LDAPディレクトリのためのモデリング言語とディレクトリ管理プログラム自動生成システム

五月女 健治[†1,†2] 近藤 誠一[†1] 大沼 聡久[†1]
小宮 崇[†1] 酒井 三四郎[†3] 水野 忠則[†3,†4]

LDAPディレクトリサービスが，企業の情報システムを構築するツールの1つとして位置付けられ，利用され始めている．しかし，ディレクトリの設計においては標準的な設計モデルがなく，ディレクトリ構造を例示する程度の不完全な図を示すのが一般的で，ディレクトリ設計のための手法が求められている．本論文では，LDAPディレクトリにおけるディレクトリ情報ツリーの設計を目的とする，UMLをベースに拡張したモデリング言語を提案する．また，本モデリング言語から，ディレクトリを管理するプログラムを自動生成するシステムを試作し評価を実施した．その結果，本システムが十分な適用性と性能を有することが分かった．

Modeling Language for LDAP Directory and Automatic Production System of Directory Management Program

Kenji Saotome,[†1,†2] Seiichi Kondo,[†1] Akihisa Oonuma,[†1]
Takashi Komiya,[†1] Sanshiro Sakai[†3] and Tadanori Mizuno[†3,†4]

LDAP Directory Service begins to be used as a tool for the development of Enterprise Information System. Nevertheless, there are not the standards of the model for the design of the Directory and generally incomplete diagrams of the Directory have been illustrated. The methods to design the Directory are expected. Then, we proposed the modeling language extending UML for the design of Directory Information Tree (DIT). We developed for trial the system that automatically produces the programs that manage the Directory, and estimated it. We found that this system is enouph applicable and efficient.

1. はじめに

近年，ディレクトリサービスの標準化が進み，企業の情報システムのデータを管理することを利用目的としたディレクトリサービス製品がリリースされるようになり，利用され始めている．しかし，システム要件により独自のデータ構造を構築する必要が生じた場合，ディレクトリ固有の標準的な設計モデルがないために，階層構造を例示する程度の不完全な図を示して，それを設計文書とすることが通例となっている．そのため，従来のディレクトリの構築手法では，ディレクトリ設計者とそれを利用するプログラム開発者やディレクトリ情報の利用者間での円滑な意思疎通を実現できず，プログラム開発者や利用者にとって所望のディレクトリを得ることが困難となっている．このため，開発の後工程や稼動間際に問題が発生するプロジェクトも見受けられる．

本論文では，広く使用されているUMLモデルをベースとして，ディレクトリのデータ構造を表現できるディレクトリサービスに特化したディレクトリ・モデリング言語を提案する．また，本モデリング言語から，ディレクトリを管理するプログラムを自動生成するシステムを試作し評価を実施した．その結果，本システムが十分な適用性と性能を有することが分かった．

モデリング言語および自動生成・自動変換に関する研究・提案が報告されているが，ディレクトリ構築を目的とする研究は他に見ない[1)~7)]．本分野において

†1 三菱電機株式会社情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation
†2 法政大学大学院イノベーション・マネージメント研究科
Hosei Business School of Innovation Management
†3 静岡大学情報学部
Faculty of Information, Shizuoka University
†4 静岡大学創造科学技術大学院
Graduate School of Science & Technology, Shizuoka University

は，それぞれの目的の分野ごとに，最適な言語の定義および自動生成・自動変換の出力を検討する必要があり，ディレクトリ構築を目的とするとき，既存の成果により代替することはできない．

本論文で対象とするディレクトリとは，ITU-T（International Telecommunication Union-Telecommunication Standardization Sector）X.500 シリーズ勧告で定義されている構造を持ち，IETF（Internet Engineering Task Force）で定義されている LDAP（Lightweight Directory Access Protocol）のインタフェースを有するものとする[8)~10)]．また，提案するディレクトリ・モデリング言語は，OMG（Object Management Group）で標準化された UML（Unified Modeling Language）をベースとする[11)~13)]．

本論文中において，ディレクトリの object class はオブジェクトクラスと呼び，UML および Java における class は単にクラスと呼び区別する．また，ディレクトリの attribute type は属性型，attribute value は属性値と呼び，UML における attribute は単に属性と呼び区別する．

2. 前提条件と範囲

本論文の前提条件と実現範囲を示す．

(1) UML のクラス図をモデリング言語のベースとする．以下の理由による．
  (a) UML は標準化されており，システム開発のシステム分析フェーズで広く利用されている．
  (b) UML は，プロファイル（ステレオタイプおよびタグ付き値）による拡張が可能である．
  (c) モデルからソースコードを生成するオープンソースのシステムが存在するため，試作する自動生成システムの作成が容易である．

(2) 提案するモデリング言語は，システム開発のシステム分析フェーズの成果を，ディレクトリの設計に活用し実装することを目的とする．ディレクトリが持つすべての機能を表現するためのモデリング言語を目的としていない．

(3) オブジェクトクラスおよび属性型などのスキーマに関連するモデル表現は，本論文では対象外とする．ユーザ定義のオブジェクトクラスおよび属性型は，手動でディレクトリサーバに登録するものとする．

3. モデリング言語の定義仕様

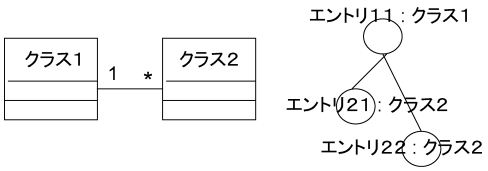
3.1 概　　要

ディレクトリのエントリ（entry）の関係を UML で示すことを考えるとき，**図 1** (a) のような UML における典型的な 2 つのクラス（class）の 1 対多の関連（association）は，図 1 (b) のような DIT（Directory Information Tree）に対応すると考えると，UML によるモデル化が妥当であることが分かる．そこで，**図 2** (a) のようなクラスとその関連に対しては，図 2 (b) のような対応が考えられる．しかし，ディレクトリは階層ツリー構造で，エントリは 2 つの上位エントリを持つことができないため，DIT の上位と下位の対応だけで図 2 (a) のようなクラスとその関連を表現することはできない．図 2 (a) のような関係においても，DN（Distingushed Name）や値など，エントリを関連付ける別の手段を選択できるように，UML をステレオタイプ（stereotype）およびタグ付き値（tagged value）で拡張したディレクトリ・モデリング言語を提案する．

