社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DBS－141 (12)
2007－GN－ 62 (12)
2007／1／26

LDAPのためのモデリング言語と自動生成システム

五月女健治[†,††] 近藤誠一[†] 酒井三四郎[†††]

LDAP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，ディレクトリの階層構造を例示する程度の不完全な図を示すのが一般的である．本稿では，LDAPディレクトリのための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たので，その仕様と利用方法を報告する．

Modeling Language for LDAP and Automatic Production System

KENJI SAOTOME,[†,††] SEIICHI KONDO,[†] and SANSHIRO SAKAI[†††]

There are not the standards of the model for the design of the Directory and generally incomplete diagrams of the Directory have been illustrated. Then, we propose the modeling language extending UML for the design of LDAP Directory. We developed for trial the system that automatically produces the programs that manage the Directory. We report the specification and method making use of these systems.

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危険性がある．

筆者らは，すでに，UMLモデルをベースとす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し，本モデリング言語からディレクトリ管理プログラムを自動生成するシステムを試作して，本システムの適用性を実証した[1]．

[†]三菱電機株式会社 情報技術総合研究所
Mitsubishi Electric Corporation,
Information Technology R&D Center
[††]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
[†††]静岡大学情報学部
The Faculty of Information, Shizuoka University

今回は，モデリング言語にオブジェクトクラスおよび属性型のスキーマ定義の表記を追加し，そのスキーマの自動生成機能を実現して，実用レベルの仕様を備えたシステムが完成した．本稿では，本モデリング言語と自動生成システムの仕様およびUMLによるシステム開発におけるこれらの活用方法について報告する．

本稿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[2)~3)]．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[4)~5)]．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，attribute syntaxは属性構文と呼び，UMLにおけるattributeは単に属性と呼び区別する．

2. モデリング言語の定義仕様

2.1 概要

ディレクトリ・モデリング言語の構文仕様および

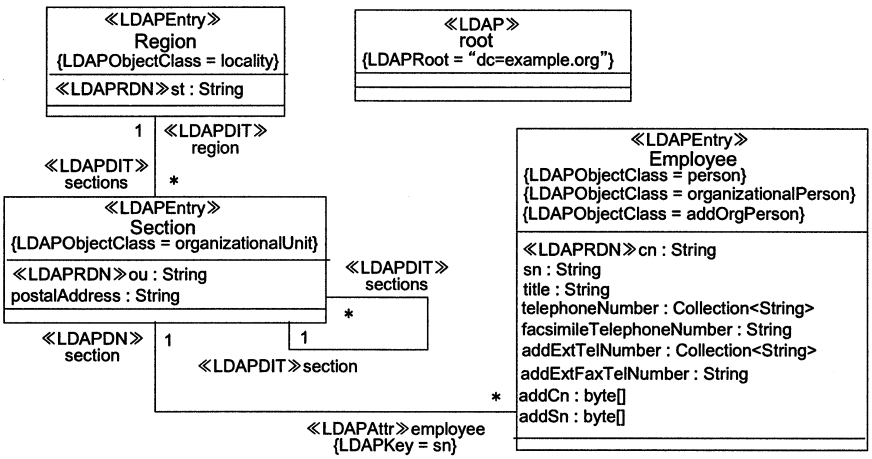


図1　ディレクトリ・モデリング言語による DIT のモデル
Fig.1　DIT Model by Directory Modeling Language.

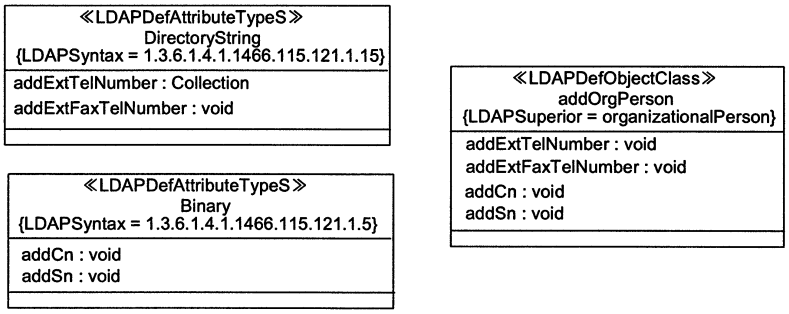


図2　ディレクトリ・モデリング言語によるスキーマ定義のモデル
Fig.2　Scheme Definition Model by Directory Modeling Language.

