分散木構造データベースJungleによる企業向け許認可システム

enterprise Authorization system on distributed tree structure database Jungle

琉球大学工学部情報工学科

金川竜己
指導教員 河野真治

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究目的

我々があつかっている知識は主に木構造である。しかし、知識の量は膨大であり、人が全てを記憶しておくのは難しいため、データベースに格納したい。しかし、RDB 上に木構造データを格納するためには、煩雑なデータ設計が必要になる。

また、データベースを使用するウェブサービスの規模も年々大きなものとなり、それに比例してデータベースへの負荷も増大し、その結果サービスが停止する自体が多々見られるようになった。そのため、データベースの処理性能はそのままサービスの質につながっている重要な項目となっている。

データベースの処理性能を向上させる代表的な方法として、ハードウェア的に高性能なマシンを用意することで処理性能を上げるスケールアップと、汎用的なマシンをいくつも用意し、処理を分散させることで処理性能を上げるスケールアウトの 2 つがある。単純に処理能力を上げたいのなら、スケールアップは有効ではあるが、単一のマシンを高性能にするのにも限界があり、いずれはそのマシンの限界を超える負荷がかかる可能性もある。それに対しスケールアウトは、処理が重くなるに連れて汎用的なマシンを順次追加していくことで性能を上げるため、ハードウェア的に高性能なマシンを要求せずにすみ、柔軟な対応を取ることが出来るため、データベースの性能を上げる方法としてはスケールアウトが求められている。本研究で扱うスケーラビリティとはスケールアウトのことをさす。

今、最も使われているデータベースである RDB は、マシンを追加して負荷を分散することが容易ではない。そのためスケーラビリティを持つことが困難である。

当研究室では、これらの問題を解決した、煩雑なデータ設計が必要ないスケーラビリティのあるデータベースを目指して、非破壊的木構造データベース Jungle を開発している。しかし、Jungle はまだ開発途中であり、データベースに必要な API 等もまだ十分に実装されていない。そこで、当研究では、共同研究を行っている Symphonies 社が開発している業務用アプリケーション maTrix に Jungle を組み込み、実装すべき API の洗い出しを行い、その後実用 DB としての性能があるかテストを行う。

第2章　Jungle

2.1　JungleCore

2.2　JungleNetwork

第3章　Jungleを使用したアプリケーション

本章では、実際に Jungle を使用したアプリケーションを紹介する。

3.1　BBS

Jungle を用いた簡易掲示板システムで、組み込みウェブサーバーである jetty をフロントエンドとし作成、そのバックエンドとして Jungle を利用している。今迄 Jungle にデータを格納する方法は実際に Java のコードから直接格納するしか無かったが、BBS を使用することで、ブラウザを使用して Jungle へデータの格納とデータの表示を行えるようになった。

3.2　XMLReader

実際にアプリケーション等で使用しているデータを書き出す際に、XML 形式を使用することが多い。JungleXMLReader は、XML ファイルを読み込み、Jungle にデータとして格納するアプリケーションである。XML はデータが木構造になっているため、Jungle にそのまま格納することが可能である。

JungleXMLReader の実装には sax(Simple API for XML) を用いて実装を行った。sax は、XML のタグやテキストデータを読んだ際にイベントを発生させる。プログラマは、ContentHandler という特殊なイベントリスナを sax の parser に登録すると、構文解析処理の過程で XML の要素を読んだ際に、発生するイベントを取得できる。イベントには、読み込んだタグの名前やテキストデータが含まれているため、プログラマは構文解析結果を随時受け取ることが出来る。

