適応型分散オブジェクト指向環境における柔軟な環境適応の実現

金澤 康祐[†1]　福田 哲也[†1]　Naldy Nirmanto Tjondronegoro[†1]　吉田 隆一[†2]

概要：分散システムでは，負荷の集中などの局所的かつ一時的な実行環境変化が発生した場合に全体の性能や信頼性が低下する．これに対処するため，我々はシステムを構成する内部モジュールを環境変化に応じて入れ替えることで適応性を実現する再構成可能なシステム Juice を提案している．
我々の先行研究ではこの内部モジュールの構成には制限があり，十分な適応性が得られなかった．そこで，本研究では，環境適応の柔軟性を高めることを目的とする．利用者は，環境変化に対するシステムへの適応の指示を適応ポリシーと呼ばれるファイルに記述するが，柔軟性を高めるため，様々な機能をもった内部モジュールの指定とそれらの間の接続関係の記述法を提案し，Juice システムで利用できるようにした．

キーワード：自律型システム，適応性，再構成，分散ミドルウェア

Flexible Adaptation for an Adaptive Distributed Object-Oriented Environment

Kanazawa Kousuke[†1]　Fukuda Tetsuya[†1]　Naldy Nirmanto Tjondronegoro[†1]
Yoshida Takaichi[†2]

Abstract:
When a temporal and partial run-time environment change in a distributed system such as unbalance of load or network congestion occurs, total performance or reliability goes down. We have proposed the Juice system which can adapt the system to environmental change automatically. Juice system exchange internal objects to adapt itself to environmental change.
In our previous work, we did not have enough adaptability of the system, because configuration of internal objects were restricted. The purpose of this paper is to enhance flexibility of environmental adaptation of the system. In the Juice system, users instruct the system how to adapt itself to environmental change, which is called an adaptation policy described in a file. We expand the adaptation policy in order that the users can describe various configuration of the internal objects and their interconnection.

Keywords: Autonomous system, Adaptabilityt, Reconfiguration, Distributed object-oriented middleware

1. はじめに

近年，複数の計算機がネットワークを介して処理を行う分散システムが普及している．分散システムでは局所的かつ一時的な実行環境の変化，例えば負荷の集中や計算機の故障が発生した場合，全体の性能や安全性，信頼性が低下する．そのためこれらの実行環境変化に対して対処する必要がある．これらの実行環境変化に対する人手での対処は多大な時間と人件費を要するため自律的な対処が求められている [1]．そこで，我々はシステムが実行環境変化に自律的に対処する適応型分散オブジェクト指向環境 Juice[2], [3], [4] を提案している．本論文では Juice システムが実行環境変化に自律的に対処することを適応と呼ぶ．

†1 現在，九州工業大学大学院情報工学研究院情報創成工学専攻
Presently with Kyushu Institute of Technology
†2 現在，九州工業大学大学院情報工学研究院情報創成工学研究系
Presently with Kyushu Institute of Technology

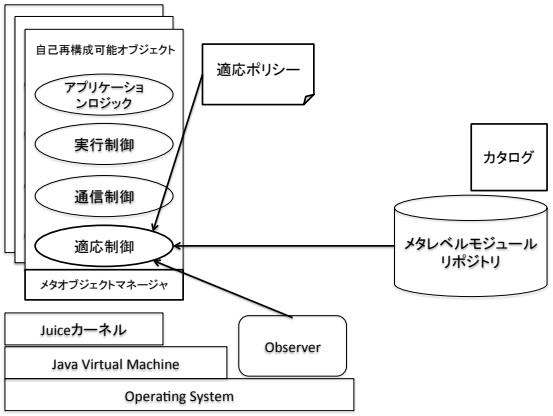

図 1 Juice システムの全体像

Juice システムでは，自己再構成可能オブジェクトと呼ぶオブジェクトを複数の内部オブジェクトによって構成している．この自己再構成可能オブジェクト内のオブジェクトをメタオブジェクトと呼びメタオブジェクトを実行環境に応じたものに入れ替えることにより実行環境変化に対処する．