3.2 構 文 仕 様

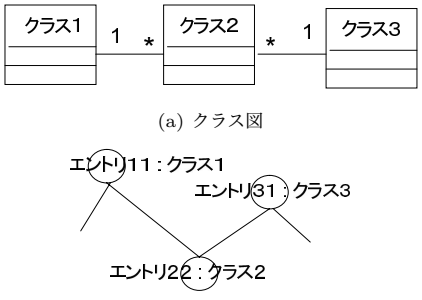
以下に，ディレクトリ・モデリング言語の構文仕様を示す．

(1) ディレクトリ・モデリング言語で記述するモデルは，UML のクラス図によって表現する．



(a) クラス図　　(b) DIT

図 1 1 対多の関連と DIT
Fig. 1 Association and DIT (One to Many).



(a) クラス図

(b) DIT

図 2 1 対多対 1 の関連と DIT
Fig. 2 Association and DIT (One to Many to One).

(2) クラスには，クラス名およびエントリに関係する定義を行うことを示す <<LDAP>> または <<LDAPEntry>> ステレオタイプを指定する．クラス名は，当モデル内で一意でなければならない．

(3) モデルに対して，<<LDAP>> ステレオタイプを指定するクラスを 1 つだけ記述する．このクラスには，1 つの {LDAPRoot} タグ付き値を指定する．{LDAPRoot} タグ付き値には，文字列を指定する．このクラスには，関連を指定してはならない．

(4) 以下の記述は，<<LDAPEntry>> ステレオタイプを指定するクラスに対する構文仕様を示す．

(5) クラスには，1 つ以上の {LDAPObjectClass} タグ付き値を指定できる．これらのタグ付き値には，文字列を指定する．

(6) クラスは，複数の属性（attribute）を持つ．

(7) 属性は，属性名とタイプ（type）からなる．指定する属性名は，当クラス内で一意でなければならない．タイプには，String または Collection を指定する．

(8) 属性には，<<LDAPRDN>> ステレオタイプを指定することができる．クラスの中で，1 つ以上の属性に対して，<<LDAPRDN>> ステレオタイプを指定しなければならない．

(9) クラス間の関連を指定することができ，関連端（association end）に対する多重度（multiplicity），誘導可能性（navigability）および役割名（role name）を指定できる．あるクラスにおける関連端とは，当クラスと他方のクラスとを結ぶ関連における他方のクラス側の端を意味する．あるクラスに関連が複数あるとき，それぞれの関連端の役割名は互いに一意でなければならない．

(10) 関連の多重度は，以下のどれかを指定する．* は，0 以上を意味する．
- 1
- *

(11) 関連の誘導は，双方向または片方向を指定する．

(12) 関連端には，以下の種類のうちの 1 つのステレオタイプを指定しなければならない．
- <<LDAPDIT>>
- <<LDAPDN>>
- <<LDAPAttr>>

(13) 関連端に <<LDAPDIT>> ステレオタイプを指定するとき，他方の関連端に (12) のステレオタイプのどれかを指定するときは <<LDAPDIT>> ステレオタイプを指定しなければならない．また，当関連の多重度は 1 対*でなければならない．

(14) <<LDAPAttr>> ステレオタイプを指定した関連端には，{LDAPKey} タグ付き値を指定しなければならない．{LDAPKey} タグ付き値には，関連端側のクラスの属性名の 1 つを指定する．

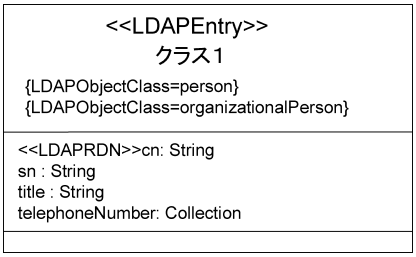


図 3 モデリング言語によるクラス
Fig. 3 Class by modeling language.

3.3 意味仕様

ディレクトリ・モデリング言語の意味仕様を，対応するディレクトリの概念と関係付けて記述する．

3.3.1 クラス

クラスは，エントリを示すために <<LDAPEntry>> ステレオタイプを指定する．

クラスには，そのクラスに属するエントリを構成するオブジェクトクラス（object class）を {LDAPObjectClass} タグ付き値で記述する．{LDAPObjectClass} タグ付き値は，複数指定することができる．ただし，top オブジェクトクラスは，すべてのエントリに存在するため省略する．

クラスに定義される属性の属性名は，{LDAPObjectClass} タグ付き値で示したオブジェクトクラスのどれかに属する属性型（attribute type）を指定する．なお，クラスに定義される属性のタイプは String または Collection を指定する．タイプが単一値を持つとき String，また多値を持つときは Collection を指定する．

<<LDAPRDN>> ステレオタイプは，エントリの RDN（Relative Distingushed Name）となる属性型であることを示す．

図 3 は，ディレクトリ・モデリング言語で記述されたクラスの例である．

<<LDAP>> ステレオタイプを指定したクラスには，モデル全体の情報を記述する．当クラスには，1 つの {LDAPRoot} タグ付き値を指定する．{LDAPRoot} タグ付き値には，当モデルのクラス群に属するすべてのエントリの最上位エントリとなる DN を指定する．

3.3.2 クラスの関連

(1) 概要

関連は，エントリ間の関係を示すために使用する．関連の実現方法を表現するために，関連端に <<LDAPDIT>>，<<LDAPDN>> または <<LDAPAttr>> ステレオタイプのいずれかを指定する．

(2) <<LDAPDIT>> による関連

<<LDAPDIT>> ステレオタイプは，その関連を DIT により実現することを示す．この関連の多重度は，1 対*でなければならない．1 を指定した関連端のクラスに属するエントリが，他方のクラスに属するエントリの直接上位エントリとなるように配置することによって関連を実現する．