saxで構文解析の途中に発生するイベントは多数あるが、今回XMLReaderで使用したイベントだけ以下に記述する。

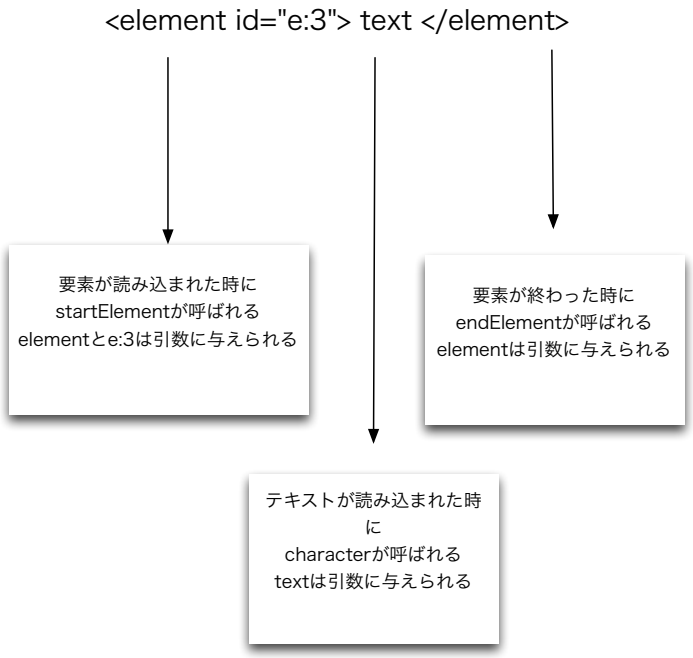

図 3.1: イベントが呼ばれるタイミングの例

表 3.1: 発生するイベント一覧

イベント名	呼ばれるタイミング
startDocument	XML の Parse を始める際に呼ばれる
startElement	要素を読み込んだ際に呼ばれる
characters	要素の Text を読み込んだ際に呼ばれる
endElement	要素が終わった時に呼ばれる
endDocument	XML の Parse が終わった際に呼ばれる

saxでは、org.xml.sax.helpers.DefaultHandlerという形で、ContentHandlerのデフォルト機能が提供されているため、プログラマはこれを継承することで、必要なイベント処理のみをoverrideして記述できるようになっている。JungleXMLReaderで使用しているReadXmlHandlerは、要素を読み込んだ時、テキストを読み込んだ時、要素が終わった時、XMLのParseが終わった時、の4つのイベントを使用している。

sax example

```
SAXParser saxParser = saxParserFactory.newSAXParser();
ReadXmlHandler readXmlHandler = new ReadXmlHandler();
saxParser.parse(new FileInputStream(xmlPath), readXmlHandler);
```

sax example は、実際に saxPerser を生成し構文解析を発生させるまでのコードである。

1. 最初に saxParserFactory.newSAXPerser() で新しい Parser を生成する
2. 次に DefaultHandler を継承した ReadXmlHandler のインスタンスを生成する
3. 最後に parser.parse(XMLFile,DefaultHandler) に読み込みたい XML ファイルと生成した ContentHandler を渡し構文解析を行う。

3.3 maTrix

maTrix とは Symphonies 社が開発しているアカウント管理、許諾判定システムのことである。人や組織の情報などを保持しており、それらを関連付けることで表現している。今現在 maTix は PostgreSQL を使用している。本研究は実用アプリケーションである maTrix 上に Jungle を組み込み評価することが目的であるので、maTrix についても本章で解説することとする。

3.3.1 maTrix におけるアプリケーションのアカウント管理

maTrix を使用しなかった場合、図 3.2 のようにアプリケーションごとにアカウントを作る必要があり、user の権限等が変わった際に、全てのアカウントの更新を行う必要があるため、手間がかかったり、ミスが発生するなど、業務に支障をきたすおそれがある。

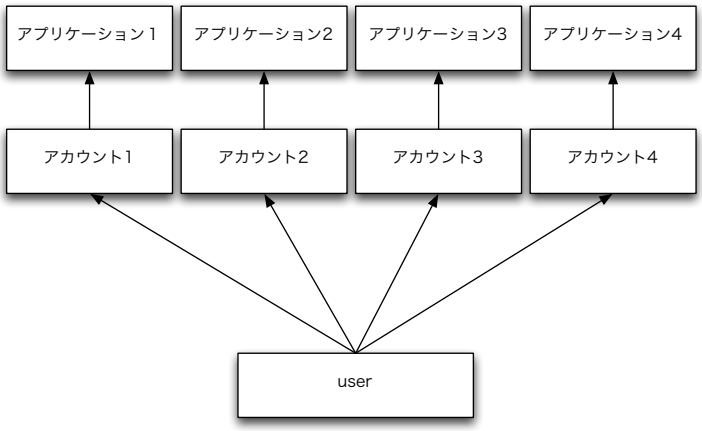

図 3.2: maTrix におけるアカウント管理例 1

しかし maTrix を用いることで図 3.3 の様に、maTrix のアカウント 1 つで全ての業務アプリケーションにアクセスできるため、user の権限が変わったとしても maTrix のアカウントの更新だけしか行う必要がないので、手間もかからず、ミスも発生しづらい。

