義を変更する方法

本論文では、一つ目の方法を用いて部品の配置に影響するダウンロードの記述指針を示した。他の関連について同様に記述指針を定めた場合、現在の定義は配置を記述することに重点をおいて定義されているため、定義の利用の仕方に限界が来るか文書に同じような表現が繰り返され、読み取りにくい文書になることが考えられる。そのため、この方法で関連を記述することは難しい。

二つ目の方法の関連を記述するための新しい関数を定義しようとすると、本論文の定義では不十分な点がある。例えばファイルの読み書きを定義する場合、SimpleFile 型ではファイルの内容に関する定義が存在しないため、既存の定義ではファイルの書き換えに関する関数を定義することが困難である。

三つ目の方法では、関連など文書を詳細に記述する場合のための定義を、現在の定義を変更するか新しく別に定義を用意する必要がある。本論文の定義に必要な要素を追加することで、ダウンロード以外の関連を記述するための関数を定義することが期待できる。

5.2 事例の文書化に関する考察

文書の記述量について

本論文では、提案した手法が実際の事例に対して適用できることを示すために、分散システムの事例 VEDICI の機能である VediciRepository と OhpDataModelServer に対して文書化を行った。これにより、ソフトウェア部品の配置と通信路に関しては適用可能であることを示した。しかし形式仕様記述言語を用いたため、既存の方法に対して文書の記述量が増え、可読性が多少劣ることは否定できない。この問題に対しては、文書の読者は、“あるレベルの詳細さ、たぶん誤解のレベルを最小にするのに十分な低さで、なおかつ役に立たない混乱する大量のデータで苦しめるほどには低くない詳細さを選ぶ” [19] ため、この問題は回避できると考えられる。

スキーマの分割について

節 4.3.2で取り上げた記述例の通信路の制約部分では、可読性向上の試みとして Z 言語のスキーマを三つに分割した。これにより読者は、多数存在する通信路の制約を順次理解することが出来る。文書内に多数存在している制約のスキーマを分割することで、文書の記述量による読者の混乱を避け、文書の可読性は向上すると考えられる。しかし制約が多くなった場合、スキーマの分け方については特に定めておらず、スキーマの分割方法を定める必要がある。

例えば、今回示したように機能ごとに分割してスキーマで記述する方法がある。OhpDataModelServer の通信路に関する記述部分では、GateKeeper や SmartAgent と OhpDataModelServer のサーバ・クライアント間通信の通信路を分けて通信路を記述した。その他、一度記述した文書を再利用したいという場合も考えられる。この場合、スキーマインクルージョンを利用することで文書の再利用を促し、これにより文書はより簡潔になると考えられる。ただし、再利用した文書がどこから参照したのかを自然言語によって明記する必要がある。

取り扱った事例について

本論文の手法では、一般的な分散システムを想定して定義や記述指針を用意し、部品の配置と部品間の通信路に関しては VEDICI の中にある機能の一部を記述することができた。しかし事例として用いたのは VEDICI のみである。そのため本論文の手法が部品の組み合わせ出来ている分散システム全般に使えることを、VEDICI 以外の事例を文書化して確認する必要がある。

第 6 章

おわりに

6.1 まとめ

本論文では、多数のソフトウェア部品で構成され構造が複雑な分散システムを、システムの運用・管理者が把握できるように形式仕様記述言語である Z 言語を用いて部品の配置と通信路に着目した文書化法を開発した。

まず分散システムの文書化のために、文書に必要な情報や文書化のための手法について検討した。部品の配置と通信路を記述する上で、部品の配置に関わる以下の要素が必要となった。

- 個々の部品の配置
- 他の部品との配置関係
- 部品間の通信路
- 部品間の関連 (ダウンロード)

文書化の手段として、文書に曖昧さを伴いにくい形式仕様記述言語の中で比較的可読性に優れている Z 言語を選択した。検討結果に基づいて記述者側と読者側の観点から文書の全体構成を定め、文書に必要な要素を記述するための形式的な定義と定義を用いた文書を記述指針を示した。

次に、分散システムの事例 VEDICI の機能に対して本論文の手法を用いて文書化を行った。その結果、ソフトウェア部品の配置と通信路に関して、本論文の手法が実際の事例に対して適用できることを示した。

しかし、本論文の手法では記述指針に非形式的な記述が用いられており、文書の記述者が誤って文書を記述してしまう懸念がある。形式的な指針を用意するか提案した手法の意味検査ツールを用いて、手法に残る曖昧さを排除する必要性があると考えられる。