システムが自律的に実行環境変化に対処するためにはシステム管理者などが環境変化に対する構成の情報を与える必要がある．この情報を記述したファイルを適応ポリシーと呼ぶ．適応ポリシーには通信方法が信頼できず情報漏洩の危険性がある場合，安全な通信を行うなどの情報を記述する．システムが現在の実行環境は情報漏洩の危険性があると判断した場合 Juice システムは通常の通信を行うメタオブジェクトを通信内容を暗号化するメタオブジェクトに入れ替えることで環境変化に対処する．

Juice システムは Java 言語を用いて実装されている．Java 言語を用いて実装されたプログラムは高い可搬性をもつ上広く普及しているオブジェクト指向言語であるため採用した．

本論文では従来の Juice システム [5] が対処できなかった実行環境変化への対処を可能にし，対処可能な実行環境変化を増加させる．これにより柔軟な環境適応が可能になる．

本論文は次のように構成される．第 2 章では適応型分散オブジェクト指向環境 Juice の詳細について述べる．第 3 章では現在の Juice システムの問題点について述べ，第 4 章では課題と解決方法について述べる．最後の第 5 章で本論文のまとめを述べる．

2. Juice システムの全体像

Juice システムの全体像を図 1 に示す．

2.1 自己再構成可能オブジェクト

アプリケーションは複数の自己再構成可能オブジェクトから成る．その自己再構成可能オブジェクトは複数のメタオブジェクトと呼ぶ内部オブジェクトから構成されている．これらメタオブジェクトは自己再構成可能オブジェクト間の通信制御や自己再構成可能オブジェクト自体の実行制御などの機能を果たす．Juice システムは自己再構成可能オブジェクト内のメタオブジェクトを入れ替えることにより実行環境変化に対処する．メタオブジェクトの入れ替えを行い実行環境に適応することを再構成と呼ぶ．この再構成の例として通信の実効帯域幅が減少した際に通常の通信を行うメタオブジェクトから通信内容の圧縮を行うメタオブジェクトに入れ替えるというものが挙げられる．

2.1.1 メタオブジェクト

自己再構成可能オブジェクトは以下の個別の役割を持つ複数のメタオブジェクトより構成される．ここで役割とは自己再構成可能オブジェクト内のメタオブジェクトを分類するための最も粒度の大きい区分のことである．

アプリケーションロジック アプリケーションロジックはアプリケーションプログラマが作成するオブジェクトである．通信の安全性などの非機能要求を他のメタオブジェクトが担うことによりアプリケーションプログラマは機能要求の実装のみに集中することが可能になる．またアプリケーションプログラムを自己再構成可能オブジェクト内のメタオブジェクトとして機能させるためには提供されているトランスレータを使用することで可能になるため，アプリケーションプログラマは Juice システムについての知識や意識が必要ない．

実行制御 実行制御メタオブジェクトはアプリケーションロジックの実行を制御する．具体例として排他制御を行い並行実行制御を提供するものなどが挙げられる．この実行制御メタオブジェクトを入れ替えるメリットの一例としてアプリケーションや実行環境によってサスペンドロックとスピンロックを入れ替えることが挙げられる．

通信制御 通信制御メタオブジェクトは外部の自己再構成可能オブジェクトとの通信を行う．具体例として他の自己再構成可能オブジェクトに送信するメッセージを暗号化するものなどが挙げられる．通信制御メタオブジェクトを入れ替えるメリットとしてはアプリケーションや実行環境毎に必要となる暗号強度は異なるため，この通信制御メタオブジェクトの入れ替えで対処可能になることが挙げられる．

適応制御 適応制御メタオブジェクトは，現在の計算機の実行環境情報を収集すること，現在の実行環境を同定すること，その実行環境に適応するために必要なメタオブジェクトの情報を決定することの 3 つの機能を果たす．適応制御メタオブジェクトは適応ポリシーを保持しており，この適応ポリシーをもとに現在の実行環境の同定，必要なメタオブジェクトの決定を行う．こ

