ポリシに基づくサービス・コミュニティのための形式モデルの研究

來間　啓伸

博士（学術）

総合研究大学院大学

複合科学研究科

情報学専攻

平成１７年度

(2005)

2006 年 3 月

本論文は総合研究大学院大学複合科学研究科情報学専攻に
博士（学術）授与の要件として提出した博士論文である。

審査委員：

中島　震（主査）	国立情報学研究所／総合研究大学院大学
本位田　真一	国立情報学研究所／東京大学
細部　博史	国立情報学研究所／総合研究大学院大学
佐藤　健	国立情報学研究所／総合研究大学院大学
武田　英明	国立情報学研究所／総合研究大学院大学
渡部　卓雄	東京工業大学

（主査以外はアルファベット順）

A Study on Formal Models for Policy Based Service Community on the Web

Hironobu Kuruma

DOCTOR OF

PHILOSOPHY

Department of Informatics
School of Multidisciplinary Sciences
The Graduate University for Advanced Studies (SOKENDAI)

March 2006

A dissertation submitted to
the Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies (SOKENDAI)
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Advisory Committee:

Shin Nakajima (Chair)	National Institute of Informatics/ The Graduate University for Advanced Studies
Shinichi Honiden	National Institute of Informatics/ Tokyo University
Hiroshi Hosobe	National Institute of Informatics/ The Graduate University for Advanced Studies
Ken Satoh	National Institute of Informatics/ The Graduate University for Advanced Studies
Hideaki Takeda	National Institute of Informatics/ The Graduate University for Advanced Studies
Takuo Watanabe	Tokyo Institute of Technology

(Alphabet order of last name except chair)

内容梗概

World Wide Web を通じて提供／利用／仲介される Web サービスは，急速に変化する社会環境に適応できる柔軟な情報システムを構築するための，基本的な基盤と考えられている．例えば，旅行代理店システムでは，Web サービスとして提供されているホテル予約機能を呼び出して利用することで，ホテルの予約をユーザに仲介することができる．このように構成されたシステムは，呼び出す Web サービスを変更することでホテルの追加や削除に対応できる，部屋の提供条件の変更などに Web サービスを提供する側で対応できる，などの特徴を持つ．また，このシステムが旅行代理店機能を Web サービスとして提供することによって，他のシステムに構成要素として組み込まれることも可能である．しかしながら，システムが複雑になるにつれて全ての構成要素を把握することが困難になり，ホテル予約サービスが当初の設計意図とは異なる使われ方をされる場合も起こり得る．このようにして構成されるシステムは，異種かつ自律的な要素から成りシステムの構成が動的に変化するオープンなシステムであり，堅牢なシステムを開発するための技術は未だ研究途上にある．特に，システムの構成要素となる Web サービスやユーザ（以下ではまとめてプレイヤと呼ぶ）を他のプレイヤの不正なアクセスから守るアクセス制御は，不正利用や情報漏えいを防ぐために不可欠であるが，システムの柔軟性を損なうことのないアクセス制御ポリシをどのように設計し，それを実現する仕組みをどのように作るかは，課題のまま残されている．

従来から，多くのアクセス制御モデルが提案されているが，オペレーティング・システムなどにおける計算資源へのアクセス制御を起源とするため，中央集約的な管理機構を持たないオープンなシステムには適用できない．また，実社会の構造に適合するアクセス制御モデルとして広く知られ標準化されている Role Based Access Control モデル（RBAC モデル）でも，任意の時点で全てのプレイヤを一意に識別できることが前提となる．さらに RBAC モデルでは，RBAC によって制御されたシステムを別の RBAC によって制御されたシステムの一部として取り込む場合のアクセス制御を，プレイヤの一意性を保つためにシステム全体について RBAC を構成し直すことなしには取り扱うことができない．このように，オープンなシステムにおけるアクセス制御の問題は未解決である．

本研究では，上記の問題を解決するために，オープンなシステムのためのアクセス制御モデルとして COAC（Community Oriented Access Control）モデルを提案する．COAC モデルでは，システムに関わるプレイヤの集まりをコミュニティと呼び，コミュニティに対してアクセス制御ポリシを設定する．より詳細には，コミュニティは，システムの中で担う役割（以下ではパートと呼ぶ）毎に集めたプレイヤの集まり（プレイヤの集まりの集まり）である．コミュニティのアクセス制御ポリシは，各々のパートに属するプレイヤ間のアクセス許可を定め，各パートのプレイヤに強制される．すなわち，COAC モデルでは，コミュニティのアクセス制御ポリシを，パートに基づいて規

定する．また，COAC モデルでは，１つのシステムを別のシステムの要素として組み込むことを，各システムに対応するコミュニティのアクセス制御ポリシの間に対応関係を与えるポリシ（連合のポリシ）に基づくコミュニティの連合とみなす．連合のポリシは，異なるコミュニティのパート間に対応関係を与える．先述の例では，ユーザが属するパート，旅行代理店システムが属するパート，ホテル予約サービスが属するパートから成るコミュニティを構成し，プレイヤ間で許可するアクセスをパートの関係として表現する．その結果，コミュニティのアクセス制御ポリシは不変のまま，ホテル予約サービスを追加あるいは削除することができる．また，旅行代理店機能を Web サービスとして提供して他のシステムに組み込む場合，例えば旅行代理店サービスを利用するシステムが属するコミュニティのパートとユーザのパートを連合のポリシで対応付け，後者を前者にも拡張したアクセス制御ポリシを設定することで，各システムを利用するプレイヤのレベルで再構築をすることなくアクセス制御ポリシを拡張できる．

また，本研究では COAC モデルのコミュニティを実現する論理的なシステム・アーキテクチャを示し，その中のアクセス制御機構を設計するための COAC フレームワークを提案した．システム・アーキテクチャは，各パートに属するプレイヤのメッセージ送受信を制御するアクセス・コントローラと，連合するコミュニティ間のメッセージ送受信を制御するバウンダリ・コントローラから構成される．COAC フレームワークは，コミュニティのアクセス制御ポリシにしたがうプレイヤの間のメッセージ送受信を，メタ階層構造を持つモデル記述言語を使って多階層で表現するためのフレームワークであり，コミュニティの連合とアクセス制御ポリシの実現のための基本構造を与える．COAC フレームワークのメタ階層の中で，上位階層は下位階層のメッセージ送受信を監視し，コミュニティのアクセス制御ポリシに違反するメッセージ送受信を拒否する機能を担う．したがって，COAC フレームワークの下位階層にはパートに属するプレイヤが行うメッセージ送受信を記述し，上位階層には論理的なアクセス制御機構を記述して，下位のメッセージ送受信がコミュニティのアクセス制御ポリシにしたがうことを示すことで，そのアクセス制御機構はコミュニティのアクセス制御ポリシを実現することが検証できる．なお，上位階層の機能は，システム・アーキテクチャにおいてアクセス・コントローラおよびバウンダリ・コントローラの機能に対応する．

COAC モデルは，アクセス制御ポリシと連合のポリシをパート間の関係に基づいて形式化したモデルであり，連合によってできたコミュニティのアクセス制御ポリシが各コミュニティのアクセス制御ポリシと整合することを検証するための基盤を与える．COAC モデルでは，個々のプレイヤはアクセス制御ポリシには現れない．したがって，各々のプレイヤを一意に識別する必要はなく，オープンなシステムに適合するアクセス制御ポリシの構築が可能となった．ホテル仲介システムと航空券仲介システムを結合するケーススタディを通じて，COAC モデルの妥当性と，単純なコミュニティから複雑なコミュニティを構成する連合の有効性を確認した．また，コミュニティのアクセス制御機構が COAC フレームワークを使って記述され，かつ，連合による結合が柔軟に行われ得ることを確認した．COAC モデルではパートを通じてプレイヤのアクセスを制御するため，RBAC モデルのように個々のプレイヤを対象とするアクセス制御は行えないが，ケーススタディの範囲では支障はなかった．現在の段階では，COAC モデルによるコミュニティの連合はパートの間に１対１ないし１対ｎの対応関係を設定できるときにのみ可能である．しかし，現実のシステムで詳細なアクセス制御を行

う場合にはパート間に明確な対応関係がないことがあり，その場合にも適用できるよう COAC モデルを拡張することは今後の課題である．

Abstract

The Web services, that enable to provide, to use, and to mediate services on the World Wide Web, are thought to be the basic components for constructing information systems that can be quickly adapted to the changes of the real world. For example, consider a travel agency system which offers customers hotel reservation services and uses hotel reservation functions provided by cooperating hotels as Web services. Such system has strong points: 1) new hotels can be easily added to the reservation services by modifying Web services used in the system, 2) the change of conditions of the hotel rooms can be managed by the providers of Web services. Further, other systems can use the travel agency system if it provides its travel agency function as a Web service. However, as the system becomes complicated, it becomes difficult to grasp all component Web services. Consequently, the hotel reservation services may be used in the way that does not consistent with the intention of its designer. The systems constructed in this way are in open environments that the components involved are heterogeneous and autonomous, and the system configurations can change dynamically. The methods for developing robust systems in open environments are still under research and development. Especially, the access control between components such as Web services and users (in the following, I call such components players) is indispensable in order to prevent illegal usage of services and information leakage. But the method for designing access policies that do not spoil the flexibility of the system and the method for implementing such policies are not provided.

In the past 30 years, many access control models are proposed. Since they are aiming at controlling the access to the resources managed by operating system, they are not appropriate for the systems in open environments that do not have central control mechanisms. For example, the Role-based Access Control (RBAC) model, which is widely accepted as the access control model that conforms to the structure of real society and standardized by America National Standard Institute, is based on the assumption that all players can be uniquely identified at arbitrary time. Furthermore, RBAC model cannot manage the access control policy of the system, which involves the subsystems each has access control policy, without reconstructing whole access control policy in order to maintain the identity of players. The problem of access control for the systems in open environments is unsolved.

In this paper, I propose the Community Oriented Access Control (COAC) model, which is an access control model for the systems in open environments. In COAC model, the collection of players involved in the system is called community, and the access control policy is assigned to the community. A community is a collection of

players that play roles (in the following, I call such roles parts) such as provider of a service, user of a service, mediator of a service, etc. The community's access control policy defines the access permissions between players of different parts. Accordingly, community's access control policy is represented based on the parts. Combining systems to make a larger system is regarded as the federation of communities. The access control policy of the community formed by the federation is described according to the policy of federation that gives the correspondence relationship between the access control policies of communities. The policy of federation is represented as the correspondence relationships between the parts of different communities. For the travel agency system, a community is formed which consists of the part of system users, the part of travel agency system, and the part of hotel reservation Web services. The access control policy of the travel agency community is represented as the accessibility relationship between parts and defines the access permissions between the players of different parts. Therefore, the administrator of the travel agency system can add new hotels to the reservation services without changing the community's access control policy. To use the travel agency system in a larger system, the administrator may extend the community's access control policy by corresponding the part of the travel agency system users and the part of the larger system, i.e. regarding the larger system as a user of the travel agency system. In this way, the access control policy of federated community can be obtained without reconstructing the policy of each community in the level of each player.

For realizing the community of the COAC model, I show logical system architecture of the community and propose the COAC framework that is a framework for designing access control mechanism in the system architecture. The access control mechanism consists of the access controllers and the boundary controllers. Each access controller controls the message transfer of the players of each part, and each boundary controller controls the message transfer between federated communities. The COAC framework provides a method for describing the model of message transfer between players that are compliant with the community's policy. The model is represented by hierarchical layers of message transfer and supplies the basic structure for implementing the access control policy of each community and their federation. In the hierarchy of message transfer layers, the upper layer observes the message transfer of the lower layer and rejects message which violates the community's access control policy. Therefore, by describing logical access control mechanism in the upper layer and showing that the message transfer of the lower layer is compliant with the community's access control policy, the system designer can verify whether the access control mechanism implements the community's access control policy. The access controllers and the boundary controllers of the system architecture implement the functions described in the upper layer.

The COAC model is a formal model of the communities' access control policies and the policy of their federation, and provides a basis for verifying that the access control policy of the federated communities is consistent with the access control policy of each community. The access control policy of community and the policy of federation are

not represented as the accessibility relationships between players but represented as the accessibility relationships between parts. The COAC model enables to describe access control policy that can be applied to the system in open environments, since it is not necessary to identify each player uniquely. The case study of combining a hotel mediation system and an air ticket mediation system shows the propriety of the COAC model and that the access control mechanism of the community which can be combined easily with that of federating community is described by using the COAC framework. Also the case study shows the advantage of the model in forming a complex community from the federation of simpler communities.

Since the access control policy in the COAC model is represented independent of the individual players, it is impossible to represent the access control policy with constrains of player level that the RBAC model provides. However, the constraints of player level are not required in the domain of the case study. At present stage, the federation of communities can be handled in COAC model only if one to one or one to N correspondence between the parts of federating communities exists. But, there are cases that the clear correspondence between parts cannot be found in real systems. The future work is to extend the COAC model to be applicable to such cases.

目 次

図 目 次

第1章　序論

本章では，まず 1.1 節で研究の動機を述べる．コミュニティの概念を 1.2 節でインフォーマルに導入し，その実現について 1.3 節で述べる．1.4 節では本研究の目的を述べ，次章以下の構成を示す．

1.1　研究の動機

ネットワークを通じたサービス提供／利用／仲介の発達は，私たちの生活を大きく変えつつある．既に，多くの人々は日常的にネットワークを通じて情報を入手し，商品を購入し，あるいは旅行を手配している．このような技術には，より高度で複雑なサービスを，より多くの人々が簡単に安心して利用できる方向へと発展を続けることが期待されている．

World Wide Web 上でサービスとして提供されるソフトウェアの機能は Web サービスと呼ばれ，急速に変化する社会環境に適応できる柔軟な情報システムを構築するための，基本的な構成要素とみなされている．Web サービスを使って構成されたシステムは，サービスを単位とする機能の変更に容易に対応できる．さらに，システムが提供する機能自体を Web サービスとして提供することで，より複雑なシステムの一部として組み込まれることも可能である．

このようなシステムは自律的な参加者から構成されるオープンなシステムであり，このことがサービスの多様性と変更に対する柔軟性を生み出している．その反面，中央集約的な管理が困難であることから，参加者が予想しないアクセスや，悪意ある参加者による不正利用の危険にもさらされている．堅牢なオープンなシステムを開発するための技術は，未だ研究開発の途上にある．特に，参加者やサービスを不正なアクセスから守るアクセス制御は不可欠であるにもかかわらず，システムの柔軟性を損なうことのないアクセス制御ポリシをどのように設計し，それを実現するしくみをどのように作るかは，課題であった．Web サービスを使って構成されるシステムの，このような課題を解決するため，本研究では一定のポリシにしたがう参加者から成るコミュニティを想定し，コミュニティ内の参加者のためのアクセス制御ポリシと，コミュニティを実現するための設計手法を提案する．

1.2　コミュニティ

コミュニティは，ネットワークの中から信頼できる参加者を見つけ出し，安心できるサービスの提供／利用を行うための枠組みの一つである [57]．参加者がコミュニティに加わる際に，コミュニティの他の参加者との間に適用される一連の契約を包括的に

結ぶことにより，コミュニティ内では参加者を相互に信頼することが可能になる．このような包括的な契約はコミュニティの参加者の変動に関わりなく適用されるため，以下ではこれをコミュニティのポリシと呼ぶ．コミュニティとして以下の性質を持つものを考える．

- コミュニティは，ポリシにしたがう参加者のみによって構成される
- 参加者はコミュニティのポリシによって緩やかに束縛されるが，自律的にふるまうことができる
- コミュニティは，他のコミュニティと結合して拡張し得る

コミュニティの結合による拡張を，本論文ではコミュニティの連合と呼ぶ．連合したコミュニティでは，あるコミュニティに属する利用者は，連合先のコミュニティに属する提供者からのサービスを受けることができる．個々のコミュニティのポリシが，連合したコミュニティのポリシにおいてどのように扱われるかは，連合する際に規定するべきである．例えば，あるコミュニティが利用者保護のために利用者と提供者の直接のアクセスを禁じ，利用者へのサービス提供を仲介者が代行することをポリシとしているならば，連合先のコミュニティの提供者との間でも直接的なアクセスが禁止されるのは自然であろう．コミュニティがどのように連合するかを規定する規則を，以下では連合のポリシと呼ぶ．

コミュニティのポリシで規定される内容は，そのコミュニティの性質を反映して，さまざまであると考えられる．例えば，商取引のコミュニティでは支払や返品の方法が規定されているかもしれないし，情報共有のコミュニティでは書き込みのマナーが規定されているかもしれない．これらのポリシのあるものは形式的に記述し参加者がポリシを遵守していることを機械的に検証し得るであろうし，あるものは記述は可能でも機械的な検証は困難であるかもしれないし，さらには参加者の暗黙の了解に基づくため記述することさえ困難なものもあるかもしれない．本研究ではコミュニティ内の参加者のアクセス制御に関する，形式的な記述と検証が可能なポリシに焦点をあて，オープンなシステムのためのアクセス制御モデルとして COAC（Community Oriented Access Control）モデルを提案する．

1.3 コミュニティの実現

オープンなシステムでは，システムに関する情報が完全ではないため構成要素の変化を予め知る事はできない．したがって，各構成要素は自己をとりまく他の構成要素（以下では環境と呼ぶ）の変化を前提とし，変化に対して自己を調整しつつ動作を継続する必要がある．オープンなシステムのモデルとして，アクターやエージェントによるモデル [1, 19, 56] が知られている．これらのモデルでは，構成要素は以下のように特徴付けられる．

- 他の構成要素とは独立に機能する．
- 相互にコミュニケーションする．

- コミュニケーションを通じてゴールを達成する．
- コミュニケーションを通じて環境に適応する．

ソフトウェア開発の面からは，このようなシステムは開発と運用の区分があいまいで，連続的に変化するシステムであると言える．ここで特徴的な点は，システムの開発と運用が同時に行われることであり，システムの変更によって動作中のソフトウェア・モジュールが受ける影響の少ないシステム構成が求められる．従来のシステム開発技術は環境の変化を前提としないため，システムの変更によって影響を受ける部分とそうでない部分が設計段階では明確に区分されず，連続的に変化するシステムの開発には適していない．

本論文では，アクセス制御ポリシに基づくコミュニティを実現するための論理的なシステム・アーキテクチャを示し，その中のアクセス制御機構を設計するためのフレームワークとしてCOAC（Community Oriented Access Control）フレームワークを提案する．このフレームワークを記述するため，エージェントと場の概念に基づくモデル記述言語を採用する．この言語に求められるのは，以下の性質である．

- 記述の単位の独立性が高いこと
- 環境の変化への適応を記述できること
- 個々の記述単位の機能から大域的な性質を導出できること

1.4 本研究の目的

コミュニティの参加者の間のアクセス制御を実現するためには，以下の項目を明らかにする必要がある．

- コミュニティのアクセス制御ポリシは，どのように規定されるのか
- コミュニティの連合のためのポリシは，どのように規定されるのか
- 参加者の間のアクセス制御は，どのような仕組みによって実現されるのか

ここで，オープンなシステムの特性から，コミュニティのアクセス制御ポリシは各々の参加者には依存しないで規定される必要がある．また，連合のためのポリシは，連合する各コミュニティのアクセス制御ポリシに基づいて規定されなければならない．

本研究のゴールは，このような課題を解決し，セキュリティ技術，ソフトウェア工学，マルチエージェント技術の視点から，アクセス制御ポリシにしたがうコミュニティの構築手法を明確にすることにある．このための大きな要素は，次の２点である．

- 自律的な参加者から構成されるコミュニティのためのアクセス制御モデル
- コミュニティを実装するためのシステム設計手法

本論文では，まずコミュニティ内の参加者間のアクセス制御ポリシ・モデルを提案する．次に，コミュニティを実現する論理的なシステム・アーキテクチャを示し，コミュニティのアクセス制御機構を上記のアクセス制御ポリシ・モデルに基づいて設計するためのフレームワークを提案する．

以下，本研究の背景となる関連研究を 2 章でサーベイする．3 章で本研究で提案するアクセス制御モデルである COAC モデルを定義し，コミュニティの連合におけるアクセス制御ポリシの分離について述べる．4 章ではオープンなシステムのモデル記述言語を導入し，コミュニティの実現モデルを記述するためのフレームワークである COAC フレームワークを，この言語を使って 5 章に示す．COAC モデルと COAC フレームワークに基づくケーススタディを，6 章に示す．7 章では，COAC モデルと COAC フレームワークを評価するとともに，関連研究と比較する．8 章では本研究をまとめ，今後の課題について述べる．

第2章　現状の技術とその課題

本章では，サービス・コミュニティのアクセス制御ポリシを考える上で，背景となる技術について論じる．まず，2.1 節でセキュリティ・ポリシ・モデルをサーベイし，Role Based Access Control（RBAC）モデルを紹介する．また，2.2 節で Web サービス技術を列挙するとともに，それらを使ってシステムを構築するための開発指針であるサービス思考アーキテクチャを紹介する．

Web サービスのためのアクセス制御モデルについては，近年さまざまな研究がなされている． 2.3 節では，本研究の関連研究となる Web サービスのためのアクセス制御モデルをサーベイし，2.4 節でサービス・コミュニティのアクセス制御の課題を位置づける．

2.1　セキュリティ・ポリシ・モデル

情報システムのセキュリティ・ポリシの研究は，軍用システムのためのセキュリティ・ポリシから，商用システムのためのセキュリティ・ポリシへと拡大されてきた．対象とするシステムの性格によって，要求されるセキュリティ・ポリシが異なるため，それらを表現するために様々なセキュリティ・ポリシ・モデルが提案されている．この節では，代表的なセキュリティ・ポリシ・モデルを示す．

2.1.1　セキュリティ・ポリシ

情報システムのセキュリティは，秘匿性（confidentiality），完全性（integrity），可用性（availability）の 3 要素を基礎とする．秘匿性は，情報が特定のユーザには開示されないことである．完全性は情報が信頼できることであり，情報が不適切な変更を受けないことを意味する．可用性は，ユーザが必要なときにアクセスできることである．文献 [9] では，これらは次のように定義される．

秘匿性　X を実体の集合，I を情報とするとき，X の全ての要素が I に関する情報を得ることができないならば，I は X に対して秘匿性を持つ．

完全性　X を実体の集合，I を情報または資源とするとき，X の全ての要素が I を信頼するならば，I は X に対して完全性を持つ．

可用性 X を実体の集合，I を資源とするとき，X の全ての要素が I にアクセスできるならば，I は X に対して可用性を持つ．

セキュリティ・ポリシは，システムの状態をセキュアな状態とセキュアでない状態に分離する言明であり，これらの 3 つの側面に関わる．セキュリティ・ポリシのうち，秘匿性に関わるものは秘匿性ポリシ，完全性に関わるものは完全性ポリシと呼ばれる．

セキュリティ・ポリシが利用するアクセス制御には，次の 2 種類がある．

- 強制アクセス制御（mandatory access control）
- 任意アクセス制御（descritionary access control）

強制アクセス制御では，オブジェクトへのアクセスがシステムによって規定され，ユーザはそれを変更することができない．これに対し，任意アクセス制御ではオブジェクトへのアクセスの可否をユーザがユーザが設定することができる．ここで，システムの安全性を，任意アクセス制御を含むシステム一般について判定する問題は，決定不能であることが HRU モデル [18] によって示された．一方，任意アクセス制御を含む場合でも，個々のシステムに関してシステムの安全性を判定するアルゴリズムは存在することが，Take-Grant モデル [23] によって示された．

2.1.2 秘匿性ポリシ・モデル

秘匿性ポリシは，情報の不正な開示を防止する．ADEPT-50 の最高水準点（High-Water Mark）[55] モデルや，Bell-LaPadula モデル [5, 6] が代表的なモデルであり，軍の階級に対応する機密情報の取り扱いをモデル化することができる．

Bell-LaPadula モデルは，多階層（multilevel）セキュリティ・ポリシの代表的なモデルであり，アクセスの主体および対象を線形順序関係にある階層に分け，階層構造に基づいてアクセスを制限する．また，Bell-LaPadula モデルは任意アクセス制御と強制アクセス制御を併用するモデルであり，強制アクセス制御によって制限されない限り，任意アクセス制御によってアクセス可否を設定することができる．Bell-LaPadula モデルの目的は，主体の階層より上位にある対象への読み出しアクセスを禁止すること，すなわち上位階層の情報が下位階層へ流れるのを禁止することである．これは，次の 2 つの特性によって定義される．

- シンプル・セキュリティ特性（simple-security property）
  主体は，対象が同じまたは下位の階層にあり，かつ対象に対して任意読み出しアクセスを持つならば，対象を読み出すことができる．
- ＊特性（star-property）
  主体は，対象が同じまたは上位の階層にあり，かつ対象に対して書き込みアクセスを持つならば，対象に書き込むことができる．

シンプル・セキュリティ特性と＊特性が共に成り立つシステムを，セキュア・システムとする．セキュア・システムにおいて，セキュアな初期状態から，シンプル・セキュリティ特性および＊特性を保持した状態遷移によって遷移した状態はセキュアであることが，証明できる．

2.1.3 完全性ポリシ・モデル

完全性ポリシは，データの不正な変更を防止する．軍用システムのセキュリティにおいて秘匿性は優先度の高い側面であるのに対し，商用システムのセキュリティにおいて完全性は優先度の高い側面である．代表的な完全性ポリシ・モデルには，Biba モデル [8]，Clark-Wilson モデル [13]，LOMAC モデル [17] がある．

Biba モデルは，情報の不正な変更を禁止するためのポリシ・モデルを提供する．Biba モデルも多階層セキュリティ・ポリシ・モデルであり，ユーザおよびデータを完全性階層に分割して，下位階層のデータが上位階層のデータを汚染しないように情報フローを制御する．Biba モデルの規則は，次の 3 つである．

- 主体は，対象が同じまたは上位の階層にある場合に限って，対象を読み出すことができる．
- 主体は，対象が同じまたは下位階層にある場合に限って，対象を変更することができる．
- 主体は，対象が同じまたは上位の階層にある場合に限って，対象に書き込むことができる．

対象 o_1 から $o_{n+1}(1 < n)$ へ情報フローがあるとき，すなわち主体 $s_1 ... s_n$ が存在して $s_i(1 \leqslant i \leqslant n)$ が o_i を読み出し o_{i+1} を変更するとき，Biba モデルの規則によって o_1 は o_{n+1} より上位階層にあることが証明できる．

2.1.4 混成型ポリシ・モデル

Bell-LaPadula モデルと Biba モデルは，各々秘匿性と完全性の一つの側面のみを対象としており，主体は Bell-LaPadula モデルでは秘匿性階層の下位，Biba モデルでは完全性階層の上位の対象を読み出すことができる．しかし，多くのシステムでは秘匿性と完全性の両側面が求められ，これらのポリシ・モデルでは不十分である．Chinese Wall モデル [11] と Role-based Access Control（RBAC）[2, 15, 16, 45] モデルに代表される混成型ポリシ・モデルは，秘匿性と完全性の両側面を提供する．

Chinese Wall モデルは，利害の衝突（conflict of interest）を含むポリシを記述することができる．基本的な構成要素は，次の 3 つである．

- オブジェクト
  企業に関する個々の情報
- 企業データセット
  1 つの企業に関するオブジェクトの集合
- 利害衝突クラス
  競合企業のデータセットの集まり

Bell-LaPadula モデルのシンプル・セキュリティ特性に対応して，Chinese Wall モデルでは次のいずれかの場合に限って主体 S は対象 O を読み出すことができる．

- O が，S が既にアクセスした対象 O' と同じ企業データセットに属している場合
- O が，S が読み出した全ての対象 O' とは異なる利害衝突クラスに属している場合

また，＊特性に対応して，次の条件が満たされるときに限って S は O を書き込むことができる．

- S が O を読み出すことが許可される．
- 全ての対象 O' について，S が O' を読み出せるならば，O と O' は同じ企業グループに属している．

RBAC モデルは，セキュリティ・ポリシの各要素を役割（role）を使って表現する．RBAC モデルについては，2.1.5 節で述べる．なお，NIST の RBAC モデル [2, 15] では，ユーザは強制アクセス制御によって制御される．アクセス制御ポリシの管理は，モデル上には陽に現れない暗黙の管理者によって行われる．一方，Sandhu 他 [45] では，アクセス制御ポリシの管理自体も RBAC によって制御される．

2.1.5 RBAC モデル

RBAC のアイディアは，主体が担う役割（ロール）に基づいて，対象に対するアクセスを制御することにある．ロールの概念は，以前からさまざまなアプリケーションやデータベースの特権管理において用いられてきた．RBAC モデルは，これらを抽象化して個々のアプリケーションとは独立な一般的なモデルへと統合したものと位置付けられる．