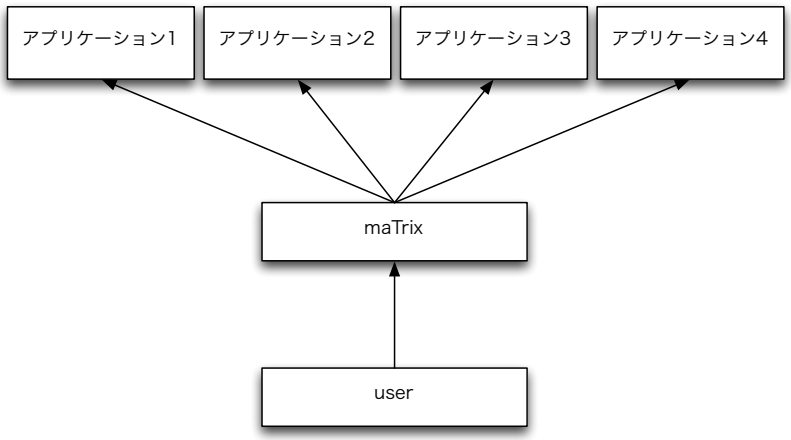

図 3.3: maTrix におけるアカウント管理例 2

3.3.2 maTrix の保持するデータ構造

matrix は人、役職、役割、権限と言った木構造の組織、許認可の判断に用いるポリシーファイルの 2 つのデータを持っている。maTrix が保持しているデータはお互いに参照している。また、過去のデータも全て保持し、それらのデータはまとめて構成情報モデルとして版管理している。

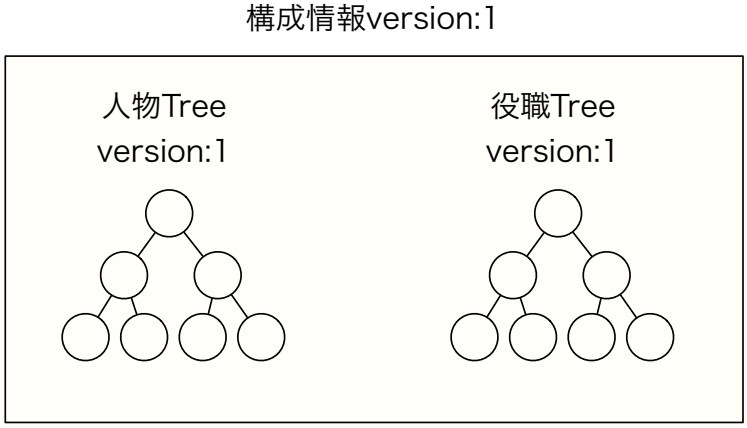

図 3.4: maTrix の構成情報例 1

人物と役職の Tree の version が共に 1 の時、構成情報モデルの version も 1 とする。

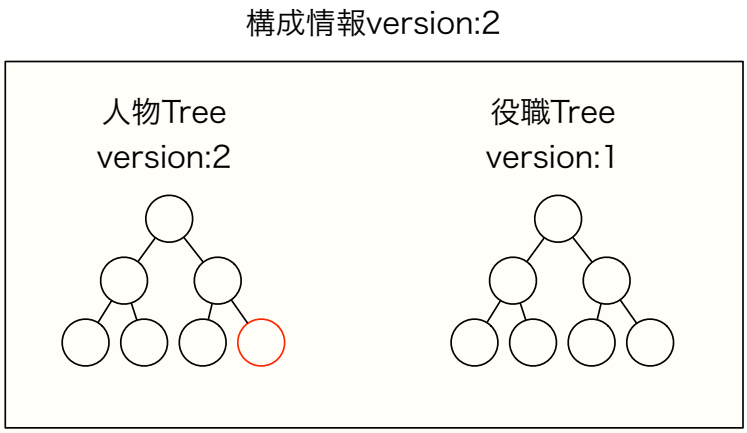

図 3.5: maTrix の構成情報例 1

構成情報モデルの version1 の人物 Tree のデータを更新した場合、新しく version:2 の構成情報が構築され、maTrix は version:1 の構成情報と version:2 の構成情報の両方を保持し、両方の構成情報にアクセスすることが可能である。

maTrix のデータ構造は、木構造であるため、組織の Tree を xml や json 形式で出力することができる。以下に人物 Tree を xml 形式で出力したデータの一部を記述する。

```
<Persons>
<Person id="p:1" type="Person">
<accountId>a:26</accountId>
<lastName>東</lastName>
<name>東俊一</name>
<nameReading>あずましゅんいちくん</nameReading>
<roleRefIds>
<roleRefId>r:10</roleRefId>
<roleRefId>r:34</roleRefId>
</roleRefIds>
<parentOrganizations type="OrganizationMappedByRole">
<OrganizationMappedByRole type="OrganizationMappedByRole">
<organizationRefId>o:2</organizationRefId>
<roleRefId>r:10</roleRefId>
</OrganizationMappedByRole>
<OrganizationMappedByRole type="OrganizationMappedByRole">
<organizationRefId>o:11</organizationRefId>
<roleRefId>r:34</roleRefId>
</OrganizationMappedByRole>
</parentOrganizations>
<priorities type="PriorityMappedByRole">
<PriorityMappedByRole type="PriorityMappedByRole">
<priority>0</priority>
<roleRefId>r:10</roleRefId>
</PriorityMappedByRole>
<PriorityMappedByRole type="PriorityMappedByRole">
<priority>1</priority>
<roleRefId>r:34</roleRefId>
</PriorityMappedByRole>
</priorities>
</Person>
</Persons>
```