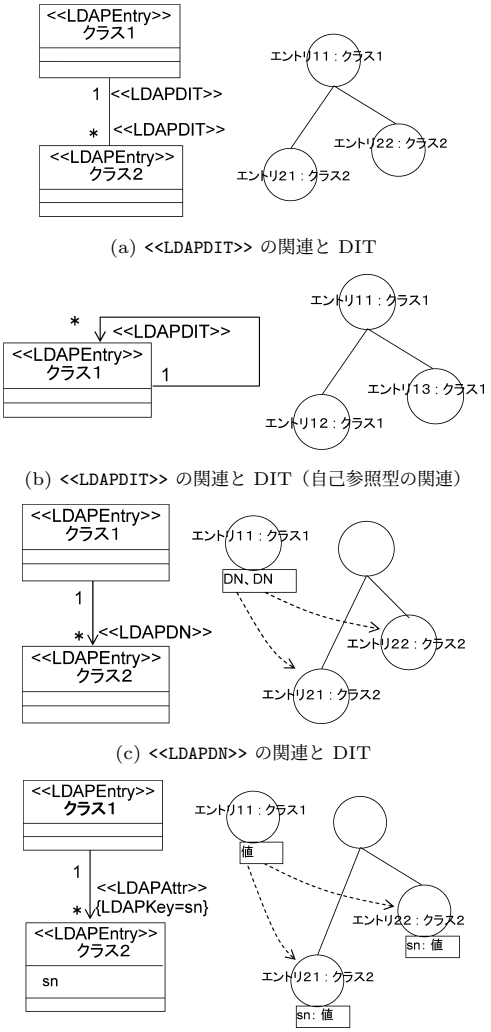
1 つのクラスに複数の <<LDAPDIT>> ステレオタイプによる関連があるとき，その関連の中で，当クラス側の関連端の多重度が*の関連を 2 つ以上持つことはできない．**図 4** (a) は，<<LDAPDIT>> ステレオタイプを使用した UML クラス図と対応する DIT の概略を示す．DIT による関連の実現は，ディレクトリとしては最も自然な構造である．また，企業の組織を表現する場合など，自身のクラスと関連を持つ自己参照型の関連は，ディレクトリの DIT によって表現できる典型である．図 4 (b) は自己参照型の関連を表す UML クラス図と対応する DIT の概略を示す．

(3) <<LDAPDN>> による関連

<<LDAPDN>> ステレオタイプは，その関連を DN により実現することを示す．あるクラスに属するエントリが，その関連端のクラスに属するエントリの DN を持つことによって関連を実現する．この関連の実現方法は，ディレクトリにおける静的グループと呼ばれる，グループとそのメンバの関係を表現するときに使用されている方法と類似する．図 4 (c) は，<<LDAPDN>> ステレオタイプを使用した UML クラス図と対応する DIT の概略を示す．なお，図中の点線は，関連するエントリを示すためのもので，実際の DIT を構成するための表記ではない．

(4) <<LDAPAttr>> による関連

<<LDAPAttr>> ステレオタイプは，その関連を属性型の属性値を利用して実現することを示す．関連端には {LDAPKey} タグ付き値を指定する．{LDAPKey} タグ付き値には，関連端のクラスの属性の中で，関連に使用する属性名を指定する．エントリ内のある属性型に保持された属性値と，関連端のクラスに属する {LDAPKey} タグ付き値で指定した属性型が保持する属性値が一致することによって関連すると見なす．この関連の実現方法は，リレーショナルデータベースにおける主キーと外部キーの関係と類似する．図 4 (d) は <<LDAPAttr>> ステレオタイプを使用した UML クラス図と対応する DIT の例である．



(a) <<LDAPDIT>> の関連と DIT

(b) <<LDAPDIT>> の関連と DIT（自己参照型の関連）

(c) <<LDAPDN>> の関連と DIT

(d) <<LDAPAttr>> の関連と DIT

図 4 モデリング言語による関連と DIT
Fig. 4 Association by modeling language and DIT.

3.4 実装方法

ディレクトリ・モデリング言語によるモデルから DIT を実装する一方法を示す．

3.4.1 エントリ

関連情報を除いて，クラスに属するエントリは，クラスに記述した {LDAPObjectClass} タグ付き値のオブジェクトクラス，属性と同じ名前の属性型および属するクラスを示す情報から構成する．**図 5** は，図 3 で示したクラスのエントリの内容を，LDIF 形式で示したものである．図中，属するクラスを示す情報を示

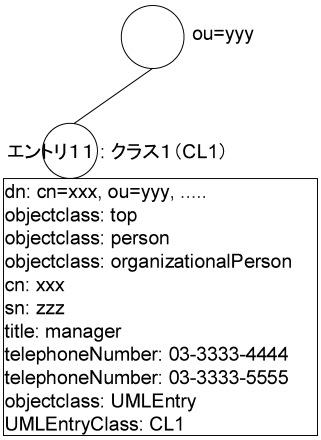
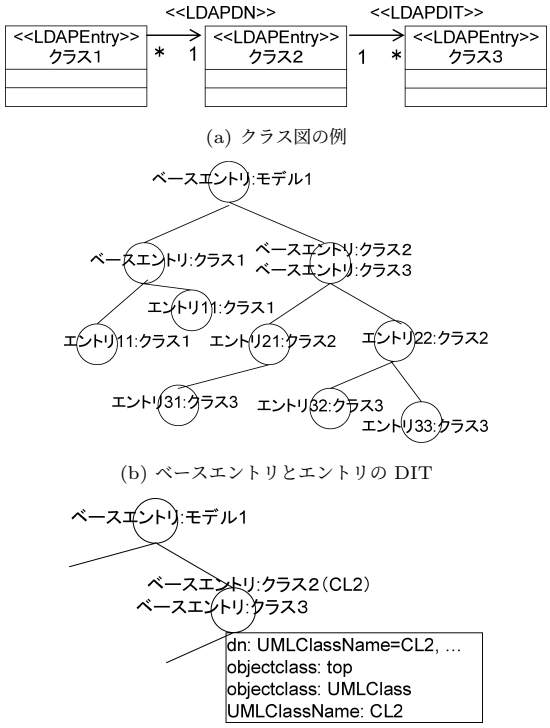


図 5 エントリの実装例
Fig. 5 Implementation of entry.