RBAC のリファレンスモデルは，ANSI 標準 [2] で形式仕様記述言語を使って定義された．しかし ANSI 標準の記述には誤りが含まれているため，誤りを修正して書き直したリファレンスモデルを，付録 A に示す．

RBAC の基本的な構成要素は，次の４つである．

- ユーザ
- ロール
- 操作
- 対象

ユーザは人間あるいは計算機やエージェントなどであり，ロールは組織内で与えられる権限や責任の観点からユーザに割り当てられる職責である．操作は，起動されるとユーザのために何らかの機能を提供する実行形式のプログラム，対象は操作が加えられる相手である．ファイルやディレクトリなどの情報を格納するもののほかに，プリンタやディスクスペースなどの消費されるシステムリソースも，対象である．保護された対象に対して操作の実行を許すことを，許可と呼ぶ．RBAC は基本的に，

- ロールに割り当てられたユーザ

- ロールに割り当てられた許可

によって定義される．ユーザと許可がロールを経由して結合されることにより，ユーザへの許可の割り当てが柔軟に行われる．

ANSI の RBAC モデル [2] では，CoreRBAC および，CoreRBAC をベースに他の 3 つの要素を組み合わせて拡張したモデルが規定されている．

1. CoreRBAC
   CoreRBAC は，RBAC の基本概念である以下の項目を規定する．

   - ユーザにはロールが割り当てられる
   - ロールには許可が割り当てられる
   - ユーザは，ロールのメンバとなることで許可を獲得する

   CoreRBAC には，以下の項目が要求として含まれる．

   - ユーザとロールの間および許可とロールの間の関係が多対多であることを許す．
   - 特定のユーザに割り当てられたロールを決定したり，特定のロールが割り当てられたユーザを決定する，ユーザ－ロール検査を含む．
   - ロールの選択的な開始や終了を許す，セッションの概念を含む．
   - ユーザが複数のロールに与えられた許可を同時に行使することを許す．

2. 階層的 RBAC
   階層的 RBAC は，ロールの間に階層関係を導入する．階層はロールの間の半順序関係であり，下位のロールに与えられた許可が上位のロールにも与えられるとともに，上位のロールのユーザメンバーシップが下位のロールにも与えられる．ANSI 標準では，次の 2 種類のロール階層を認める．

   - 無制限階層的 RBAC
     ロール階層として，任意の半順序関係を許す．
   - 制限つき階層的 RBAC
     ロール階層に制約を課す．

3. 責務関係の静的な分離（SSD）
   責務関係の静的な分離は，ロールへのユーザの割り当てを制約する．SSD は，2 つ以上のロールからなる集合と，そのうちユーザに同時に割り当てることが禁止されるロールの数を示す 2 以上の数によって規定される．

4. 責務関係の動的な分離（DSD）
   責務関係の動的な分離は，ユーザが開始できるロールにセッションに基づく制約を設けることで，ユーザが利用できる許可を制約する．

SSD は，潜在的に責務の衝突を起こす可能性があるロールに，ユーザが割り当てられるのを排除する．例えば，一人のユーザが経理関係のロール４つのうちの３つ以上に割り当てられてはならない，というポリシは SSD を使って表現することができる．一方 DSD は，セッションの中で同時に開始すると責務の衝突を起こすロールに，ユーザが割り当てられるのを排除する．ここで，DSD はこれらのロールが同一セッションで開始されることを排除するのであり，これらのロールが一人のユーザに割り当てられること自体は排除しない．その意味で，SSD は DSD よりも強い制約である．

2.2 ネットワーク上のサービス

この節では，ネットワーク上のサービス提供／利用に関する現在の技術を，システムを構成する要素技術と，それらを使ったシステム構築技術の２つの側面から，サーベイする．まず，Web サービスのための主な標準技術を，2.2.1 節に列挙する．次に，Web サービスを使ってシステムを構築するための開発指針となる SOA（Service Oriented Architecture）について，2.2.2 節にまとめる．

2.2.1 Web サービス

インターネットの拡大，高性能な計算機の遍在化，プラットフォームや計算機言語に依存しないメッセージング技術の登場により，個々のシステムやソフトウェアの構成に依存しない「サービス」[52] をネットワークを通じて提供／利用する，Web サービスが実現しつつある．Web サービスを使って構成されるシステムは，関係する構成要素が異種（heterogeneous）かつ自律的（autonomous）であるとともに，システムの構成が動的に変化し得るという意味で，オープンなシステムである．Web サービスのアーキテクチャには，一般に次の３種類の参加者が関与する．

- サービスの提供者
- サービスの利用者
- サービスの仲介者

サービスの仲介者は，提供されるサービスに関する情報を管理し，利用者の要求に適合するサービスを仲介する．これら３者の間を結ぶための基本技術として，次の３つが広く知られている．

- XML/SOAP
  参加者間でデータを交換するための共通言語と手続きを提供する．
- WSDL
  サービスに関する情報を表現する枠組みを提供する．
- UDDI
  サービスに関する情報のリポジトリを提供する．

XML

XML（eXtensible Markup Language）[66] はHTML（HyperText Markup Language）の拡張であり，記述の構造を規定するタグを必要に応じて定義することができる点で，HTML とは異なる．記述の構造を明示的に規定できることから，XML はアプリケーションによって機械的に処理することができる．このため，Web 上でデータを交換するための基盤となる記述言語として利用されている．

SOAP

SOAP（Simple Object Access Protocol）[60] は，構造を持つデータを分散環境のもとで交換するためのプロトコルを提供する．SOAP は HTTP（Hypertext Transfer Protocol）などの Web の標準プロトコルを利用するため，Web 上で XML メッセージを交換するためのプロトコルとして広く使われている．

WSDL

WSDL（Web Services Description Language）[63] は，ネットワーク・サービスを記述するための言語を提供する．WSDL は XML を使って定義された言語であり，Web サービスに関する情報を Web 上で交換するための基盤を与える．

UDDI

UDDI（Universal Description, Discovery, and Integration）[61] は，Web サービスの提供者がサービスに関する情報を公表し，利用者がサービスを発見するための方法を提供する．

WSCI

WSCI（Web Services Choreography Interface）[62] はXML ベースのインタフェース記述言語であり，相互にインタラクションを行う Web サービスのメッセージ・フローを記述する言語を提供する．WSCI は WSDL とともに機能し，Web サービスの観測可能な振る舞いを記述する．

BPEL4WS

BPEL4WS（Business Process Execution Language for Web Services）[58] は，ビジネス・プロセスとインタラクション・プロトコルを形式的に規定する方法を与える．BPEL4WS では，WSDL で記述された情報に基づいて，複数の Web サービスを連携するビジネス・プロセス記述言語が提供される．

セマンティック Web

セマンティック Web[7, 24, 40] は，Web 上の情報に機械処理可能なメタデータを付与するための基本的な枠組みを提供する．セマンティック Web はさまざまな技術から構成されており，その中には Web サービスでも有効であることが期待される技術も多い．特にオントロジー技術 [59] は，Web サービスに関する情報に共通の意味を与えるための基盤として期待されている [49].

WS-Security

WS-Security（Web Services Security）[36, 64] は，メッセージの完全性，秘匿性を実現するための SOAP 拡張機能を提供する．WS-Security ではメッセージにセキュリティ・トークンを付与するメカニズムを与え，アプリケーションによる安全な SOAP メッセージ送受信を可能にする．

OASIS のロードマップ [65] によれば，管理ドメイン間にセキュアで相互運用可能な Web サービスを構成することを目的として，WS-Security の上位に以下の仕様が定義される予定である．

- WS-Policy
  メッセージ送受信の相手が持つセキュリティ要件を記述するための仕組みを提供する．
- WS-Trust
  セキュリティ・トークンを第 3 者機関から取得するためのプロトコルを規定する．
- WS-Privacy
  プライバシー情報の扱いを規定する．

これらのさらに上位に，連合を扱うための以下の仕様が構成される．

- WS-SecureConversation
  メッセージ送受信の相手を相互に認証する方法を規定する．
- WS-Federation
  異なる管理ドメイン間で認証情報を交換する方法を規定する．
- WS-Authorization
  Web サービスのアクセス・ポリシがどのように規定され管理されるかを記述する．

これらの仕様が標準化されれば，サービス提供／利用／仲介のアクセス制御を実装するための基盤技術となる．

2.2.2 サービス指向アーキテクチャ

実世界のビジネス環境は急速に変化しており，適切なビジネスパートナーと速やかにビジネスを開始することが重要になっている．Web サービスは，異種の構成要素を動的に結合し得る点で，急速に変化するビジネス環境のためのアプリケーション構築の基盤として有効であると考えられている．ここで，Web サービスは基本的に１つの提供者が１つのサービスを１つの利用者に提供することを前提としている．したがって，現実の要求に適合する複雑度を持つシステムを開発し管理するためには，Web サービスの概念だけでは十分ではない．サービス指向アーキテクチャ（Service Oriented Architecture）[26] は，現実の要求に適合するアプリケーションを，Web サービスの連携によって構築するためのソフトウェア・アーキテクチャである．サービス指向アーキテクチャの目標は，次の２点である．

- オープンなシステムのための柔軟なアプリケーションを可能にする．
- オープンなシステムにおけるアプリケーションの開発と保守の生産性を向上する．

サービス指向アーキテクチャは，次の要素から成るソフトウェア・アーキテクチャであると定義される．

- システムの全ての活動を開始／制御する能動的なプレイヤ
- 上位のビジネス概念をカプセル化した機能単位であるサービス
- サービスの発見や，サービスの利用に関する情報を提供するリポジトリ
- プレイヤやサービスを相互に結合するネットワーク

ここで，サービスもシステムに対して能動的に働きかけ得る（サービスが他のサービスを利用するなど）ため，本研究ではサービスを含めたシステムの構成要素をプレイヤと呼ぶ．

サービス指向アーキテクチャの基本的な考え方は，マルチエージェント・システム [10] の考え方と共通する．この意味では，サービス指向アーキテクチャは，標準化された Web サービス技術のもとにマルチエージェント・システムを実現するための，システム開発技術と位置づけることもできる．

サービス指向アーキテクチャでは，以下のような要件が規定されている [47] ．

1. 疎結合
   構成要素間に密なトランザクション結合がないこと．構成要素をまたがってデータの整合性を求めることは，一般に適切ではない．一方，構成要素が高レベルの契約的な関係のもとに相互作用することは許容される．

2. 実装非依存
   構成要素間のインタフェースを規定すべきであり，相互作用する構成要素の実装の詳細には依存してはならない．特に，特定のプログラミング言語に依存するべきではない．

3. 柔軟な構成
   システムは，構成要素の柔軟な動的束縛によって構築されなければならない．言い換えれば，構成要素が開発過程の後の方で結合されるとともに，構成が動的に変更可能でなければならない．

4. 長い寿命
   オープンなシステムを扱っているので，常に例外処理を扱うことが可能でなければならない．このことから，構成要素は関係する例外を検出し，修正動作を行い，他の構成要素の修正動作に反応するのに十分な時間，存在する必要がある．

5. 粗い粒度
   サービス指向アーキテクチャの構成要素は，粗い粒度で理解されるべきである．すなわち，構成要素の間のビジネスのために本質的な特質によって，動作や相互作用をモデル化するのがよい．粗い粒度は構成要素間の依存性を縮小し，コミュニケーションをより重要な少数のメッセージに集約する．

6. チーム
   計算を中央集約的にまとめる代わりに，計算が自律的な組織によってどのように実現されるかを考えるのが良い．オープンなシステムにおける計算では，相手に命令する構成要素の代わりに，チームとして働くビジネスパートナーが重要である．したがって，個々の構成要素ではなく，協調する構成要素のチームがより良いモデル化の単位となる．

サービス指向アーキテクチャは，オープンなシステムのための開発指針を与える．しかし，複雑なシステムを開発するためには，Web サービスの個々の要素技術と，それを使うための開発指針が提供されるだけでは不十分である．サービス指向アーキテクチャと整合性があり，システム開発を把握可能な複雑度の部分へと分割する，オープンなシステムのモデルが必要である．

2.3 Web サービスのためのアクセス制御モデル

セキュリティ・ポリシ・モデルの研究成果を，Web 上のサービス，情報，リソースのアクセス管理に応用する研究が，近年盛んに行われている．この節では，その主なものを列挙する．これらは本研究の関連研究であり，7.5 節で詳細に論じる．

2.3.1 RBAC/Web

RBAC/Web[27, 14] は，World Wide Web のための RBAC の実現であり，Web 上での権限の管理と権限ポリシの強制のための方法を与える．RBAC/Web は，Web 上の情報へのアクセスを，ユーザに割り当てられたロールに基づいて制御する．ユーザの権限をロールに基づいて管理することで，機能の変更や拡張に柔軟なシステムを構成することができる．

RBAC/Web では，ユーザはただ 1 つのログイン名を持ち，ユーザに割り当てられた全てのロールを任意の時点で知ることができなければならない．しかし，Web はオー

プンな環境であることから，その実現は容易ではなく，実装の段階で RBAC の各機能は制限される．また，２つのシステムが連合する際には，連合してできたシステムのためにロールを再構成する必要があるとともに，個々のユーザについて割り当てられるロールを変更する必要がある．

2.3.2 URA97/WWW

RBAC/Web ではユーザを中央集約的に認証する必要があるため，Web 上での実装では各機能が制限される．URA97/WWW [46] は，URA97 と呼ばれる管理モデルを RBAC/Web に適用することで，ユーザへのロール割り当ての非集中的な管理を実現し，この問題を解決する．

URA97/WWW では，RBAC/Web のようにユーザへのロール割り当てを中央集約的に行う必要はない．しかし，ユーザに割り当てられている全てのロールを知ることができることを前提としているため，個々のユーザについて割り当てられたロール情報を管理するサブシステムは連携する必要がある．また，連合の際には個々のユーザについてロールを割り当て直す必要がある．

2.3.3 Web 上での RBAC 実現アーキテクチャ

Park 他 [42] は，Web 上で RBAC の実現するため，ユーザに割り当てられるロールやアイデンティティなどの属性を Web 上で得るアーキテクチャを提案した．

しかし，これらのアーキテクチャにおいてロールは一元的に管理される必要があり，ロールの変更に対する柔軟性は低い．

2.3.4 I-RBAC

I-RBAC [50] は，ロールの概念に基づくアクセス制御をイントラネット上で実現することを目的としている．I-RBAC では，個々のサーバ上にあるオブジェクトへのアクセス許可を定めるローカル・ロールと，イントラネット上にあるサーバ全体へのアクセス許可を定めるグローバル・ロールの２種類のロールが使われる．

ローカル・ロールが各サーバに分散して管理されるのに対し，グローバル・ロールはイントラネット上で一意に管理される必要があり，各サーバにはそのレプリカが置かれる．このため，グローバル・ロールあるいはそれを構成するローカル・ロールに変更が起こった場合には，イントラネット上の全てのサーバのグローバル・ロールを更新しなければならない．

2.3.5 連合のアクセス制御

データベースの分野では，独立に開発された異種データベースを統合し協調させるデータベースの連合に関する研究の進展とともに，連合したデータベースシステムに対するアクセス制御の問題が議論されるようになった．

Taylor 他 [51] は，Web 上に分散して存在する異種データベースの連合のための，認証とアクセス制御のモデルを提案した．このモデルでは，一定の契約を結んだ関係者のコミュニティを前提とし，コミュニティが緩やかに結合した連合を考える．ユーザには，コミュニティ内でどのような情報にアクセス可能かを示すプロファイルを割り当てる．ユーザからのアクセス要求に対して，データベースを管理するゲートウェイがプロファイルに基づいてアクセス可否を判断することにより，ユーザ－ロール割り当てをアクセス対象であるデータベースのゲートウェイで局所的に管理することができる．

ユーザに割り当てられた全てのロールを知るためには全てのゲートウェイを調べる必要があるため，RBAC の責務関係の分離は実現できない．しかし，Taylor 他は，連合データベースの読み取りのみを利用するアプリケーション分野では責務関係の分離は不要であるとする．一方，ユーザ側ではどのような情報にアクセス可能であるかを陽に知ることができないため，ユーザはゲートウェイへのアクセスを通じてアクセス許可を確認する必要がある．ここで，連合関係が変化した際にロールを再構成する枠組みが提供されていないので，ユーザは必要な情報へのアクセス許可が与えられるゲートウェイに到達するまで，連合関係の変化によって再構成されたゲートウェイに順次アクセスする必要がある．

2.4 未解決の課題

Web サービスのためのアクセス制御は，ユーザや Web サービスを他のユーザや Web サービスによる不正なアクセスから守り，不正利用や情報漏えいからシステムを守るために必要不可欠である．2.1 節のセキュリティ・ポリシ・モデルは，Web サービスのためのアクセス制御においても基盤技術となる．

しかし，Web サービスを利用して構成されるシステムはオープンなシステムであり，全ての構成要素を把握できることを前提とするセキュリティ・ポリシ・モデルをそのまま適用することはできない．2.2.2 節のサービス指向アーキテクチャの要件から，オープンなシステムのためのアクセス制御の要件は，次のように整理することができる．

1. 疎結合について
   構成要素間に密な結合を生じさせてはならない．

2. 実装非依存性について
   構成要素の実装の詳細に依存してはならない．

3. 柔軟な構成について
   構成要素の柔軟な動的束縛を妨げてはならない．

4. 長い寿命について
   構成要素が存在する時間アクセスを制御しなければならない．

5. 粗い粒度について
   粗い粒度の構成要素のアクセスを制御できなければならない．

6. チームについて
   協調する構成要素のチームを単位とするアクセス制御を行わなければならない.

2.3 節に示したように，Web 上に構成されるシステムのためのアクセス制御モデルについて様々な提案がなされている．これらのモデルは，上記の要件を部分的に満たすが，満たされない要件もあり，十分とは言えない．オープンなシステムのためのアクセス制御において，高い優先度で求められる性質をまとめると，以下のようになる．

- システムを構成する要素に対するアクセスを制御すること
- 中央集約的な管理機構を前提としないアクセス制御であること
- システムの連合に対応できるアクセス制御であること

第3章　COACモデル

本章では，オープンなシステムのためのアクセス制御モデルとしてCOAC（Community Oriented Access Control）モデルを提案し，コミュニティのためのアクセス制御ポリシを厳密に形式化する．まず，形式化の基本要素を 3.1 節に列挙する．この基本要素に基づいて，3.2 節でコミュニティ内のアクセス制御ポリシのモデルを，3.3 節ではコミュニティの連合のためのポリシのモデルを示す．また，サービス仲介の基本パターンを対象とした，COAC モデルによるアクセス制御ポリシの記述例を，3.4 節に示す．

3.1　基本構成要素

3.1.1　概要

コミュニティと，それを構成する要素の間の関係を，模式的に図 3.1 に示す．この図で，長方形はコミュニティを，円はパートを，破線の楕円はプレイヤを表す．本研究では，ネットワーク上のサービス提供／利用を，一般に複数の，コミュニティによって表現する．各コミュニティはパートの集まりから構成され，各パートはプレイヤの集まりから構成される．

3.1.2　プレイヤとパート

本論文では，サービスの提供／利用／仲介に関わる参加者を，包括的にプレイヤと呼ぶ．例えば，サービスが計算機システムによって自動処理される場合にはその計算機システムを，人間が直接関与して処理される場合にはその人間をプレイヤとみなす．

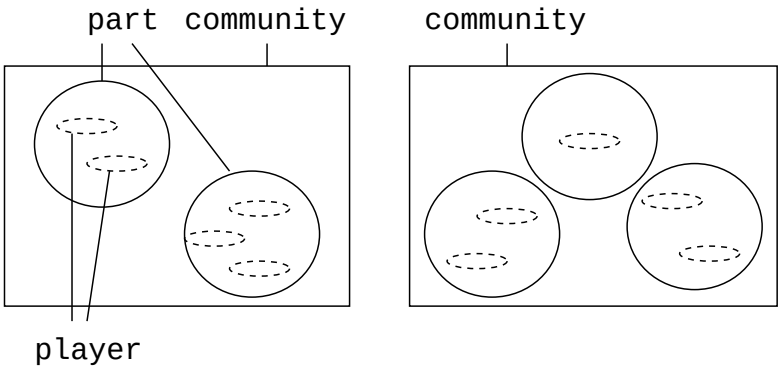

図 3.1: 基本構成要素の間の関係

ここで，人間のように外部に対して能動的に作用するアクティブな実体も，データベース・システムのように外部からの要求に反応して結果を返すリアクティブな実体も，本論文では区別せずにプレイヤとして扱う．以下では，全てのプレイヤの集合を

$$A$$

と表す．

プレイヤがサービスの提供／利用／仲介に関して担う役割を，パートと呼ぶ．個々のサービスについて，サービス提供者，利用者，仲介者は典型的なパートである．パートはユニークに識別可能であるとし，以下では全てのパートの集合を

$$R$$

と表す．

各々のパートには，プレイヤの集合が対応付けられる．ここで，１つのプレイヤが複数のパートに属し得るものとする．パートと，そのパートに属するプレイヤの対応を関数

$$g : R \to 2^A$$

によって表す．

3.1.3 オペレーションとコミュニティ

プレイヤが他のプレイヤに対して行う操作を，オペレーションと呼ぶ．例えば，サービスを利用するプレイヤがサービスを提供するプレイヤに対してサービスを要求する操作は，オペレーションである．RBAC [15] のオペレーションの概念と同様に，起動されるとプレイヤのために何らかの機能を提供するプログラムはオペレーションであり得る．また，オブジェクト指向方法論 [43] の観点からは，各プレイヤが外部に対して提供するメソッドがオペレーションに該当する．以下では，全てのオペレーションの集合を

$$OP$$

と表す．

各々のパートに属するプレイヤは，役割を遂行するために必要なオペレーションを共通して提供する．パート（すなわち，そのパートに属するプレイヤ）が提供するオペレーションの集合を関数

$$f : R \to 2^{OP}$$

によって表す．

コミュニティは，オペレーションを通じて関連するパートの集まりと定義する．コミュニティはユニークに識別可能である一方，パートは複数のコミュニティに同時に属することができるものとする．以下では，全てのコミュニティの集合を

$$C$$

とし，コミュニティ $c \in C$ を構成するパートの集合を

$$R_c$$

と表す．$R_c \subseteq R$ である．

コミュニティ c に包含されるプレイヤの集合 A_c は，次のように表すことができる．

$$A_c = \bigcup_{r \in R_c} g(r)$$

3.2 コミュニティのアクセス制御ポリシ・モデル

3.1 節で述べた用語にしたがうと，プレイヤ $a \in A$ がパート $r \in R$ に属するとき，a は他のプレイヤに対してオペレーションの集合 $f(r)$ を提供する．したがって，a に対するオペレーション $op \in f(r)$ の使用許可を他のどのプレイヤに対して与えるかを規定することで，a に関するアクセスを制御することができる．

本論文ではコミュニティを個々のプレイヤとは独立に論じるため，このようなパート r とオペレーション op の組をアクセス許可とするとともに，アクセス許可をパートに与え，アクセス許可とそれが与えられるパートの組によってアクセス制御ポリシを表現する．

3.2.1 アクセス許可

コミュニティを構成するプレイヤに対するアクセス許可を，パート R とオペレーション OP の間の 2 項関係 $PRM \subseteq R \times OP$ と定義する．ここで，任意の $r \in R$ と $op \in OP$ に対して，$(r, op) \in PRM$ ならば f が r について定義されかつ $op \in f(r)$ ．

$(r, op) \in PRM$ は，パート r に属するプレイヤのどれに対してもオペレーション op を行うための，許可を表す．

コミュニティ $c \in C$ に対して，c を構成するプレイヤに対するアクセス許可を PRM_c とすると，PRM_c は次のように表すことができる．

$$PRM_c = \{(r, op) \mid (r, op) \in PRM \wedge r \in R_c\}$$

3.2.2 コミュニティのアクセス制御ポリシ

任意のコミュニティ c に対して，コミュニティのアクセス制御ポリシ p_c を R と許可 PRM_c の間の 2 項関係

$$p_c \subseteq R \times PRM_c$$

と定義する．

p_c はパート間のアクセス許可を表し，任意のパート $X \in R$ に対して $(X, prm) \in p_c$ ならば，X に属するプレイヤに対してアクセス許可 prm が与えられることを表す．

図 3.2 は，$(X, (Y, op)) \in p_c$ のときの各要素の関係を模式的に示した図である．この図で，矢印はアクセス許可を表す．図 3.2 は，X に属するプレイヤに Y に属するプレイヤに対して op を行う許可が与えられていることを意味する．

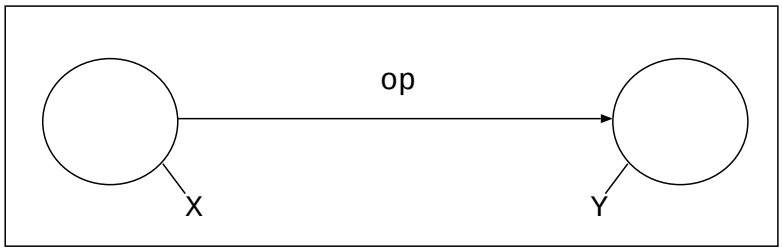

図 3.2: コミュニティのアクセス制御ポリシ

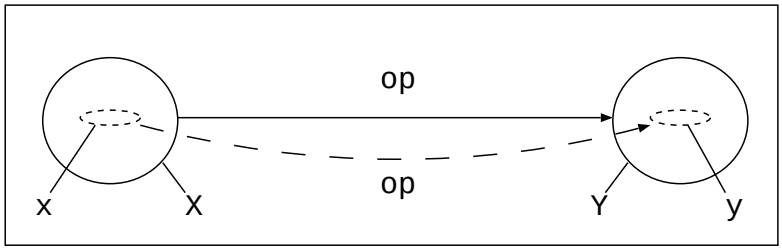

図 3.3: コミュニティのアクセス制御ポリシにしたがうプレイヤ

3.2.3 コミュニティのアクセス制御ポリシにしたがうプレイヤ

プレイヤ x が，次の条件が満たされるときに限ってコミュニティ c のプレイヤ y にオペレーション op でアクセスするとき，x は c のコミュニティのアクセス制御ポリシ p_c にしたがうプレイヤであると言う．

条件
$(X, prm) \in p_c$ となる X と prm が存在し，$X \in g(x)$ かつ $Y \in g(y)$ となる Y に対して $(Y, op) \in prm$ ．

図 3.3 は，コミュニティのアクセス制御ポリシとプレイヤによるアクセスの関係を模式的に示した図である．この図で，破線の矢印はプレイヤによるアクセスを表す．図 3.3 において，パート X に属するプレイヤ x がパート Y に属するプレイヤ y にオペレーション op でアクセスするとき，コミュニティのアクセス制御ポリシ $(X, (op, Y)) \in p_c$ であるならば，x はコミュニティのアクセス制御ポリシにしたがうプレイヤである．

3.3 コミュニティの連合のポリシ・モデル

複数のコミュニティから，各コミュニティのパートを包含する一つのコミュニティを構成することを，本論文ではコミュニティの連合と呼ぶ．連合によって構成されたコミュニティのアクセス制御ポリシは，3.2.2 節のコミュニティのアクセス制御ポリシで表現される．COAC モデルでは，各コミュニティのパートが持つアクセス許可を，他のコミュニティのパートに与えることで，連合によって構成されたコミュニティのアクセス制御ポリシを構成する．連合のポリシは，連合の際に，あるパートのアクセス

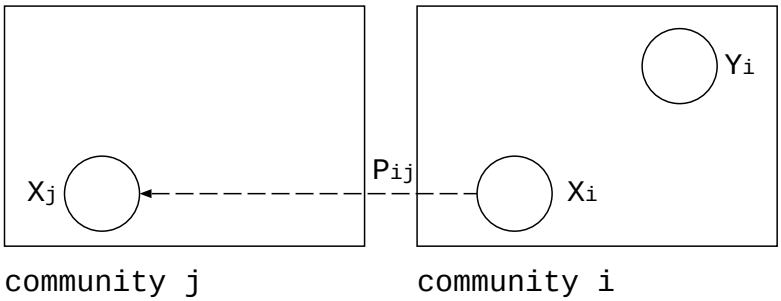

図 3.4: アクセス許可の委譲

許可を他のパートに与えることの可否を規定する．本論文では，アクセス許可を与えることを許可することを，アクセス許可の委譲と呼ぶ．

3.3.1 連合のポリシ

コミュニティmとnが連合する際のmからnへのアクセス許可の委譲を，パート間の 2 項関係

$$P_{mn} \subseteq R_m \times R_n$$

によって定める．

$(X, Y) \in P_{mn}$ は，X に与えられたコミュニティ m のパートに関するアクセス許可を，コミュニティ n のパート Y が持ち得ることとする．図 3.4 は，コミュニティiからjへのアクセス許可の委譲のようすを示す．破線の矢印はパート間のアクセス許可の委譲を表し，この例ではコミュニティiのパートX_iからコミュニティjのパートX_jにアクセス許可が委譲される．

コミュニティの連合では，アクセス許可の委譲関係は推移的であると定める．委譲関係の推移閉包 $P^\dagger$ を，連合のポリシと定義する．$P^\dagger$ は，連合に含まれる全てのコミュニティの組 m と n に対して以下を満たす最小の集合である．

- $P_{mn} \subseteq P^\dagger$ かつ $P_{nm} \subseteq P^\dagger$
- 任意のパート X，Y，Z に対して，$(X, Y) \in P^\dagger$ かつ $(Y, Z) \in P^\dagger$ ならば，$(X, Z) \in P^\dagger$

$(X, Y) \in P^\dagger$ は，X に与えられた各コミュニティのパートに関するアクセス許可を Y に委譲することを意味する．

3.3.2 隔離的な連合のポリシ

連合を構成する全てのコミュニティ c のパートの集合 R_c に対して，$X, Y \in R_c$ である全てのパートの組 X, Y について

$$X \neq Y \Longrightarrow (X, Y) \notin P^\dagger$$

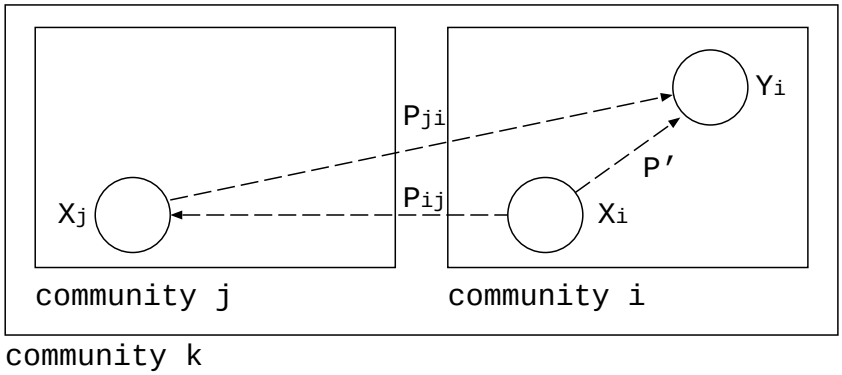

図 3.5: 隔離的でない連合のポリシ

であるとき，連合のポリシ $P^{\dagger}$ は隔離的であると呼ぶ.