表 3.2: Person.xml の要素

Persons	この要素以下に Person の情報があることを意味する
Person	人の情報がこれ以下にあることを示す。uniqueId が割り振られている
accontId	その Person のアカウント Id
lastName	苗字
name	フルネーム
nameReading	名前のふりがな
roleRefIds	この要素以下にその人が保持する役割を記述する
roleRefId	役割の Id を記述する
parentOrganizations	この要素以下にその人が所属している組織の Id を記述する
OrganizationMappedByRole	この要素以下に組織と、その組織の役割を記述する
organizationRefId	所属している組織の Id
priorities	人物に割り振られている役割の優先順位を以下に記述する
PriorityMappedByRole	この要素以下に役割と優先順位をペアで記述する
priority	役割の優先順位を記述する

Person.xml を例で上げたが、役職 Tree や役割 Tree も同じような構造でデータを保持している。

3.3.3 Jungle 上での maTrix のデータ構造の表現

本節では、前節で述べた、maTrix のデータ構造をどのように Jungle で表現するかを記述する。人物 Tree や役職 Tree は、木構造のデータであるためそのまま Jungle に格納することができる。実際に人物 Tree を格納した際のデータの形 (図 3.6) を以下に示す。

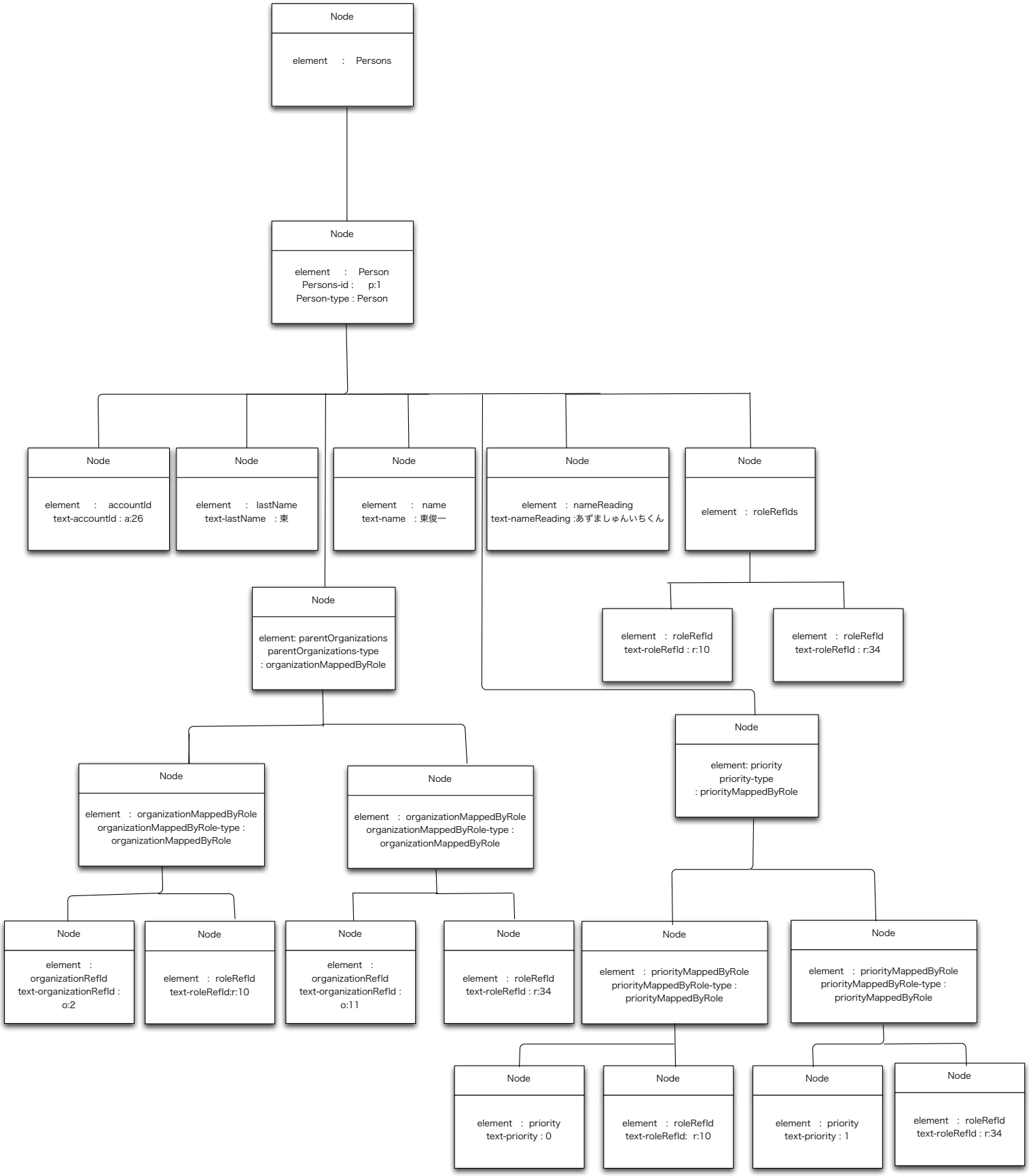