の適応ポリシーについては 2.2 で詳細に述べる．

2.1.2 メタオブジェクト間通信

メタオブジェクトは互いに通信を行いながら動作する．Juice システムではこのメタオブジェクト間の通信をメッセージ通信により実現している．このメッセージの送受信を相手の参照を保持しメタオブジェクト間で直接行うとメタオブジェクト間の関係が密になりメタオブジェクトの入れ替えが困難になる．そこで Juice システムではマイクロポートという機構を用いている．マイクロポートはメタオブジェクト毎に存在し，他のメタオブジェクトにメッセージを送信する際には相手のマイクロポートに送信し，メッセージを受信する際には自分のマイクロポートからメッセージを取り出す．このようにマイクロポートを用いて間接的にメッセージを送受信することでメタオブジェクト間の関係を疎にし再構成を可能にしている．

2.1.3 メタオブジェクトマネージャ

各自己再構成可能オブジェクトにメタオブジェクトマネージャを置く．メタオブジェクトマネージャはメタオブジェクト間の通信，メタオブジェクトの入れ替えなどを行う．メタオブジェクト間の通信をメタオブジェクトマネージャを介して行うことによりメタオブジェクト同士の関係が疎になりメタオブジェクトの入れ替えが可能になる．

2.2 適応ポリシー

適応ポリシーは計算機の実行環境変化に対してどのように対処を行うかを利用者が記述したファイルである．この適応ポリシーは以下の 3 つのファイルで構成されている．

適応戦略定義 適応戦略定義には実行環境の条件とその実行環境に対処するための自己再構成可能オブジェクト内のメタオブジェクトの構成を記述する．
例：通信方法が信頼できない場合セキュアな通信を行う

条件定義 条件定義は収集した計算機の実行環境情報から現在の計算機の状態を同定するための情報を記述する．
例：通信方法が信頼できないとは無線通信を行っている状態

構成定義 構成定義はメタオブジェクトの構成を特定するための構成名とその構成に必要なメタオブジェクトの情報を記述する．
例：セキュアな通信とは暗号化形式が RSA かつ暗号化 bit が 1024 以上

2.3 メタレベルモジュールリポジトリ

メタオブジェクトはメタオブジェクトプログラマが実装し，メタオブジェクトの実装をメタレベルモジュールと呼ぶ．メタレベルモジュールリポジトリはメタオブジェクトの実装コードを保存し複数の自己再構成可能オブジェクト間で共有する．またメタレベルモジュールリポジトリは保存しているメタレベルモジュールの中から実行環境変化に対処するための最適なものを検索するためにカタログを保持している．このカタログにはメタレベルモジュールリポジトリ内に保存されているメタレベルモジュールの満たす非機能要求が記述されている．

2.4 オブザーバ

オブザーバは計算機の CPU 使用率，メモリ使用率，通信の実効帯域幅などの実行環境の情報を取得し自己再構成可能オブジェクトに送信する．Java Virtual Machine(JVM) によって計算機の状態が隠蔽されるため外部プロセスであるオブザーバを介して計算機の実行環境情報を取得している．

2.5 Juice カーネル

JVM 毎に存在し自己再構成可能オブジェクトの生成，異なる JVM 上で動作する Juice システムとの通信を行う．

2.6 実行環境変化への対処の流れ

Juice システムが実行環境変化に対して自律的に対処する流れを次に示す．

1. 自己再構成可能オブジェクトはオブザーバから実行環境情報を取得する．
   自己再構成可能オブジェクト内の適応制御メタオブジェクトがこの動作を行う．
2. 適応制御メタオブジェクトは取得した実行環境と自己内に保持する適応ポリシーを照らし合わせ現在の実行環境を同定する．その後適応に必要なメタオブジェクトの構成を決定する．
3. 適応制御メタオブジェクトは必要なメタオブジェクト情報をクエリとしてメタレベルモジュールリポジトリに送信する．
4. メタレベルモジュールリポジトリはカタログを検索し，必要なメタレベルモジュールを特定し，自己再構成可能オブジェクトに送信する．
5. 自己再構成可能オブジェクト内のメタオブジェクトマネージャはメタレベルモジュールをインスタンス化させる．
6. メタオブジェクトマネージャはインスタンス化したメタオブジェクトに対してマイクロポートを対応付ける．

3. 現状の問題点

現状の Juice システムはプロトタイプとして実装を容易に行うために自己再構成可能オブジェクト内のメタオブジェクトについて 2.1.1 で述べた役割と役割毎のメタオブジェクトの生成数を 1 つに固定していた．そのため例えば役割としてデバックや動作のトレースを行うロガーを新た