(a) クラス図の例

(b) ベースエントリとエントリの DIT

(c) ベースエントリの属性型と属性値

図 6 ベースエントリの実装
Fig. 6 Implementation of base-entry.

す属性型を UMLEntryClass，その属性型を持つオブジェクトクラスを UMLEntry としている．

3.4.2 ベースエントリ

当モデルで示すクラス群の最上位エントリとなるモデルベースエントリを配置する．モデルベースエントリは，モデル全体を 1 つの DIT とするためのルートとしての役割を持つ．また，クラスに属するすべてのエントリを下位に持つクラスベースエントリを配置する．モデルベースエントリ，クラスベースエントリおよび各クラスに属するエントリによる DIT は以下のように決める．

(i) あるクラスにおいて，自己参照型の関連を除いて，<<LDAPDIT>> ステレオタイプが指定され，かつ当クラスと関連端のクラスが*対 1 の多重度の関連が含まれている（たかだか 1 組のみ）とき，関連端のクラスベースエントリを，当クラスのクラスベースエントリとする．関連端のクラスベースエントリが決まっていないときは，本節の方法で関連端のクラスベースエントリを決めて，そのクラスベースエントリを当クラスのクラスベースエントリとする．関連端のクラスベースエントリを決める手順を進めた結果，当クラスに到達したときは，このモデルを実装することはできない．当クラスに属するエントリは，関連端に属するエントリの直接下位となるよう配置する．

(ii) (i) 以外のとき，モデルベースエントリの直接下位にエントリを作成し，それを当クラスに対するクラスベースエントリとする．当クラスに属するエントリは，当クラスベースエントリの直接下位となるよう配置する．<<LDAPDIT>> ステレオタイプによる自己参照型の関連の場合，配置済の当クラスに属するエントリの直接下位となるよう配置する．

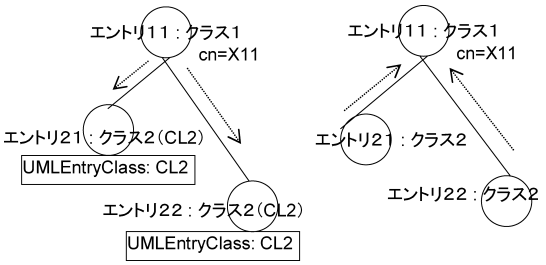
図 6 (a) のクラス図の例を，上述のように構築したときの DIT を図 6 (b) に示す．また，図 6 (c) は，<<LDAPDIT>> ステレオタイプの関連があるときのクラスベースエントリの実装例を示す．クラスベースエントリは，クラス名を保持する UMLClassName 属性型からなる UMLClass オブジェクトクラスを持つ．

3.4.3 <<LDAPDIT>> による関連

<<LDAPDIT>> ステレオタイプの関連は DIT により表現できるため，図 5 のエントリの例と同じ実装方法で実現できる．**図 7** は，図 4 (a) のクラスとその関連を上述のように実装したときの DIT の例である．図 7 (a) は，クラス 1 に属する上位エントリからクラス 2 に属する下位エントリへの誘導の場合の実装例で，この場合次のような検索条件で関連するエントリを検索する．

・検索ベース：cn=X11，...
・検索スコープ：one
・検索フィルタ：UMLEntryClass=CL2

図 7 で，エントリの右下に記述した cn=X11 は当エ



(a) 下位への誘導 (b) 上位への誘導

図 7 <<LDAPDIT>>の関連の実装例
Fig. 7 Implementation of association <<LDAPDIT>>.

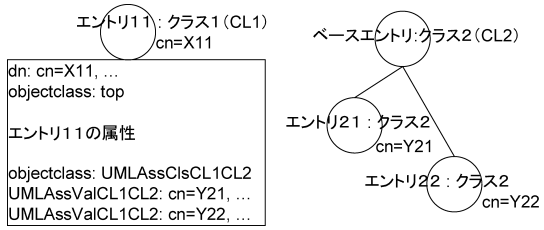


図 8 <<LDAPDN>> の関連の実装例
Fig. 8 Implementation of association <<LDAPDN>>.

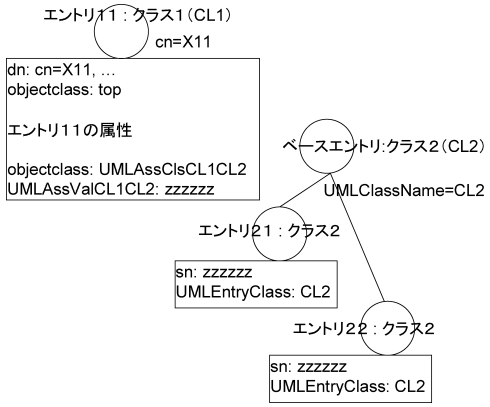


図 9 <<LDAPAttr>> の関連の実装例
Fig. 9 Implementation of association <<LDAPAttr>>.

ントリの RDN を示す.

図 7 (b) は，クラス 2 に属する下位エントリからクラス 1 に属する上位エントリへの誘導の場合の実装例で，この場合次のような検索条件で関連するエントリを検索する.

・検索ベース：cn=X11, ...
・検索スコープ：base
・検索フィルタ：objectclass=*

なお，検索ベースは，下位エントリの DN から RDN を除いた DN を使用する.

3.4.4 <<LDAPDN>> による関連

<<LDAPDN>> ステレオタイプの関連は，以下のように実装する．クラスに属するエントリは，その関連端のクラスに属するエントリの DN を，オブジェクトクラス UMLAssClsCL1CL2 を構成する属性型 UMLAssValCL1CL2 に保管する．ここで，CL1 はクラス名で，CL2 は関連端の役割名とする．個々の関連情報を保管するオブジェクトクラスおよび属性型の名前は，モデル内で一意とする必要があるため，このようにクラス名と役割名から生成する.

図 8 は，図 4 (c) のクラスとその関連を上述のように実装したときの DIT の例である．図中の CL1 はクラス 1 のクラス名，CL2 はクラス 2 の関連端の役割名である．次のような検索条件で関連するエントリを検索する.