図 3.6: Jungle 上での人物 Tree 表現例

Jungle 上での、構成情報モデルの表現は、構成情報モデル Tree を作成し管理する (図 3.7)。

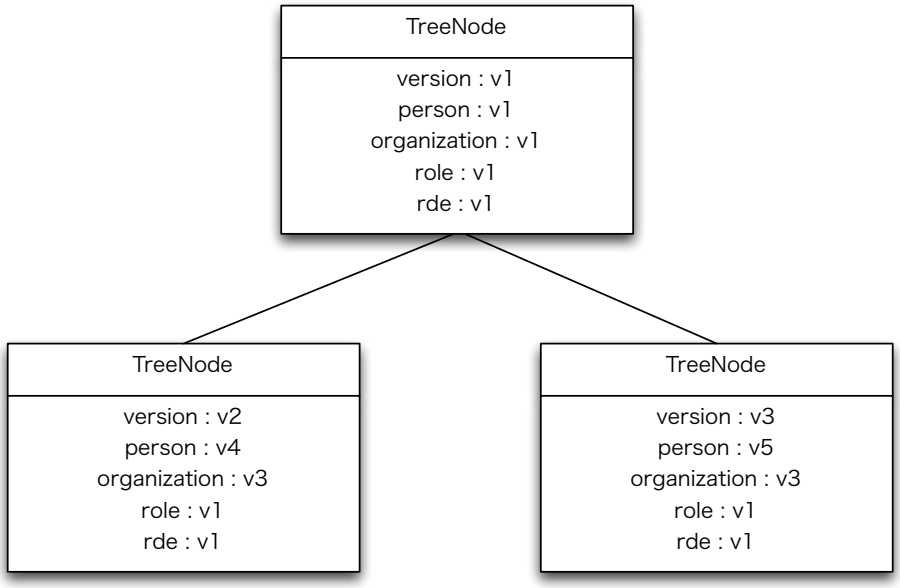

図 3.7: 構成情報モデル Tree 表現例

表 3.3: 構成情報 Tree の TreeNode が保持している Attribute

version	構成情報の version
person	構成情報の version に対応する人物 Tree の version
organization	構成情報の version に対応する組織 Tree の version
role	構成情報の version に対応する役割 Tree の version
rde	構成情報の version に対応する役割記述要素 Tree の version

実際にどのように Jungle 上で過去の構成情報モデルにアクセスするか、例題を用いて説明する。構成情報モデル version:3 に対応する人物 Tree にアクセスする手順を以下に示す。

1. 構成情報 Tree からアクセスしたい version 情報を保持している Node を取得する (今回の例題では version3)
2. 取得した TreeNode には、構成情報:version:3 に対応した人物 Tree などの version が記述されている (図 3.7) ので、人物 Tree の verson(v:5) を取得する
3. ２で取得した version の人物 Tree にアクセスする。