に自己再構成可能オブジェクト内に生成することができない．また非機能要求は直交する機能を複数求められることも考えられる．例えばグループ通信と通信内容の暗号化の2つが求められる場合が挙げられる．この機能は通信制御メタオブジェクトが担っているため現状ではこの両者の要求を満たすことができない．

そこで2.1.1以外の役割をもつメタオブジェクトを自己再構成可能オブジェクト内に生成可能にすること，同じ役割を持つメタオブジェクトでも複数生成可能にすること，以上の2点を本論文の目的としている．これによりJuiceシステムが提供可能な非機能要求の幅が広くなる．例としてデバックという役割をもつメタオブジェクトをアプリケーションロジックと実行制御メタオブジェクトの間に生成し，アプリケーションロジックがどのように呼び出されているかを知ることが可能になる．またグループ通信を行う通信制御メタオブジェクトと通信内容の暗号化形式が異なる通信制御を組み合わせることで，宛先ごとに暗号化形式を変更して通信を行うことが可能になる．

4. 提案手法

4.1 課題

柔軟な環境適応を行うためには以下に示す課題が存在する．

ユーザーの記述面

適応ポリシーの記述内容 次の2点が必要になる．

(1) 新たな役割をもつメタオブジェクトの情報を記述可能にする．

(2) メタオブジェクト間の接続関係を記述可能にする．

(1) について，従来は自己再構成可能オブジェクト内に生成することができるメタオブジェクトの役割を限定していた．そのため適応ポリシーに記述する内容も限定的であった．しかし自己再構成可能オブジェクト内に生成可能なメタオブジェクトの役割制限を無くすことで，従来は存在しなかった役割をもつメタオブジェクトについても記述可能にする必要がある．

(2) について，従来はメタオブジェクト間の通信は役割で宛先を決定していた．しかし自己再構成可能メタオブジェクト内の役割と生成数の制限を無くしたことにより通信相手の指定が必要になる．そのためメタオブジェクトの接続関係を適応ポリシーに記述する．

適応ポリシーとカタログ間の一貫性 実行環境変化が発生し自己再構成可能オブジェクト内のメタオブジェクトの再構成が必要になった場合，適応ポリシーの記述内容からクエリを生成しカタログを検索する．しかし適応ポリシーはアプリケーション運用者，システム運用者などが記述し，カタログはメタオブジェクトプログラマが記述する．そのため2者の間で用語を統一する必要がある．

Juiceシステムの機能面

適応ポリシーに従ったメタオブジェクト間通信 メタオブジェクト間の通信は役割を宛先としていた．これを適応ポリシー内で指定された相手との通信が行えるようにする．

カタログの検索 適応ポリシーからカタログを検索するためのクエリ生成器を作成しカタログを検索できるようにする．

4.2 適応ポリシーの改良

適応ポリシーには宣言的な記述が必要になる．記述の容易さからYAML Ain't Markup Language(YAML)[6], JavaScript Object Notation(JSON)[7]などを候補としたが，普及度およびXQuery[8]などのツールの豊富さからExtensible Markup Language(XML)[9]を用いることにした．

様々な役割をもつメタオブジェクトの内容も適応ポリシーに記述可能にするためにあらかじめ適応ポリシーに記述する内容のテンプレートを決めておき，それらについての値を記述していくことにした．適応ポリシーは「適応戦略定義」「条件定義」「構成定義」「モジュール定義」の4要素から構成する．適応ポリシーを4要素に分けたのは動作する計算機やアプリケーションによって異なる記述が必要なためである．例として構成定義に記述する「安全な通信」という要求は動作する環境によって異なる．また名前付けをして状態を定義することにより異なる実行環境やアプリケーションでも再利用が可能になる．例えば高負荷の場合計算機内の処理を移送すると記述しておけば，計算機によって高負荷を示すCPUの使用率を変化させることが可能になる．