隔離的な連合のポリシでは，連合を構成する各コミュニティの内部ではアクセス許可の委譲が起こらない．言い換えれば，連合を構成する各コミュニティの内部は，連合によってできるコミュニティのためのアクセス許可の委譲から隔離される.

隔離的でない連合のポリシの例を，図 3.5 に示す．この例では，コミュニティ i のパート X_i からコミュニティ j のパート X_j にアクセス許可が委譲されるとともに，コミュニティ j のパート X_j からコミュニティ i のパート Y_i にアクセス許可が委譲される．この結果,

$$(X_i, Y_i) \in P^{\dagger}$$

となり，コミュニティ i の内部のパート間でアクセス許可が委譲される.

3.3.3 連合のポリシにしたがうコミュニティの連合

m 個のコミュニティ $c_1, ..., c_m$ が連合して，コミュニティ c を構成するとする．c のアクセス制御ポリシを p_c，$c_1, ..., c_m$ のアクセス制御ポリシをそれぞれ $p_1, ..., p_m$ とすると，p_c が全ての $i(1 \leqslant i \leqslant m)$ について以下を満たすとき，この連合は連合のポリシ $P^{\dagger}$ にしたがう連合であると言う.

- $p_i \subseteq p_c$
- c_i のパート Y に対して $(X, (Y, op)) \in p_c$ ならば，
  $(X', X) \in P^{\dagger}$ かつ $(X', (Y, op)) \in p_i$ である X' が存在する.

図 3.6 は，コミュニティ i のパート Y_i に対してオペレーション op でアクセスするアクセス許可を X_i が持つとき，連合のポリシ $P^{\dagger}$ にしたがって連合したコミュニティ k では，コミュニティ j のパート X_j がそのアクセス許可を持ち得ることを示す.

3.3.4 連合におけるコミュニティのアクセス制御ポリシの分離

m 個のコミュニティ $c_1, ..., c_m$ が連合してコミュニティ c を構成するとする．c のコミュニティのアクセス制御ポリシを p_c，$c_1, ..., c_m$ のコミュニティのアクセス制御ポリ

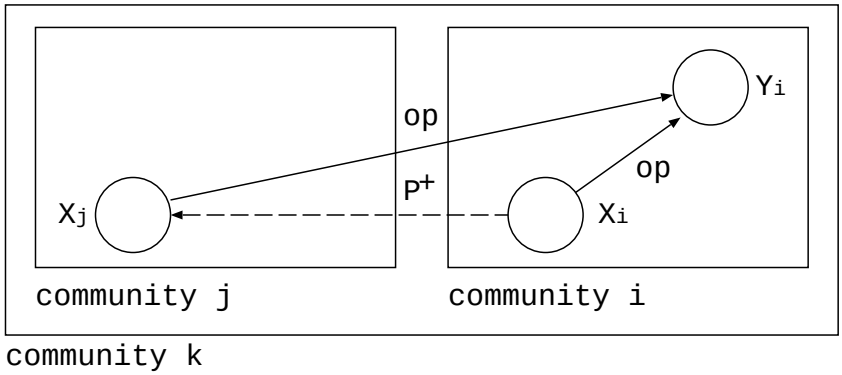

図 3.6: 連合のポリシにしたがうコミュニティ

シをそれぞれ $p_1, ..., p_m$，$c_1, ..., c_m$ のパートの集合をそれぞれ $R_1, ..., R_m$ とすると，全ての $i(1 \leqslant i \leqslant m)$ について以下が成り立つとき，この連合はコミュニティのアクセス制御ポリシを分離した連合であると言う．

$$\forall X, Y \in R_i;\ \forall op \in OP \mid (X, (Y, op)) \in p_i \Leftrightarrow (X, (Y, op)) \in p_c \tag{3.1}$$

すなわち，連合を構成する全てのコミュニティ c_i について，c_i の各パートに与えられた c_i の他のパートに関するアクセス許可が連合によって変化しない時，コミュニティのアクセス制御ポリシを分離した連合であると言う．

定理
隔離的な連合のポリシにしたがうコミュニティの連合は，コミュニティのアクセス制御ポリシを分離した連合である．

証明
連合のポリシにしたがう連合なので，

$$p_i \subseteq p_c.$$

すなわち，任意の $X, Y \in R_i$ と $op \in OP$ について

$$(X, (Y, op)) \in p_i \Longrightarrow (X, (Y, op)) \in p_c. \tag{3.2}$$

いま，ある $X' \in R_i$ に対して，

$$(X', (Y, op)) \in p_c \wedge (X', (Y, op)) \notin p_i \tag{3.3}$$

であると仮定する．このとき，連合のポリシにしたがう連合の定義から

$$(X', X'') \in P^{\dagger} \wedge (X'', (Y, op)) \in p_i \tag{3.4}$$

である $X'' \in R_i$ が存在しなければならない．ところが，隔離的な連合の定義から，$X' \neq X''$ ならば

$$(X', X'') \notin P^{\dagger}$$

であり，3.4 を満たす X'' は存在しない．したがって，どのような X' に対しても 3.3 は成り立たず，

$$(X,(Y,op)) \in p_c \Longrightarrow (X,(Y,op)) \in p_i. \tag{3.5}$$

3.2 と 3.5 から，3.1 が言える．

3.4 サービス仲介のアクセス制御ポリシの例

この節では，COAC モデルの表現力を検証するため，サービス仲介の基本パターンについて，COAC モデルに基づくアクセス制御ポリシの記述例を示す．サービス仲介の基本パターンとして，マッチメイキング・パターンを採り上げる．

3.4.1 マッチメイキングのパターン

マッチメイキングは，情報を必要とするエージェントに，必要な情報を有するエージェントを，１対１の関係の下で割り当てることを言う [22]．KQML（Knowledge Query and Manipulation Language）では，図 3.7 に示す５つのマッチメイキング・パターンが規定される [10]．マッチメイキングでは，要求者エージェントは必要な情報に関するメタデータを提示し，提供者エージェントは提供できる情報に関するメタデータを提示して，マッチメーカとなるエージェントが適合するエージェント同士を結びつける．図 3.7 では，A が要求者，B が提供者，F がマッチメーカである．矢印は情報の流れを表し，数字はその順番を表す．

(a) の point-to-point パターンはマッチメーカが関与しないパターンであり，A は必要な情報を提供する B を知っていて，直接 B に要求を送る．一方，(b) から (e) は A が B を知らない場合のパターンであり，F が A と B の間を仲介する．(b) の subscribe パターンでは，まず A が F に必要な情報に関するメタデータを提示し，B から F へ情報が送られた場合に F から A へ情報が転送される．(c) の broker パターンでは，B は提供できる情報に関するメタデータをあらかじめ F に提示し，F が A からの要求に対して B に情報を要求し，B からの情報を F が中継して A に送る．(d) の recruit パターンでは，B は提供できる情報に関するメタデータをあらかじめ F に提示し，F が A からの要求に対して B に情報を要求し，B は情報を直接 A に送る．(e) の recommend パターンでは，B は提供できる情報に関するメタデータをあらかじめ F に提示し，F は A からの要求に対して B を紹介する．このパターンでは，A は直接 B に情報を要求し，B からの応答を受け取る．

3.4.2 コミュニティのアクセス制御ポリシ

KQML のマッチメイキング・パターンの各々について，そのようなマッチメイキングを行うコミュニティを考える．コミュニティの中には，3.4.1 節の A，F，B に対応するエージェントが一般に複数あり，それらは各々コミュニティのマッチメイキング・パターンにしたがったマッチメイキングを行う．したがって，COAC モデルでは 3.4.1 節の A，F，B をパートとみなし，各々のパートに属するプレイヤ（エージェント）が

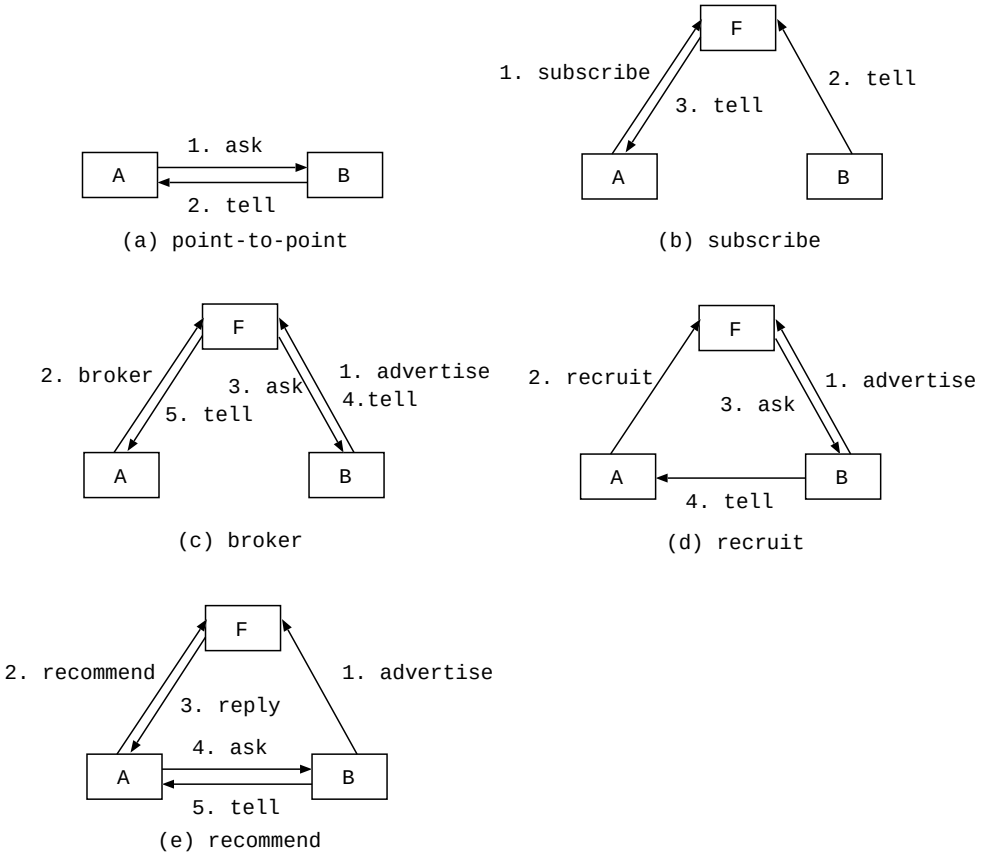

図 3.7: KQML のマッチメイキング・パターン

マッチメイキングを行うものとして，マッチメイキングのアクセス制御ポリシを規定すれば良い．また，ask のようにエージェント間で受け渡される語彙は，オペレーションとみなす．ここで，point-to-point パターンの ask に対する tell のように，オペレーションに対する応答となる情報の流れは，はじめのオペレーションに従属する．これらは新たなアクセスを発生させるものではないので，ファイルへの read オペレーションに対する応答と同様，アクセス制御ポリシには取り上げないこととする．

各マッチメイキング・パターンに対するコミュニティのアクセス制御ポリシの例を，以下に示す．

- point-to-point パターン
  A に対応するパートを AR，B に対応するパートを AP とする．

$$p_{point-to-point} = \{(AR, (AP, ask))\}$$

- subscribe パターン
  A に対応するパートを BR，F に対応するパートを BM，B に対応するパートを BP とする．

$$p_{subscribe} = \{(BR, (BM, subscribe)), (BP, (BM, tell))\}$$

- broker パターン
  A に対応するパートを CR，F に対応するパートを CM，B に対応するパートを CP とする．

$$p_{broker} = \{(CP,(CM,advertise)),(CR,(CM,broker)),\\ (CM,(CP,ask))\}$$

- recruit パターン
  A に対応するパートを DR，F に対応するパートを DM，B に対応するパートを DP とする．

$$p_{recruit} = \{(DP,(DM,advertise)),(DR,(DM,recruit)),\\ (DM,(DP,ask)),(DP,(DR,tell))\}$$

- recommend パターン
  A に対応するパートを ER，F に対応するパートを EM，B に対応するパートを EP とする．

$$p_{recommend} = \{(EP,(EM,advertise)),(ER,(EM,recommend)),\\ (ER,(EP,ask))\}$$

これらのコミュニティのアクセス制御ポリシを図式化したものを，図 3.8 に示す．

3.4.3 コミュニティの連合

連合の例として，3.4.2 節の broker コミュニティと recruit コミュニティの連合を考える．この場合，broker コミュニティの CR パートに属するプレイヤは recruit コミュニティの DP パートに属するプレイヤが提供する情報を間接的に利用し，recruit コミュニティの DR パートに属するプレイヤは broker コミュニティの CP パートに属するプレイヤから情報を受け取ることが許可されるものとする．このためには，アクセス許可の委譲を

$$p_{broker-recruit} = \{(CM,DM),(CP,DP)\}$$

$$p_{recruit-broker} = \{(DM,CM),(DP,CP)\}$$

とすれば良い．したがって，連合のポリシは

$$P^{\dagger} = \{(CM,DM),(DM,CM),(CP,DP),(DP,CP),\\ (CM,CM),(DM,DM),(CP,CP),(DP,DP)\}$$

と表せる．ここで，$P^{\dagger}$ は隔離的な連合のポリシの条件を満たす．すなわち，

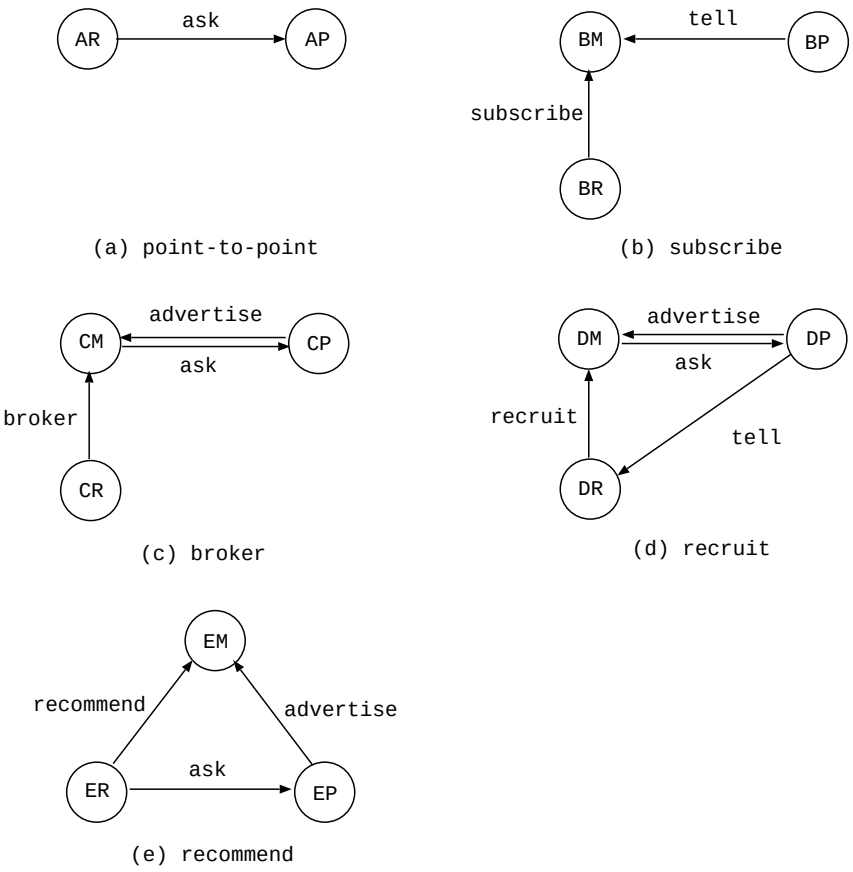

図 3.8: コミュニティのアクセス制御ポリシ

- broker コミュニティのパートの集合

  $$R_{broker} = \{CR, CM, CP\}$$

  の任意の要素 X と Y に対して，

  $$X \neq Y \Longrightarrow (X, Y) \notin P^{\dagger}$$

- recruit コミュニティのパートの集合

  $$R_{recruit} = \{DR, DM, DP\}$$

  の任意の要素 X と Y に対して，

  $$X \neq Y \Longrightarrow (X, Y) \notin P^{\dagger}$$

連合したコミュニティのアクセス制御ポリシ $p_{federation}$ を以下のように設定する．

$$\begin{aligned} p_{federation} \quad = \quad & p_{broker} \cup p_{recruit} \cup \\ & (CM, (DP, ask)), (CP, (DM, advertise)), (CP, (DR, tell)) \\ & (DM, (CP, ask)), (DP, (CM, advertise))\} \end{aligned}$$

3.3.3 節の定義より，この連合は連合のポリシ $P^{\dagger}$ にしたがう連合である．すなわち，

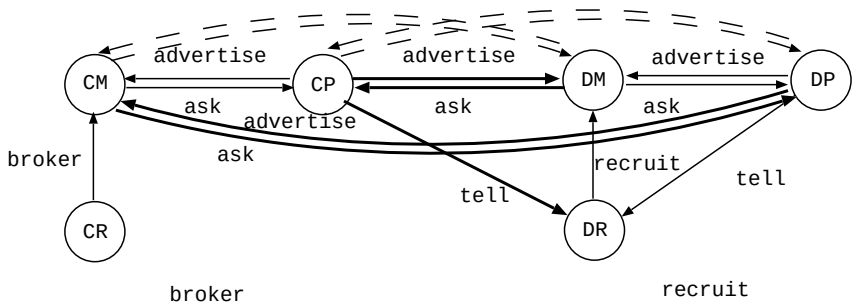

図 3.9: 連合したコミュニティのアクセス制御ポリシ

- $p_{broker} \subseteq p_{federation}$ かつ $p_{recruit} \subseteq p_{federation}$
- $(CM, (DP, ask)) \in p_{federation}$ に対して DM が存在し，
  $(DM, CM) \in P^{\dagger}$ かつ $(DM, (DP, ask)) \in p_{recruit}$
- $(CP, (DM, advertise)) \in p_{federation}$ に対して DP が存在し，
  $(DP, CP) \in P^{\dagger}$ かつ $(DP, (DM, advertise)) \in p_{recruit}$
- $(CP, (DR, tell)) \in p_{federation}$ に対して DP が存在し，
  $(DP, CP) \in P^{\dagger}$ かつ $(DP, (DR, tell)) \in p_{recruit}$
- $(DM, (CP, ask)) \in p_{federation}$ に対して CM が存在し，
  $(CM, DM) \in P^{\dagger}$ かつ $(CM, (CP, ask)) \in p_{broker}$
- $(DP, (CM, advertise)) \in p_{federation}$ に対して CP が存在し，
  $(CP, DP) \in P^{\dagger}$ かつ $(CP, (CM, advertise)) \in p_{broker}$

さらに，連合のポリシ $P^{\dagger}$ が隔離的であることから，この連合はコミュニティのアクセス制御ポリシを分離した連合である．

図 3.9 に，連合のポリシを破線で，連合によってできたコミュニティのアクセス制御ポリシを実線で示す．ここで，太い実線は連合によって付け加わるアクセス許可を表す．読み易くするため，この図では CM，CP，DM，DP の自分自身に対するアクセス許可の委譲は省略した．

3.4.4 COAC モデルの評価

3.4.2 節に示したように，KQML の 5 つのマッチメイキング・パターンについて，そのアクセス制御ポリシを COAC モデルで表現することができた．このことから，COAC モデルはサービス仲介の基本的なアクセス制御ポリシを表現するのに十分な表現力を持つと考えられる．

ここで，COAC モデルは RBAC モデルなどと同様に，アクセス順序に関する制約を表現することはできない．例えば recommend パターンで，オペレーション `advertise` が

recommend の前に行われることは，COAC モデルのコミュニティのアクセス制御ポリシからは読み取れない．しかし，個々の利用者や提供者を特定できないオープンなシステムでは，個々の利用者／仲介者／提供者についてアクセス順序を制約することはできない．サービス仲介のアクセス制御ポリシでは，集団としての利用者／仲介者／提供者についてアクセス制御ポリシを設定することが求められるので，この場合はアクセス順序に関する制約は意味を持たない．すなわち，利用者からの recommend 要求を受けた仲介者は，その時点で advertise している提供者を仲介すれば十分であり，特定の提供者 b が advertise した後に特定の利用者 a が recommend 要求を出すことをアクセス制御ポリシで規定する必要はない．

一方，3.4.3 節に示したように，COAC モデルではより複雑なコミュニティを連合によって構成する枠組みを提供する．

第4章　モデル記述言語

本章では，環境の変化に柔軟に対応する要素から構成されるシステムを表現することを目的とした，オープン・システムのモデルを記述するための言語 [28] について述べる．この言語は，5 章のコミュニティの実装モデルの記述言語となる．本章のモデル記述言語は，メッセージ送受信するエージェントを記述の基本単位とし，環境の変化への適応をエージェント間のメッセージ送受信の動的な組み替えによって表現する．エージェント間のメッセージ送受信は，場によって構造化する．エージェントの記述には階層化したメタ構造を用い，上位レベルから下位レベルを監視し操作することで，メッセージ送受信の柔軟な組み替えを表現する．

4.1 節では，コミュニティの実装モデルの基本構造となるエージェントと場の概念について述べる．4.1.1 節でメッセージ送受信を階層的に記述するしくみとしてのメタ記述について述べ，4.1.2 節でメッセージ送受信がどのように表現されるかを示す．モデル記述言語の構文の概要を付録 C に示す．

4.1　エージェント＋場モデル

本研究のモデル記述言語では，対象をエージェントと場の集まりによって表現する．各エージェントは固有の状態を持ち，自己の状態に応じてメッセージ送受信を行なう．また，エージェントは１つ以上の場に属し，属する場を変えることができる．場は有限個のエージェントを含み，エージェント間のメッセージ送受信の範囲を規定する．ここで，エージェントが直接メッセージ送受信できるのは同一場内のエージェントに限られる．同一場内のメッセージ送受信は，成功あるいは失敗のいずれかに定められる．他の場に属するエージェントとのメッセージ送受信は，複数の場に属するエージェントが協調し，場間でメッセージを中継することによって行なう事ができる．

エージェントと場の関係の概略を図 4.1 に示す．ここで，灰色の楕円はエージェントを，平行四辺形は場を表わし，楕円の平行四辺形への包含関係はエージェントの場への所属関係を表わす．また，縦の直線によって円筒状に結ばれた灰色の楕円は，複数の場に属する一つのエージェントを表わす．矢印はエージェント間のメッセージ伝達を示す．

図 4.2 は，エージェントの協調によるメッセージ伝達のようすを表わす．ここでは，エージェントａは場Ａに，エージェントｂは場ＡとＢに，エージェントｃは場ＢとＣに，エージェントｄは場Ｃに属する．ａが直接メッセージを送ることができるのは同一場内のｂに限られるが，ｂがｃに，ｃがｄにメッセージを中継することにより，ａは場Ａ，Ｂ，Ｃを経由してｄにメッセージを送ることができる．直観的には，場を知人関係として，ａが知人ｂにメッセージを託し，ｂがｃに，ｃがｄにメッセージを受

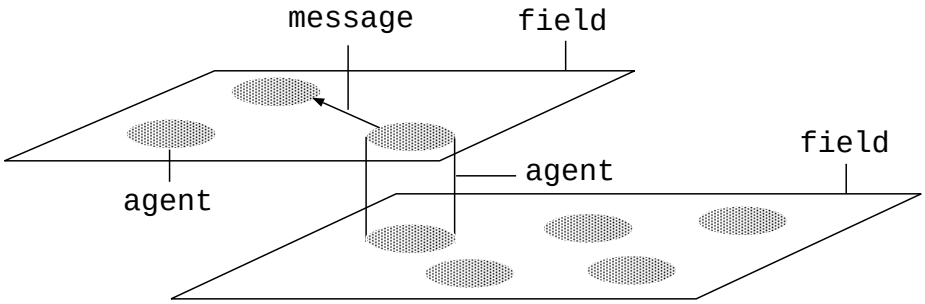

図 4.1: エージェント＋場モデル

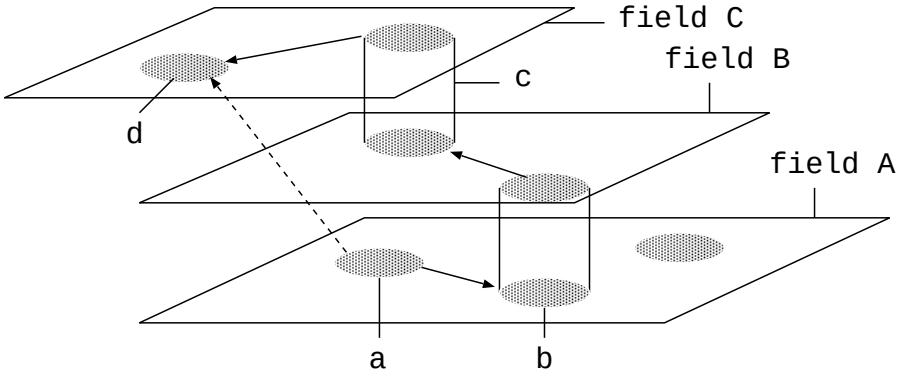

図 4.2: メッセージ伝達

け渡すことによって，ａから未知のｄへとメッセージが渡される，と解釈することができる．

エージェントは，エージェントの名前と所属する場によって識別することができる．しかし，場およびエージェントは消滅あるいは変化し得るので，そのエージェントに至るまでに経由する場や中継するエージェントは，常に同じとは限らない．一般には，目的エージェントとのメッセージ送受信毎に，中継するエージェントとのメッセージ送受信が必要となる．このようなメッセージ伝達のためのメッセージ送受信を階層的に表現し，エージェントや場の変化に柔軟に対応するエージェントを記述するしくみとして，メタ記述を導入する．

4.1.1 メタ記述

１つのエージェントを，0 レベルから n レベルまでの $n+1$ 階層に分けて記述する．各レベルには，以下の操作を記述することができる．

- そのレベルの内部状態の更新
- 同一レベル間のメッセージ送受信

さらに，$m+1$ レベル $(0 \leqslant m < n)$ では m レベルの実行状態の表現を持ち，これに対する操作を記述することができる．ここで，m レベルの実行状態は，m レベルの内部