といった手順で Jungle では maTrix の構成情報モデルを表現する。以下に Jungle 上での構成情報モデル表現の図 (3.8) を記す。

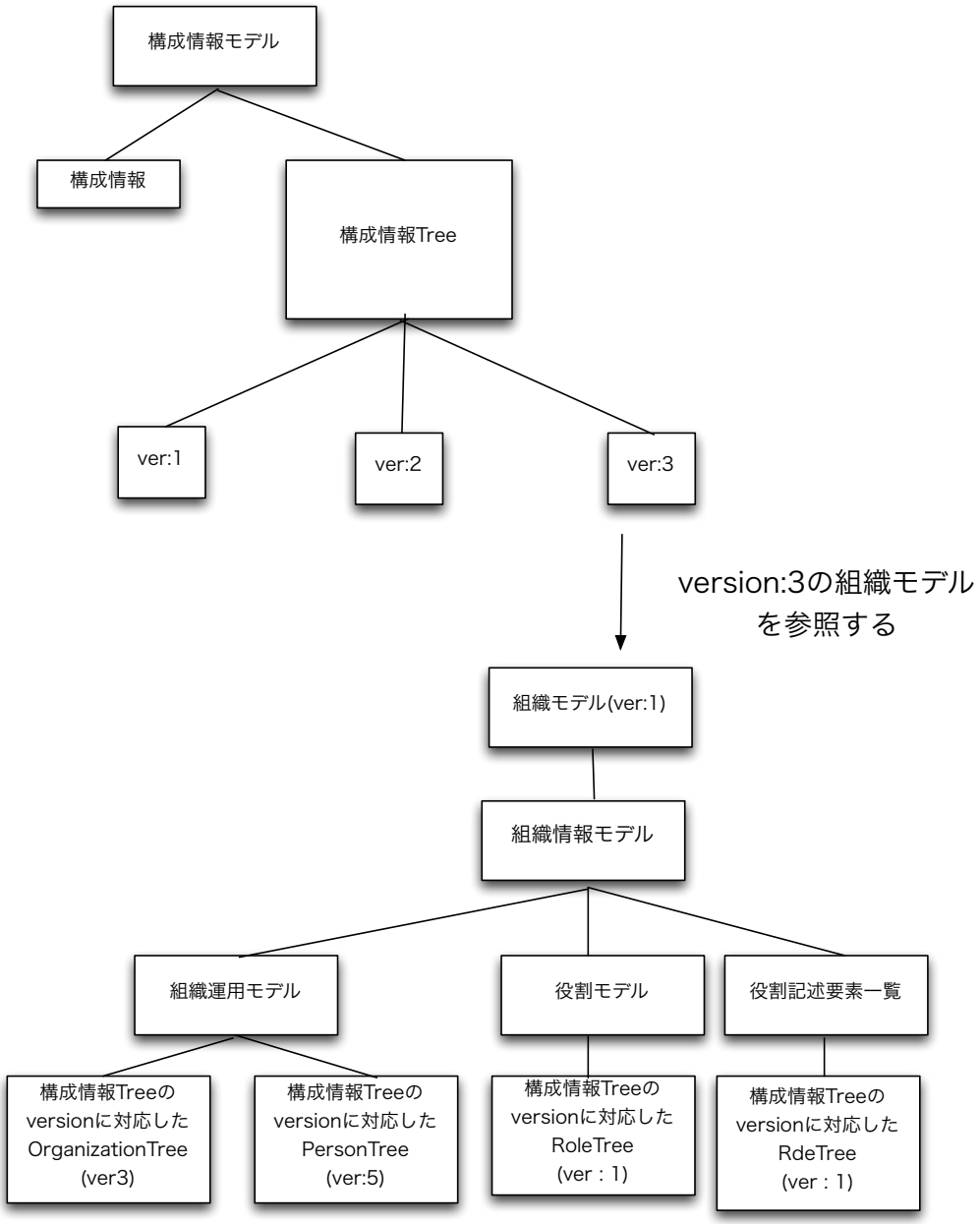

図 3.8: 構成情報モデル Tree 表現例 2

3.3.4 XACML

XACML では、データに関するアクセス要求について、その要求元の情報と要求の内容、アクセス対象の組み合わせから、そのアクセス要求が許可されるか否認されるかを判断するためのルールを記述できる。実際に使用する際は、XACMl 自体がアクセス制御を行うのではなく、アクセス管理アプリケーションが XACML を参照しアクセス制御を行う。XACML は以下の様なデータ構造を持つ (図 3.9)