今後、より詳細な定義を用意することで、正確に分散システム全体を読者に伝えられることが期待できる。

6.2 今後の課題

以下に挙げた点が今後の課題である。

- より詳細な記述
  本論文では、ダウンロード以外の通信の内容を全て通信路として記述した。今後、文書の読者により多くの情報を伝えるためにどのような通信であるかがわかるようにする必要がある。

- 記述指針に残る曖昧さの排除
  本論文で示した手法では、記述指針を一部自然言語により与えているところがあり、その指針に従って文書を記述する時、記述者が必要な情報を欠落する危険性がある。曖昧さの残る記述指針について、形式的な記述指針を用意するか、計算機による支援で記述の検査を行うことで、記述指針に残る曖昧さを排除する必要があると考えられる。

- VEDICI 以外の分散システム
  本論文で事例として扱った分散システムは、VEDICI のみである。本論文の記述方法が、VEDICI 以外の分散システムでも文書化できることを確認してみる必要がある。

謝辞

本研究を進めるにあたり、落水浩一郎教授には日頃よりご指導、ご助言いただき心より深く感謝致します。

また、藤枝和宏助手には日頃よりご指導をいただき深く感謝致します。

信州大学の海谷治彦助教授には、本研究へのコメントをいただき深く感謝致します。

論文審査にあたり、二木厚吉教授、篠田陽一助教授にはご助言、ご意見をいただき深く感謝致します。

そして、村越広享助手、服部哲助手、猪股敦夫氏、藤田充典氏には多大なご助言をいただき深く感謝致します。最後に、落水研究室、篠田研究室の皆様に深く感謝致します。

参考文献

[1] 長尾真 他. 岩波情報科学辞典. 岩波書店, 1990.

[2] Jonathan Bowen. *Formal Specification and Documentation Using Z: A Case Study Approach*. INTERNATIONAL THOMSON COMPUTER PRESS, 1996.

[3] World Wide Web Consortium. Extensible markup lnaguage (xml), 2000. http://www.w3.org/XML/.

[4] J.M.Spivey. *The Z Notation Second Edition*. Prentice Hall, 1992.

[5] Igor Mejuev, Akira Kumagai, Manuel Chakravarty, and Tetsuo Ida. Vedici: A conceptual framework and implementation strategy. In *Software Symposium '99*, pages 108–118, 1999.

[6] 落水浩一郎 熊谷章 他. ギガネットを利用した知識の創成とオンデマンド学習支援システムの開発. 産学連携支援型研究開発支援制度, 2000.

[7] Grady Booch. *UML* ユーザーガイド. ピアソンエディケーション, 1999.

[8] A.van Hoff, H.Partovi, and T.Thai. *The Open Software Descrition Format(OSD)*. Microsoft Corp. and Marimba, Inc., aug 1997. http://www.w3.org/TR/NOTE-OSD.

[9] R. S. Hall, D. Heimbigner, and A. L. Wolf. Specifying the deployable software description format in xml. Technical Report CU-SERL-207-00, University of Colorado Software Engineering Research Laboratory, 1999.

[10] 片山卓也 土居範久 鳥居宏次. ソフトウェア工学大事典. 朝倉書店, 1998.

[11] B.Potter, J.Sinclair, and D.Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1991. (田中武二 監訳. ソフトウェア仕様記述の先進技法-Z 言語. トッパン, 1993).

[12] Jones and Cliff B. *Systematic Software Development Using VDM.* Prentice-Hall, 1990.

[13] J-R Abrial. *The B-Book.* CAMBRIDGE UNIVERSITY PRESS, 1996.

[14] Goguen J.A., Winkler T., Futatsugi K., and Jouannaud J.P. *Introducing OBJ, Technical Report SRI-CSL-92-03.* SRI International, 1992.

[15] Futatsugi K. and Nakagawa A. *An OverView of CAFE Specification Environment*, chapter an algebraic approach for creating, verifying, and maintaining formal specifications over networks. 1st IEEE ICFEM, 1997.

[16] Guttag, John V., Horning, and James J. *Larch: Languages and Tools for Formal Specification.* Springer-Verlag, 1993.

[17] ISO: Information Processing System. *Open Systems Interconnecton -LOTOS- A Formal Description Technique based on the Temporal Ordering of Observational Behaviour.* IS 8807, 1989.