次に適応ポリシー記述を具体例も含めて説明する．

適応戦略定義

図2に適応戦略定義の例を示す．適応戦略定義には「条件名」と「構成名」の対を記述する．「条件名」とは計算機の状態を示したもので「構成名」は自己再構成可能オブジェクト内のメタオブジェクトの構成を示す．適応戦略定義は計算機の置かれている状態が「条件名」に当てはまる場合，自己再構成可能オブジェクト内のメタオブジェクトを「構成名」に記述された内容に再構成することで実行環境変化に対処する．以下に適応戦略定義内の要素についての意味を示す．

Condition 計算機がCondition要素に示された状態（実行環境）の場合，自己再構成可能オブジェクト内のメタオブジェクトを次に記述されるConfiguration要素の内容に入れ替える．この要素の記述内容は条件定義内のName要素と対応する．

```
計算機の状態を表す「条件名」とその条件に対応するための
メタオブジェクトの「構成名」を記述する.
<Policy>
   <Strategy>
      <Condition>UnreliableNetwork</Condition>
      <Configuration>
      SecureCommunication
      </Configuration>
   </Strategy>
</Policy>
```

図 2 適応戦略定義

```
「条件名」を定義するための論理式を「条件式」に記述する.
<Conditions>
   <Condition>
      <Name>UnreliableNetwork</Name>
      <Environment>NetworkType=0</Environment>
   </Condition>
</Conditions>
```

図 3 条件定義

Configuration 自己再構成可能オブジェクト内のメタオブジェクトの構成を示す．この要素の記述内容は構成定義内の Name 要素と対応する．

図 2 は信頼できない通信 (Unreliable Network) の場合は安全な通信 (SecureCommunication) を行う構成に入れ替えるという内容を記述している．

条件定義

条件定義には「条件名」と「条件式」の対を記述する．「条件式」に記述された論理式が真となる場合，現在計算機は「条件名」で記述された状態であると判断する．図 3 に条件定義の例を示す．以下に条件定義内の要素についての意味を示す．

Name 計算機の実行環境を表現するための名称を示す．この記述内容は適応戦略定義内の Condition 要素と対応する．

Environment ここに記述された条件式が真となる時，実行環境が Name 要素で名付けられた状態であると判断する．

図 3 は信頼できない通信 (Unreliable Network) とは実行環境が無線通信 (NetworkType=0) という内容を記述している．

構成定義

構成定義には「構成名」と「コンポーネント群」の対を記述する．自己再構成可能オブジェクト内のメタオブジェクトを「構成名」で示された構成に入れ替える際に「コンポーネント群」で指定されたメタオブジェクト群を使用する．図 4 に構成定義の例を示す．以下に構成定義内の要素についての意味を示す．

```
「構成名」に必要なメタオブジェクト群を
「コンポーネント群」に記述する.
<Configurations>
   <Configuration>
      <Name>SecureCommunication</Name>
      <Component>
         <Tree>
            <Module>FIFO</Module>
            <Tree>
               <Module>RSA1024</Module>
            </Tree>
         </Tree>
      </Component>
   </Configuration>
</Configurations>
```

図 4 構成定義

Name 構成名に付与する名前を記述する．適応戦略定義の Configuration 要素と対応する．

Component この要素内にメタオブジェクトの構成と接続関係を記述する．

Tree メタオブジェクトの接続関係は木構造であるのでその木構造を定義する．このメタオブジェクトの接続関係についての詳細は 4.5 で述べる．

Module 実行環境への適応に必要なモジュール名を示す．この記述内容はモジュール定義の ModuleName 要素と対応する．

図 4 は安全な通信 (SecureCommunication) を行うメタオブジェクトの構成は FIFO と RSA1024 のコンポーネントから構成されることを記述している．

モジュール定義

モジュール定義は「コンポーネント群」と「要求」の対を記述する．「コンポーネント群」がどのような動作を提供するかを「要求」に記述する．モジュール定義についての例を図 5 に示す．なお XML 文書内で大小記号（<および>）を使用することはできないが可読性のために図 5 では大小記号を使用している．以下にモジュール定義内の要素についての意味を示す．