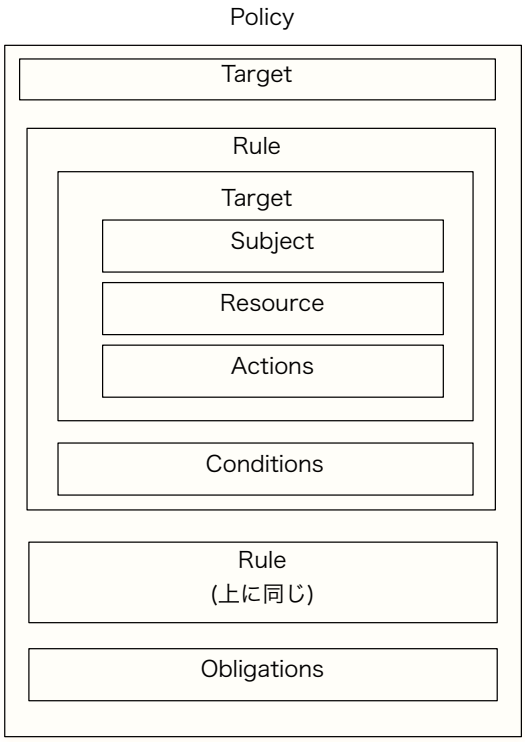

図 3.9: XACML のデータ構造

Target は 3 つの要素を持ち、誰に対して (Subject)、どのような資源を (Resource)、どのように扱うか (Action) と分類され、それぞれの要素で、評価関数を用いて評価を行う。評価の結果は、Match、No-Match、Indeterminate(評価不能) の 3 つである。

Rule は、Target に対する規則を定めるもので、ルールにそれぞれ適合した場合に決定する値 (Permit、Deny) を、Effect として設定しておく。また、それとは別にオプションとして Conditions を付けることも可能である。ルールの評価は、ルール結合アルゴリズム (表 3.4) にそって結合する。

表 3.4: Rule の評価

Target	Condition	評価
Match	true	Efferct(Permit or Deny)
Match	false	NotApplicable
Match	Indeteminate	Indeteminate
No-Macth	-	NotApplicate
Indeterminate	-	Indeteminate

Policy

複数の Rule をまとめたものを Policy という。Policy の子要素には、Target、Rule、Obligations(責務) がある。もしも、Policy に Obligations が規定された場合は、たとえ、ルール結合後の結果が permit であったとしても、Obiligations に記述されていることを同時に実施することが出来なかった場合、承認を拒否する必要がある。また Policy が複数の Rule を評価するときは、ルール結合アルゴリズム (表 3.5) を用いる。

表 3.5: XACML のルール結合アルゴリズム

Deny-overrides	どれか 1 つのルールの Effect が Deny であれば結果は Deny とする
Permit-overrides	どれか 1 つのルールの Effect がが Permit であれば結果は Permit とする
First-applicable	ポリシー内のすべてのルールについて順番に評価して、対象がマッチした場合、対象の評価結果を Match とし、次に Condition を評価し、これが True なら結果は Effect で指定された Permit または Deny とする。

maTrix では、許認可を行う際に使用する PolicyFile を XACML で記述している。

xacml の記述例

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
PolicyId="urn:srl-oasis:names:matrix:tank:aso-sample:01-01:policy"
RuleCombiningAlgId = ここに XACML のルール結合アルゴリズムを記述する‘
<Target>
//誰が、どこに、何を出来るかのルールを記述する
</Target>
<Rule
RuleId="urn:srl-oasis:names:matrix:tank:aso-sample:01-01-01:rule"
Effect="Deny">
<Target>
<Subjects>
<Subject>//操作できる人の条件を記述</Subject>
</Subjects>
<Redources>
<Redource>//どのリソースに対してアクセスできるかを記述</Redource>
</Redources>
<Actions>
<Action>//どんな操作を許可するかを記述</Action>
</Actions>
</Target>
<Condition>
ターゲット意外に対する規則があればここに記述する
</Condition>
</Rule>
<Rule>
//省略
</Rule>
</Policy>
```

3.3.5 Jungle 上での maTrix の許認可

Jungle 上で maTrix の許認可を行う際は、XACML を読み込み、各 Tree に対して検索を行いその結果から許認可を行う。

例えば、A さんが、管理者しか閲覧出来ない資料を閲覧しようとした場合は

1. 人物 Tree から、A さんのデータを取得し、A さんに割り振られている役割の ID を取得する。
2. 役割 Tree から、1 で取得した役割 ID のデータを取得し、その役割 ID に割り振られている役割記述要素 ID を取得する
3. 役割記述要素 Tree から、2 で取得した役割記述要素 ID のデータを取得し、役割記述要素名を取得する
4. 3 で取得した役割記述要素名が、管理者かどうかを調べ、一致した場合はアクセスを許可し、一致しなかった場合はアクセスを拒否する。

といった流れになる。

3.4 XACMLInterpreter

XMLReader と同じよう sax にを用いて実装している。XACMLInterpreter は、引数に、使用する policyFile 名、どの Resource にアクセスするか、どんな処理を行うか (action)、許認可を求める人の UserID 等を与える。

XAMLInterpreter で Handler が使用しているイベントは、XMLReader と同じで、startElement、charactor、endElement、endDocument の 4 つを使用している。

startElement では、主に評価関数や、評価関数の引数等を取得する。また評価関数が入れ子になっていることがあるため、functionStack と attributeStack を用いて評価関数と引数を関連付けている。3.10

charactor では、評価関数の引数を取得し、AttributeStack に push する

endElement では。評価関数の実行を行う。以下に入れ子になっている評価関数例と、その時の処理の流れを記述する。

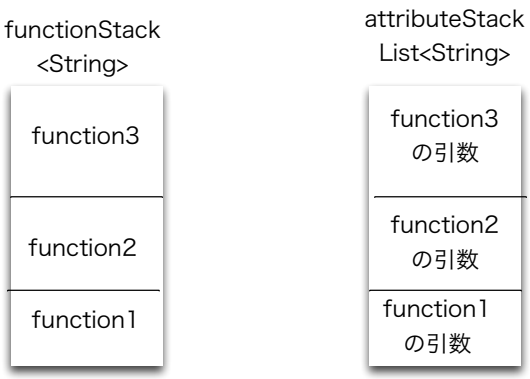

図 3.10: XACML のデータ構造

入れ子になっている関数の例