・検索ベース：cn=Y21, ...
・検索スコープ：base
・検索フィルタ：objectclass=*

3.4.5 <<LDAPAttr>> による関連

<< LDAPAttr>> ステレオタイプの関連は，以下のように実装する．クラスに属するエントリは，その関連端のクラスに属するエントリを検索するための値を，オブジェクトクラス UMLAssClsCL1CL2 を構成する属性型 UMLAssValCL1CL2 に保管する．ここで，CL1 および CL2 は，<<LDAPDN>> ステレオタイプの場合と同一の命名方法である．**図 9** は，図 4 (d) のクラスとその関連を上述のように実装したときの DIT の例である．次のような検索条件で関連するエントリを検索する.

・検索ベース：UMLClassName=CL2, ...
・検索スコープ：sub
・検索フィルタ：

(&(sn=zzzzzz)(UMLEntryClass=CL2))

4. 自動生成システム

4.1 概　　要

本論文で提案するディレクトリ・モデリング言語がディレクトリを構築するために十分な記述能力を持つこと，および当モデリング言語で記述したモデルからディレクトリを管理するプログラム（以下，ディレクトリ管理プログラム）を自動生成できることを検証することを目的として，ディレクトリ管理プログラムの自動生成システムを試作した.

ディレクトリ管理プログラムは，試作を前提としているため，ディレクトリシステムとして典型的な運用方法を想定した次のような仕様とする.

・データの投入は全データの一括投入方式のみ.
・アプリケーションにより投入データを参照.

実現する DIT は，3.4 節で示した実装方法の構造を採用する．

4.2 実 現 方 式

自動生成システムは，ディレクトリ・モデリング言語で記述したモデルからディレクトリ投入コンパイラとディレクトリ・アクセス API を生成するフェーズ（自動生成フェーズ），およびディレクトリ投入コンパイラとディレクトリ・アクセス API を利用してアプリケーション・システムを開発・運用するフェーズ（アプリケーション・フェーズ）からなる．

自動生成システムは，Poseidon for UML（以下，Poseidon）および AndroMDA を利用し，変換定義およびディレクトリ管理プログラム・生成プログラムを作成することで実現した[14),15)]．

図 10 (a) は，自動生成フェーズの流れを示す．以下に，各コンポーネントの役割を示す．

- Poseidon は UML モデリングツールで，ディレクトリ・モデリング言語で記述したモデルから，XMI 形式の全モデル情報を出力する．
- AndroMDA は MDA（Model Driven Architecture）フレームワークで，変換定義に基づいて，XMI 形式の全モデル情報から必要な情報を抽出し，自動生成システムで定義した独自スクリプトのディレクトリ・モデル情報に変換し出力する[7)]．
- ディレクトリ管理プログラム・生成プログラムは，ディレクトリ・モデル情報を入力し，ディレクトリ投入コンパイラの JavaCC ソースコードとディレクトリ・アクセス API の Java ソースコードを出力する[16),17)]．ディレクトリ・モデル情報の構文仕様・意味仕様のチェックは，当プログラムで実施する．当プログラムは，構文仕様・意味仕様のチェック処理に適した JavaCC で記述した．ディレクトリ管理プログラムは，このディレクトリ投入コンパイラとディレクトリ・アクセス API からなる．

図 10 (b) は，アプリケーション・フェーズの流れを示す．以下に，各コンポーネントの役割を示す．

- ディレクトリ投入コンパイラは，自動生成フェーズで生成されたコンポーネントで，独自投入スクリプトで記述された投入データを入力し，ディレクトリサーバが解釈可能な LDIF 形式の投入データを出力する．ディレクトリサーバに LDIF 形式の投入データをインポートすることで，ディレクトリを構築する．
- ユーザ APP は，自動生成フェーズで生成されたディレクトリ・アクセス API を使用することで，ディレクトリ情報を参照することができる．

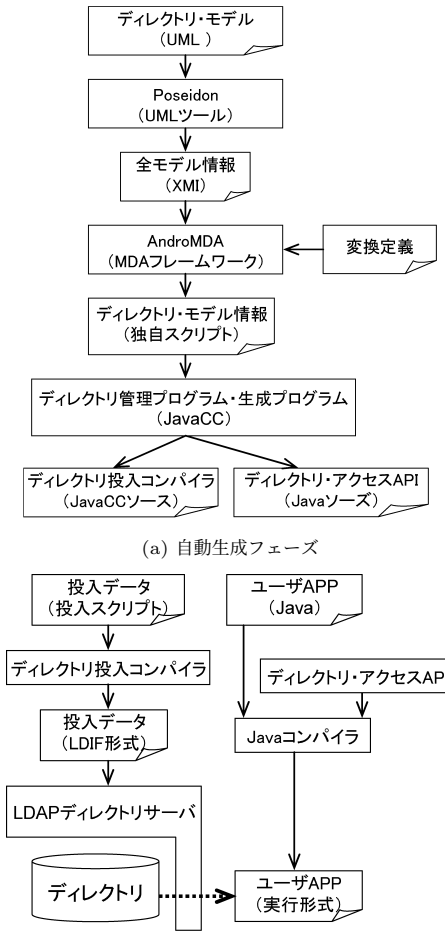


図 10 自動生成システムの流れ
Fig. 10 Flow of automatic production system.