[18] Xiaoping Jia. *ZTC: A Type Checker for Z Notation User's Guide*, 1998. http://se.cs.depaul.edu/fm/ztc.html.

[19] A.Littleford. Artificial intelligence and hypermedia. In E.Berk and J.Devlin, editors, *Hypertext/Hypermedia Handbook.* McGraw Hill Publishing Company, 1991.

付録 A

読者の指針

A.1 定義一覧

定義

$[NAME]$

ファイル型

$$TYPE ::= directory \mid file \mid component$$

$$FILE ::= def \langle\langle NAME \times TYPE \times \mathbb{P}\, FILE \rangle\rangle$$

ホスト型

$$HOST == \mathbb{P}\, NAME \times \mathbb{P}\, FILE$$

ネットワーク型

$$NETWORK == \mathbb{P}\, HOST$$

型からそれぞれの要素を取り出す

$$\begin{array}{|l} FILE\_name : FILE \pfun NAME \\ FILE\_type : FILE \pfun TYPE \\ FILE\_file : FILE \pfun \mathbb{P}\, FILE \\ \hline \forall\, x : NAME;\ y : TYPE;\ z : \mathbb{P}\, FILE \bullet \\ \qquad FILE\_name(def(x, y, z)) = x \land \\ \qquad FILE\_type(def(x, y, z)) = y \land \\ \qquad FILE\_file(def(x, y, z)) = z \end{array}$$

$$
\begin{array}{|l}
HOST\_name : HOST \pfun \mathbb{P}\, NAME \\
HOST\_file : HOST \pfun \mathbb{P}\, FILE \\
\hline
\forall\, x : \mathbb{P}\, NAME;\ y : \mathbb{P}\, FILE \bullet \\
\qquad HOST\_name(x, y) = x \land \\
\qquad HOST\_file(x, y) = y
\end{array}
$$

ファイル、コンポーネント、ディレクトリの区別

$$SimpleFile == \{f : FILE \mid FILE\_type(f) = file \land FILE\_file(f) = \varnothing\}$$

$$Component == \{f : FILE \mid FILE\_type(f) = component\}$$

$$Directory == \{f : FILE \mid FILE\_type(f) = directory\}$$

通信路に関する定義

ポート型

$$PORT == \mathbb{N}$$

使用するプロトコルは、それぞれの文書でデータ型 ($PROTOCOL$) として定義する。

$$
\begin{array}{|l}
comm : \mathrm{seq}\, FILE \pfun PROTOCOL \\
broadcast : FILE \pfun NETWORK \\
port : FILE \pfun PORT
\end{array}
$$

A.2 読む指針

文書は、まずはじめに分散システム名とそのシステムで使用されるプロトコルを定義する。文書は機能ごとに構成され、機能ごとの文書は以下のように構成される。

- 文書名
- 部品の列挙
- 配置の制約
- 部品間の通信路
- 部品間の関連

文書名は機能名である。そのあとに簡単な機能の紹介がある。

部品の列挙では、その機能の文書内で使用される文書を列挙してある。

配置の制約では、その機能の文書内に存在する配置の制約を記述してある。以下に配置の制約の記述例と読み方を説明する。

- ファイル (f) が、ホスト (host) 内の、ディレクトリ (dir) に含まれる。

$$dir \in HOST\_file(host) \wedge f \in FILE\_file(dir)$$

- ある二つの異なる部品 (component1, component2) が同一のホスト (host) に存在する。

$$\{component1, component2\} \subseteq HOST\_file(host)$$

- 異なるホスト (host1,host2) に異なる部品が存在する。

$$\begin{aligned} & host1 \neq host2 \\ & \quad \wedge component1 \in HOST\_file(host1) \\ & \quad\quad \wedge component2 \in HOST\_file(host2) \end{aligned}$$

- 同一のネットワーク (network) に二つの異なる部品が存在する。

$$\begin{aligned} & \{host1, host2\} \subseteq network \\ & \quad \wedge component1 \in HOST\_file(host1) \\ & \quad\quad \wedge component2 \in HOST\_file(host2) \end{aligned}$$

- 異なるネットワーク (network1,network2) に、異なる部品が存在する。

$$\begin{aligned} & network1 \neq network2 \wedge host1 \neq host2 \\ & \quad \wedge host1 \in network1 \wedge host2 \in network2 \\ & \quad\quad \Rightarrow component1 \in HOST\_file(host1) \\ & \quad\quad\quad \wedge component2 \in HOST\_file(host2) \end{aligned}$$