意味仕様について記述する．図 1 および図 2 は，当モデリング言語で記述したモデルの例である．

2.2 構文仕様

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．クラスには，そのクラスの役割を示すステレオタイプを指定する．以下に，ステレオタイプごとに，本モデリング言語の構文仕様を示す．

2.2.1 <<LDAP>>クラス

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

2.2.2 <<LDAPEntry>>クラス

(1)　クラス名は，当モデルの他の<<LDAPEntry>>ステレオタイプを指定するクラス内で一意でなければならない．
(2)　クラスには，ひとつ以上の{LDAPObjectClass}タグ付き値を指定しなければならない．{LDAPObjectClass}タグ付き値には，文字列を指定する．
(3)　クラスは，複数の属性(attribute)をもつ．
(4)　属性は，属性の名前およびタイプ(type)からなる．

指定する属性の名前は，当クラス内で一意でなければならない．

(5) タイプには， String， byte[]， Collection， Collection<String>または Collection<byte[]>を指定する．

(6) 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．

(7) 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．

(8) 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
- *

(9) 関連の誘導は，双方向または片方向を指定する．

(10) 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>>

(11) 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(10)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．

(12) <<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の名前のひとつを指定する．

2.2.3 <<LDAPDefAttributeTypeS>>クラス

(1) このクラスには，1 つの{LDAPSyntax}タグ付き値を指定しなければならない．{LDAPSyntax}タグ付き値には，文字列を指定する．

(2) クラスは，複数の属性をもつ．

(3) 属性は，属性の名前およびタイプからなる．

(4) 属性の名前は他の<<LDAPDefAttributeTypeS>>ステレオタイプのクラスに指定された属性の名前の内で，一意でなければならない．

(5) タイプには，void または Collection を指定する．

(6) この属性には，必要なら，{LDAPEquality}，{LDAPOrdering}または{LDAPSubstr}タグ付き値を，それぞれ 1 つ指定することができる．これらのタグ付き値には，文字列を指定する．

(7) このクラスには，関連を指定してはならない．

2.2.4 <<LDAPDefObjectClass>>クラス

(1) このクラスには，1 つの{LDAPSuperior}タグ付き値を指定しなければならない．{LDAPSuperior}タグ付き値には，文字列を指定する．

(2) クラスは，複数の属性をもつ．

(3) 属性は，属性の名前およびタイプからなる．

(4) 属性の名前は，クラス内で，一意でなければならない．

(5) タイプは，指定しないか，指定するときは void を指定する．

(6) それぞれの属性に対して，<<LDAPMust>>ステレオタイプを指定できる．

(7) このクラスには，関連を指定してはならない．

2.3 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．

2.3.1 クラス

2.3.1.1 <<LDAP>>クラス

<<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ディレクトリ・エントリの最上位エントリとなる DN (Distingushed Name)を指定する．

2.3.1.2 <<LDAPEntry>>クラス

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を指定する．

クラスに定義される属性のタイプは，自動生成システムが生成する API で使用する Java のタイプを示すもので次のように指定する．その属性構文が Binary, Octet String または Certificate などのバイナリデータを示すとき，単一値では byte[]，多値では Collection <byte[]>を指定する．その属性構文が Directory String, Boolean または Integer などの文字列データを示すとき，

単一値では String，多値では Collection または Collection<String>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．

2.3.1.3 <<LDAPDefAttributeTypeS>>クラス

<<LDAPDefAttributeTypeS>>ステレオタイプを指定したクラスは，共通の属性構文をもつ，ユーザ定義の属性型を定義するときに使用する．共通の属性構文は，クラスに指定する{LDAPSyntax}タグ付き値に指定する．{LDAPSyntax}タグ付き値には属性構文(属性型定義の SYNTAX キーワードに対応)を指定する．