4.3 ディレクトリ投入コンパイラ

4.3.1 特　　長

ディレクトリ投入コンパイラは，以下のような特長を持つ．

- 投入スクリプトは，コンパイル時に DIT の構造上の誤りを検出できるので，不完全なディレクトリ情報が残ることがない．
- 投入スクリプトは，関連についての定義・参照関係をチェックできる．
- LDIF 形式では，各エントリ自身や関連するエントリの DN を指定する必要があるが，投入スクリプトを利用すれば，DN を意識することなく，投入データの記述ができる．

4.3.2 仕　　様

ディレクトリ投入コンパイラは，ディレクトリデータ

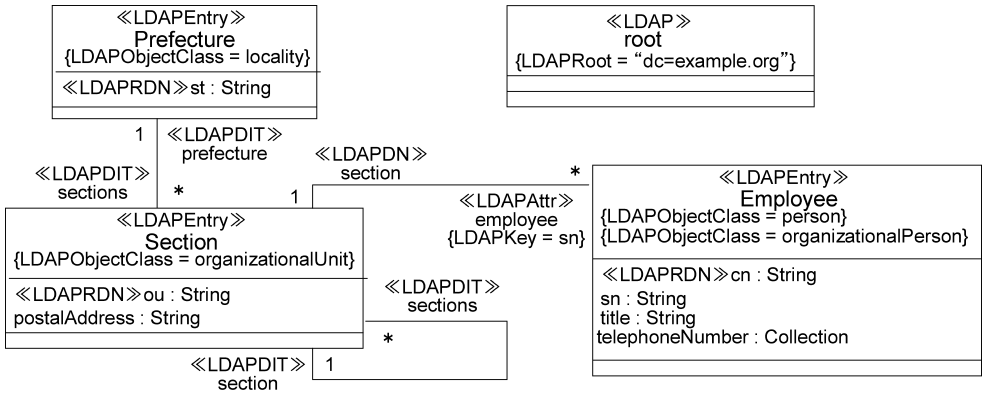


図 11 ディレクトリ・モデリング言語で記述されたモデル
Fig. 11 Model by modeling language for LDAP directory.

```
Prefecture {
 #entry (st="東京都") {
    Section {
       #entry (ou="営業部", postalAddress="東京都新宿区", employee=<sn="A"> <sn="B">) ;
       #entry (ou="開発部", postalAddress="東京都千代田区1", employee=<sn="C">) {
          Section {
             #entry (ou="開発一課", postalAddress="東京都千代田区2", employee=<sn="D"> <sn="E">) ;
             #entry (ou="開発二課", postalAddress="東京都千代田区3", employee=<sn="F"> <sn="G">) ;
          }
       }
    }
 }
 #entry (st="大阪府") {
    Section {
       #entry (ou="人事部", postalAddress="大阪市北区", employee=<sn="H"> <sn="I">) ;
    }
 }
}
Employee {
   #entry (cn="A", sn="A", title="部長", telephoneNumber="03-1111-0000", section=<ou="営業部">) ;
   #entry (cn="B", sn="B", title="担当", telephoneNumber="03-1111-0002", section=<ou="営業部"> ) ;
   #entry (cn="C", sn="C", title="部長", telephoneNumber="03-1111-0003" "03-1111-1113", section=<ou="開発部">) ;
   #entry (cn="D", sn="D", title="課長", telephoneNumber="03-1111-0004", section=<ou="開発部"> ) ;
   #entry (cn="E", sn="E", title="担当", telephoneNumber="03-1111-0005", section=<ou="開発部"> ) ;
   #entry (cn="F", sn="F", title="課長", telephoneNumber="03-1111-0006", section=<ou="開発部"> ) ;
   #entry (cn="G", sn="G", title="担当", telephoneNumber="03-1111-0007", section=<ou="開発部"> ) ;
   #entry (cn="H", sn="H", title="部長", telephoneNumber="06-2222-0001", section=<ou="人事部">) ;
   #entry (cn="I", sn="I", title="担当", telephoneNumber="06-2222-0002", section=<ou="人事部"> ) ;
}
```

図 12 ディレクトリ投入コンパイラの入力
Fig. 12 Input of directory loading compiler.

を一括投入するための投入スクリプトを解釈し，LDIF 形式の投入データを生成する．**図 12** は，**図 11** のモデルから生成されたディレクトリ投入コンパイラの仕様に基づいて記述したスクリプトである．図 12 の `Prefecture`，`Section` および `Employee` はクラス名で，それぞれのクラスに対応するエントリを `#entry()` に記述する．ここには，“パラメータ=値” の形式で，属性型および属性値を指定する．たとえば，`Prefecture` のエントリの `st="東京都"` は，`st` が属性型で，`"東京都"` が属性値である．

`#entry()` の後の `{}` は，DIT による下位エントリとなるエントリを指定する．前述の `st="東京都"` を RDN とするエントリには，`Section` クラスの `ou="営業部"` および `ou="開発部"` のエントリが下位エントリとなる．このように，各エントリの入れ子構造と各エントリの RDN によって DIT 構造を表現し，各エントリの DN を確定する．

DIT 以外の関連を表す方法は，クラス図の役割名を使って表現する．`Employee` クラスのエントリに指定する `section` パラメータはクラス図の `<<LDAPDN>>`

```
dn: cn=A, UMLClassName=Employee, dc=example.org
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: UMLEntry
UMLEntryClass: Employee
cn: A
sn: A
title: 部長
telephoneNumber: 03-1111-0000
objectclass: UMLAssocClassEmployeesection
UMLAssValEmployeesection: ou=営業部, st=東京都, UMLClassName=Prefecture, dc=example.org
```

図 13 ディレクトリ投入コンパイラの出力（一部）
Fig. 13 Output of directory loading compiler.

ステレオタイプが指定された役割名で，ここに関連するエントリを指定する．**図 13** は，図 12 における `Employee` クラスの `cn="A"` のエントリに対して生成される LDIF 形式データである．図 13 の `UMLAssValEmployeesection` 属性型の属性値として指定されている DN は，図 12 の `cn="A"` のエントリの `section=<ou="営業部">` から生成されたものである．

4.4 ディレクトリ・アクセス API

4.4.1 特　　長

ディレクトリ・アクセス API は，以下のような特長を持つ．

- LDAP ディレクトリ・アクセスの Java 標準インタフェースである JNDI（Java Naming and Directory Interface）を使用しており，ディレクトリサーバに依存しない API として実現している．
- EJB（Enterprise Java Beans）などで採用されている Java のデザインパターンを踏襲したインタフェースでアクセスが可能である．

4.4.2 仕　　様

ディレクトリ・アクセス API は，エントリの取得および取得したエントリの属性値やその関連するエントリの取得などの機能を提供する．

図 11 の `Section` クラスに属するエントリに対応する API の機能を，**図 14** に示す．`SectionHome` クラスの `findByPrimaryKey()` メソッドは，RDN 属性をキーにして該当するエントリを `Section` クラスのインスタンスとして取得する．また，`findAll()` メソッドは，`Section` クラスに属するエントリをすべて取得する．`Section` クラスの `getDn()`，`getOu()` および `getPostalAddress()` は，エントリの属性値を取得する．また，`getPrefecture()`，`getSection()`，`getSections()` および `getEmployees()` は，関連するエントリを取得する．`SectionDTO` は，属性値を保持する `java.io.Serializable` を実装した JavaBeans である．