```
<Apply FunctionId=評価関数 1>
 <Apply FunctionId=評価関数 2>
  <ResourceAttributeDesignator
  AttributeId=評価関数 2 の引数
  DataType=評価関数 2 の引数のデータ型>
 </Apply>
 <SubjectAttributeDesignator
 AttributeId=評価関数 1
 DataType=評価関数 1 のデータ型>
</Apply>
```

1. <Apply> を読み込んだ際に、functionStack に評価関数 1 を push する
2. 新しい評価関数 1 に対応する引数 (List¡String¿) を attributeStack に push する。
3. ２つ目の <Apply> を読み込んだ際に、functionStack に評価関数 2 を push する
4. 新しい評価関数 2 に対応する引数 (List¡String¿) を attributeStack に push する。
5. <ResourceAttributeDesignator> を読み込んだ際に、attributeList から評価関数 2 に対応する attirbuteList を取得する。
6. attributeList に評価関数 2 の引数を add し、attirbuteStack に push する。
7. </Apply> を読み込んだ際に、attributeStack から評価関数 2 の引数を取得する
8. 7 で取得した引数を用いて functionStack から取得した評価関数 2 を実行する
9. attributeStack から評価関数 1 の attributeList を取得し、評価関数 2 の結果を add する
10. 評価関数 1 の attribtueList を attributeStack に push する
11. <SubjectAttributeDesignator> を読み込んだ際に、attributeStack から評価関数 1 の attributeList を取得し、評価関数 1 の引数を add する。
12. </Apply> を読み込んだ際に、attributeStack から評価関数 1 の引数を取得する
13. 12 で取得した引数を用いて functionStack から取得した評価関数 1 を実行する

endDocument では、これまでに実行した評価関数等の結果から今回の許認可を判断する。

第4章　Jungle上でmaTrixを構築するのに必要なAPIの設計

4.1　maTrixのデータ構造の表現に必要なAPI

4.2　Jungle上でのmaTrixの許認可に必要なAPI

第5章　Jungle上でmaTrixを構築するのに必要なAPIの実装

5.1　過去のTreeに対するアクセス

5.2　Treeに対する検索

5.3　Index

第6章　FunctionalJava

6.1　FunctionalJava とは

6.2　Index で TreeMap を使用するメリット

6.3　TreeMap のバグ

第7章　実装の評価

7.1　検索のAPIの測定

7.2　Indexの作成時間

7.3　Node数とTreeの構築時間の測定

7.4　transactionPerSecond

7.5　FunctionalJavaのTreeMapのget

第8章　結論

8.1　まとめ

8.2　今後の課題

8.2.1　push/pop

8.2.2　index の IncrementalUpdate

8.2.3　differencialList

8.2.4　exponential backoff

参考文献

[1] hoge

謝辞

本研究の遂行，また本論文の作成にあたり、御多忙にも関わらず終始懇切なる御指導と御教授を賜わりました hoge 助教授に深く感謝したします。

また、本研究の遂行及び本論文の作成にあたり、日頃より終始懇切なる御教授と御指導を賜わりました hoge 教授に心より深く感謝致します。

数々の貴重な御助言と細かな御配慮を戴いた hoge 研究室の hoge 氏に深く感謝致します。

また一年間共に研究を行い、暖かな気遣いと励ましをもって支えてくれた hoge 研究室の hoge 君、hoge 君、hoge さん並びに hoge 研究室の hoge、hoge 君、hoge 君、hoge 君、hoge 君に感謝致します。

最後に、有意義な時間を共に過ごした情報工学科の学友、並びに物心両面で支えてくれた両親に深く感謝致します。

2010 年 3 月

hoge