クラスの属性はディレクトリの属性型を表し，属性の名前は属性型名を示す．また，タイプには，単一値属性型のときは void，多値属性型のときは Collection をそれぞれ指定する．クラスの各属性に指定できる {LDAPEquality} ， {LDAPOrdering} および {LDAPESubstr}タグ付き値は照合規則を表し，それぞれ等価性(EQUALITY)，順序性(ORDERRING)および部分文字列一致(SUBSTR)の照合規則を表す．これらのタグ付き値に指定する文字列には，LDAP の属性型定義で指定する照合規則のキーワードを指定する．

2.3.1.4 <<LDAPDefObjectClass>>クラス

<<LDAPDefObjectClass>>ステレオタイプを指定したクラスは，ユーザ定義のオブジェクトクラスを定義するときに使用する．クラスに指定する {LDAPSuperior}タグ付き値には基底オブジェクトクラス(オブジェクトクラス定義の SUP キーワードに対応)を指定する．

クラスの属性は，当オブジェクトクラスに属する属性型を示す．属性の名前は，属性型名を表す．各属性に指定する<<LDAPMust>>ステレオタイプは，この属性型が当オブジェクトクラスの必須属性型であることを示す．

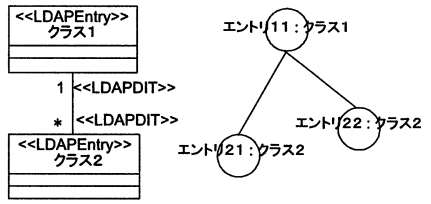
2.3.2 クラスの関連

2.3.2.1 概要

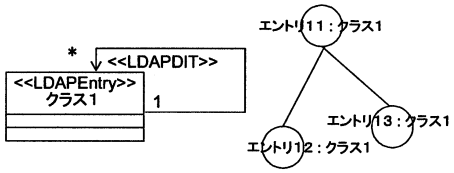
関連は，エントリ間の関係を示すために使用する．ディレクトリにおける関連の実現方法を表すために，関連端に <<LDAPDIT>>， <<LDAPDN>> または <<LDAPAttr>>ステレオタイプのいずれかを指定する．

2.3.2.2 <<LDAPDIT>>による関連

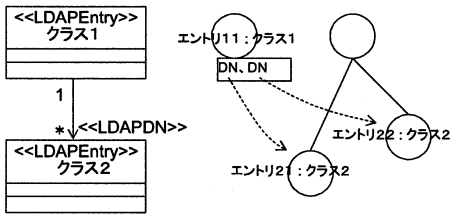
<<LDAPDIT>>ステレオタイプは，その関連を DIT



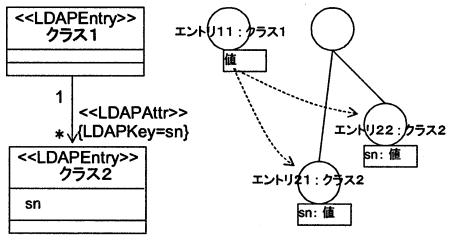
(a) <<LDAPDIT>>の関連と DIT



(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 3　モデリング言語による関連と DIT

Fig.3　Association by Modeling Language and DIT.

（Directory Information Tree）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 つ以上もつことはできない．**図 3(a)**は，<<LDAPDIT>>ステレオタイプを使用したクラス図と対応する DIT の概略図を示す．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も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 3(b)**は自己参照型の関連を表すクラス図と対応する DIT の概略図を示す．

2.3.2.3 <<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 をも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ー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 3(c)**は，<<LDAPDN>>ステレオタイプを使用したクラス図と対応する DIT の概略図を示す．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．

2.3.2.4 <<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みなす．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 3(d)**は<<LDAPAttr>>ステレオタイプを使用したクラス図と対応する DIT の例である．