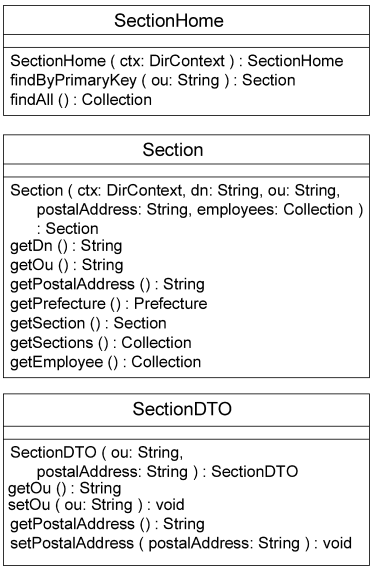


図 14 `Section` クラスで使用できるメソッド
Fig. 14 Methods of class "`Section`".

5. 評　　価

5.1 概　　要

ディレクトリを利用する既存のアプリケーションを対象として，ディレクトリの構築，動作検証および性能評価を実施する．

評価対象は，三菱電機（株）の「情報漏洩防止ソリューション」におけるディレクトリシステムとし，当該ディレクトリの組織・従業員情報を対象として同等のディレクトリをディレクトリ・モデリング言語で記述し，自動生成システムと「情報漏洩防止ソリューション」のそれぞれの API で作成したプログラムにより性能比較を実施する[18)~20)]．

5.2 DIT の構築

ディレクトリ・モデリング言語により記述したモデルから自動生成システムを利用して，対象ディレクトリとほぼ同一の DIT を構築できることが確認できた．

表 1 測定用データの構造
Table 1 Data structure of measurement.

	組織数	従業員数	役職
本社	1	1	社長
本部	5	5	本部長
事業所	25	25	事業所長
部	125	125	部長
課	625	5,000	課長，担当
合計	781	5,156	

記述したモデルの規模は，クラス数が 9，属性数が 30，関連の数が 13 である．

5.3 性能測定

5.3.1 測定方法

(1) データ構造

従業員 5,000 人規模の企業を想定して，**表 1** のような測定用データを利用して，性能比較を実施する．

(2) 測定プログラム

性能比較を実施するプログラムは，ディレクトリによりアドレス帳の運用を実施することを想定して，以下のような機能を持つ 2 つのプログラムとする．情報漏洩防止ソリューションの API は，ユーザ向けに公開している組織・従業員参照 API を使用する．

- プログラム 1
  組織構造から従業員を見つけることを想定したプログラム．本社から下位組織を次々と参照し，その組織とそこに属する従業員すべてのデータを取得する．参照する対象とそのアクセス数は，組織数 781，および従業員数 5156 である．
- プログラム 2
  従業員の属性から，直接従業員を見つけることを想定したプログラム．ランダムに従業員名を生成し，その従業員名をキーに従業員のデータを取得する．参照する対象とそのアクセス数は，従業員数 5000，組織数 5000，従業員の上長数 4999（社長以外の従業員）である．

(3) 測定環境

以下の 2 台のコンピュータを 100BaseT で接続し，評価プログラム動作 PC からディレクトリサーバへ LDAP インタフェースでアクセスするものとする．

(a) 評価プログラム動作 PC
H/W CPU：Intel Pentium4 2.8 GHz
メモリ：760 MB，HDD：35 GB
S/W Java 1.4.2_06
Windows XP Professional

(b) ディレクトリサーバ
H/W CPU：Intel Xeon 3.2 GHz
メモリ：2 GB，HDD：292 GB
S/W SunONE Directory Server 5.2
Windows Server 2003

表 2 測定結果
Table 2 Result of measurement.

測定プログラム	比較システム	実行時間（秒）	転送エントリ数
プログラム 1	情報漏洩防止	20	19,842
	自動生成	24	19,061
プログラム 2	情報漏洩防止	54	40,081
	自動生成	50	40,000

5.3.2 測定結果

測定結果を**表 2** に示す．実行性能は，プログラムの開始と終了時に記述した，日時表示 API の表示結果の差で算出した．また，転送エントリ数は，SunONE Directory Server が提供するシステム管理情報による．

5.4 考察

自動生成システムは，情報漏洩防止ソリューションとほぼ同一の DIT を構築できたことおよび同一機能のアプリケーション作成が可能であることから，ディレクトリ・モデリング言語は十分な記述能力があると判断できる．

転送エントリ数においては，プログラム 1 は差が 781 で，プログラム 2 は 81 のみとなっている．このことから，自動生成システムの API は，ディレクトリ・アクセスにおいて，情報漏洩防止ソリューションの API とほぼ同一の処理方式で実現できていることが分かる．

実行時間においては，プログラム 1 およびプログラム 2 の実行時間合計が同一であり，自動生成システムの API は，実用的な性能で動作可能であることが分かる．なお，両者の性能が逆転していることに対する考察を以下に示す．

情報漏洩防止ソリューションと自動生成システムにおける性能に影響する処理の相違点は次のようである．

- 自動生成システムの API はすべての属性値を取得するのに対して，情報漏洩防止ソリューションには，エントリの一部の属性値のみを取得する API がある．したがって，自動生成システムは，データ転送量が増大し，性能上不利となる．
- 情報漏洩防止ソリューションは，エントリを取得するときにディレクトリ構造の矛盾をチェックするためにエントリ転送をともなわないディレクトリのアクセスを行うが，自動生成システムは投入コンパイラでデータ投入するため矛盾チェックが不要である．したがって，情報漏洩防止ソリューションは，ディレクトリのアクセス数が増大し，性能上不利となる．