部品間の通信路では、その部品間に存在する通信路の制約を記述する。以下に部品間の通信路の例とその読み方を示す。

- 二つの部品 (component1, component2) が通信し、プロトコルに HTTP を用いて、component2 のポート番号が 80 で待ち受ける

$$comm\langle component1, component2\rangle = HTTP \land port(component1) = 80$$

- 異なる二つの部品 (component1, component3) が、別のコンポーネント (component2) を中継して、プロトコル IIOP で通信する。

$$comm\langle component1, component2, component3\rangle = IIOP$$
$$\land\ port(component2) = 15000 \land port(component3) = 15000$$

- 部品 (component) が、あるネットワーク (network) に対して、ブロードキャストを行う。

$$broadcast(component) = network$$

部品間の関連では、その部品間に存在する通信路によっておこるダウンロードについて記述する。

ダウンロードに関しては、以下のように記述する。

- component3 は component1 と同じホストにあって、component2 の存在するホストにダウンロードされる。

documentbuhin

$$component1, component2, component3 : Component$$

documentCommunicationPath

$$\Xi documentbuhin$$
$$WellKnownPorts$$

$$comm\langle component1, component2\rangle = HTTP$$
$$\Rightarrow (\exists h1, h2 : HOST \bullet$$
$$\{component1, component3\} \subseteq HOST\_file(h1)$$
$$\land \{component2, component3'\} \subseteq HOST\_file(h2))$$

付録 B

文書化の例

分散システム VEDICI

ここで紹介する機能は以下の通り。

- VediciRepository
- OhpDataModelServer サーバ間通信
- OhpDataModelServer サーバ・クライアント間通信

本文書で使用されるプロコルは以下の通り。

$$PROTOCOL ::= HTTP \mid IIOP \mid UDP$$

本文書で使用されるポート番号

$$\begin{array}{|l}
\hline
WellKnownPorts \\
\hline
Ports : PROTOCOL \pfun PORT \\
\hline
Ports = \{(HTTP \mapsto 80), \\
\qquad\qquad (IIOP \mapsto 15000)\} \\
\hline
\end{array}$$

B.1 VediciRepository

VediciRepository

機能；Web ブラウザから、Web サーバ上におかれたファイルを編集できる機能

必要となるコンポーネント

VediciRepositorybuhin

repository.pl, *ActivePerl*, *Vedici.jar*,
HTTP サーバ, *Web* ブラウザ, *Java* プラグイン : *Component*

配置の制約

VediciRepositoryDeployment

VediciRepositorybuhin

$\exists\, h : HOST \bullet \{\, Web\ \text{ブラウザ}, Java\ \text{プラグイン}\, \} \subseteq HOST\_file(h)$

部品間の通信路

VediciRepositoryCommunicationPath

Ξ *VediciRepositorybuhin*
VediciRepositoryDeployment
WellKnownPorts

$comm\langle HTTP\ \text{サーバ}, ActivePerl \rangle = HTTP$

$comm\langle ActivePerl, repository.pl \rangle = HTTP$

$comm\langle Web\ \text{ブラウザ}, HTTP\ \text{サーバ} \rangle = HTTP \land$
$\quad \land\ port(HTTP\ \text{サーバ}) = 8880$

$comm\langle Java\ \text{プラグイン}, HTTP\ \text{サーバ} \rangle = HTTP$
$\quad \land\ port(HTTP\ \text{サーバ}) = 8880$
$\quad \Rightarrow (\exists\, h1, h2 : HOST \bullet \{HTTP\ \text{サーバ}, Vedici.jar\} \subseteq HOST\_file(h1)$
$\quad \land\ \{Java\ \text{プラグイン}, Vedici.jar'\} \subseteq HOST\_file(h2))$

$\exists f : SimpleFile;\ h : HOST \bullet \{HTTP\ \text{サーバ}, f\} \subseteq HOST\_file(h)$
$\quad \land\ comm\langle repository.pl, f \rangle = HTTP$