3. ディレクトリ管理プログラム

3.1 概要

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）を自動生成する，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

ディレクトリ管理プログラムは，ディレクトリ投入コンパイラ，ディレクトリ・アクセス API およびユーザ定義のスキーマ情報からなる．

3.2 ディレクトリ投入コンパイラ

3.2.1 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もつ．

・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．

3.2.2 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 4** は，**図 1** 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 4** の Region，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，Region 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 におけ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>>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．

3.3 ディレクトリ・アクセス API

3.3.1 特長

ディレクトリ・アクセス API は，以下のような特長をもつ．

```
Region {
  #entry (st="東京都") {
    Section {
      #entry (ou="営業部", postalAddress="東京都新宿区", employees=<sn="A"> <sn="B">) ;
      #entry (ou="開発部", postalAddress="東京都千代田区1", employees=<sn="C">) {
        Section {
          #entry (ou="開発一課", postalAddress="東京都千代田区2", employees=<sn="D"> <sn="E">) ;
          #entry (ou="開発二課", postalAddress="東京都千代田区3", employees=<sn="F"> <sn="G">) ;
（省略）
}
Employee {
  #entry (cn="A", sn="A", title="部長", telephoneNumber="03-1111-0001", facsimileTelephoneNumber="03-1111-1001",
    addCn="gmA=", addSn="gmA=", addExtTelNumber="11-0001", addExtFaxTelNumber="11-1001", section=<ou="営業部">) ;
  #entry (cn="B", sn="B", title="担当", telephoneNumber="03-1111-0002", facsimileTelephoneNumber="03-1111-1002",
    addCn="gmE=", addSn="gmE=", addExtTelNumber="11-0002", addExtFaxTelNumber="11-1002", section=<ou="営業部"> ) ;
（省略）
  #entry (cn="I", sn="I", title="担当", telephoneNumber="06-2222-0002", facsimileTelephoneNumber="03-2222-1002",
    addCn="gmg=", addSn="gmg=", addExtTelNumber="22-0002", addExtFaxTelNumber="22-1002", section=<ou="人事部"> ) ;
}
```

図4　ディレクトリ投入コンパイラの入力
Fig.4　Input of Directory Loading Compiler.

・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永続オブジェクト参照のインタフェースでアクセスが可能である．

3.3.2 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1のSectionクラスに属するエントリに対応するAPIの機能を，**図5**に示す．SectionHomeクラスのfindByPrimaryKey()メソッドは，RDNとなる属性型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，getRegion()，getSection()，getSections()およびgetEmployee()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．

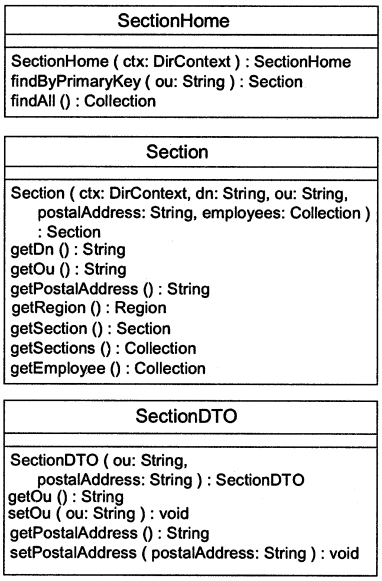


図5　Sectionクラスで使用できるメソッド
Fig.5　Methods of Class "Section".