プログラム 1 においては，このプログラムの実現には，情報漏洩防止ソリューションがエントリの一部の属性値のみを取得する API を使用するため，比較上，自動生成システム側のデータ転送量が増大し，その影響が顕在化した．なお，情報漏洩防止ソリューション側を自動生成システムと同等の属性情報を取得するように修正することも考えられるが，これにより転送エントリ数および実行時間が増大し，比較として有効な測定値を得られなかったので，この測定値は採用しなかった．

プログラム 2 においては，転送エントリ数が多いプログラムであり，情報漏洩防止ソリューションが矛盾チェックのために行うディレクトリへのアクセスが増大し，その影響が顕在化した．

6. おわりに

本論文で提案したモデリング言語および試作した自動生成システムを既存のシステムへ適用し比較した結果から，ディレクトリ設計・構築に十分に適用可能という結論を得た．このシステムを利用することによって，①ディレクトリ設計情報をモデルにより可視化することですべての開発関係者の理解が可能となり上流工程での問題抽出を可能とすることによる品質向上，②API の自動生成およびデータ投入を容易にすることによる生産性向上，③設計情報であるモデルとそこから生成されたアプリケーションが一致することによる保守性向上が，期待できる．

このシステムの本来の利用目的は，新規アプリケーション・システム開発において貢献することである．今後は，ディレクトリサービスを利用する新規プロジェクトにおいて，本論文で示したモデリング言語および自動生成システムを適用し，効果の評価を実施して，モデリング言語の拡張または洗練化を実施する必要がある．

ディレクトリ・モデリング言語の拡張仕様としては，本論文では対象外としたスキーマ定義に関連する記述やユーザ定義の操作（operation）などがあげられる．

参考文献

1) 北島聡史，中村義幸ほか：ビジネスプロセスモデリングと実行可能 UML を連結させた情報システム構築法の試作と評価，情報処理学会研究報告，DPS-121, pp.1–6 (2005).
2) 北島聡史，小泉寿男ほか：ビジネスプロセスモデリングによる情報システム構築の一手法，情報処理学会研究報告，DPS-119, pp.15–20 (2004).
3) 小澤陽平，細川卓誠ほか：組み込みソフトウェア向け UML における状態遷移表・ソースコード変換方式，FIT（情報科学技術フォーラム），pp.165–66 (2003).
4) 斉藤　亮，田中文基ほか：ISO 19100 地理空間データの SemanticWeb での利用に関する研究，FIT（情報科学技術フォーラム），pp.7–8 (2002).
5) Milicev, D.: Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments, *IEEE Trans. Softw. Eng.*, Vol.28, No.4, pp.413–431 (2002).
6) Apvrille, L. and Court, J.-P.: TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit, *IEEE Trans. Softw. Eng.*, Vol.30, No.7, pp.473–487 (2004).
7) MDA (Model Driven Architecture). http://www.omg.org/mda/
8) ITU-T (International Telecommunication Union-Telecommunication Standardization Sector). http://www.itu.int/ITU-T/
9) IETF (Internet Engineering Task Force). http://www.ietf.org/
10) Howes, T.: *Understanding and Deploying Ldap Directory Services*, Addison-Wesley Professional (2003).
11) OMG (Object Management Group). http://www.omg.org/
12) UML (Unified Modeling Language). http://www.uml.org/
13) グラディ・ブーチ（著），羽生田栄一（訳）：UML ユーザーガイド，ピアソン・エデュケーション (1999).
14) Poseidon for UML. http://gentleware.com/
15) AndroMDA. http://www.andromda.org/
16) JavaCC (Java Compiler Compiler). https://javacc.dev.java.net/
17) 五月女健治：JavaCC コンパイラ・コンパイラ for Java，テクノプレス (2003).
18) 情報漏洩防止ソリューション. http://www.mitsubishielectric.co.jp/security/info/
19) 情報漏洩防止ソリューション. http://www.mdis.co.jp/security/solution02/
20) 近藤誠一，白木弘明ほか：ロールベースアクセス制御情報の多バージョン並行処理制御を利用した監査ログトラッキング手法，情報処理学会論文誌：データベース，Vol.46, No.SIG18(TOD28), pp.103–115 (2005).

(平成 18 年 3 月 16 日受付)
(平成 18 年 7 月 24 日採録)

（担当編集委員　高倉 弘喜）

五月女健治（正会員）

1979 年大阪大学基礎工学部情報工学科卒業．同年三菱電機（株）入社．現在，同社より法政大学大学院イノベーション・マネジメント研究科に出向中．Web シングルサインオンシステム，LDAP ディレクトリサービスの応用等の研究に従事．

近藤　誠一（正会員）

1984 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年三菱電機（株）入社．1989〜1992 年（財）新世代コンピュータ技術開発機構（ICOT）出向．統合データベース，システム間連携，情報セキュリティシステムに関する研究開発に従事．

大沼　聡久

1986 年早稲田大学理工学部機械工学科卒業．同年三菱電機（株）入社．言語プロセッサの開発を経て，現在，情報セキュリティシステムに関する研究開発に従事．

小宮　　崇

2001 年佐賀大学大学院工学系研究科電子工学専攻修了．同年三菱電機（株）入社．Web シングルサインオンシステムに関する研究開発に従事．

酒井三四郎（正会員）

1956 年生．1984 年静岡大学大学院電子科学研究科博士後期課程（電子応用工学専攻）修了．学習院大学，新潟産業大学，静岡大学工学部を経て，1998 年静岡大学情報学部助教授．現在，同学部教授．工学博士．ソフトウェア開発支援環境，プログラミング教育支援環境，遠隔学習，協調学習に関する研究・開発に従事．電子情報通信学会，教育システム情報学会，日本 e-Learning 学会各会員．

水野　忠則（フェロー）

1945 年生．1968 年名古屋工業大学経営工学科卒業．同年三菱電機（株）入社．1993 年静岡大学工学部情報知識工学科教授，現在，静岡大学創造科学技術大学院長，情報学部情報科学科教授．工学博士．情報ネットワーク，モバイルコンピューティング，放送コンピューティングに関する研究に従事．著訳書としては『コンピュータネットワーク概論』（日経 BP），『モダンオペレーティングシステム』（ピアソン・エデュケーション）等がある．電子情報通信学会，IEEE，ACM 各会員．当会フェロー，監事．