B.2 OhpDataModelServer

B.2.1 サーバ間通信

OhpDataModelServer サーバ間通信

機能；OhpModelServer が、InterBase にアクセスする。

必要となるコンポーネント

OhpDataModelServer サーバ間通信 *buhin*

CORBA サーバ, *JDBC−ODBC* ブリッジ,
InterBase サーバ, *ODBC* ドライバ : *Component*

FILE_name(*CORBA* サーバ) = *OhpDataModelServer*

配置の制約

OhpDataModelServer サーバ間通信 *Deployment*

OhpDataModelServer サーバ間通信 *buhin*

$\exists h : HOST \bullet$
$\{CORBA\text{ サーバ}, JDBC-ODBC\text{ ブリッジ}, ODBC\text{ ドライバ}, InterBase\text{ サーバ}\}$
$\subseteq HOST\_file(h)$

B.2.2 サーバ・クライアント間通信

OhpDataModelServer サーバ・クライアント間通信

機能；Lecture1.OhpBean が、OhpDataModelServer の持つ分散オブジェクトのオペレーションを実行して、その結果として得られる OHP のデータを表示する。

必要となるソフトウェア部品

OhpDataModelServer サーバ・クライアント間通信 *buhin*

$CORBA$ サーバ, $CORBA$ クライアント,
$GateKeeper$, $SmartAgent$,
Web サーバ, Web ブラウザ, $Java$ プラグイン : $Component$

$FILE\_name(CORBA$ サーバ$) = OhpDataModelServer$

$FILE\_name(CORBA$ クライアント$) = Lecture1.OhpBean$

配置の制約

OhpDataModelServer サーバ・クライアント間通信 *Deployment*

OhpDataModelServer サーバ・クライアント間通信 *buhin*

$$\exists h : HOST \bullet \{GateKeeper, Web \text{ サーバ}\} \subset HOST\_file(h)$$

$$\exists h1, h2 : HOST;\ n1, n2 : NETWORK \bullet$$
$$n1 \neq n2 \land h1 \neq h2 \land h1 \in n1 \land h2 \in n2$$
$$\land\ CORBA \text{ サーバ} \in HOST\_file(h1)$$
$$\land\ CORBA \text{ クライアント} \in HOST\_file(h2)$$
$$\Rightarrow (\exists h3, h4 : HOST \bullet h3 \neq h4 \land h3 \in n1 \land h4 \in n2$$
$$\land\ SmartAgent \in HOST\_file(h3)$$
$$\land\ SmartAgent \in HOST\_file(h4))$$

部品間の通信路

GateKeeper の通信路

GateKeeperCommunicationPath

OhpDataModelServer サーバ・クライアント間通信 *Deployment*

$$
\begin{aligned}
&\exists\, h : HOST \bullet \neg \{ Web\ \text{サーバ}, CORBA\ \text{サーバ} \} \subseteq HOST\_file(h) \\
&\qquad \Rightarrow comm\langle Java\ \text{プラグイン}, GateKeeper, CORBA\ \text{サーバ} \rangle = IIOP \\
&\vee \\
&\{ Web\ \text{サーバ}, CORBA\ \text{サーバ} \} \subseteq HOST\_file(h) \\
&\qquad \Rightarrow comm\langle Java\ \text{プラグイン}, CORBA\ \text{サーバ} \rangle = IIOP
\end{aligned}
$$

SmartAgent の通信路

SmartAgentCommunicationPath

OhpDataModelServer サーバ・クライアント間通信 *Deployment*

$$
\begin{aligned}
&\exists\, n : NETWORK;\ h1, h2 : HOST \bullet \{h1, h2\} \subseteq n \\
&\qquad \wedge\ CORBAServer \in HOST\_file(h1) \\
&\qquad \wedge\ SmartAgent \in HOST\_file(h2) \\
&\qquad\qquad \Rightarrow broadcast(CORBA\ \text{サーバ}) = n \\
&\qquad\qquad\qquad \wedge\ comm\langle CORBA\ \text{サーバ}, SmartAgent \rangle = UDP \\
&\qquad\qquad\qquad \wedge\ port(SmartAgent) = 14000 \\
&\exists\, n : NETWORK \bullet broadcast(CORBA\ \text{クライアント}) = n \\
&\qquad \wedge\ comm\langle CORBA\ \text{クライアント}, SmartAgent \rangle = UDP \\
&\qquad\qquad\qquad \wedge\ port(SmartAgent) = 14000
\end{aligned}
$$

サーバ・クライアント間の通信路

OhpDataModelServer サーバ・クライアント間通信 *CommunicationPath*

GateKeeperCommunicationPath

SmartAgentCommunicationPath

$$
comm\langle Web\ \text{ブラウザ}, Web\ \text{サーバ} \rangle = HTTP \wedge port(Web\ \text{サーバ}) = 8880
$$