3.4 ユーザ定義のスキーマ情報

3.4.1 特長

ユーザ定義のスキーマ情報は，以下のような特長をもつ．

・ユーザ定義のスキーマ情報は，標準のLDIF形式で生成されるため，ディレクトリサーバに依存しない．

3.4.2 仕様

<<LDAPDefAttributeTypeS>>ステレオタイプのクラスの情報から生成されたユーザ定義の属性型および<<LDAPDefObjectClass>>ステレオタイプのクラスの情報から生成されたユーザ定義のオブジェクトクラスを登録するLDIF形式のデータからなる．**図6**は，**図2**の<<LDAPDefAttributeTypeS>>クラスから生成された属性型定義のデータである．また，**図7**は，**図2**の<<LDAPDefObjectClass>>クラスから生成されたオブジェクトクラス定義のデータである．

```
dn: cn=schema
changetype: modify
add: attributetypes
attributeTypes: ( addExtTelNumber-oid
 NAME 'addExtTelNumber'
 DESC 'User Defined Attribute'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
 X-ORIGIN 'user defined' )
attributeTypes: ( addExtFaxTelNumber-oid
 NAME 'addExtFaxTelNumber'
 DESC 'User Defined Attribute'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
 SINGLE-VALUE
 X-ORIGIN 'user defined' )
attributeTypes: ( addCn-oid
 NAME 'addCn'
 DESC 'User Defined Attribute'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
 SINGLE-VALUE
 X-ORIGIN 'user defined' )
attributeTypes: ( addSn-oid
 NAME 'addSn'
 DESC 'User Defined Attribute'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
 SINGLE-VALUE
 X-ORIGIN 'user defined' )
```

図 6　属性型を定義する LDIF
Fig.6 LDIF for Definition of Attribute Types.

```
dn: cn=schema
changetype: modify
add: objectclasses
objectClasses: ( addOrgPerson-oid
 NAME 'addOrgPerson'
 SUP organizationalPerson STRUCTURAL
 MUST ( addCn $ addSn )
 MAY ( addExtTelNumber
 $ addExtFaxTelNumber )
 X-ORIGIN 'user defined' )
```

図 7　オブジェクトクラスを定義する LDIF
Fig.7 LDIF for Definition of Objectclasses.

4. アプリケーション開発プロセス

ディレクトリ・モデリング言語および自動生成システムをアプリケーション開発の開発プロセスに適応する方法を，主にディレクトリまわりの開発に着目して記述する．

(1)　要求分析フェーズ

図 8 のような要求分析モデルを作成する．当モデルでは、エンティティに関連するクラスのみを示す．

(2)　システム分析フェーズ

要求分析モデルにおいて，用途およびアクセスの形態・頻度などから，各クラスの実装方法を決定する．図 8 では，商品および売上クラスは，リレーショナルデータベースで，地域，部署および従業員クラスをディレクトリで実装する．次に，ディレクトリ実装のクラスとそれらの関連のすべてがディレクトリ標準の

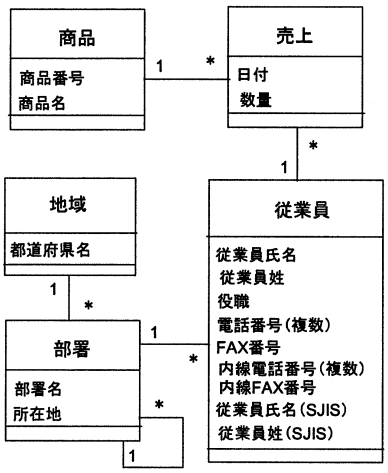


図 8　要求分析モデル
Fig.8　Requirement Analysis Model.

構造または標準構造の部分的な拡張で実現可能な場合は，標準構造を採用するかどうかを検討する．標準の構造を採用すれば，ディレクトリ製品または第三者ソフトウェアが提供する機能を利用することで，開発・運用コスト削減の効果が期待できる．しかし，拡張仕様によっては，他ソフトウェアが期待する認証機能などのディレクトリ製品が提供する機能の動作に悪影響またはそのリスクとなる場合があり，その際ディレクトリ製品提供者とのトラブルとなる可能性があることを考慮する必要がある．また，将来拡張が予定される場合も同等の判断が必要である．標準の構造をベースに目的の機能を実現する場合は，設計フェーズで，要求分析モデルから標準の構造への対応作業を実施する．

(3)　設計フェーズ

独自の DIT 構造を実装する場合，本稿のシステムを利用する．要求分析モデルから当モデリング言語のモデルへの変換を，以下のように行う．この方法で図 8 を変換した結果が，図 1 および図 2 である．