RoleName 自己再構成可能オブジェクト内の役割名を示す．

ModuleName モジュールの名前を示す．この要素は構成定義の Module 要素と対応する．

Function モジュールを分類するための機能を記述する．カタログと一貫性を取るために使用する．

Requirement モジュールがメタレベルモジュールリポジトリに問い合わせるための情報群を記述する．

Quality メタレベルモジュールリポジトリに問い合わせる情報を記述する．この情報とはメタオブジェクトがどのように非機能要求を満たすかを示す．

```
「コンポーネント群」がどのような
「要求」を満たすかを記述する.
<ModuleDefinition>
   <Role>
      <RoleName>Communicator</RoleName>
      <MetaObjects>
         <MetaObject>
            <ModuleName>RSA1024</ModuleName>
            <Function>Encryption</Function>
            <Requirement>
               <Quality>EncryptionType=RSA</Quality>
               <Quality>EncryptionBit>1024</Quality>
            </Requirement>
         </MetaObject>
      </MetaObjects>
   </Role>
</ModuleDefinition>
```

図 5 モジュール定義

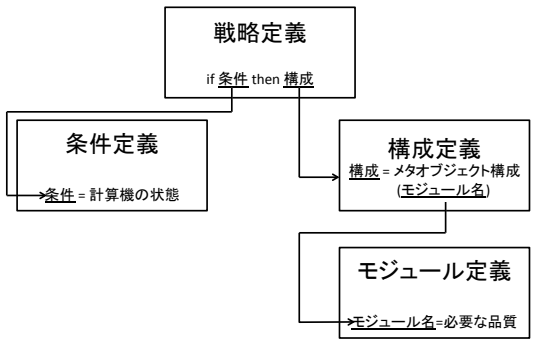

図 6 適応ポリシー間の関連

モジュール定義に記述した Function 要素はメタオブジェクトが果たす機能を示している．機能とは役割よりも粒度が小さい区分を示す．この機能毎に記述する Quality 要素を決める．例として機能が Encryption の場合 Quality 要素は EncryptionType 及び EncryptionBit を記述する．

図 5 は暗号化形式が RSA で暗号化ビット数が 1024 のメタオブジェクトに RSA1024 という名前を付け参照するための内容を記述している．

適応ポリシーの関連は図 6 のようになる．これらのファイルは互いの情報を参照して動作するためお互いの対応付けが必要になる．例えば適応戦略定義の「条件名」は条件定義に詳細が記述されている．これらの対応付けをとるために自己再構成可能オブジェクトごとに適応ポリシーの参照を保持する．この参照のために適応ポリシーに ID を付与する．この ID を参照することで自己再構成可能オブジェクトは自身に必要な適応ポリシーを判断する．

```
<Catalog>
   <Role>
      <RoleName>Communicator</RoleName>
      <MetaModules>
         <MetaModule>
            <FQCN>
         jp.ac.kyutech.ai.ylab.communicator.RSA1024
            </FQCN>
            <Function>Encryption</Function>
            <Quality>
               <Entry>
                  <Name>EncryptionType</Name>
                  <StringValue>RSA</Value>
               </Entry>
               <Entry>
                  <Name>EncryptionBit</Name>
                  <NumberValue>1024</Value>
               </Entry>
            </Quality>
            <TimeCost>10</TimeCost>
         </MetaModule>
      </MetaModules>
   </Role>
</Catalog>
```

図 7 メタレベルモジュールリポジトリのカタログ

4.3 メタレベルモジュールリポジトリのカタログの改良

カタログとはメタレベルモジュールリポジトリ内に存在する適切なメタオブジェクトを特定するために使用するファイルである．記述言語は適応ポリシーと同じく XML を使用している．図 7 にカタログの具体例を示す．次にカタログ内の記述内容について説明する．

RoleName モジュールの役割名を記述する．

FQCN Fully Qualified Class Name の略称で唯一名を記述する．メタレベルモジュールリポジトリはこの FQCN をもとにメタモジュールを特定する．

Function メタレベルモジュールを分類するための機能名を記述する．適応ポリシーとの一貫性を取るために使用する．

Quality メタモジュールが果たす機能の品質を示す．

Entry 品質名と品質値の対を記述する．品質とはメタレベルモジュールが果たすことができる非機能要求を示す．

Name 品質名を記述する．

StringValue/NumberValue 品質値が文字列の場合は StringValue を品質値が数値の場合は NumberValue を用いる．

図 7 は FQCN が jp.ac.kyutech.ai.ylab.communicator.RSA1024 のメタレベルモジュールは暗号化形式 (EncryptionType) が RSA かつ暗号化ビット数 (EncryptionBit) が