状態とメッセージ送受信の進行状態から構成される．$m+1$ レベルの実行状態の表現はmレベルの実行状態と対応しており，前者に対する操作の結果は後者に直ちに反映される．すなわち，上位レベルが下位レベルの実行状態の表現を持ち，それに対する操作を行なうことができる点において，各レベルは結合している．

m レベルの実行状態は $m+1$ レベルからの操作によって更新され，それ以外の方法では更新されない．すなわち，m レベルで記述された内部状態の更新やメッセージ送受信は，$m+1$ レベルからの操作によって実行状態が更新されることで実行される．この意味で，$m+1$ レベルの記述は m レベルの記述に解釈を与えることになる．ここで，n レベルの記述の解釈は他の記述によって変えることはできない．したがって，n レベルの記述はエージェント＋場モデルに率直に従う．以下では，m レベルをベースレベル，$m+1$ レベルをメタレベルと相対的に呼ぶことにする．0 レベルは対象の記述であり，0 レベルをオブジェクトレベルと呼ぶ．記述言語上は n の値は記述の必要に応じて増やせるものとし，特には定めない．

ここでは，ベースレベルの実行状態を，ベースレベルのメッセージ送受信を単位として表現する．メタレベルにおけるベースレベルの実行状態の表現を，メッセージ閉包と呼ぶことにする．メッセージ閉包は，次の構造を持つ．
＜タグ，メッセージ・パケット，継続＞

- タグはメッセージの種類の表現．
- メッセージ・パケットは，ベースレベルのメッセージの表現．
- 継続は，ベースレベルの実行を継続するのに必要な情報の表現．

メッセージ・パケットは，次の構造を持つ．
＜受信先，送信元，メッセージ本体＞

- 受信先は，メッセージを受け取るエージェントのエージェント名パターン．
- 送信元は，メッセージを送るエージェントのエージェント名パターン．
- メッセージ本体は送信／受信メッセージのメッセージパターンの表現で，次の構造を持つ．
  ＜メッセージ識別子，引数のリスト＞

継続は，次の構造を持つ．
＜局所環境，履歴＞

- 局所環境は，メッセージが評価されるベースレベルの内部状態や処理の進行状態など．
- 履歴はメッセージに対する返信を評価する局所環境の退避スタック．

メッセージ閉包は，メタレベルのメッセージ・キューに格納される．ベースレベルのメッセージ送受信を統一的に扱うため，メッセージ閉包は送信側エージェントのメッセージ・キューに一旦格納された後，受信側エージェントのメッセージ・キューに渡される．

この構造の下では，メタレベルからの以下の操作の記述により，ベースレベルの特性の変更を表現することができる．

1. メッセージ・キューの操作
   - メッセージ処理順序の変更
   - メッセージの削除
2. メッセージ・パケットの操作
   - メッセージの転送
   - メッセージの内容の変更
   - 返信メッセージの送り先の変更
3. 継続の操作
   - メッセージ送信から返信メッセージが到着するまでの間の処理の続行／停止

4.1.2 メッセージ送受信

ベースレベルのメッセージ送受信は，メタレベルのメッセージ閉包の受け渡しによって表現される（図 4.3）．例えば，エージェントａからエージェントｂへの m レベルのメッセージ送信は，$m+1$ レベルの次の記述によって表現される．

1. ａの $m+1$ レベルのメッセージ・キューからメッセージ閉包を取り出す．
2. メッセージ閉包をｂに伝達する．
3. メッセージ閉包をｂの $m+1$ レベルのメッセージ・キューに受信メッセージとして格納する．
4. ｂの $m+1$ レベルのメッセージ・キューからメッセージ閉包を取り出し，解釈する．

メタレベルの情報はベースレベルからは操作することができず，メタレベルで行われる操作はベースレベルからは完全に隠蔽される．したがって，メタレベルでメッセージ閉包がエージェントａからエージェントｃを経由してエージェントｂに渡されたとしても，ベースレベルではエージェントａからエージェントｂへメッセージが送られたこと以上の影響を受けることはない．この結果，例えば図 4.2 のメッセージ伝達の場合，エージェントａから受け取ったメッセージ閉包をエージェントｂ，ｃが中継してエージェントｄに渡すようメタレベルに記述することにより，ベースレベルではエージェントａからｄへ直接メッセージを渡す記述を行なうことができる．

メッセージ閉包はメタレベルではデータであり，その伝達はメッセージ送受信によって行われる．したがって，ベースレベルのメッセージ送受信の記述に対し，メタレベルでは，他エージェントの状態を把握して最適なエージェントにメッセージ閉包を渡すことができる．言い換えれば，メタレベルの記述はベースレベルの記述の実現方法についての記述であり，ベースレベルのメッセージ送受信を操作対象とすることができる．ベースレベルとメタレベルの間のこのような関係は，エージェント間の協調のように場や他のエージェントの状態に強く依存する処理を，そうでない部分から分離して記述する上で有効に機能する．

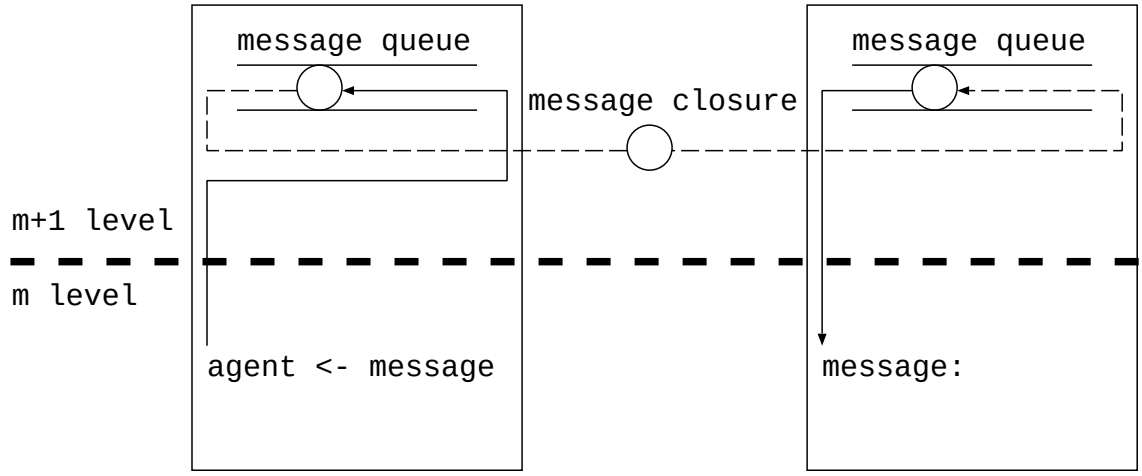

図 4.3: メッセージ送受信のモデル

4.2 モデル記述言語の記法

記述言語の構文規則を，付録 C に示す．エージェント記述子は記述上の識別子である．メタレベルでは，オブジェクトレベルのエージェント名に対してレベルの数だけ m を付けたものをエージェント記述子とする．メッセージパターンには空メッセージの記述も許すものとし，アクティブなメソッドは空メッセージによって起動されるパッシブなメソッドとして記述する．空メッセージのメッセージパターンは [] とし，対応するメッセージ閉包は次の形式とする．

```
<receive,[],[]>
```

また，エージェントへの返信は，メッセージ識別子が return のメッセージ送信によって表す．パターン式は変数への代入の省略記法であり，左辺のパターンに現れる変数と右辺のパターンに現われる変数に，各々対応する値が代入される．式の値は，変数への適切な代入によって左辺と右辺が等しくなれば真，そうでなければ偽とする．代入文中のパターン式の値が偽となる場合は，代入文はエラーとなる．整数，文字などの基本データ型の値は各々エージェントとみなし，これらのエージェントは，各場で暗黙のうちに定義されているものとする．また，次の 2 つの式は，以下の値をとる．

- forsome 変数名 with（ 式 1 ） 式 2

  変数名で示される変数の，式 1 を満たす少なくとも 1 つの値について，式 2 を評価した結果が真ならば真．値の探索は逐次的に行うものとする．

- forall 変数名 with（ 式 1 ） 式 2

  変数名で示される変数の，式 1 を満たす全ての値について，式 2 を評価した結果が真ならば真．

各レベルの各メソッドでは，次の特別変数を使うことができる．

- field: そのエージェントの属する場の集合

- msg_queue: そのエージェント自身のメッセージ・キュー
- self: 自身を示すエージェント名パターン
- sender: そのメソッドを起動したエージェントを示すエージェント名パターン

m レベルの記述は，m レベルのメッセージ・パケットと継続を引き数とする $m+1$ レベルのメソッド execute によって実行される．

execute <メッセージ・パケット> <継続>

このメソッドは，以下の処理を行なう．

1. 引き数の局所環境が空（新規メッセージ）の場合，mレベルの属性の値とメッセージパターンにマッチするメソッドから局所環境を作る．
2. メッセージ本体を局所環境の下で評価する．
3. 評価の過程でメッセージ送信が現れたならば，その時の局所環境からメッセージ閉包を作って処理を終わる．ここで，メッセージ閉包の局所環境にはその時の環境を，履歴には引き数として与えられた履歴の値をそのまま格納する．

例えば，場Fのエージェント A が次のように記述された m レベルメソッド example を持つとする．このメソッドは，場Fのエージェント B にメッセージ a_message を送り，その結果に 1 を足した値を返す．

```
example:
  let x = 1;
  y = ( B.F <- a_message );
  sender <- return x+y
```

エージェント A に場Fのエージェント C から m レベルでメッセージ example が送られた時，その実行は $m+1$ レベルのメソッド execute によって次のように進行する．

1. execute
   <A.F, C.F, <example, []>>
   <[], History>
   の形式でメソッド execute が呼び出される．ここで，History はメッセージ閉包に格納された履歴．
2. execute は，メッセージ本体＜ example, [] ＞にマッチする m レベルのメソッド
   example と属性の値から局所環境を作る．説明上，ここでは仮に局所環境を

   <State, Example>

   とする．State は m レベルの属性名と値の束縛のリスト，Example はメソッド example のコードである．

3. <State, Example>

   の下でメッセージを評価する.

4. エージェントBへのメッセージ送信が出現した時点で, execute の処理を終わる. execute は次のメッセージ閉包を返す.

```
<send, <B.F, A.F, <a_message, []>>,
 <<[x=1, sender=C.F, State], Cont>,
  History>>
```

   ここで, [x=1, sender=C.F, State] は State に束縛 x=1 と sender=C.F が加わったことを示す. また, Cont は, 返信を受け取った後に example の処理を継続するためのコードである.

5. エージェントBからの返信によって, example の処理を再開する. メソッド execute は次の形式で呼び出される.

```
execute
  <A.F, B.F, <return, [Result]>>
  <<[x=1, sender=C.F, State], Cont>,
   History>
```

   ここで, Result はエージェントBからの戻り値である.

6. <[x=1, sender=C.F, State], Cont>

   の下で Result を評価する.

7. 送信元へのメッセージ返信が出現した時点で, execute の処理を終わる. execute は次のメッセージ閉包を返す.

```
<send, <C.F, A.F, <return, [Value]>>,
 <[x=1, sender=C.F, State],
  History>>
```

   ここで, Value は Result+1 を計算した結果である.

4.3 記述例

4.1 節の記法に基づいて, メタレベルの記述例を 2 つ示す. 1 つはベースレベルの同一場内のメッセージ送受信に関する例である. もう 1 つは場を越えたメッセージ送受信に関する例であり, 同一場内のコミュニケーションの記述例を基に, ベースレベルのメッセージ送受信を異なる場に属するエージェント間にまで拡張する.

4.3.1 同一場内のメッセージ送受信

同一場内のエージェント間のメッセージ送受信を表現する，メタレベルの記述例を付録 D.1 に示す．この記述例では，目的エージェントの指定形式を除いて，メタレベルはベースレベルに依存しない．したがって，このメタレベルには様々なベースレベルを組み合わせることができる．逆に，ベースレベルからはメタレベルを意識する必要はない．この意味で，メタレベルとベースレベルは分離している．ここでは，エージェント名を仮に AGENT とした．各メソッドは，以下の処理を行なう．

- []
  メッセージ・キューの先頭要素から順に送受信処理を行なう．
- send_closure
  目的エージェントにメッセージ閉包を伝達する．
- receive_closure
  メッセージ閉包を受け取ってメッセージ・キューに格納する．

ここで，ベースレベルでは目的エージェントを「エージェント名．場の名前」の形式で指定するものとする．

また，メタレベルからはベースレベルの実行に関わる特性を陽に操作することができる．この記述例では，ベースレベルのメッセージ送信および実行の継続は次のように規定される．

- ベースレベルのメッセージ送信は，メタレベルのメッセージ・キューからメッセージ閉包が取り出され，その伝達が成功するかあるいはエラー処理が行われることによって終了する．
- メッセージを送信したベースレベルのメソッドの実行は，メッセージ閉包が作成された時点で停止し，メッセージ・キューから返信メッセージ閉包が取り出されて
  execute が呼び出された時点で再開される．

したがって，ベースレベルのメッセージ送信と返信の受信の間の実行の継続は，メタレベルでは execute を呼び出す位置によって表現される．このため，例えばメッセージ送信が終わった時点で execute を呼び出すよう記述を変更すれば，ベースレベルではメッセージ送信の後，返信を待たずに実行を続行させることになる．

また，ベースレベルのメソッドの実行順序は，メタレベルではメッセージ・キュー内の順序によって表現される．したがって，ベースレベルの特定のメソッドの優先実行は，キュー内のメッセージ閉包の順序を入れ替える記述を加えることによって表現される．また，記述例のように，キュー中の送信／返信メッセージと新規メッセージを区別せず受け付ける場合には，複数のメソッドを同時に起動する可能性を許すことを表わす．

4.3.2 場を越えたメッセージ送受信

図 4.2 のような，エージェント間の協調によるメッセージ伝達を考える．ここで，異なる場の間でメッセージを受け渡すエージェントをファシリテータと呼ぶ．エージェントのメタレベルの記述例を付録 D.2 に示す．付録 D.2 の m(AGENT) のメソッドは，付録 D.1 の m(AGENT) のメソッドに対する変更部分を示す．ファシリテータのメタ記述は，これにさらに m(FACILITATOR) のメソッドを加えたものである．

この記述例では，まずメソッド send_closure によって，所属する全ての場について「エージェント名」のエージェントを探す．もしそこに「エージェント名」に適合するエージェントが見つからなければ，場内の全てのエージェントに forward_message メッセージを送って転送を依頼する．forward_message メッセージを受け取ることのできるエージェントは，同様の手順で次々と転送の範囲を広げて行く．

この記述例も，目的エージェントの指定方法を除いて，メタレベルはベースレベルに依存しない．ここで，メッセージ閉包の中継はメタレベルで行われ，目的エージェントに到達した時点で初めてベースレベルの受信メッセージとしてメッセージ・キューに格納される．したがって，ベースレベルでは，目的エージェントが同一場内に存在するかどうかに無関係に，目的エージェントへの直接のメッセージ送信として記述される．

4.4 環境の変化への適応

環境の変化に適応するエージェントの記述における，メタ記述と場の効果について触れる．4.1.1 節で述べたように，メタレベルからはベースレベルの次の項目を操作することができる．

- メッセージの送り先
- メッセージの内容
- メッセージの処理順序

したがって，メタ記述によって，実行時の環境に合わせてベースレベルのメッセージの転送を行なうことができる．例えばメッセージの送り先は，実行時の環境に合わせて受信側エージェントを動的に探し出す方法をメタレベルに記述することにより，エージェント名と場の名前による指定だけではなく，受信側エージェントの持つメソッド名や属性の値など実行時の環境により強く依存する形式で指定することもできる．4.3.2 節の例では全てのメッセージが同様に転送されるが，転送するメッセージや受信側エージェントへの制約をメタレベルに陽に記述することにより，特定のメッセージあるいは特定の受信側エージェントについて転送の方法を変えることも可能である．メタ記述では，上記の操作によって表現できる範囲内で，環境の変化に合わせたエージェント間のメッセージ送受信の変更を記述することができる．さらに，m レベルの記述では適応できない変化への適応を $m+1$ レベルを用いることによって記述することも構造上は可能である．しかし，一般にメタレベルが上がるにつれて記述は複雑になるので，高いメタレベルのもとで多数のエージェントの間のメッセージ送受信を正確に記

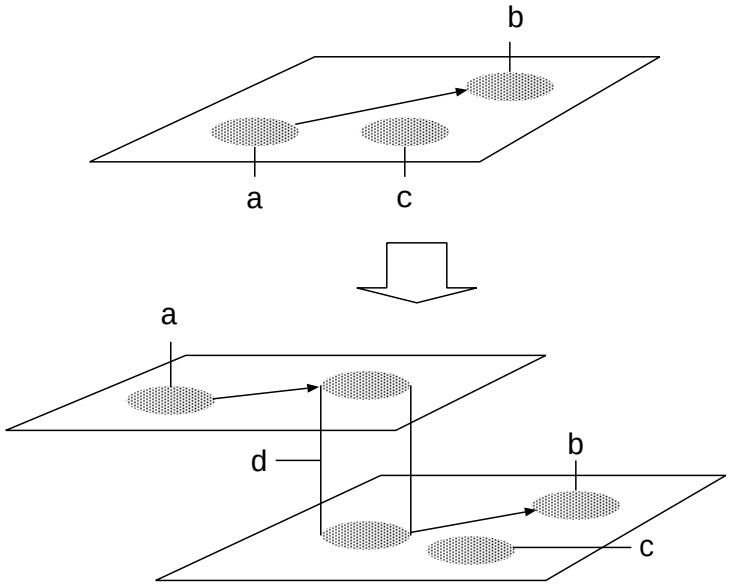

図 4.4: メッセージ送受信の構造化

述することは困難であり，この点では，環境の変化への適応を各エージェントに記述したメタ記述のみで行なうことには限界がある．

エージェント間のメッセージ送受信の，より微細な組み替えは，場を用いてメッセージ送受信を構造化することによって行なう．これは，場を用いることによって変化の影響が及ぶ範囲を限定するとともに，複数の場に属するエージェントを設定することによって，メッセージ送受信の変更をこのエージェントに局所化することで行なう．図 4.4 は，場を用いたメッセージ送受信の構造化の概略を示す．この例では，同一場内にあって直接メッセージを送受信していたエージェントａ，ｂ，ｃに対して，エージェントａを別の場に分離し，ファシリテータｄを介してｂ，ｃとメッセージ送受信するようにメッセージ送受信を構造化している．この結果，ａとｂ，ｃ間のメッセージ送受信は常にｄを経由することになり，ｄのメタレベルからの操作によってａとｂ，ｃ間のメッセージ送受信に介入し，それらを変更することが可能になる．例えば，ａからｂへ送られたメッセージｘを，ｄの操作によってｃへのメッセージｙに変更し，ｙに対するｃからの返信をメッセージｘに対する返信としてａに返すことができる．特に，ｄが転送メッセージの一部をｄ自身のベースレベルへのメッセージ受信に書き替えることにより，ベースレベルではａ，ｂ，ｃ，ｄ間のメッセージ送受信が行なわれることになり，ｄのベースレベルを使って機能を拡張することが可能となる．ここで，エージェントａ，ｂ，ｃが 4.3.2 節の例のようにファシリテータを介して場間のメッセージ送受信を行なうメタ記述を持っていれば，このような組み替えに対してａ，ｂ，ｃが変更される必要はない．

上記のように，本論文の記述言語では，環境の変化に適応するためのメッセージ送受信の組み替えを行なうために，基本的に次の２つの方法を提供する．

- メッセージの転送
- ファシリテータによるメッセージの変更

これらによって，エージェントが他のエージェントの生成や消滅などの環境の変化に適応するための機能を記述する枠組みを与えるとともに，各エージェントに記述された適応機能を利用して，エージェントの集団の一定の環境の下での機能を変化させる方法を与える．各エージェントの適応メカニズムの記述は，このような変化に対して各エージェントがどのように機能するかを規定する．

第5章　COACフレームワーク

この章では，3 章で提案した COAC モデルのアクセス制御ポリシを実現するコミュニティの論理的なシステム・アーキテクチャを示し，その中のアクセス制御を設計するための COAC フレームワークを提案する．まず，5.1 節に COAC モデルにしたがうコミュニティを実現するための課題を挙げ，5.2 節で論理的なシステム・アーキテクチャを示し，コミュニティの実現プロセスについて述べる．5.3 節では，COAC フレームワークを 4 章の記法を使って表す．COAC フレームワークを使って記述されたコミュニティの実現モデルが，コミュニティのアクセス制御ポリシの実現であるための条件を，5.5 節に示す．

5.1　コミュニティの実装における課題

COAC モデルのコミュニティを実現するためには，ポリシにしたがったインタラクションを行うプレイヤが実装されていなければならない．しかし連合を通じて変化するコミュニティの場合には，連合先のプレイヤにアクセスするための多様なインタラクションを，短時間で多数のプレイヤに実現することは困難である．連合を含むコミュニティを効率的に構成するためには，以下の課題を解決する必要がある．

1. 連合関係がプレイヤに及ぼす影響の局所化
2. 連合関係の変化に柔軟に対応するための枠組の提供
3. 各コミュニティのプレイヤのふるまいがコミュニティのアクセス制御ポリシを損なわないことの保証

（1）はスケーラビリティへの対応の点からの課題である．プレイヤは所属するコミュニティのアクセス制御ポリシにしたがう必要があるため，コミュニティが連合する際には連合先のパートの組合せに対してアクセスを調整することが求められる．コミュニティをスケーラブルに実現するためには，このような調整を個々のプレイヤとは独立に行うためのしくみを持つ設計フレームワークが必要となる．

（2）はオープン環境への適応の点からの課題である．Web サービス・システムでは提携関係の変化などによる連合関係の変更や拡張がしばしば起こり得るので，それに対応する際のシステムの変更が少ないことが求められる．このためには，連合関係の変更に対してシステムの特定箇所の変更で対応できる，設計フレームワークが必要である．

（3）は検証の点からの課題であり，システムが連合したコミュニティのアクセス制御ポリシを遵守していることの保証が求められる．そのためには，コミュニティを実現するシステムの仕様レベルで検証できる設計フレームワークが必要である．

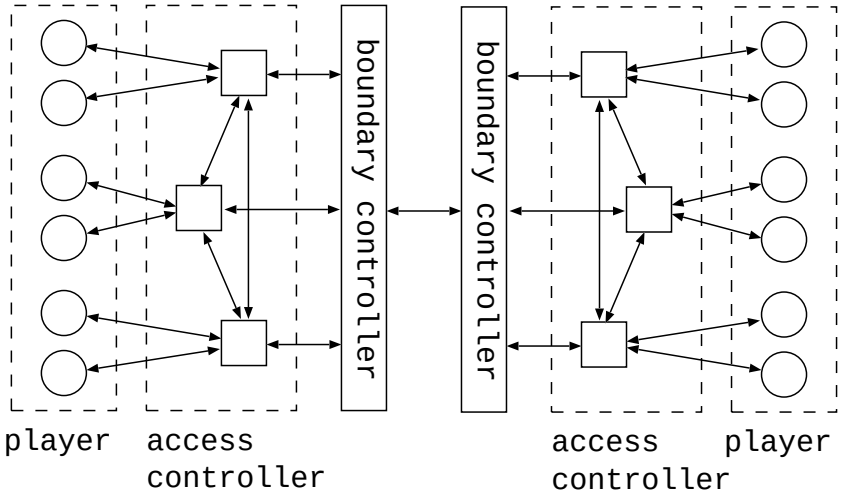

図 5.1: 論理的なシステム・アーキテクチャ

5.2 コミュニティの実現プロセス

本節ではまず，コミュニティを実現する論理的なシステム・アーキテクチャを示す．次に，このシステム・アーキテクチャに適合するコミュニティの実現モデルを示し，その記述過程を示す．

5.2.1 論理的なシステム・アーキテクチャ

コミュニティを実現する論理的なシステム・アーキテクチャを，図 5.1 に示す．システム・アーキテクチャは，パートのレベルのアクセスを制御するアクセス・コントローラと，コミュニティに対応するサブシステム間のアクセスを制御するバウンダリ・コントローラから構成される．

アクセス・コントローラの機能には，以下のものが含まれる．

- パートに属するプレイヤの認証
- パートに与えられた許可の範囲内のプレイヤ間メッセージ送受信の通過
- 許可の範囲外のプレイヤ間メッセージ送受信の拒否

また，バウンダリ・コントローラの機能には，以下のものが含まれる．

- 他サブシステムからのメッセージの許可／拒否
- 他サブシステムからのメッセージの自サブシステムのメッセージへの翻訳
- 他サブシステムからのメッセージの記録

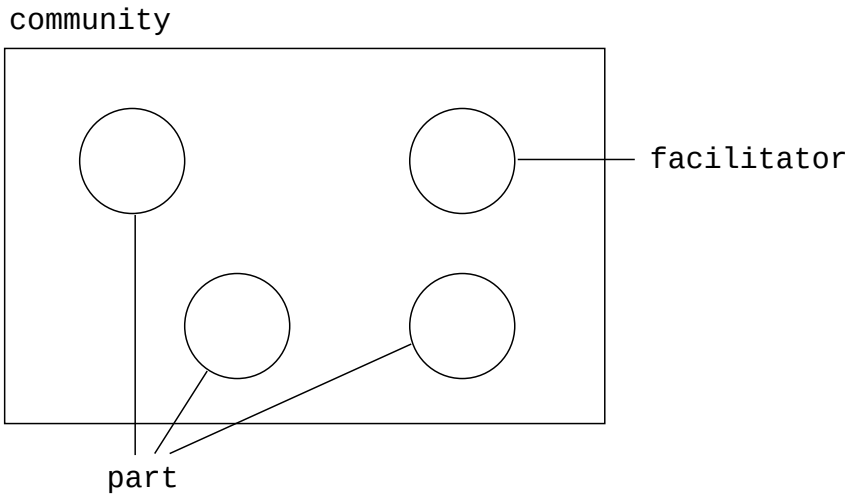

図 5.2: コミュニティの実現モデルの構成要素

5.2.2 コミュニティの実現モデル

コミュニティの実現モデルは，ポリシにしたがうプレイヤから構成されるコミュニティを，4 章の場とエージェントによって表現したモデルである．コミュニティの実現モデルの基本構成要素を，図 5.2 に示す．

コミュニティの実現モデルでは，パートをエージェントを使って表現する．また，コミュニティを場を使って表現する．それぞれの場には，場間のメッセージ送受信を管理するファシリテータを一つ置く．ファシリテータも，エージェントを使って表現する．ファシリテータは，複数の場に属して各場のファシリテータとメッセージ送受信を行うことができる．連合は，ファシリテータを介して複数の場を接続することで表現する．

コミュニティの実現モデルの各エージェントの機能は，論理的なソフトウェア・アーキテクチャのアクセス・コントローラの機能に対応する．また，各ファシリテータの機能は，バウンダリ・コントローラの機能に対応する．

5.2.3 コミュニティの実現プロセス

コミュニティの実現プロセスを，図 5.3 に示す．コミュニティの実現プロセスでは，まずポリシにしたがうコミュニティをモデル化した実現モデルを作成し，エージェント間のインタラクションがコミュニティのアクセス制御ポリシに違反しないことを検証する．その後，コミュニティの実現モデルに表現されたメッセージ送受信を行う各部分を抽出し，アクセス・コントローラおよびバウンダリ・コントローラとして実現する．コミュニティは，最終的には 2.2.1 節の Web サービス技術を使って実装される．実装の過程は各々の実装環境に依存して様々であるので，本論文ではコミュニティの実現モデルにおけるアクセス制御ポリシの実現について論じるにとどめる．実現モデルに表現されたアクセス制御機構を Web サービス技術を使って実装することは，コミュニティの実現モデルと論理的なシステム・アーキテクチャの対応から，アクセス・コ