- ディレクトリで実装するクラスを定義し，<<LDAPEntry>>ステレオタイプを指定する．
- クラスに適したディレクトリ標準のオブジェクトクラスを選定し，属性の名前を当オブジェクトクラスに属する属性型名に置換える．対応する属性型が存在しない場合は，ユーザ定義の属性型を<<LDAPDefAttributeTypeS>>クラスで定義し，その属性型に置換える．
- 属性型に対応する属性のタイプは，当属性型の属性構文および多値か単一値によって決める．特に，バ

イナリ値をもつ属性型にはbyte[]とする．バイナリ値をもつ属性型には，パスワードやデジタル認証書などがある．特殊なケースでは外字を含む姓名に対応するため，ディレクトリ標準のUTF-8コード系ではなくSJISコード系でデータを保管する属性型が必要となる場合にバイナリ値とする．

・上述のユーザ定義の属性型をもつオブジェクトクラスを<<LDAPDefObjectClass>>クラスで定義する．

・当クラスに関係するオブジェクトクラスを，{LDAPObjectClass}タグ付き値で指定する．RDNとなるクラス属性に，<<LDAPRDN>>ステレオタイプを指定する．

・関連について，その実現方法を決定し，関連端に対応するステレオタイプ，タグ付き値および役割名を指定する．

(4) 実装・テストフェーズ

自動生成システムによって，設計フェーズで作成したモデルからディレクトリ管理プログラムを生成する．当システムから生成された成果とその利用方法を以下に示す．このフェーズで改善点または誤りが検出されたときは，(3)の設計フェーズに戻る．

・ユーザ定義のスキーマ情報
アプリケーションのテスト環境の準備のため，必要な追加のスキーマ情報を登録する．

・ディレクトリ・アクセスAPI
アプリケーションのコード作成時に，ディレクトリ参照において当APIを使用する．

・ディレクトリ投入コンパイラ
ディレクトリ投入コンパイラを利用して，アプリケーションのテストデータの生成およびディレクトリへの投入を行う．

(5) 完了

アプリケーション開発が完了したら，モデリング言語で記述したモデル情報を，設計文書および保守情報として利用する．

5. 評価

ディレクトリを利用する既存のアプリケーションである三菱電機（株）の「情報漏洩防止ソリューション」を対象として，同等のディレクトリの構築を実施する[6]．既に，当該ディレクトリの組織・従業員情報を対象として，DIT構築，動作確認および性能評価を実施し，十分な機能と性能を確認している[1]．今回，モデリング言語にオブジェクトクラスおよび属性型のスキーマ定義機能を追加したので，当該機能の評価のため，「情報漏洩防止ソリューション」のユーザ定義のオブジェクトクラスを35個および属性型を136個定義し，同一のスキーマ定義を確認できた．この定義で，当モデリング言語で作成したモデルのクラスは，<<LDAPDefObjectClass>>クラスが35個，<<LDAPDefAttributeTypeS>>クラスが7個であった．属性型の136個が，<<LDAPDefAttributeTypeS>>クラス7個で表現できたのは，これらの属性型の属性構文が7種類であったことを示している。

6. おわりに

本稿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，スキーマ定義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．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，本稿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．

参考文献

1) 五月女健治，近藤誠一ほか：LDAPディレクトリのためのモデリング言語およびディレクトリ管理プログラム自動生成システム，情報処理学会論文誌 ：データベース，Vol.47， No.SIG 13 (TOD31)，pp.28-39 (2006).

2) ITU-T(International Telecommunication Union -Telecommunication Standardization Sector)．http://www.itu.int/ITU-T/

3) IETF(Internet Engineering Task Force)．http://www.ietf.org/

4) OMG(Object Management Group)．http://www.omg.org/

5) UML(Unified Modeling Language)．http://www.uml.org/

6) 情報漏洩防止ソリューション．http://www.mitsubishielectric.co.jp/security/info/solution/ http://www.mdis.co.jp/security/solution02/