1024 である暗号化に用いる通信制御メタオブジェクトであることを記述している．

4.4 適応ポリシーとカタログ間の一貫性

モジュール定義およびカタログ内の Function 要素のことを機能と呼びモジュール定義とカタログ間の一貫性をとるために使用する．適応ポリシーはシステム管理者やアプリケーション運用者が記述するが，カタログはメタレベルモジュールプログラマが記述する．実行環境変化への対処に必要なメタオブジェクトは，適応ポリシーをもとにクエリを作成しカタログを検索することで特定する．そのため適応ポリシーとカタログ間で用語の統一を取る必要がある．そこで新たに追加した機能要素を用いる．この機能要素に記述するべき品質名および品質値を指定する．これにより両者の間で一貫性を取ることが可能になる．

次に適応ポリシーおよびカタログの記述に使用する用語の統一範囲について述べる．この使用する用語とは適応ポリシー，カタログの記述に使用する EncryptionType などを示す．メタオブジェクトはサードパーティが追加することも可能である．そのため全てのメタオブジェクトに対して適応ポリシーおよびカタログ記述に使用する用語を統一することは不可能である．そこで同一アプリケーションを共有するコミュニティや開発組織毎に使用する用語を統一する．

4.5 メタオブジェクトの接続関係

4.2 の構成定義で述べたメタオブジェクトの接続関係について詳細に説明する．メタオブジェクトの接続関係は木構造で表現できる．この自己再構成可能オブジェクト内のメタオブジェクト間の接続関係は適応ポリシーに記述する．自己再構成可能オブジェクトはアプリケーションロジックを実行するために存在し，その他のメタオブジェクトはアプリケーションロジックの実行を支援するために存在する．そこで適応ポリシー内に記述する木構造はアプリケーションロジックを根として記述する．この木構造の例を図で表すと図 8 のようになる．

4.6 カタログの検索

メタレベルモジュールリポジトリはカタログを検索することで適切なメタレベルモジュールを特定する．この検索を行うためのクエリとして XPath[10] も候補としたが XQuery を使用することにした．その理由は XQuery は検索結果の並び替えなど XPath と比較して豊富な操作を行うことが可能なためである．この操作を用いる例としてカタログの検索結果として得られたメタレベルモジュール候補の優先順を並び替えることが挙げられる．一般にカタログの検索結果からメタオブジェクトの候補は複数存在する．これらの候補の並び替えの順序を変えることで品質重視，実行コスト重視などメタレベルモジュールの選択の幅が広くなる．そのため単純な操作のみを行うことが可能な XPath ではなく豊富な操作を行うことができる XQuery が最適だと判断した．

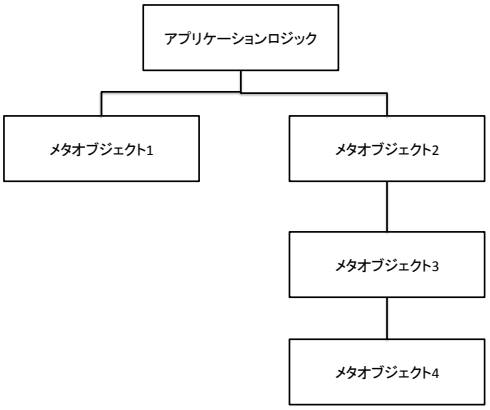

図 8 メタオブジェクト接続の木構造

4.7 メタオブジェクトのキャッシュ

メタオブジェクトの再構成を行う際にメタレベルモジュールリポジトリに問い合わせることは外部通信を含むため多大な時間と通信の帯域幅が必要となる．また Juice システムがメタオブジェクトの再構成を行う上で以前使用したメタオブジェクトが必要になることがある．このような場合に備えて適応制御メタオブジェクトはメタオブジェクトのキャッシュを保持する．このキャッシュはキーとしてモジュール定義内の ModuleName 要素を，値としてメタオブジェクトのバイトコードを保持する．実際の Juice システムの動作では例えば早朝に処理要求が集中しその時間のみ負荷分散が必要となるがその他の時間は負荷分散が必要ないというような事も考えられる．このような場合，キャッシュを利用することで Juice システムは外部通信を行うことなく素早く実行環境変化に対処可能になる上メタレベルモジュールリポジトリの負荷軽減にもつながる．