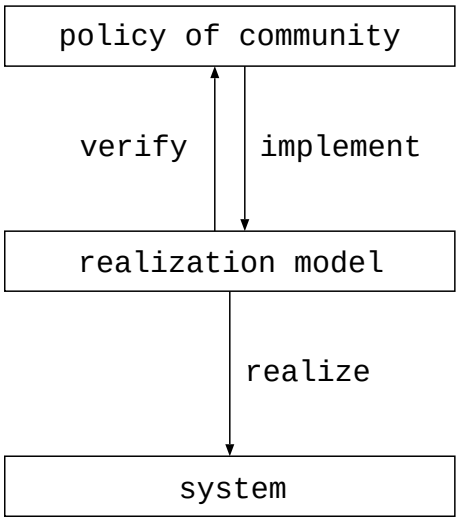

図 5.3: コミュニティの実装プロセス

ントローラおよびバウンダリ・コントローラを Web サービス技術を使って実装することに対応付けられ，個々の実装技術の問題と位置付けられる．

5.2.4 コミュニティの実現モデルの記述過程

コミュニティの実現モデルは，パートに属するプレイヤが行う次の 2 つのインタラクションを，エージェント内でメタ階層を使って分離して表現する．

- サービスの提供／利用／仲介に直接関わるインタラクション
- 連合の際のコミュニティの協調に関わるインタラクション

コミュニティの実現モデルの記述は，基本的に以下の段階からなる．

1. サービスの提供／利用／仲介のための基本的なメッセージ送受信のモデル化
2. コミュニティの協調に関わるメッセージ送受信のモデル化
3. 連合したシステムでのサービスの提供／利用／仲介のためのメッセージ送受信のモデル化

ここで，コミュニティの協調に関わるメッセージ送受信は，各コミュニティのサービス提供／利用／仲介の形態によらず共通部分を考えることができる．COAC フレームワークは，コミュニティの協調をモデル化するための骨格を提供する．

5.3 COAC フレームワークの構成

COAC フレームワークは，次の 3 つの要素から構成される．

- コミュニティを表す場

- コミュニティ内のパートを表すエージェント
- コミュニティを結合するファシリテータ

サービスの提供／利用／仲介のための処理は，各エージェントのベースレベルに記述する．これに対し，コミュニティ間の協調に関する処理は，メッセージを操作できる点から，各エージェントおよびファシリテータのメタレベルに記述するのが適切である．COAC フレームワークは，コミュニティの実現モデルの記述において，エージェントおよびファシリテータのメタレベル記述のためのフレームワークを与える．4 章のモデル記述言語で表現した COAC フレームワークを，付録 E に示す．

5.3.1 エージェントのメタレベル記述

各エージェントのメタレベルでは，ベースレベルのメッセージ送受信のための処理を行う．これは，以下の処理から構成される．

- メッセージの送信処理
  - 送信メッセージがコミュニティのアクセス制御ポリシにしたがう場合，以下を行う．
    1. 受信エージェントが同一場内にあるとき，メッセージを伝達する．
    2. 受信エージェントが同一場内にないとき，ファシリテータにメッセージの転送を依頼する．
  - コミュニティのアクセス制御ポリシにしたがわない場合には，送信メッセージを失敗させる
- メッセージの受信処理
  - 受信メッセージが正当なメッセージである場合，ベースレベルのメソッドを呼び出して実行する．
  - メッセージが不当な場合，不明なメッセージであることを送信エージェントに通知する．

モデル記述言語による記述を，付録 E.1 に示す．

5.3.2 ファシリテータのメタレベル記述

各ファシリテータのメタレベルでは，ベースレベルのメッセージを場間で転送する処理を行う．これは，以下の処理から構成される．

- 管理する場内に受信エージェントがある場合，以下を行う．
  - 受信メッセージが返信ならば，ポリシをチェックせずにメッセージを伝達する．

  - 受信メッセージがコミュニティのアクセス制御ポリシにしたがうならば，メッセージを伝達する．
  - コミュニティのアクセス制御ポリシにしたがわない場合には，受信メッセージを失敗させる．

- 管理する場内に受信エージェントがない場合，他の場を管理するファシリテータに転送を依頼する．

モデル記述言語による記述を，付録 E.2 に示す．

5.4 実現モデルの解析

COAC フレームワークを使って構成された実現モデルのメッセージ送受信は，以下のように解析することができる．ここでは，メタ階層のレベルを示す 0 以上の整数の集合を N，メッセージ・パターンの集合を M，場の集合を F とする．

5.4.1 メッセージの実行

$i \leqslant n$ レベルのメッセージ処理を示す関数

$$acceptable : M \times A \times N \rightarrow bool$$

を次のように定義する．

- エージェント a の i レベルに m を処理するメソッドが記述されているとき
  $acceptable(m, a, i) = true$
- それ以外のとき
  $acceptable(m, a, i) = false$

i レベルのメッセージは $i+1$ レベルから操作可能であるため，メッセージ実行を示す関数

$$executable : M \times A \times N \rightarrow bool$$

を次のように定義する．

- $i = n$ のとき
  $executable(m, a, i) = acceptable(m, a, i)$
- $i < n$ のとき
  $i+1$ レベルで `execute` が陽に適用される場合，
  $executable(m, a, i) = acceptable(m, a, i)$
  $i+1$ レベルからの操作により i レベルで m が m' に書き換えられる場合，
  $executable(m, a, i) = acceptable(m', a, i)$
  それ以外の場合，
  $executable(m, a, i) = false$

5.4.2 メッセージの伝達

場 f_1 にあるエージェント a_1 が，場 f_2 にあるエージェント a_2 に i レベルでメッセージ m を送信する場合を考える．このメッセージは，$i+1$ レベルからの操作によって a_2 以外のエージェントに渡されて実行され得る．場 f にあるエージェント a へのメッセージ伝達を示す関数

$$transfer : M \times (A \times F) \times (A \times F) \times N \rightarrow bool$$

は次のように定義することができる．

$$\begin{aligned} &transfer(m, (a_1, f_1), (a, f), i) = \\ &forward(wrap(m), (a_1, f_1), (a, f), i+1) \wedge \\ &executable(m, a, i) \end{aligned}$$

i レベルのメッセージ m_i に対して，$wrap(m_i) \in M$ は m_i に対応するデータを $i+1$ レベルで受け渡すメッセージを表す．$i+1$ レベルに記述がない場合には，全ての $a \in A$ に対して $acceptable(wrap(m_i), a, i+1) = true$ とする．

$$forward : M \times (A \times F) \times (A \times F) \times N \rightarrow bool$$

は上位レベルでのメッセージ受け渡しを示す関数で，場 f_1 のエージェント a_1 から場 f_2 のエージェント a_2 への $i+1$ レベルのメッセージ m_{i+1} に対して，次のように定義できる．

- $i = n$ のとき
  $f = f_1 = f_2$ かつ $a = a_2$ の場合，
  $forward(m_{i+1}, (a_1, f_1), (a, f), i+1) = true$
  それ以外の場合，
  $forward(m_{i+1}, (a_1, f_1), (a, f), i+1) = false$

- $0 \leqslant i < n$ のとき
  メッセージを転送するエージェント a_3 が存在する場合，

  $$\begin{aligned} &forward(m_{i+1}, (a_1, f_1), (a, f), i+1) = \\ &forward(m_{i+1}, (a_1, f_1), (a_3, f_3), i+1) \wedge \\ &transfer(m_{i+1}, (a_3, f_4), (a, f), i+1) \end{aligned}$$

  それ以外の場合，

  $$\begin{aligned} &forward(m_{i+1}, (a_1, f_1), (a, f), i+1) = \\ &transfer(m_{i+1}, (a_1, f_1), (a, f), i+1) \end{aligned}$$

  ここで f_3 と f_4 は a_3 が所属する場で，a_3 が複数の場に属する時 $f_3 \neq f_4$ であっても良い．

5.5 アクセス制御ポリシの実現の検証

コミュニティの実現モデルがコミュニティのアクセス制御ポリシの実現であることの検証は，エージェントによるメッセージ送受信を 5.4 節にしたがって解析することで行うことができる．

エージェント a が送ったベースレベル・メッセージ m がエージェント b に伝達されるためには，a の属する場のエージェントと b の属する場のエージェントの間に，メタレベルでメッセージ転送する経路が存在すれば良い．このことは，5.4.2 節の関数 *transfer* を使って以下のように表せる．

$$\exists\, u, v \mid transfer(m, (a, u), (b, v), 0) = true$$

したがって，コミュニティの実現モデルのエージェント間の全てのメッセージ送受信 *Access* は，次のように定義できる．

$$\begin{aligned} Access \;=\; & \{(a, (b, m)) \mid a, b \in A \wedge m \in M \wedge \\ & (\exists\, u, v \in F \wedge transfer(m, (a, u), (b, v), 0))\} \end{aligned}$$

このことから，コミュニティの実現モデルがコミュニティのアクセス制御ポリシ p の実現であるための必要十分条件は，

$$Access \subseteq p$$

と表すことができる．

第6章　ケーススタディ

本章では，3 章で述べた COAC モデルおよび 5 章で述べた COAC フレームワークを用いたケーススタディを 2 つ示す．6.1 節では，組織内の情報共有を例として，COAC モデルのコミュニティや連合が，組織変更に適応できるアクセス制御をどのように提供するかを示す．一方，6.2 節では，旅行予約の仲介サービスを例として，仲介サービスの連合が COAC モデルの下でどのように行われるかを示す．

6.1　情報サービス・コミュニティ

6.1.1　コミュニティのアクセス制御ポリシ

ある組織で，A 部に所属する従業員は部内の各課の Web サイトにアクセスして情報を引き出すことが許可されていたとする．A 部を情報サービス提供／利用のコミュニティ A とみなすと，A 部の従業員と各課の Web サイトは，サービスの提供／利用に参加するので，プレイヤとみなすことができる．コミュニティ A の中でプレイヤが担う役割は，従業員として情報を引き出す役割と，Web サーバとして情報を提供する役割の 2 つに分かれる．そこで，前者をパート AR，後者をパート AP と表す．

Web サイトにアクセスして情報を引き出すオペレーションを包括的に `get_information` とすると，A 部の各課の Web サイトにアクセスして情報を引き出すアクセス許可は，

$$(AP, get\_information)$$

である．この許可が A 部の従業員に与えられているので，コミュニティ A のポリシ p_A は次のように表すことができる．

$$p_A = \{(AR, (AP, get\_information))\}$$

プレイヤが p_A にしたがう時，コミュニティ A は p_A から形成されるコミュニティである．コミュニティ A を，図 6.1 に示す．

B 部でも同様に，B 部に所属する従業員は部内の各課の Web サイトにアクセスして情報を引き出すことが許可されていたとすると，コミュニティ B のポリシ p_B は次のように表すことができる．

$$p_B = \{(BR, (BP, get\_information))\}$$

ここで，BR と BP はそれぞれ B 部の従業員と B 部の各課の Web サーバが属するパートを表す．

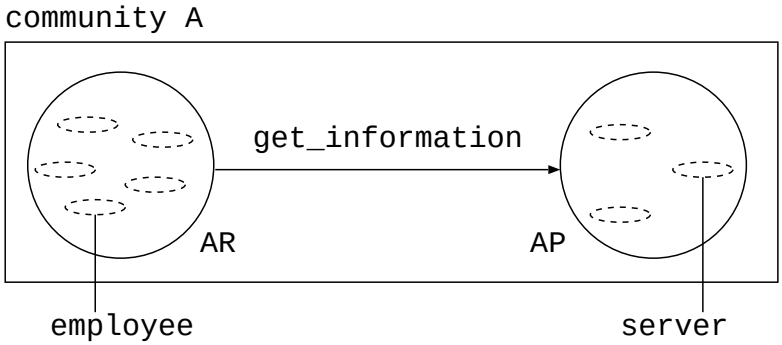

図 6.1: コミュニティA

6.1.2 コミュニティA とB の実現モデル

付録E の COAC フレームワークにしたがって，コミュニティA とB の実現モデルを構成する．

まず，コミュニティA を表す場を場A とし，各コミュニティのパートに対応するエージェントとファシリテータを場A に配置する．利用者と提供者のパートに対応するエージェントをそれぞれ AR，AP とし，A のファシリテータを AC とすると，これらのエージェントが場A に属する．

各エージェントのベースレベルには，図 6.1 のメッセージ授受を行う処理を記述する．コミュニティA の各エージェントの記述の概要を，付録 F.1 に示す．この例では，AR のメソッド `some_method` が起動されると，AR から AP へメッセージ `get_information` が，希望する情報の内容を表す引数 `ARGUMENTS` とともに送られる．このメッセージは，メタレベルの標準処理によって送られると AP の `get_information` メソッドを起動し，適切な情報が AR に提供される．また，AC はベースレベルの記述を持たない．

各エージェントのメタレベルには，ポリシにしたがったベースレベルのメッセージ送受信処理を記述する．この例では，付録 E.1 の共通記述に，送信メッセージがポリシにしたがうことをチェックする記述を加える．一方，この時点ではコミュニティ外とのメッセージ送受信をもたないので，AC のメタレベルは付録 E.2 の共通記述のままで良い．

コミュニティB についても同様であり，利用者と提供者のパートに対応するエージェントをそれぞれ BR，BP とし，B のファシリテータを BC として，これらのエージェントを場B に配置する．各エージェントのベースレベルおよびメタレベル記述の概要を，付録 F.2 に示す．

6.1.3 連合のポリシ

いま，A 部と B 部が相互に情報を利用し合うことになったとする．すなわち，A 部の従業員は B 部の各課の Web サイトにアクセスして情報を引き出すことが許可され，B 部の従業員は A 部の各課の Web サイトにアクセスして情報を引き出すことが許可されるものとする．このことは，6.1.1 節の記述に基づいて，次のように表される．

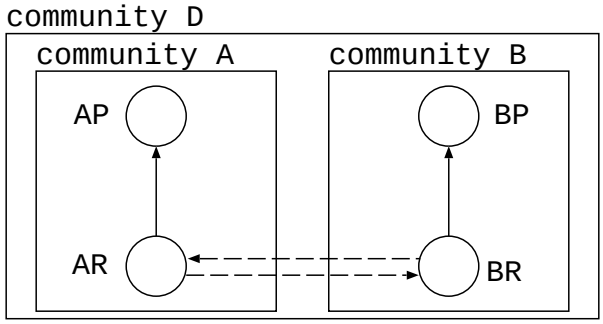

図 6.2: A と B の連合のポリシ

- BR に与えられたアクセス許可を AR に委譲する
- AR に与えられたアクセス許可を BR に委譲する

したがって，コミュニティA と B の連合のポリシは，

$$P^{\dagger} = \{(AR, BR), (BR, AR), (AR, AR), (BR, BR)\}$$

と表せる．$P^{\dagger}$ は隔離的な連合のポリシの条件を満たす．コミュニティA と B の連合のポリシを図6.2 に示す．この図で，破線の矢印が連合のポリシを表す．簡単のため，AR および BR の自分自身に対するアクセス許可の委譲は省略した．

6.1.4 連合したコミュニティのアクセス制御ポリシ

コミュニティA とコミュニティB が連合することによって構成されるコミュニティを D とし，コミュニティのアクセス制御ポリシを p_D を以下のように設定する．

$$p_D = p_A \cup p_B \cup \{(AR, (BP, get\_information)), (BR, (AP, get\_information))\}$$

3.3.3 節の定義より，コミュニティD は連合のポリシ $P^{\dagger}$ にしたがう連合である．すなわち，

- $p_A \subseteq p_D$ かつ $p_B \subseteq p_D$
- $(AR, (BP, get\_information)) \in p_D$ に対して BR が存在し，$(BR, AR) \in P^{\dagger}$ かつ $(BR, (BP, get\_information)) \in p_B$
- $(BR, (AP, get\_information)) \in p_D$ に対して AR が存在し，$(AR, BR) \in P^{\dagger}$ かつ $(AR, (AP, get\_information)) \in p_A$

さらに，$P^{\dagger}$ が隔離的な連合のポリシであることから，この連合はコミュニティのアクセス制御ポリシを分離した連合である．

D のコミュニティのアクセス制御ポリシを，図 6.3 に示す．

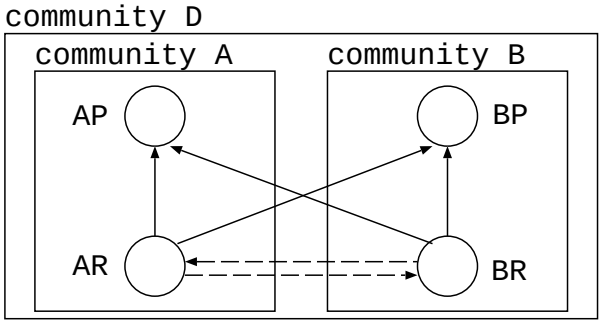

図 6.3: コミュニティD のポリシ

6.1.5 コミュニティD の実現モデル

コミュニティA と B の実現モデルを結合して，コミュニティD の実現モデルを構成する．モデルの結合は，A のファシリテータ AC を場 A と B に属させるとともに，B のファシリテータ BC も場 A と B に属させることによって行う．さらに，5.2.4 節で述べたように，次の 2 つの段階から連合したコミュニティのモデルを構成する．

- コミュニティの協調に関わるメッセージ送受信のモデル化
- 連合したシステムでのサービスの提供／利用／仲介のためのメッセージ送受信のモデル化

まず，連合によって生じる，他のコミュニティとのベースレベル・メッセージの送受信処理を記述する．この例では，AR にコミュニティB の情報を利用するためのメッセージ送信処理を加えるとともに，BR にコミュニティA の情報を利用するためのメッセージ送信処理を加える．一方，AP，BP は受信したメッセージに返信するだけなので，付録 E.1 の共通記述のもとでは，新たな処理を加える必要はない．

メッセージ送信先が増えたことにともなって，AR と BR のメタレベルではメッセージ送信がポリシにしたがうことのチェック条件が緩和される．また，AC と BC のメタレベルでは A と B の協調のための処理を加える．これは，共通記述のポリシチェックポイント 2 に場外から受け入れるメッセージの条件を加え，場間のメッセージ送受信を仲介することによって行う．D の各エージェントの記述を，付録 F.3 に示す．

6.1.6 連合関係の連鎖

連合によって構成されたコミュニティがさらに他のコミュニティと連合することで，より大きな連合が構成される．A 部や B 部と同様な次のコミュニティのアクセス制御ポリシ p_C を持つ C 部が，コミュニティD と連合する場合を考える．

$$p_C = \{(CR, (CP, get\_information))\}$$

ここで，CR と CP はそれぞれ C 部の従業員と C 部の各課の Web サーバが属するパートを表す．

アクセス許可の委譲関係は，次のように表される．

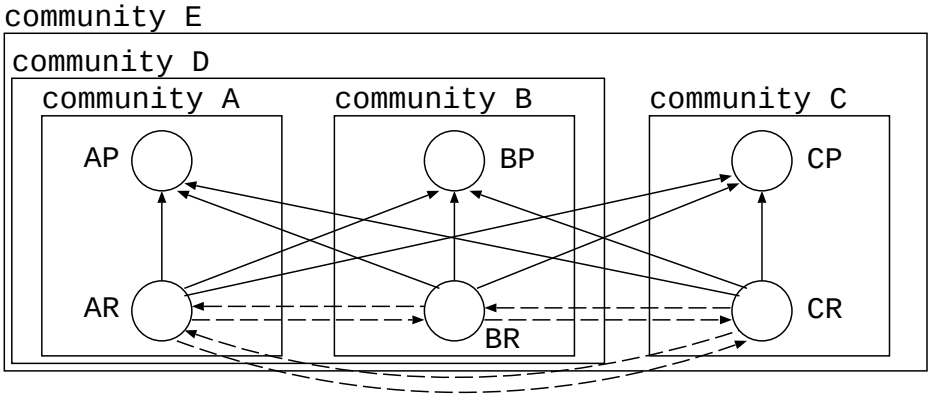

図 6.4: コミュニティE のポリシ

- CR に与えられたアクセス許可を AR および BR に委譲する
- AR および BR に与えられたアクセス許可を CR に委譲する

したがって，コミュニティC と D の連合のポリシは，

$$\begin{aligned} P^{\dagger} = \ \{ \ & (AR, BR), (AR, CR), (BR, AR), (BR, CR), \\ & (CR, AR), (CR, BR), (AR, AR), (BR, BR), (CR, CR)\}. \end{aligned}$$

コミュニティC とコミュニティD が連合することによって構成されるコミュニティを E とし，コミュニティのアクセス制御ポリシを p_E を以下のように設定する．

$$\begin{aligned} p_E \ = \ & p_C \cup p_D \cup \\ & \{(AR, (CP, get\_information)), (BR, (CP, get\_information)), \\ & (CR, (AP, get\_information)), (CR, (BP, get\_information))\} \end{aligned}$$

コミュニティE のポリシを，図 6.4 に示す．

$P^{\dagger}$ は，隔離的な連合のポリシではない．例えばコミュニティD について，

$$A.R, B.R \in R_D \text{かつ } A.R \neq B.R$$

であるにもかかわらず，

$$(AR, BR) \in P^{\dagger}.$$

一方，p_E は 3.3.4 節の条件を満たすので，この連合はコミュニティのアクセス制御ポリシを分離した連合である．

6.1.7 コミュニティE の実現モデル

6.1.6 節のコミュニティC の実現モデルは，コミュニティB と同じように構成することができる．コミュニティC を表す場を場 C とし，利用者，提供者のパートに対応す

るエージェントをそれぞれ CR，CP，ファシリテータを CC として，これらを場 C に配置する．コミュニティ C の各エージェントの記述は，付録 F.2 のコミュニティ B の各エージェントの記述と，基本的に同じである．

コミュニティ C と D の実現モデルを結合して，コミュニティ E の実現モデルを構成する．モデルの結合は，AC と BC がさらに場 C に属し，CC が場 A と B に属することで行う．連合による拡張は，6.1.5 節と同じく，次の段階によって行う．

- 連合によって生じる，他のコミュニティとのベースレベル・メッセージの送受信処理の追加
- メッセージ送信先が増えることによるメタレベルのチェック条件の緩和
- コミュニティ間の協調処理のファシリテータのメタレベルへの追加

コミュニティ E の各エージェント記述を，付録 F.4 に示す．この実現モデルにより，6.1.6 節の連合関係が実現される．

6.2 仲介サービス・コミュニティ

6.2.1 コミュニティのアクセス制御ポリシ

Web を通じてホテルの予約サービスを仲介するコミュニティ A を考える．A の利用者はブローカに希望するホテルの種類を伝え，ホテルの予約を依頼する．ブローカは適切な提供者にアクセスしてホテルを予約し，利用者に伝える．ここで，ブローカは利用者の要求に適合する提供者を探すため，メディエータに問い合わせる．A では利用者の匿名性確保のため，これ以外のアクセスは許さないものとする．この様子を，図 6.5 に示す．図中のメッセージ左側の数字は，メッセージの順序を表す．利用者と提供者はコミュニティに加わる際に情報の授受に関する包括的な規約に合意するものとすると，この過程は個々のプレイヤによらないパート間のアクセス許可ととらえることができる．利用者，ブローカ，メディエータ，提供者のパートを各々 AR，AB，AM，AP とすると，A のアクセス制御ポリシは，次のものである．

$$p_A = \{ \quad (AR,(AB,broker)),(AB,(AM,h\_recommend)), \\ (AB,(AP,h\_service\_request)).\}$$

一方，航空券の予約を行うコミュニティ B では，図 6.6 のように利用者がメディエータに希望する情報の種類を示し，メディエータから指示された提供者にアクセスして情報を得るものとする．利用者，メディエータ，提供者のパートを各々BR，BM，BP とすると，B のアクセス制御ポリシは，次のものである．

$$p_B = \{(BR,(BM,f\_recommend)),(BR,(BP,f\_service\_request))\}$$

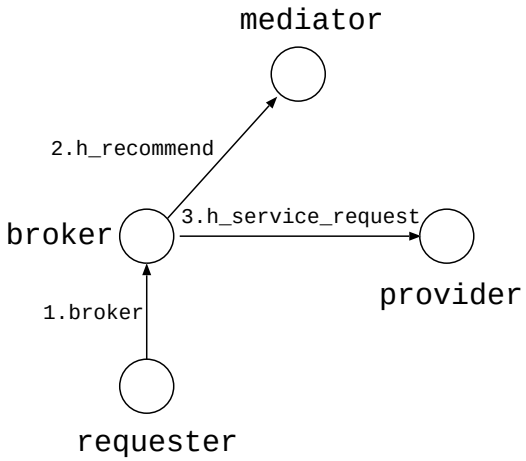

図 6.5: コミュニティA

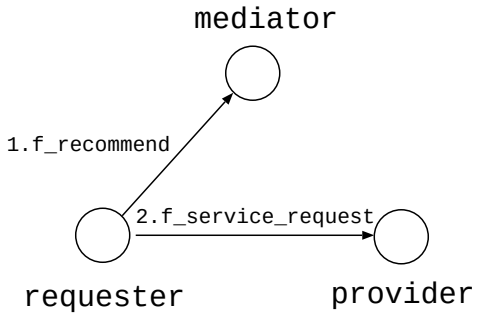

図 6.6: コミュニティB

6.2.2 コミュニティA の実現モデル

コミュニティA の実現モデルは，付録 E の COAC フレームワークにしたがって，コミュニティに場を割り当て，コミュニティの各パートに属するプレイヤのふるまいをエージェントを使って記述することによって構成する．エージェントの記述には，付録 E.1 を利用することができる．

まず，コミュニティA を表す場を場 A とし，各コミュニティのパートに対応するエージェントとファシリテータを場 A に配置する．利用者，ブローカ，メディエータ，提供者のパートに対応するエージェントをそれぞれ AR，AB，AM，AP とし，A のファシリテータを AC とすると，これらのエージェントが場 A に属する．

各エージェントのベースレベルには，図 6.5 のメッセージ授受を行う処理を記述する．コミュニティA の各エージェントのメッセージ授受の概要を，付録 G.1 に示す．この例では，AR のメソッド `some_method` が起動されると，AR から AB へメッセージ `broker` が，希望する部屋の条件を表す引数 `ARGUMENTS` とともに送られる．このメッセージは，メタレベルの標準処理によって送られると AB の `broker` メソッドを起動し，AB から AM へ `h_recommend` メッセージが推薦条件 ARGUMENTS1 とともに送られる．AM からの戻り値 Res は ARGUMENTS1 に適合する提供者へのアクセスキーであり，Res と予約に必要な情報 ARGUMENTS2 を引数として AP へ `h_service_request` メッセー

ジを送ることで，予約が行われる．また，ここでは AC はベースレベルの記述を持たない．

各エージェントのメタレベルには，ポリシにしたがったベースレベルのメッセージ送受信処理を記述する．この例では，付録 E.1 の共通記述に，送信メッセージがポリシにしたがうことをチェックする記述を加える．一方，この時点ではコミュニティ外とのメッセージ送受信をもたないので，AC のメタレベルは付録 E.2 の共通記述のままで良い．

6.2.3 コミュニティ B の実現モデル

コミュニティ B の実現モデルも，コミュニティ A の実現モデルと同様に構成する．

コミュニティ B を表す場を場 B とし，利用者，メディエータ，提供者のパートに対応するエージェントをそれぞれ BR，BM，BP とし，B のファシリテータを BC として，これらのエージェントを場 B に属させる．

各エージェントのベースレベルには，図 6.6 のメッセージ授受を行う処理を記述する．コミュニティ B の各エージェントの記述を，付録 G.2 に示す．この例では，BR のメソッド `some_method` が起動されると，BR から BM へメッセージ `f_recommend` が推薦条件を表す引数 ARGUMENTS1 とともに送られ，BM から条件に適合する提供者へのアクセスキーが返される．このアクセスキーと予約に必要な情報を引数として，BP へメッセージ `f_service_request` が送られ，予約が行われる．AC と同様 BC も，ベースレベルの記述は持たない．

各エージェントのメタレベルでは，付録 E.1 の共通記述のメッセージ送受信処理に，送信メッセージがポリシにしたがうことをチェックする記述を加える．BC のメタレベルは，付録 E.2 の共通記述のまま変更する必要はない．

6.2.4 連合のポリシ

ホテル予約サービスと航空券予約サービスが提携し，ホテル予約サービスの利用者が航空券も予約でき，航空券予約サービスの利用者がホテルも予約ができるように，システムを拡張する場合を考える．ここで，ホテル予約サービスの利用者に与えられた匿名性が航空券予約の際に不用意に損なわれることは，利用者にとって好ましいことではない．

そこで，コミュニティ A と B の連合では，A の利用者が A と同様の匿名性の下に B の情報も利用できることが求められているとする．この場合，A の仲介者は B の利用者に与えられているアクセス許可を委譲される必要がある．したがって，B から A へアクセス許可を次のように委譲する．

$$P_{BA} = \{(BR, AB)\}$$

一方，B の利用者が A の情報も利用できることが求められているので，B の利用者は A の利用者に与えられているアクセス許可を委譲される必要がある．したがって，

$$P_{AB} = \{(AB, BR)\}$$

これらより，連合のポリシは

$$\begin{aligned} P^{\dagger} &= P_{AB} \cup P_{BA} \cup \{(AB, AB), (BR, BR)\} \\ &= \{(AB, BR), (BR, AB), (AB, AB), (BR, BR)\} \end{aligned}$$

3.3.2 節の条件を満たすので，$P^{\dagger}$ は隔離的である．すなわち，

$$R_A = \{AR, AB, AM, AP\}$$

の任意の要素 X と Y に対して，

$$X \neq Y \Longrightarrow (X, Y) \notin P^{\dagger}.$$

同様に，

$$R_B = \{BR, BM, BP\}$$

の任意の要素 X と Y に対して，

$$X \neq Y \Longrightarrow (X, Y) \notin P^{\dagger}.$$

6.2.5 連合したコミュニティのアクセス制御ポリシ

拡張したポリシに対して，A と B の連合によって構成されるコミュニティを D とし，そのアクセス制御ポリシ p_D を以下のように設定する．

$$\begin{aligned} p_D &= p_A \cup p_B \cup \\ &\quad \{(AB, (BM, f\_recommend)), (AB, (BP, f\_service\_request)), \\ &\quad (BR, (AM, h\_recommend)), (BR, (AP, h\_service\_request))\} \end{aligned}$$

この連合は，連合のポリシにしたがうコミュニティの連合であるための 3.3.3 節の条件を満たす．すなわち，

- $p_A \subseteq p_D$ かつ $p_B \subseteq p_D$
- $(AB, (BM, f\_recommend)) \in p_D$ に対して BR が存在し，
  $(BR, AB) \in P^{\dagger}$ かつ $(BR, (BM, f\_recommend)) \in p_B$
- $(AB, (BP, f\_service\_request)) \in p_D$ に対して BR が存在し，
  $(BR, AB) \in P^{\dagger}$ かつ $(BR, (BP, f\_service\_request)) \in p_B$
- $(BR, (AM, h\_recommend)) \in p_D$ に対して AB が存在し，$(AB, BR) \in P^{\dagger}$ かつ
  $(AB, (AM, broker)) \in p_A$
- $(BR, (AP, h\_service\_request)) \in p_D$ に対して AB が存在し，
  $(AB, BR) \in P^{\dagger}$ かつ $(AB, (AP, h\_service\_request)) \in p_A$

連合のポリシが隔離的であることから，この連合はコミュニティのアクセス制御ポリシを分離した連合である．

D のポリシを図 6.7 に示す．この図で，丸はパートを，破線の矢印は連合のポリシを表す．実線の矢印はコミュニティのアクセス制御ポリシを示し，始点が X，終点が Y ならば，$(X, Y) \in p_D$ である．

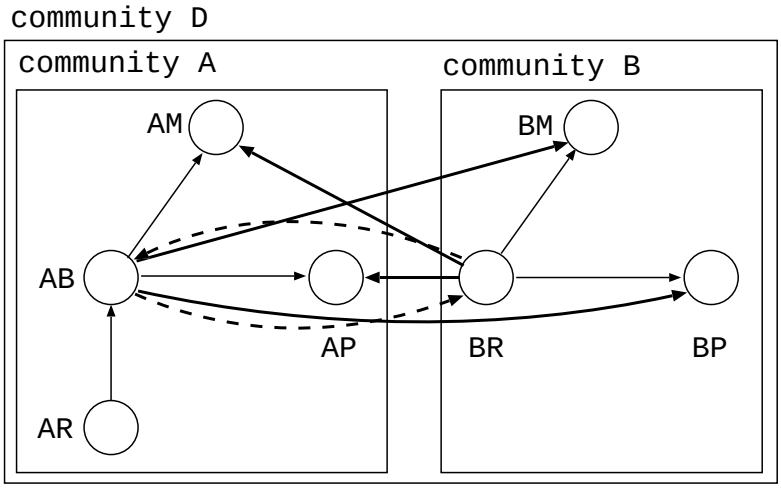

図 6.7: 連合したコミュニティのアクセス制御ポリシ

6.2.6 コミュニティD の実現モデル

コミュニティA と B の実現モデルを結合して，コミュニティD の実現モデルを構成する．モデルの結合は，A のファシリテータ AC を場 A と B に属させるとともに，B のファシリテータ BC も場 A と B に属させることによって行う．さらに，5.2.4 節で述べたように，次の 2 つの段階から連合したコミュニティのモデルを構成する．

- コミュニティの協調に関わるメッセージ送受信のモデル化
- 連合したシステムでのサービスの提供／利用／仲介のためのメッセージ送受信のモデル化

まず，連合によって生じる，他のコミュニティとのベースレベル・メッセージの送受信処理を記述する．この例では，AB のメソッド `broker` に航空券予約のための BM および BP へのメッセージ送信処理を加えるとともに，BR にホテル予約のための AB および AP へのメッセージ送信処理を加える．一方，AM，AP，BM，BP は受信したメッセージに返信するだけなので，付録 E.1 の共通記述のもとでは，新たな処理を加える必要はない．また，AR は `broker` メッセージの引数で航空券予約の条件も指定する必要があるが，ここではメッセージ授受に影響しないため省略する．

メッセージ送信先が増えたことにともなって，AB と BR のメタレベルではメッセージ送信がポリシにしたがうことのチェック条件が緩和される．また，AC と BC のメタレベルでは A と B の協調のための処理を加える．これは，共通記述のポリシチェックポイント 2 に場外からメッセージの条件を加え，場間のメッセージ送受信を仲介することによって行う．D の各エージェントの記述を，付録 G.3 に示す．

6.2.7 D のインタラクションの概要

図 6.8 は，6.2.6 節のコミュニティA の利用者に対応するエージェントが，コミュニティB の提供者に対応するエージェントにサービスを要求する際のインタラクション

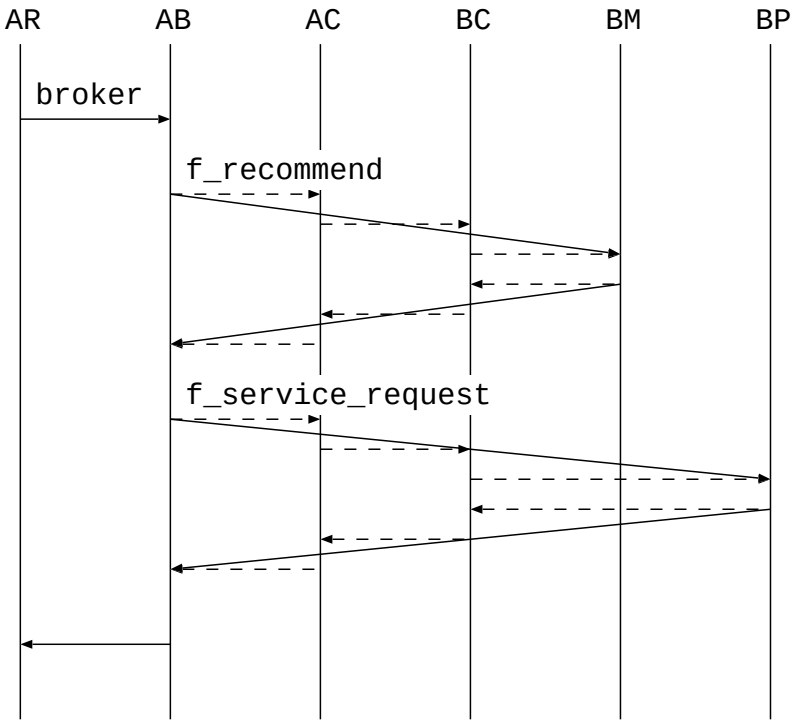

図 6.8: A から B へのメッセージ・シーケンス

の概要である．ここで，実線の矢印はベースレベルのメッセージ送受信を，破線の矢印はメタレベルのメッセージ送受信を表す．

6.2.6 節の例では，図 6.8 のように，コミュニティ A の利用者 AR がブローカ AB に送った `broker` メッセージは，AR から AB へメタレベルでメッセージ閉包として渡され，AB のベースレベルで処理される．この処理の際に AB が BM に送った `f_recommend` メッセージと BP に送った `f_service_request` メッセージは，メタレベルでファシリテータ AC に渡された後 BC に転送され，それぞれ BM と BP にで渡される．メタレベルのメッセージ受け渡しの結果，ベースレベルでは `f_recommend` メッセージと `f_service_request` メッセージが見かけ上は場を越えて送受信されるが，これらのメッセージがコミュニティ B のポリシにしたがうことが BC のメタレベルで検査される．また，各ベースレベルメッセージの送信元エージェントのメタレベルでは，送信するメッセージがコミュニティのアクセス制御ポリシにしたがうことが検査される．一方この例では，送信先エージェントからの戻りメッセージは検査されることなくメタレベルを使って受け渡される．

同様に，図 6.9 は，6.2.5 節のコミュニティ B の利用者に対応するエージェントが，コミュニティ A の提供者に対応するエージェントにサービスを要求する際のインタラクションの概要である．

6.2.8 連合関係の連鎖

連合により構成されたコミュニティがさらに別のコミュニティと連合することにより，連合関係の連鎖が起こる．例えば，6.2.5 節のコミュニティ D が，地上交通のチケッ

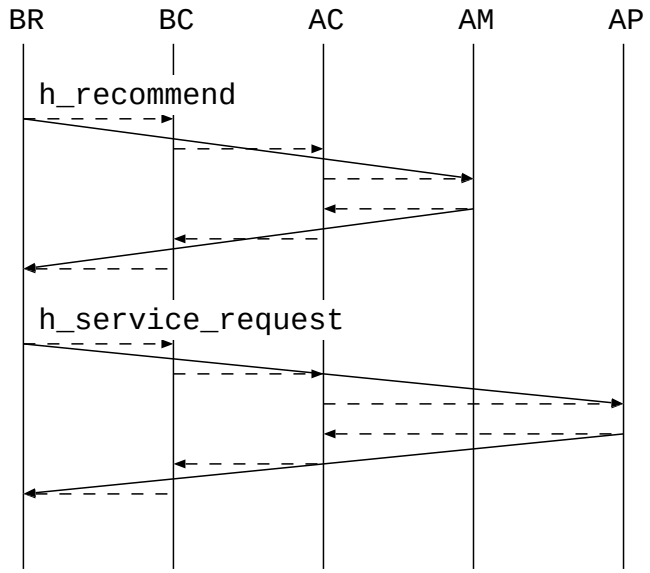

図 6.9: B から A へのメッセージ・シーケンス

ト予約サービスを提供するコミュニティC と連合する場合を考える．C では B と同様の手順で情報へのアクセスが行なわれるものとする．利用者，メディエータ，提供者のパートを各々CR，CM，CP とすると，C のアクセス制御ポリシは次のようになる．

$$p_C = \{(CR, (CM, ll\_recommend)), (CR, (CP, l\_service\_request))\}$$

C と D の連合にあたって，A の仲介者と B の利用者には C の利用者に与えられているアクセス許可が委譲され，C の利用者には A と B の利用者に与えられているアクセス許可が委譲されるとすると，

$$\begin{aligned} P_{CD} &= \{(CR, BR), (CR, AM)\} \\ P_{DC} &= \{(AR, CR), (BR, CR)\} \end{aligned}$$

C と D の連合によって構成されるコミュニティを E とすると，次の p_E は，C と D のポリシを分離した連合の，コミュニティのポリシである．

$$\begin{aligned} p_E = \; & p_D \cup p_C \cup \{(CR, (AM, broker)), \\ & (CR, (BM, f\_recommend)), (CR, (BP, f\_service\_request)), \\ & (AM, (CM, l\_recommend)), (AM, (CP, l\_service\_request)), \\ & (BR, (CM, l\_recommend)), (BR, (CP, l\_service\_request))\} \end{aligned}$$

E のコミュニティのアクセス制御ポリシを図 6.10 に示す．

6.2.9 コミュニティE の実現モデル

6.2.8 節のコミュニティC の実現モデルは，コミュニティB と同じように構成することができる．コミュニティC を表す場を場 C とし，利用者，メディエータ，提供者の

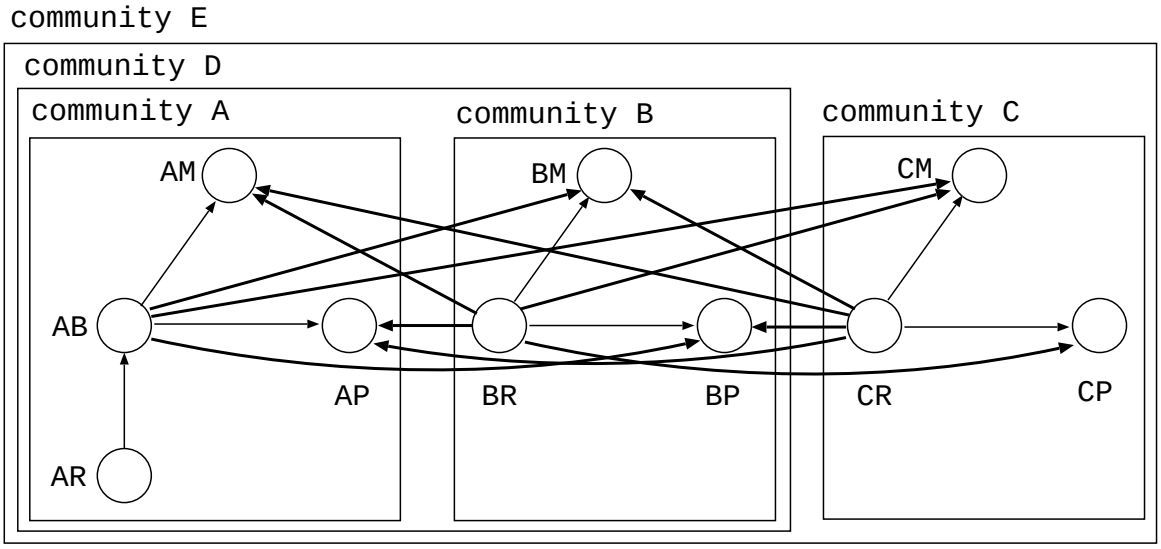

図 6.10: 連合関係の連鎖

パートに対応するエージェントをそれぞれCR，CM，CP，ファシリテータをCCとして，これらを場Cに配置する．コミュニティCの各エージェントの記述は，付録G.2のコミュニティBの各エージェントの記述と，基本的に同じである．

コミュニティCとDの実現モデルを結合して，コミュニティEの実現モデルを構成する．モデルの結合は，ACとBCがさらに場Cに属し，CCが場AとBに属することで行う．連合による拡張は，6.2.6節と同じく，次の段階によって行う．

- 連合によって生じる，他のコミュニティとのベースレベル・メッセージの送受信処理の追加
- メッセージ送信先が増えることによるメタレベルのチェック条件の緩和
- コミュニティ間の協調処理のファシリテータのメタレベルへの追加

コミュニティEの各エージェント記述を，付録G.4に示す．このモデルにより，6.2.8節の連合関係が実現される．

6.2.10 Eのインタラクションの概要

コミュニティAの利用者に対応するエージェントが，コミュニティCの提供者に対応するエージェントにサービスを要求する際のインタラクションの概要を，図6.11に示す．同様に，コミュニティCの利用者に対応するエージェントが，コミュニティAの提供者に対応するエージェントにサービスを要求する際のインタラクションの概要を，図6.12に示す．

また，コミュニティBの利用者に対応するエージェントが，コミュニティCの提供者に対応するエージェントにサービスを要求する際のインタラクションの概要を，図6.13に示す．同様に，コミュニティCの利用者に対応するエージェントが，コミュニティB

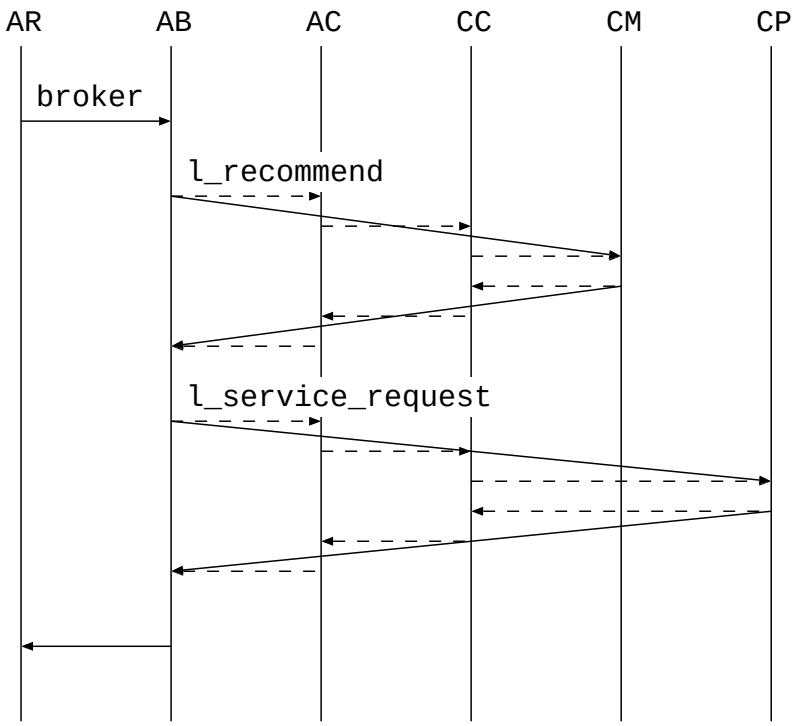

図 6.11: A から C へのメッセージ・シーケンス

の提供者に対応するエージェントにサービスを要求する際のインタラクションの概要を，図 6.14 に示す.

なお，コミュニティ A の利用者に対応するエージェントがコミュニティ B の提供者に対応するエージェントにサービスを要求する際のインタラクションと，コミュニティ B の利用者に対応するエージェントがコミュニティ A の提供者に対応するエージェントにサービスを要求する際のインタラクションは，それぞれ図 6.8 と図 6.9 のままで変更はない.

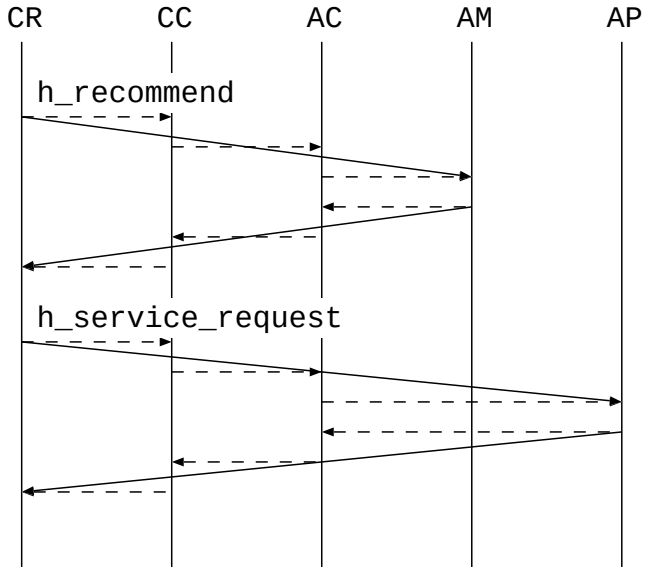

図 6.12: C から A へのメッセージ・シーケンス

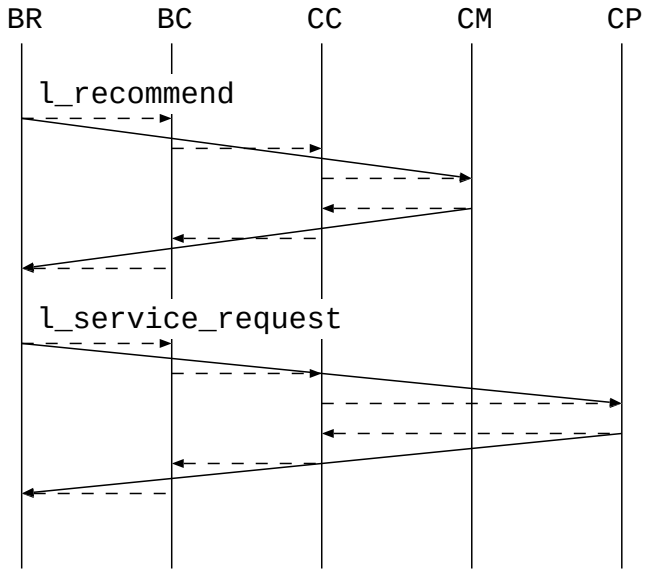

図 6.13: B から C へのメッセージ・シーケンス

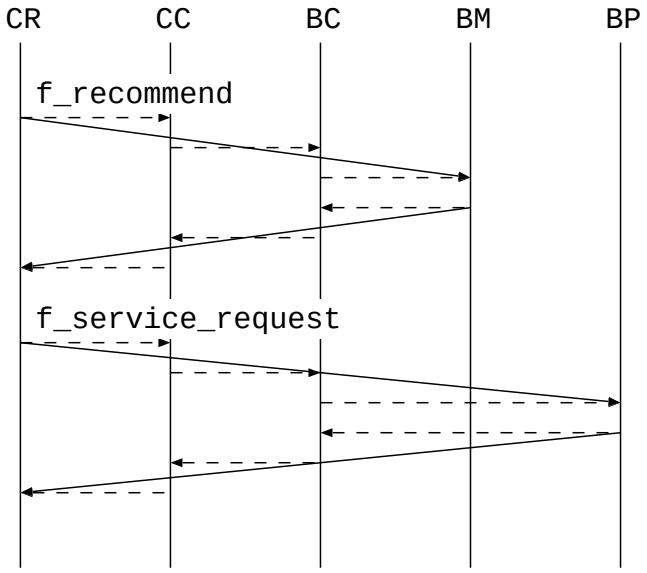

図 6.14: C から B へのメッセージ・シーケンス

第7章　考察

本章では，3 章で提案した COAC モデルを，既存のアクセス制御モデルと比較することによって位置付ける．また，6 章に示したコミュニティの特性を検証することによって，5 章で提案した COAC フレームワークを位置付ける．7.3 節では，本研究の手法の適用範囲について述べる．本研究ののアプローチを 7.4 節で議論し，関連研究との比較を 7.5 節で行う．

7.1　COAC モデルの位置付け

この節では，COAC モデルを 2.1.5 節の Role Based Access Control(RBAC) [15, 16, 45] モデルと比較し，その特性を評価する．RBAC モデルは実社会での要求を分析して作られたアクセス制御モデルであり，さまざまな領域に適用できる幅広いアクセス制御ポリシの表現が可能である．このような背景から，RBAC モデルは商用システムのための実用的なアクセス制御モデルとして広く認知されているとともに，さまざまなバリエーションが提案されている．ここでは比較対象として RBAC のリファレンスモデルとなる ANSI 標準勧告 [2] を採り上げる．

7.1.1　RBAC モデルとの比較

COAC モデルと RBAC モデルの基本機能の違いを，次の表に示す．ここで，各機能について○はサポートしていることを，×はサポートしていないことを，△は前提を緩和することで拡張可能であることを示す．

機能	COAC モデル	RBAC モデル
ロール階層	△	○
連合のポリシ	○	×
責務関係の静的な分離	△	○
責務関係の動的な分離	△	○

閉じた計算機システムにおけるアクセス制御では，中央集権的な管理者が強い運用方針の下にユーザにアカウントを割り当てる．このため，RBAC モデルでは実世界の実体と計算機内部のユーザが一対一に対応することを前提とすることができ，この前提の下に責務関係の分離を規定することが可能となる．これに対し，COAC モデルではオープンなシステムを前提としていることから，あるコミュニティのプレイヤ a と，別のコミュニティのプレイヤ b が，同一の実体であるかどうかを確認する手段が存在することを仮定しない．言い換えれば，コミュニティ間でのプレイヤの同一性を仮定

しない．さらに，これらのコミュニティが緩やかに連合する場合には，実体が同一の異なるプレイヤが連合によってできたコミュニティ内に存在することになり，コミュニティ内でもプレイヤの同一性が保証されない．このような前提の下に，COAC モデルではアクセス制御ポリシをパート間の関係のみによって規定する．パートとプレイヤの対応は，プレイヤがコミュニティに参加する際に決定されているものとして，ポリシ・モデルでは扱わない．この結果，プレイヤの同一性に依存する制約はサポートしない．

7.1.2 責務の分離について

RBAC モデルではセッションの概念を導入することで，ユーザに与えられるロール（SSD に関わるロール）とユーザがタスクを実行する際に開始するロール（DSD に関わるロール）を分離する．COAC モデルではプレイヤの同一性を前提としないため，任意の時点でプレイヤが担っているパートを全て列挙できるという仮定をとらず，プレイヤは担い得る全てのパートを担っているものと考える．このことは，DSD と SSD を同一視していることと同じである．さらに，プレイヤが担い得るパートはプレイヤがコミュニティに参加する際に解決されるものとして，SSD をポリシ・モデルの対象外としている．この結果，SSD と DSD はポリシ・モデル上には現れない．

COAC モデルのこのような前提は，構成要素が緩やかに結合したオープンなシステムを考えるうえでは妥当なものであると思われる．一方，プレイヤがコミュニティに参加する際に外的な手段で確認することにより，プレイヤの同一性を保証するコミュニティも存在し得る．例えば銀行口座の場合，開設時に公的な身分証明の提示を要求するため，実体とプレイヤ（口座名義人）の一対一対応を保証することができる．このようなコミュニティでは，連合の際にも各コミュニティが密に連絡することで，プレイヤの同一性を確保することが可能であるとともに，任意の時点でプレイヤが担っているパートを列挙することができる．COAC モデルを，プレイヤの同一性が保証されたコミュニティとその連合に適合するように拡張し，静的および動的な責務関係の分離を導入することは容易である．その場合，プレイヤとパートの束縛をポリシ・モデル上で陽に扱うとともに，連合の際にも責務関係の分離が保たれるための条件がコミュニティの連合のポリシに付加される．

7.1.3 ロール階層について

プレイヤとなる実体に与えられるべき許可の順序関係がコミュニティによらず一定であると仮定できる場合には，COAC モデルにもパート間の階層関係を導入することができる．この仮定は，あるコミュニティでプレイヤ a にプレイヤ a' より上位の許可が与えられるとき，他のコミュニティでも a の実体がとるプレイヤ b に a' の実体がとるプレイヤ b' より上位の許可が与えられることを，プレイヤの同一性を要求することなく想定できることを意味する．例えば，プレイヤが属することのできるパートを信用情報に基づいて決定するコミュニティの間では，高い信用を持つ実体は，どのコミュニティでも上位のパートに属するプレイヤとして振舞えることが想定できる．この場合，a と a' および b と b' が属するパートの間の順序関係が，a と b および a' と b' の同

一性を考慮することなく，各コミュニティの独立した判断によって保たれる．パート間の階層関係を導入して拡張した COAC モデルを，付録 H に示す．このようなコミュニティが連合する際には，連合によってパート間の階層関係が損なわれないように，連合のポリシは H.3 節の条件を満たす必要がある．

7.1.4 連合のポリシについて

連合のポリシは，異なるコミュニティのパートの間の許可の委譲関係である．RBAC モデルでも，ロールの間の許可の委譲関係を導入して，モデルを拡張することは可能であると考えられる．しかし，RBAC モデルでは１つの計算機システムが全宇宙であり，その段階的な拡張はモデルの範囲内で扱うことができるが，異なる RBAC を持つ他の計算機システムとの結合はモデルの対象外である．したがって，RBAC モデルにロール間の許可の委譲関係を導入しても，１つの計算機システム内のロールの間の許可の委譲関係にしかならない．この点で，COAC モデルと RBAC モデルは本質的に異なると考えられる．COAC モデルのこのような特性は，7.2 節で述べるように 2.2.2 節の疎結合や柔軟な構成をサポートするアクセス制御ポリシを表現する上で有効である．

7.2 COAC フレームワークの位置付け

5.1 節で挙げた課題（1）（2）（3）は，COAC フレームワークによって各々以下のように解決される．

7.2.1 連合関係に対するプレイヤの独立性

連合にともなうインタラクションの調整の大部分はファシリテータが行うため，実現モデル上では連合関係の有無によって個々のエージェントが受ける影響は少ない．例えば，6.1 節や 6.2 節の例では，連合先のコミュニティとの間のインタラクションはファシリテータが担う．この結果，図 6.13 のように C の要求者に対応するエージェントが送った `f_service_request` メッセージは各場のファシリテータによって転送され，B の提供者に対応するエージェントによって航空券の予約が実行される．エージェントには，拡張されたサービスを利用するための処理がベースレベルに，その処理がポリシにしたがうことを検査する処理がメタレベルに追加されるのみであり，連合のために担う調整は限定的である．