5. おわりに

本論文では実行環境変化に自律的に対処する分散オブジェクト指向環境 Juice について述べた．Juice システムはメタオブジェクトと呼ぶ複数の内部オブジェクトから構成される自己再構成可能オブジェクトから成る．このメタオブジェクトを入れ替えることで実行環境変化に対処する．従来の Juice システムでは実装の容易化のために自己再構成可能オブジェクト内のメタオブジェクトについて役割とメタオブジェクトの生成数を固定していた．そのため対処可能な実行環境変化が制限されていた．そこで自己再構成可能オブジェクト内のメタオブジェクトに対する役割と生成数に関する制限をなくした．本論文はそのために適

応ポリシーとメタレベルモジュールのカタログの改良に取り組んだ．適応ポリシーとは Juice システムが実行環境変化に対処する際に使用するファイルである．またメタレベルモジュールのカタログとは入れ替えに必要なメタレベルモジュールを検索するために使用する．

自己再構成可能オブジェクト内のメタオブジェクトに関する制限を無くすためには次の課題が存在していた．

（1）適応ポリシーの改良．
（2）適応ポリシーとカタログ間の一貫性．
（3）適応ポリシーに従ったメタオブジェクト間通信．

（1）については従来存在しなかった役割をもつメタオブジェクトについても適応ポリシーに記述可能にするために記述する項目をテンプレート化した．またメタオブジェクトの接続関係についても適応ポリシーに記述可能にした．(2) については適応ポリシーとメタレベルモジュールのカタログに共通する「機能」という項目を追加した．この「機能」項目毎に記述しなければならない項目が定められておりこの機能項目により適応ポリシーとカタログ間で記述内容を統一する．この新たな適応ポリシーとメタレベルモジュールのカタログについて Juice システムで使用可能にした．

今後は (3) について取り組む．これは適応ポリシーに記述したメタオブジェクトの接続関係をもとに自己再構成可能オブジェクト内のメタオブジェクトを通信可能にする．また適応ポリシーをもとにメタレベルモジュールのカタログを検索するがその検索のためのクエリ生成器を作成する．

参考文献

[1] Jeffry O. Kephart, David M. Chess, The Vision of Autonomic Computing, Computer 36, p.p41-50, 2003
[2] 小田謙太郎, 適応型分散オブジェクト指向計算モデルとその応用に関する研究, 九州工業大学博士学位論文, 2008
[3] 加藤建士，小田謙太郎，吉田隆一，適応型分散オブジェクト指向計算環境の実現，情報処理学会論文誌 44 巻 1 号 pp.39〜47, 2003
[4] Kentaro Oda, Shinobu Izumi, Yoshihiro Yasutake, Takaichi Yoshida, A Simple Reconfigurable Object Model for a Ubiquitous Computing Environment, International Journal of Computer Science and Network Security, Vol.7, No.5, 2007
[5] Hiromu Saito, Naldy Nirmanto Tjondronegoro, Kyota Hashimoto, Takaichi Yoshida, A Mechanism for Self-Reconfigurable Objects for an Adaptive Distributed Object-Oriented Environment, International Workshop on ICT, 2013
[6] The Official YAML Web Site, http://yaml.org/, (2017 年 1 月 18 日アクセス)
[7] JSON, http://www.json.org/, (2017 年 1 月 18 日アクセス)
[8] Priscilla Walmsley, XQuery Search Across a Variety of XML Data, O'Reilly Media, Inc., 2015
[9] Extensible Markup Language (XML) 1.0 (Fifth Edition) https://www.w3.org/TR/REC-xml/, (2017 年 1 月 18 日アクセス)
[10] XML Path Language(XPath), https://www.w3.org/TR/xpath/, (2017 年 1 月 18 日アクセス)