COAC フレームワークでは，メタ階層を使って以下を表現できる．

- メッセージ送受信先の変更
- 複数のメッセージを一つのメッセージに結合
- 一つのメッセージを複数のメッセージに分割

これらの操作をさらに上位階層から操作することによって，より高次の操作も表現できる．

COAC フレームワークにおける，エージェントがポリシにしたがったインタラクションを行うためのメタレベルからの調整は，5.2.1 節で述べたようにアクセス・コントローラとバウンダリ・コントローラによって実装することが考えられる．この時，各エージェントのメタレベルはアクセス・コントローラの機能に，ファシリテータのメタレベルはバウンダリ・コントローラの機能によって実装される．各プレイヤは，自らが属するパートに応じたアクセス・コントローラを通じて他のパートに属するプレイヤにアクセスするとともに，アクセス・コントローラからバウンダリ・コントローラを経由して他のコミュニティのプレイヤにアクセスする．連合にともなうアクセスの調整は，メタレベルに抽出された機能を実装するアクセス・コントローラおよびバウンダリ・コントローラが行う．このような構成のもとでは，コミュニティの連合によってプレイヤに要求されるのは拡張されたサービスへの対応のみであり，多くのプレイヤをスケーラブルに結合することができる．

7.2.2 連合関係の変化への対応

図 6.8 は，6.2 節の仲介サービス・コミュニティの連合において，コミュニティ A の利用者がコミュニティ B の提供者のサービスを利用する際のインタラクションである．この図は，場 A のブローカによるベースレベル・メッセージ `f_recommend` と `f_service_request` が，各場のファシリテータによって転送されてコミュニティ B のメディエータおよび提供者に送られることを表す．COAC フレームワークでは，このようなベースレベル・メッセージの転送をコミュニティ間に共通の方法でメタレベルで行う．この結果，6.2.8 節のようにコミュニティ A と B の連合がさらにコミュニティ C と連合する場合においても，付録 G.4 に示したように A と B のエージェントを大きく変更する必要はない．COAC フレームワークによって，以下が得られる．

- コミュニティ間の協調のための共通基盤
- 連合関係の変化による影響範囲の限定

7.2.3 コミュニティのアクセス制御ポリシの分離の検証

6.2 節のコミュニティ A と B の連合であるコミュニティ D について，コミュニティのアクセス制御ポリシが分離されることを検証する．6.2.5 節に示したように，p_D はコミュニティ A，B のポリシを分離したポリシである．したがって，6.2.6 節のコミュニティの実現モデルが，p_D の実現であることを検証すれば良い．

A と B のエージェント間のベースレベルのメッセージ送受信に関して，5.4.2 節で述べた関数 *transfer* の値が真になるのは以下の組合せのみである．

$$transfer(broker, (AR, A), (AB, A), 0)$$

$$transfer(h\_recommend, (AB, A), (AM, A), 0)$$

$transfer(f\_recommend, (AB, A), (BM, B), 0)$

$transfer(h\_service\_request, (AB, A), (AP, A), 0)$

$transfer(f\_servive\_request, (AB, A), (BP, B), 0)$

$transfer(f\_recommend, (BR, B), (BM, B), 0)$

$transfer(h\_recommend, (BR, B), (AM, A), 0)$

$transfer(f\_service\_request, (BR, B), (BP, B), 0)$

$transfer(h\_service\_request, (BR, B), (AP, A), 0)$

これらから 5.5 節にしたがって,

$$\begin{aligned} Access \;=\; & \{(AR, (AB, broker)), (AB, (AM, h\_recommend)), \\ & (AB, (BM, f\_recommend)), (AB, (AP, h\_service\_request)), \\ & (AB, (BP, f\_service\_request)), (BR, (BM, f\_recommend)), \\ & (BR, (AM, h\_recommend)), (BR, (BP, f\_service\_request)), \\ & (BR, (AP, h\_service\_request))\} \end{aligned}$$

一方，6.2.5 節より,

$$\begin{aligned} p_D \;=\; & \{(AR, (AB, broker)), (AB, (AM, h\_recommend)), \\ & (AB, (BM, f\_recommend)), (AB, (AP, h\_service\_request)), \\ & (AB, (BP, f\_service\_request)), (BR, (BM, f\_recommend)), \\ & (BR, (AM, h\_recommend)), (BR, (BP, f\_service\_request)), \\ & (BR, (AP, h\_service\_request))\} \end{aligned}$$

これは,

$Access \subseteq p_D$

であるため，D のポリシを満たす．したがって，6.2.6 節のコミュニティの実現モデルは，D のコミュニティのアクセス制御ポリシの実現であることが言える．

コミュニティ E についても同様に p_E がコミュニティ C と D のポリシを分離したポリシであり，6.2.9 節のコミュニティの実現モデルが E のコミュニティのアクセス制御ポリシの実現であることを検証できる．

7.3 手法の適用範囲

本研究の手法の適用範囲を，COAC モデルの表現能力，COAC フレームワークの表現能力，コミュニティの実装環境の 3 点から考察する．ここで，実装環境は多様であり，現在の段階で論じるのは適切ではない．ここではコミュニティの実装モデルは何らかの方法で実装されるものとして，COAC モデルの表現能力と COAC フレームワークの表現能力の点から手法の適用範囲を論じる．

7.3.1 コミュニティのアクセス制御ポリシ

COAC モデルの適用範囲は，アクセス制御がパート間の素朴な関係によって表現できる範囲に限定される．すなわち，7.1 節で述べたように，RBAC が持つ責務関係の分離に関する制約は取り扱うことができない．これは，本研究ではプレイヤを一意に認証する認証機構の存在を前提としないことから生じる違いであり，プレイヤが緩やかに結合するオープンなシステムでは妥当な前提であると思われる．

また，RBAC と同様に，アクセス順序に関する制約を表現することはできない．例えば,「サービス提供者にアクセスするには，その前に必ず仲介者にアクセスしなければならない」という制約を，コミュニティのアクセス制御ポリシとして表現することはできない．アクセス順序に関するポリシの表現は，今後の検討課題である．

さらに，禁止ポリシを直接表現することはできない．例えば,「B 会員は A 会員向けのサービスにアクセスしてはならない」という制約を，コミュニティのアクセス制御ポリシとして表現することはできない．COAC モデルは許可ポリシを明示的に表現するものであり，許可ポリシで陽に許可されていないアクセスは全て禁止されているものと解釈する．したがって，上記の制約は例えば「A 会員は A 会員向けのサービスにアクセスすることができる」と表現することになる．禁止ポリシが有効である局面も少なくはないが，アクセスの範囲が明示される許可ポリシは安全性の検証が容易であり，管理下にある資源を保護する点からは禁止ポリシは必ずしも必要ではないと考える．

7.3.2 連合のポリシ

COAC モデルの連合のポリシの適用範囲は，パート間の対応関係によって表現できる範囲内に限定される．連合するコミュニティは独立に構成されているため，適切なパートの対応が定められない場合は容易に起こり得る．例えば，男女を区別しないサービスのコミュニティ A と男女を区別するサービスのコミュニティ B の連合において，A のパート a に属するプレイヤのうち男性は B のパート b に，女性は B のパート c に対応する場合が考えられる．この場合，次の 2 つの対応があり得る．

- コミュニティ A を変更し，パート a を分割する
- 連合のポリシで，a と，b および c を対応付ける

コミュニティのパート間に最小限の委譲関係を設定することが難しい連合の場合，想定されるパート間のあらゆる対応を連合のポリシとして，連合したコミュニティのア

クセス制御ポリシの設定の際にコミュニティのアクセス制御ポリシを分離した連合を構成することは可能である．しかし，その場合には不必要に広い連合のポリシが与えられることになり，セキュリティの原則である最小権限の付与が損なわれることになる．本研究では連合のポリシのモデルとしてパート間の素朴な対応関係をとったことにより，連合したコミュニティのアクセス制御ポリシの解析が直感的で容易になった反面，その適用範囲が制限されている．より高度な連合のポリシ・モデルの設定は，今後の課題である．

また，連合のポリシが動的に変化する場合のためのポリシ・モデル，すなわち連合の持続時間より短い時間内に変化するパート間の委譲関係を表現するためのモデルも，今度の課題である．

7.3.3 COAC フレームワーク

本研究では，4 章で述べたメタ階層に基づく言語を使っており，モデル記述言語自体の表現能力は高い．しかし，コミュニティの実現モデルではコミュニティ間で共通の協調処理を行わせるため，記述を 5.3 節の COAC フレームワークによって制限している

COAC フレームワークの下では，場とメタ階層を使って表現できる範囲は，メッセージ操作に関わる以下の項目に限定される．

- メッセージ送受信先の変更
- メッセージ内容の書換え
- 複数のメッセージを１つのメッセージに統合
- １つのメッセージを複数のメッセージに分割

この制限は，コミュニティの実現モデルからの実装を容易にする点で有効である．一方で，パートやプレイヤの間のネゴシエーションなどは，より上位のメタ階層からの操作として記述できることが予想されるが，現時点では未検討である．

7.4 議論

オープンなシステムでは，システムの非集中性や柔軟な拡張性を損なわないために，アクセス制御のために構成要素を集中的に管理することはできない．また，連合によって構成されるシステムのアクセス制御を行うために連合するシステムのアクセス制御を作り直すことは困難であり，連合するシステムのアクセス制御と整合性のあるアクセス制御が必要となる．

本研究では，これらの課題に対応するため，以下の項目に重点を置いた．

- プレイヤの認証が非集中的に行われること
- プレイヤに割り当てられたパートが非集中的に管理されること
- 連合によってパート全体を再構成する必要がないこと

- 連合によって各プレイヤにパートを割り当て直す必要がないこと

一方，プレイヤとパートを非集中的に管理することから，責務関係の分離をアクセス制御機構によって強制することができない．責務関係の分離は，プレイヤがコミュニティに参加する際に課せられる契約などの，非形式的なポリシによって解決しなければならない．

2.4 節の要件に対して，本研究では次のアプローチをとる．

- 疎結合について
  ポリシによって規定されるコミュニティのメンバとなることで，プレイヤ同士は疎に結合する．

- 実装非依存性について
  COAC フレームワークを使うことにより，ポリシにしたがうインタラクションを実装に依存することなく明確化する．

- 柔軟な構成について
  パートの概念を導入することでコミュニティとプレイヤの結合を間接化し，パートに属するプレイヤが変わることによるシステムの変化を許容する．また，コミュニティの連合のためのポリシを提供し，連合によるコミュニティの柔軟な拡張をサポートする．

- 長い寿命と粗い粒度について
  プレイヤに対するアクセスをパートを通じて制御することで，長い寿命と粗い粒度のアクセス制御を提供する．

- チームについて
  コミュニティを単位としたアクセス制御を行う．

COAC モデルでは，プレイヤを一意に認証するメカニズムの存在を前提としない．これに対し，全てのプレイヤを一意に認証することによって，より高度なアクセス制御を実現するアプローチも存在する．このようなアプローチでは，RBAC モデルのように閉じたシステムのためのアクセス制御モデルが有効であろう．また，両者の中間的なアプローチとして，コミュニティ内のプレイヤは一意に識別されていることを前提とするアプローチも考え得る．この場合，コミュニティ内のサービスについて高度なアクセス制御を実現することができる．その一方，コミュニティの連合関係が変化する毎に，コミュニティ内のプレイヤが一意に識別されていることを何らかのメカニズムによって保証する必要がある．このような前提の下でのコミュニティのアクセス制御モデルについては，7.1.1 節で述べた．

本研究は標準化された Web サービス技術を使ったサービス利用／提供／仲介システムの実装を目的としており，コミュニティの実現モデルはシステム構築のための機能仕様と位置づけられる．このため，コミュニティの実現モデルのエージェントは COAC モデルのパートに対応し，実装段階でアクセス・コントローラやバウンダリ・コントローラとプレイヤに分離される．一方，マルチ・エージェントの開発技術を使って，コミュニティの実現モデルのエージェントを直接実装するアプローチもあり得る．その

場合には，各エージェントはアクセス・コントローラやバウンダリ・コントローラを介することなく，各々が自律的にポリシを遵守したインタラクションを行うことで，より柔軟なシステムを構成することが可能になる．ポリシ違反の動的な監視やエージェント間のネゴシエーションの記述において，より上位のメタ階層からの操作が有効であると予想される．その反面，一般にメタ階層に強く依存するモデルは実装が困難であり，このようなエージェントの効率的な実装方法については不明な点も多い．

7.5 関連研究との比較

7.5.1 RBAC/Web

RBAC/Web[27, 14] は，World Wide Web のための RBAC の実現であり，Web 上での権限の管理と権限ポリシの強制のための方法を与える．RBAC/Web は，Web 上の情報へのアクセスを，ユーザに割り当てられたロールに基づいて制御する．ユーザの権限をロールに基づいて管理することで，機能の変更や拡張に柔軟なシステムを構成することができる．例えば，機能拡張にともなって新たな権限を付与する場合には，ロールに対して権限を付与すれば十分であり，個々のユーザに個別に権限を付与する必要はない．RBAC/Web では，Web 上に構築されたシステムが一つの RBAC によって制御されるため，ロール階層や責務の静的および動的な分離がサポートされる．

ただし，Web はオープンな環境であることから，実装の段階で RBAC の各機能は制限される．まず，RBAC/Web の実装ではユーザはただ１つのログイン名を持つことが仮定され，ユーザとセッションが同一視される．また，ユーザに与えられた許可を任意の時点で知ることができないため，責務の動的な分離は制限されて実装されるか，あるいは実装されない．さらに，２つのシステムを連合する際には，連合してできたシステムのためにロールを再構成する必要があるとともに，個々のユーザについて割り当てられるロールを変更する必要がある．

COAC モデルでは，プレイヤはパートによって管理されるため，RBAC/Web の責務関係の分離を提供することはできない．その一方，連合のポリシをパート間の関係として規定しており，連合によってパートを再構成する必要はない．また，連合によって個々のプレイヤのレベルでパートとの関係が変化することもない．

7.5.2 URA97/WWW

RBAC/Web ではユーザを集中的に認証する必要があるため，前節に述べたように，Web 上での実装では各機能が制限される．URA97/WWW [46] は，URA97 と呼ばれる非集中的なユーザ－ロール割り当て管理のためのモデルを RBAC/Web に適用することで，この問題を解決する．

URA97 は Sandhu と Bhamidipati によって開発されたモデルであり，ユーザ－ロール割り当てを管理するために RBAC96 モデル [45] を利用する．すなわち，URA97/WWW では RBAC の管理を RBAC に基づいて行う．URA97/WWW のロールは通常（regular）ロールと管理（administratibe）ロールに分割され，両者は互いに疎である．管理ロールを割り当てられたユーザは，一定の条件を満たすユーザに通常ロールを割り当てるこ

とができる．この割り当ては，どの管理ロールを割り当てられたユーザが，どのような条件を満たすユーザに，どの通常ロールを割り当てることができるかを規定した，can-assign 関係に基づいて行われる．割り当ての条件は必須条件（prerequisite condition）と呼ばれ，通常ロールを割り当てるユーザが，ロール階層の下で，ある通常ロールに割り当てられている／いないを指定する論理式である．したがって，can-assign 関係では排他的なロール割り当てを規定することができ，静的な責務関係の分離を行うことができる．

URA97/WWW では，ユーザへの通常ロールの割り当てを，管理ロールを割り当てられた複数のユーザが非集中的に行っても，can-assign 関係の下でシステム全体のロール－ユーザ割り当ての整合性が確保される．このため，RBAC/Web のようにユーザ－ロール割り当てを集中的に行う必要はない．しかし，can-assign 関係はユーザに割り当てられている全てのロールを知ることができることを前提としているため，個々のユーザについてユーザ－ロール割り当て情報を管理するサブシステムは連携する必要がある．また，２つのシステムが連合する際には can-assign 関係を再構成する必要があるが，その方法は URA97/WWW では提供されない．さらに，連合の際には個々のユーザについてロールを割り当て直す必要がある．

RBAC/Web の場合と同様，COAC モデルでは URA97/WWW の責務関係の分離を提供することはできない．その一方，プレイヤに割り当てられたパートを知るために，アクセス・コントローラが連携する必要はない．また，連合によって個々のプレイヤのレベルでパートとの関係が変化することもない．さらに，COAC モデルでは連合のポリシによって，連合の際のパート間の関係を指定する方法を提供する．

7.5.3 Web 上での RBAC 実現アーキテクチャ

Park 他 [42] は，Web 上で RBAC の実現するため，ユーザに割り当てられるロールやアイデンティティなどの属性を Web 上で得るアーキテクチャを提案した．この提案には，user-pull と server-pull の２つのアーキテクチャが含まれる．user-pull は，ユーザがロールサーバからロールを得て Web サーバに与える．このため，ロールが無効になるまではユーザは異なるセッションや異なる Web サーバでロールを使うことができる．server-pull では，Web サーバがロールサーバからユーザのロールを得て RBAC で利用する．このため，Web サーバはセッションごとに必要なロール情報を得ることができ，ユーザに割り当てられるロールを動的に変えることが可能である．

しかし，これらのアーキテクチャにおいてロールは一元的に管理される必要があり，ロールの変更に対する柔軟性は低い．

RBAC/Web や URA97/WWW の場合と同様，Park 他の方法が提供する責務関係の分離を，COAC モデルでは提供することはできない．その一方，パートの一元的な管理を行わないため，パートの変更に対する柔軟性は高い．

7.5.4 I-RBAC

I-RBAC [50] は，ロールの概念に基づくアクセス制御をイントラネット上で実現することを目的としている．I-RBAC では，個々のサーバ上にあるオブジェクトへのアク

セス許可を定めるローカル・ロールと，イントラネット上にあるサーバ全体へのアクセス許可を定めるグローバル・ロールの２種類のロールが使われる．ユーザは，認証の後に割り当てられたローカル・ロールおよびグローバル・ロールにしたがって，イントラネット上のサーバのオブジェクトへのアクセスを許可される．ここで，グローバル・ロールは各サーバのローカル・ロールのリストから構成されており，イントラネット上のどのサーバのどのローカル・ロールがユーザに割り当てられるかを規定する．

ローカル・ロールが各サーバに分散して管理されるのに対し，グローバル・ロールはイントラネット上で一意に管理される必要があり，各サーバにはそのレプリカが置かれる．このため，グローバル・ロールあるいはそれを構成するローカル・ロールに変更が起こった場合には，イントラネット上の全てのサーバのグローバル・ロールを更新しなければならない．この点から，I-RBAC はスケーラビリティに問題があり，イントラネットの連合には対応できないとされる．

COAC モデルでは，連合によってコミュニティが拡張するため，グローバル・ロールに対応するパートはない．パートは，I-RBAC ローカル・ロールに対応すると考えられる．この結果，パートに変化が起こった場合には，関連するパートを更新すれば十分である．さらに，連合のポリシをパート間の関係として規定するので，連合によってパートを再構成する必要はない．

7.5.5 連合のアクセス制御

データベースの分野では，独立に開発された異種データベースを統合し協調させるデータベースの連合に関する研究の進展とともに，連合したデータベースシステムに対するアクセス制御の問題が議論されるようになった．

Taylor 他 [51] は，Web 上に分散して存在する異種データベースの連合のための，認証とアクセス制御のモデルを提案した．このモデルでは，一定の契約を結んだ関係者のコミュニティを前提とし，コミュニティが緩やかに結合した連合を考える．このようなコミュニティでは，ユーザ－ロール割り当ては局所的に管理される必要があり，個々のユーザをコミュニティを超えて管理することは困難である．この問題を解決するため，Taylor 他は，ユーザにコミュニティ内でどのような情報にアクセス可能かを示すプロファイルを割り当てる．ユーザからのアクセス要求に対して，データベースを管理するゲートウェイがプロファイルに基づいてアクセス可否を判断する．このことにより，ユーザ－ロール割り当てをアクセス対象であるデータベースのゲートウェイで局所的に管理することができる．

この方法では，ユーザに割り当てられた全てのロールを知るためには全てのゲートウェイを調べる必要があるため，RBAC の責務関係の分離は実現できない．しかし，Taylor 他は，連合データベースの読み取りのみを利用するアプリケーション分野では責務関係の分離は不要であるとする．一方，ユーザ側ではどのような情報にアクセス可能であるかを陽に知ることができないため，ユーザはゲートウェイへのアクセスを通じてアクセス許可を確認する必要がある．ここで，連合関係が変化した際にロールを再構成する枠組みが提供されていないので，ユーザは必要な情報へのアクセス許可が与えられるゲートウェイに到達するまで，連合関係の変化によって再構成されたゲートウェイに順次アクセスする必要がある．

COAC モデルでは，コミュニティ内のアクセス制御はコミュニティのアクセス制御ポリシにしたがって行われる．コミュニティのアクセス制御ポリシの変更は，コミュニティ内の複数のパートに影響する可能性があり，Taylor 他の方法のように 1 つのゲートウェイのみでアクセス制御ポリシを変更することはできない．その一方，どのパートがどのパートにアクセス可能であるかは，コミュニティのアクセス制御ポリシによってコミュニティ内で共通に認識される．また，連合関係は連合のポリシによって規定され，連合によってできたコミュニティのアクセス制御ポリシもコミュニティ内で共通に認識される．したがって，パートや連合関係が変化した場合にも，個々のプレイヤが他のパートに順次アクセスして新しく与えられるアクセス許可を確認する必要はない．

7.5.6 自律的なエージェント

1980 年代後半以降，メタレベル・アーキテクチャおよびリフレクションに関して多くの研究がなされている [35, 54]．また，エージェントの面からはリフレクションを使った自己再構成可能なエージェントに関する研究 [12] などが挙げられる．

これらの研究の多くが実行時にメタレベルからの介入によって動作を変えるシステムの実現を目的とするのに対し，本研究ではメタ階層を使って記述した実装モデルをもとに人手であるいは半自動的に，実行可能なプログラムを作成するアプローチをとる．本研究の目的は標準化された Web サービス技術によるコミュニティの実現にあり，コミュニティ間の協調機能を持ったシステムの実現を意図している．この結果，本研究の実現モデルは基本的に人間が読み解析できるものであれば十分であり，実行可能であることが要求される場合に比べて記述の自由度を高く設定できる．その一方，実現モデルからの実装にはメタ記述からプログラムコードへの変換が含まれるので，一般にプログラムの自動生成は容易ではない．本研究では，論理的なシステム・アーキテクチャに基づいて，COAC フレームワークによってメタ記述の範囲を制限することで，実現モデルからの実装の困難を避けた．しかし，このことは実現モデルの表現能力を制限することであり，パート間の動的なネゴシエーションなどは扱えない．将来リフレクション分野の研究が進めば，コミュニティのアクセス制御ポリシからアクセス制御機構を自動生成し，自律的なエージェントによる直接実装が可能になることも考えられる．

第8章　結論

ネットワークを通じたサービスの提供／利用／仲介技術は，急速に変化する社会環境に適用する柔軟なシステムを構築するための技術として不可欠な存在となっている．このような技術に基づくシステムは中央集約的な管理が困難なオープンなシステムであり，プレイヤを不正なアクセスから守るアクセス制御のしくみを，システムの柔軟性を損なうことなく作ることは重要な問題である．

この問題は，オープンなシステムのためのアクセス制御モデルが存在しないため，未解決であった．例えばアクセス制御モデルとして広く知られているRBACモデルでは，システムの全ての構成要素を知ることができる閉じたシステムを前提としており，任意の時点で全てのプレイヤを一意に識別できることが前提である．また，RBACモデルでは，RBACによって制御されたシステムと別のRBACによって制御されたシステムを連合させる場合のアクセス制御を，プレイヤの一意性を保つためのRBACの再構築なしには取り扱うことができない．

本研究では，オープンなシステムのアクセス制御で優先されるべき要件として，以下を設定した．

- プレイヤの認証が非集中的に行われること
- プレイヤへのパートの割り当てが非集中的に管理されること
- 連合によってパート全体を再構成する必要がないこと
- 連合によって各プレイヤにパートを割り当てなおす必要がないこと

このような要件を満たすために，本研究ではオープンなシステムのためのアクセス制御モデルとしてCOACモデルを提案した．COACモデルでは，システムを構成するプレイヤの集まりをコミュニティと呼び，コミュニティに対してアクセス制御ポリシを設定する．また，COACモデルのコミュニティを実現するため，COACフレームワークを提案した．

RBACモデルと異なり，COACモデルでは個々のプレイヤはアクセス制御ポリシには現れない．したがって，各々のプレイヤを一意に識別する必要はなく，プレイヤへのパートの割り当ても各パート毎に管理すれば良い．COACモデルでは，コミュニティの連合を取り扱うため，アクセス制御ポリシ・モデルの中に連合のポリシの概念を含む．連合のポリシは，連合するコミュニティのパート間のアクセス許可の委譲を定め，連合によってできたコミュニティのアクセス制御ポリシを規定する．このことで，連合によるパートの再構成とプレイヤへのパートの再割り当てを避けることができる．またCOACモデルでは，連合するコミュニティのアクセス制御ポリシと連合によってできたコミュニティのアクセス制御ポリシが分離されるための条件について示した．

一方で，COAC モデルではプレイヤが属するパートに基づいてアクセス制御を行うため，RBAC モデルのような責務の分離を行うことはできない．すなわち，プレイヤが複数のパートに属することで生じるパート間の情報フローを，アクセス制御機構によって禁止することができない．このような情報フローの禁止が優先度の高い問題になるかどうかは，アプリケーション・ドメインに依存する．本研究では，プレイヤがコミュニティに参加する際に課せられる契約などの，非形式的なポリシによって解決できるものとして，この問題の優先度は低くした．ケーススタディの範囲では，Taylor 他 [51] と同様，責務の分離が強制できないことに問題はなかった．

COAC モデルでは，連合するコミュニティのパート間に１対１の対応関係がない場合には，隔離的な連合のポリシを設定することが難しい．パートのグループの概念を導入するなどによる，１対１の対応関係がない場合への隔離的な連合のポリシの範囲の拡大は，今後の課題である．また，RBAC などのアクセス制御ポリシと同様に，コミュニティのアクセス制御ポリシではアクセスの可否を規定するのみで，アクセスの順番を規定することはできない．アクセス順序を含めたアクセス制御ポリシ・モデルへの拡張も，今後の検討課題である．

謝辞

本論文の執筆にあたり，御指導や御助言をいただいた多くの方々に心より感謝いたします．

はじめに，本研究全般に関して御指導と御鞭撻を賜りました，国立情報学研究所／東京大学の本位田真一教授に深い感謝の意を表します．本研究は，本位田教授のソフトウェア工学とエージェント技術に関する深い知見に導いていただきました．そして，本論文をまとめるにあたり御指導と御鞭撻を賜りました，国立情報学研究所／総合研究大学院大学の中島震教授，武田英明教授，佐藤健教授に深い感謝の意を表します．また，本論文をまとめるにあたり御教示を賜りました，東京工業大学の渡部卓雄助教授，国立情報学研究所／総合研究大学院大学の細部博史助教授に深い感謝の意を表します．

本研究の一部は，情報処理振興事業協会（現 独立行政法人 情報処理推進機構）が産業科学技術研究開発制度の一環として新エネルギー・産業技術総合開発機構から委託を受けて実施した「新ソフトウェア構造化モデルの研究開発」プロジェクトにおいて行われました．このプロジェクトで日頃御討論いただいた研究員の方々に，心より感謝いたします．特に，株式会社 東芝の大須賀昭彦氏，株式会社 三菱総合研究所の粂野文洋氏，株式会社 管理工学研究所の松浦佐江子氏（現 芝浦工業大学）には，本研究に関して計り知れない御示唆と御助言をいただきました．厚く御礼申し上げます．

北陸先端科学技術大学院大学の二木厚吉教授，スイス連邦工科大学の David Basin 教授には，形式手法について手解きいただきました．深く感謝いたします．

本研究の機会を与えて下さり，御指導，御鞭撻を賜った，株式会社 日立製作所の多くの方々に心より感謝いたします．特に，システム開発研究所の小坂満隆前所長と前田章所長，舩橋誠壽主管研究長，宝木和夫主管研究長，田中厚センタ長の御理解と御鞭撻は，本論文をまとめるうえで大きな支えとなりました．厚く御礼申し上げます．さらに，舩橋誠壽主管研究長には自律分散サービスシステムについて，宝木和夫主管研究長にはコンピュータ・セキュリティについて，御指導，御教示いただきました．また，山野紘一氏（現 ブナソフト），高野明彦氏（現 国立情報学研究所）には，計算機言語について御教示いただきました．野木兼六氏（現 神奈川工科大学），大槻繁氏（現 株式会社 一），金藤栄孝氏には，ソフトウェア工学について御教示いただきました．桜庭健年氏には，セキュリティ・ポリシ・モデルについて御教示いただきました．他にも，日頃御討論いただいている多くの方々に，感謝いたします．

最後に，心の支えとなり元気づけてくれた家族に感謝します．

参考文献

[1] Agha, G., S. Frolund, W.Y. Kim, R. Panwar, A. Patterson, and D. Sturman: Abstraction and Modularity for Concurrent Computing, *Research Directions in Concurrent Object-Oriented Programming*, the MIT Press, pp. 3–21 (1993)

[2] *Role Based Access Control*, American National Standards Institute, Inc. (2004)

[3] Badger,L., Sterne,D.F., Sherman,D.L., Walker,K.M., and Haghighat,S.A.: Practical Domain and Type Enforcement for UNIX, *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pp.66–77 (1995)

[4] Basin, D., Kuruma, H., Takaragi, K., and Wolff, B.: Verification of a Signature Architecture with HOL-Z, *Proceedings of FM05*, LNCS 3582, Springer-Verlag (2005)

[5] Bell, D.E., and LaPadula, L.J.: Secure Computer Systems: Mathematical Foundations, *Technical Report MTR-2547*, MITRE Corporation (1973)

[6] Bell, D.E., and LaPadula, L.J.: Secure Computer Systems: Unified Exposision and Mulics Interpritation, *Technical Report MTR-2997*, MITRE Corporation (1976)

[7] Berners-Lee,Tim,Hendler, James, and Lassila,Ora: The Semantic Web, *Scientific American*, May (2001)

[8] Biba, K.J.: Integrity Considerations for Secure Computer System, *Technical Report MTR-3153*, MITRE Corporation (1975)

[9] Bishop, M.: *Computer Security*, Addison-Wesley (2003)

[10] Bradshaw, J.M. ed.: *Software Agents*, AAAI Press (1997)

[11] Brewer, D., and Nash, M.: The Chinese Wall Security Policy, *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pp. 206–214 (1989)

[12] Charlton, P.: Self-Configurable Software Agents, *Advances in Object-Oriented Metalevel Architectures and Reflection*, CRC Press, pp. 103–127 (1996)

[13] Clark, D.D., and Wilson, D.R.: A Comparison of Commercial and Military Computer Security Policies, *Proceedings of 1987 IEEE Symposium on Security and Privacy*, pp.184–194 (1987)

[14] Ferraiolo, D.F., Barkley, J.F., and Kuhn, D.R.: A Role-Based Access Control Model and Reference Inplementation Within a Corporate Intranet, *ACM Trans. on Information and System Security*, Vol.2, No.1, pp.34–64 (1999)

[15] Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, R., and Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control, *ACM Trans. on Information and System Security*, Vol.4, No.3, pp.224–274 (2001)

[16] Ferraiolo, D.F., Kuhn, D.R, and Chandramouli, R: *Role-Based Access Control*, Artech House (2003)

[17] Fraser,T.: LOMAC: Low Water-Mark Integrity Protection for COTS Environments, *Proceedings of the 2000 IEEE Symposium on Security and Privary*, pp.230–245 (2000)

[18] Harrison, M., Ruzzo, W., Ullman, J.: Protection in Operating Systems, *Communication of the ACM*, Vol.19, No.8, pp.337–363 (1977)

[19] Hewitt, C. and Peter de Jong: Open Systems, *On Conceptual Modelling*, Springer-Verlag, pp. 147–162 (1984)

[20] オペレーティングシステムのアクセスコントロール機能におけるセキュリティポリシーモデル, 情報処理進行事業協会セキュリティセンター, http://www.ipa.go.jp/security/awareness/administrator/security-model.pdf

[21] アクセス制御に関するセキュリティポリシーモデルの調査報告書, 情報処理推進機構 (2004)

[22] 本位田真一, 飯島正, 大須賀昭彦: エージェント技術, 共立出版 (1999)

[23] Jones,A.K., Lipton,R.J., and Snyder,L.: A Linear Time Algorithm for Deciding Security, *Proceedings of the 17th Symposium on the Foundations of Computer Science*, pp.33–41 (1976)

[24] 清野正樹, 来間啓伸, 今村誠: セマンティック Web とオントロジ記述言語, 情報処理, Vol. 43, No. 7, pp.727–733 (2002)

[25] Koch, M., Mancini, L.V., and Parisi-Presicce, F.: On the Specification and Evolution of Access Control Policies, *Proceedings of SACMAT'01*, pp.121–130 (2001)

[26] Krafzig, D., Banke, K., and Slama, D.: *Enterprise SOA – Service-Oriented Architecture Best Practices*, Pearson Education, Inc. (2005)

[27] Kuhn, D., Barkley, J., Cincotta, V., Ferraiolo, D., and Gavriella, S.: Role Based Access Control for the World Wide Web, *Proceedings of 20th National Information Systems Security Conference*, pp.331–340 (1997)

[28] 来間啓伸, 大須賀昭彦, 本位田真一: 協調アーキテクチャに基づくソフトウェア・モジュールの仕様記述モデル, 情報処理学会論文誌, Vol. 37, No. 6, pp. 1171–1186 (1996)

[29] Kuruma, H., and Futatsugi, K.: Incremental Specification based on the Combination of Data Types, Futatsugi, K., Goguen, J., Meseguar, J. eds, *OBJ/CafeOBJ/Maude at Formal Methods '99*, The Theata Foundation, pp.95–114 (1999)

[30] 来間啓伸, 本位田真一: ネットワーク上のサービスの動的な検索／連携のための多階層コミュニケーション・モデル, 杉山, 藤田編, ソフトウェア工学の基礎 VIII, 近代科学社, pp.187–190 (2001)

[31] 来間啓伸, 本位田真一: Web サービス・コミュニティ連合のためのポリシ・モデル, 第 1 回ディペンダブルソフトウェアワークショップ論文集 (2004)

[32] 来間啓伸, 本位田真一: ポリシに基づく Web サービス・コミュニティ連合のモデル, 情報処理学会論文誌, Vol. 45, No. 6, pp. 1593–1602 (2004)

[33] 来間啓伸, 谷津弘一, 中島震, 本位田真一: アドホックネットワーク・プロトコル仕様の形式記述と検証, 第 2 回ディペンダブルソフトウェアワークショップ論文集, pp.97–104 (2005)

[34] Kuruma, H., and Honiden, S.: A Model for Policy Based Service Community, *Proceedings of 7th International Conference on Enterprise Information Systems*, Vol. 3, pp.360–366 (2005)

[35] Maes, P.: Issues in Computational Reflection, *Meta-Level Architectures and Reflection*, Elsevier Science Publishers B.V. (North-Holland), pp. 21–35 (1988)

[36] 丸山宏: XML と Web サービスのセキュリティ, 共立出版 (2004)

[37] Matsuura, S., Kuruma, H., and Honiden, S.: EVA: A Flexible Programming Method for Evolving Systems, *IEEE Trans. on Software Engineering*, Vol. 23, No. 5, pp.296–313 (1997)

[38] 松浦佐江子, 来間啓伸, 本位田真一: EVA: 仕様変更プロセスを用いたプログラム開発支援システム, 情報処理学会論文誌, Vol. 38, No. 1, pp.114–130 (1997)

[39] 宮崎邦彦,Basin, D., 来間啓伸, 宝木和夫, 手塚悟: 形式手法を用いたディジタル署名システムの安全性評価, コンピュータソフトウェア, Vol. 22, No. 2, pp.74–84 (2005)

[40] 森田幸伯, 津田宏, 清水昇, 布目光生, 来間啓伸, 佐藤宏之: セマンティック Web のツール, 情報処理, Vol. 43, No. 7, pp.734–741 (2002)

[41] Murthy, C. Siva Ram., and Manoj, B. S.: *Ad Hoc Wireless Networks*, Prentice Hall (2004)

[42] Park, J.S., Sandhu, R., and Ahn, G.-J.: Role-Based Access Control on the Web, *ACM Trans. on Information and System Security*, Vol.4, No.1, pp. 37–71 (2001)

[43] ランボー,J., プラハ,M., プレメラニ,W., エディ,F., and ローレンセン,W. 著, 羽生田栄一監訳: オブジェクト指向方法論OMT, トッパン (1991)

[44] Samarati, P., and Bertino, E., and Jajodia, S.: An Authorization Model for a Distributed Hypertext Systems, *IEEE Transactions on Knowledge and Data Engineering*, Vol.8, No. 4, pp. 555 – 562 (1996)

[45] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., and Youman, C.E.: Role Based Access Control Models, *IEEE Computer*, pp.38–47 (1996)

[46] Sandhu, R., and Park, J. S.: Decentralized User-Role Assignment for Web-based Intranets, *Proceedings of the third ACM Workshop on Role-based Access Control*, pp. 1 – 12 1998

[47] Singh, M. P., and Huhns, M. N.: *Service-Oriented Computing*, John Wiley & Sons (2005)

[48] Spivey, J.M.: *The Z Notation 2nd edition*, Prentice Hall (1992)

[49] 武田英明: Webインテリジェンスの可能性–知識情報基盤としてのWWW–, 計測と制御, Vol. 42, No. 6, pp.508–511 (2003)

[50] Tari, Z., and Chan, S.-W.: A Role-based Access Control for Intranet Security, *IEEE Internet Computing*, Vol. 1, No.5, pp. 24 – 34 1997

[51] Taylor, K., and Murty, J.: Implementing Role Based Access Control for Federated Information Systems on the Web, *Proceedings of the Australasian Information Security Workshop 2003* (2003)

[52] Toyouchi, J., Funabashi, M. and Strick, L.: Service Integration Platform based on TINA 3-tier Model and Interfaces, *TINA 2000 CONFERENCE Conference Proceedings*, pp. 21–26 (2000)

[53] De Capitani di Vimercati, S., and Samarati, P.: Access Control in Federated Systems, *Proceedings of the 1996 Workshop on New Security Paradigms*, pp. 87 – 99 (1996)

[54] 渡部卓雄: リフレクション, コンピュータソフトウェア, Vol. 11, No. 3, pp. 5–14 (1994)

[55] Weissman,C.: Security Controls in the ADEPT-50 Time Sharing System, *AFIPS Conference Proceedings*, Vol. 35, pp.119–133 (1969)

[56] Wooldridge, M. and N. R. Jennings: Agent Theories, Architectures, and Languages: A Survey, *Intelligent Agents*, LNAI890, pp. 1–39, Springer-Verlag (1995)

[57] *Autonomous Decentralized Service Systems Whitepaper Version 1.0*, OMG ADSS Domain Special Interest Group, ads/97-12-01 (1997)

[58] *Business Process Execution Language for Web Services version 1.1*, IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel (2003)

[59] *OWL Web Ontology Language*, W3C Recommendation 10 February 2004, http://www.w3.org (2004)

[60] *SOAP Version 1.2*, W3C Recommendation 24 June 2003, http://www.w3.org (2003)

[61] *UDDI Technical White Paper*, Universal Description, Discovery and Integration, http://www.uddi.org (2000)

[62] *Web Services Choreography Interface (WSCI) 1.0*, W3C Note 8 August 2002, http://www.w3.org (2002)

[63] *Web Services Description Language (WSDL) 1.1*, W3C Note 15 March 2001, http://www.w3.org (2001)

[64] *Web Services Security: SOAP Message Security 1.0*, OASIS Standard 200401, http://www.oasis-open.org (2004)

[65] *Security in a Web Services World: A Proposed Architecture and Roadmap*, A joint security whitepaper from IBM Corporation and Microsoft Corporation, Version 1.0, http://www-106.ibm.com/developerworks/library/ws-secmap (2002)

[66] *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation 04 February 2004, http://www.w3.org (2004)

研究業績

主著論文

学術論文（査読付き）

1. 来間啓伸, 本位田真一: ポリシに基づく Web サービス・コミュニティ連合のモデル, 情報処理学会論文誌, Vol. 45, No. 6, pp.1593–1602 (2004)

2. 来間啓伸, 大須賀昭彦, 本位田真一: 協調アーキテクチャに基づくソフトウェア・モジュールの仕様記述モデル, 情報処理学会論文誌, Vol. 37, No. 6, pp.1171–1186 (1996)

国際会議

1. Kuruma, H., and Honiden, S.: A Model for Policy Based Service Community, *Proceedings of 7th International Conference on Enterprise Information Systems*, Vol. 3, pp. 360–366 (2005)

2. Kuruma, H., Futatsugi, K.: Incremental Specification Based on the Combination of Data Types, K. Futatsugi, J. Goguen, J. Meseguer eds, *OBJ/CafeOBJ/Maude at Formal Methods '99*, The Theta Foundation, pp.95–114 (1999)

全国大会・研究会

1. 来間啓伸, 谷津弘一, 中島震, 本位田真一: アドホックネットワーク・プロトコル仕様の形式記述と検証, 第２回ディペンダブルソフトウェアワークショップ (DSW05) (2005)

2. 来間啓伸, 本位田真一: Web サービス・コミュニティ連合のためのポリシ・モデル, 第１回ディペンダブルソフトウェアワークショップ (DSW04) (2004)

3. 来間啓伸, 本位田真一: ネットワーク上のサービスの動的な検索/連携のための多階層コミュニケーション・モデル, 杉山, 藤田編, ソフトウェア工学の基礎 VIII, 近代科学社, pp.187–196 (2001)

4. 来間啓伸, 本位田真一: サービス仲介システムの連合のための抽象エージェントモデル, 電気学会情報システム研究会 IS-01-30, 13 (2001)

5. 来間啓伸, 二木厚吉: 形式仕様言語 CafeOBJ を用いたテキストエディタの仕様記述, 情報処理学会第５６回全国大会, 3K-07 (1998)

6. 来間啓伸, 粂野文洋, 蓬莱尚幸, 松浦佐江子, 本位田真一: エージェント指向言語 Flage, 第 14 回 IPA 技術発表会論文集 (1995)

7. 来間啓伸, 本位田真一: Flage アーキテクチャにおける記述モデル, 情報処理学会第５１回全国大会, 4N-7, Vol. 5, pp.221–222 (1995)

8. 来間啓伸, 本位田真一: 協調型ソフトウェア・アーキテクチャに基づく開放型システムの仕様記述モデル, 情処研報, Vol. 95, No. 11, pp.135–140 (1995)

9. 来間啓伸, 大槻繁: 集合に基づく形式的言語を使ったソフトウェア仕様の記述形態について, 情報処理学会ソフトウェア工学研究会, 83-13, pp.97–104 (1992)

10. 来間啓伸, 高野明彦, 山野紘一: 関数型言語への構文処理機能の導入, 情報処理学会第３６回全国大会, pp.755–756 (1988)

11. 来間啓伸, 高野明彦, 山野紘一: 関数型言語 VULCAN への文法記述機能の導入, 情報処理学会第３２回全国大会, pp.551-552 (1986)

主著以外の論文

学術論文（査読付き）

1. 宮崎邦彦, Basin, D., 来間啓伸, 宝木和夫, 手塚悟: 形式手法を用いたディジタル署名システムの安全性評価, コンピュータソフトウェア, Vol. 22, No. 2, pp.74–84 (2005)

2. Matsuura, S., Kuruma, H., and Honiden, S.: EVA: A Flexible Programming Method for Evolving Systems, *IEEE Trans. on Software Engineering*, Vol. 23, No. 5, pp.296–313 (1997)

3. 松浦佐江子, 来間啓伸, 本位田真一: EVA: 仕様変更プロセスを用いたプログラム開発支援システム, 情報処理学会論文誌, Vol. 38, No. 1, pp.114–130 (1997)

国際会議

1. Basin, D., Kuruma, H., Takaragi, K., and Wolff, B.: Verification of a Signature Architecture with HOL-Z, *Proceedings of FM05*, LNCS 3582, Springer-Verlag (2005)

その他（解説記事）

1. 清野正樹, 来間啓伸, 今村誠: セマンティック Web とオントロジ記述言語, 情報処理学会誌, Vol. 43, No. 7, pp.727–733 (2002)

2. 森田幸伯, 津田宏, 清水昇, 布目光生, 来間啓伸, 佐藤宏之: セマンティック Web のツール, 情報処理学会誌, Vol. 43, No. 7, pp.734–741 (2002)

付 録A　RBACリファレンスモデルの形式記述

ANSI 標準 [2] では，RBAC のリファレンスモデルを Z 記法 [48] のサブセットを使って形式的に定義しているが，その中には少なからぬ誤りが含まれている．ここでは，誤りを修正して定義し直したリファレンスモデルを示す．なお，以下の記述に使った Z 記法の簡単な説明を，付録 B に掲げる．

A.1　CoreRBAC

Core RBAC は，次の 6 つの基本データ要素を導入する．

- USERS
  ユーザの集合
- ROLES
  ロールの集合
- PRMS
  許可の集合
- OPS
  操作の集合
- OBS
  対象の集合
- SESSIONS
  セッションの集合

[*NAME*]

CoreRBAC

$$
\begin{aligned}
&USERS : \mathbb{P}\, NAME \\
&ROLES : \mathbb{P}\, NAME \\
&PRMS : OPS \leftrightarrow OBS \\
&OPS : \mathbb{P}\, NAME \\
&OBS : \mathbb{P}\, NAME \\
&SESSIONS : \mathbb{P}\, NAME \\
&UA : USERS \leftrightarrow ROLES \\
&PA : PRMS \leftrightarrow ROLES \\
&session\_users : SESSIONS \rightarrow USERS \\
&session\_roles : SESSIONS \rightarrow \mathbb{P}\, ROLES \\
&avail\_session\_perms : SESSION \rightarrow \mathbb{P}\, PRMS \\
&assigned\_users : ROLES \rightarrow \mathbb{P}\, USERS \\
&assigned\_permissions : ROLES \rightarrow \mathbb{P}\, PRMS
\end{aligned}
$$

$$
\begin{aligned}
&\forall\, s : SESSIONS \bullet session\_roles(s) \subseteq \\
&\quad \{r : ROLES \mid (session\_users(s), r) \in UA \bullet r\} \\
&\forall\, s : SESSIONS \bullet avail\_session\_perms(s) = \\
&\quad \bigcup \{r : ROLES \mid r \in session\_roles(s) \bullet \\
&\qquad assigned\_permissions(r)\} \\
&\forall\, r : ROLES \bullet assigned\_users(r) = \\
&\quad \{u : USERS \mid (u, r) \in UA \bullet u\} \\
&\forall\, r : ROLES \bullet assigned\_permissions(r) = \\
&\quad \{p : PRMS \mid (p, r) \in PA \bullet p\}
\end{aligned}
$$

A.2 ロール階層

ロール階層は，許可に基づくロール間の継承（inheritance）関係を定義する．つまり，ロール r_1 がロール r_2 を継承する必要十分条件は，r_2 に与えられる全ての特権が r_1 に与えられること，と定義する．階層的 RBAC では，ロール階層 RH が導入される．

無制限ロール階層は，次の基本要素を導入する．

- $\succeq$
  継承関係 (RH)
- *authorized_permissions*
  ロールに与えられる許可
- *authorized_users*
  ロール階層のもとでロールに属するユーザ

$r_1 \succeq r_2$ である時かつその時に限り，r_2 の許可は同時に r_1 の許可であり，r_1 のユーザは同時に r_2 のユーザである．

GeneralRoleHierarchies

$$
\begin{array}{l}
CoreRBAC \\
\succeq : \mathbb{P}(ROLES \times ROLES) \\
authorized\_users : ROLES \to \mathbb{P}\, USERS \\
authorized\_permissions : ROLES \to \mathbb{P}\, PRMS \\
\hline
\forall\, r_1 \in ROLES \bullet r_1 \succeq r_1 \\
\forall\, r_1, r_2 \in ROLES \bullet r_1 \succeq r_2 \land r_2 \succeq r_1 \Rightarrow r_1 = r_2 \\
\forall\, r_1, r_2, r_3 \in ROLES \bullet r_1 \succeq r_2 \land r_2 \succeq r_3 \Rightarrow r_1 \succeq r_3 \\
\forall\, r \in ROLES \bullet \\
\quad authorized\_users(r) = \{r' \succeq r \land (u, r') \in UA \bullet u\} \\
\forall\, r \in ROLES \bullet \\
\quad authorized\_permissions(r) = \{r \succeq r' \land (p, r') \in PA \bullet p\}
\end{array}
$$

以下では，ロールの直接の継承関係を $\gg$ で表す．すなわち，

$$r_1 \gg r_2 \Leftrightarrow r_1 \succeq r_2 \land r_1 \neq r_2 \land (\forall\, r_3, r_1 \succeq r_3 \succeq r_2 \Rightarrow r_3 = r_1 \lor r_3 = r_2)$$

無制限ロール階層ではロール階層として任意の半順序関係が許されるため，2つ以上のロールから許可やユーザメンバーシップを継承する機能（多重継承）が提供される．

制限付きロール階層は，ロールが2つ以上の下位ロールを持たないように無制限ロール階層の継承関係を制限する．

LimitedRoleHierarchies

$$
\begin{array}{l}
GeneralRoleHierarchies \\
\hline
\forall\, r, r_1, r_2 \in ROLES \bullet r \succeq r_1 \land r \succeq r_2 \Rightarrow r1 = r2
\end{array}
$$

A.3 責務関係の静的な分離

責務の静的な分離では，次の基本要素が導入される．

- SSD
  責務の静的な分離制約

SSD の要素はロールの集合と 2 以上の自然数（個数）の対であり，各々の対について，ロールの集合から指定された個数以上のロールがユーザに割当られないことを要求する．以下では，SSD の要素であるロールの集合と個数の対を SSD ロール集合と呼ぶ．

StaticSeparationOfDuty

$CoreRBAC$
$SSD : \mathbb{P}\, ROLES \times \mathbb{N}$
$ssd\_set : SSD \to \mathbb{P}\, ROLES$
$ssd\_card : SSD \to \mathbb{N}$

$\forall (rs, n) \in SSD;\ t \subseteq rs \bullet \#t \geq n \Rightarrow$
$\quad \bigcap \{r \in t \bullet assigned\_users(r)\} = \emptyset$
$\forall (rs, n) \in SSD \bullet ssd\_set((rs, n)) = rs$
$\forall (rs, n) \in SSD \bullet ssd\_card((rs, n)) = n$
$\forall\, ssd \in SSD \bullet ssd\_card(ssd) \geq 2 \wedge$
$\quad ssd\_card(ssd) \leq \#ssd\_set(ssd)$

ここで，基本要素以外の変数は以下を表す.

- ssd_set
  SSD からロール集合を取り出す関数
- ssd_card
  SSD から個数を取り出す関数

A.4 責務関係の動的な分離

責務の動的な分離では，次の基本要素が導入される.

- DSD
  責務の動的な分離制約

DSD の要素はロールの集合と 2 以上の自然数（個数）の対であり，各々の対について，ロールの集合から指定された個数以上のロールを同時に活生化する主体がないことを要求する．以下では，DSD の要素を DSD ロール集合と呼ぶ.

DynamicSeparationOfDuty

$CoreRBAC$
$DSD : \mathbb{P}\, ROLES \times \mathbb{N}$
$dsd\_set : DSD \to \mathbb{P}\, ROLES$
$dsd\_card : DSD \to \mathbb{N}$

$\forall\, s \in SESSIONS;\ rs \in \mathbb{P}\, ROLES;\ role\_subset \in \mathbb{P}\, ROLES;\ n \in \mathbb{N} \bullet$
$\quad (rs, n) \in DSD \wedge role\_subset \subseteq rs \wedge role\_subset \in session\_roles(s)$
$\quad \Rightarrow \#role\_subset < n$
$\forall (rs, n) \in DSD \bullet dsd\_set((rs, n)) = rs$
$\forall (rs, n) \in DSD \bullet dsd\_card((rs, n)) = n$
$\forall\, dsd \in DSD \bullet dsd\_card(dsd) \geq 2 \wedge$
$\quad dsd\_card(dsd) \leq \#dsd\_set(dsd)$

ここで，基本要素以外の変数は以下を表す.

- dsd_set
  DSD からロール集合を取り出す関数
- dsd_card
  DSD から個数を取り出す関数

付 録B　RBACリファレンスモデルの記法

ここでは，付録 A の RBAC リファレンスモデルの記述で使った Z 記法を，簡単に説明する．

B.1　スキーマ

リフォレンスモデルは，次の枠組みで記述される．

リファレンスモデル名
＜宣言部＞
＜述語部＞

これは，リファレンスモデルが取り得る状態空間を規定する。

＜宣言部＞には，リファレンスモデルで導入される変数名と，その型が＜述語＞を使って規定される。

例：集合 *NAME* のべき集合を型に持つ変数 *USER* （つまり，*USER* の値は *NAME* の部分集合）は，次のように宣言される。

$$USER : \mathbb{P}\, NAME$$

また，他のリファレンスモデルを包含する場合には，包含されるリファレンスモデル名も＜宣言部＞に記述される。その場合，包含するリファレンスモデルの＜宣言部＞は包含されるリファレンスモデルの＜宣言部＞で，＜述語部＞は包含されるリファレンスモデルの＜述語部＞で，それぞれ拡張される。

＜述語部＞には，宣言部で規定した変数の取り得る値の間の関係が，＜述語＞を使って規定される。＜述語部＞には複数の＜述語＞を改行をはさんで列挙することができ，論理的にはそれらの＜述語＞は論理積によって結合されたものとみなされる。

B.2　＜述語＞の記法

1. Equality

＜式＞ = ＜式＞

＜述語＞ $x = y$ は，x と y が同一の対象であるとき真

2. Membership

   ＜式＞ ∈ ＜式＞

   ＜述語＞ $x \in S$ は，x が集合 S の要素であるとき真

3. Propositional connectives

   ¬ ＜述語＞
   ＜述語＞ ∧ ＜述語＞
   ＜述語＞ ∨ ＜述語＞
   ＜述語＞ ⇒ ＜述語＞
   ＜述語＞ ⇔ ＜述語＞

   - $\neg P$
     P が真でない
   - $P_1 \wedge P_2$
     P_1 と P_2 がともに真
   - $P_1 \vee P_2$
     P_1 と P_2 の少なくとも一方が真
   - $P_1 \Rightarrow P_2$
     P_1 が真でないか，あるいは P_1 と P_2 がともに真
   - $P_1 \Leftrightarrow P_2$
     P_1 と P_2 がともに真であるか，あるいはともに真でない

4. Quantifiers

   ∀＜スキーマテキスト＞ • ＜述語＞
   ∃＜スキーマテキスト＞ • ＜述語＞

   $\forall S \bullet P$ は，S で導入された変数の S の制約を満たす全ての値について，＜述語＞ P が真ならば真．$\exists S \bullet P$ は，S で導入された変数の少なくともひとつの値が，S の制約と＜述語＞ P を満たすならば真．ここで，＜スキーマテキスト＞は以下の要素からなる．

   - 変数とその型の宣言
   - 変数の値を制約する＜述語＞

   例：

   $$\forall s : SESSIONS \bullet \\ session\_roles(s) \subseteq \{r : ROLES \mid (session\_users(s), r) \in UA \bullet r\}$$

B.3 ＜式＞の記法

1. Set display

   { ＜式＞, .., ＜式＞ }

   $\{x_1, ..., x_n\}$ の値は，$x_1, ..., x_n$ を要素とする集合．{} は空集合．

2. Tuple

   (＜式＞, .., ＜式＞)

   $(x_1, ..., x_n)$ の値は，$x_1, ..., x_n$ を要素とする n-タプル．

3. Set complehension

   { ＜スキーマテキスト＞ • ＜式＞ }

   $\{S \bullet E\}$ の値は，S で導入された変数が S の制約を満たす全ての値について，＜式＞ E がとる値を要素とする集合．例：

   $$\{r : ROLES \mid (session\_users(s), r) \in UA \bullet r\}$$

   RBAC の形式記述の中では，変数とその型の宣言は暗黙のものとして，＜述語＞のみから構成された＜スキーマテキスト＞も使われている．

4. Power set

   $\mathbb{P}$ ＜式＞

   $\mathbb{P} S$ の値は，集合 S の全ての部分集合の集合．

5. Cartesian product

   ＜式＞ × ... × ＜式＞

   $S_1, ..., S_n$ が集合であるとき，$S_1 \times ... \times S_n$ の値は，タプル $(x_1, ..., x_n)$ の集合．ここで，各々の $i(1 \leq i \leq n)$ に対して $x_i \in S_i$．

6. Function application

   ＜式＞ ＜式＞

   $f\ x$ は，関数 f の引数 x への適用．$f(x)$ とも書かれる．

7. Conditional expression

   if ＜述語＞ *then* ＜式＞ *else* ＜式＞

   if P *then* E_1 *else* E_2 の値は，P が真ならば E_1 の値，そうでなければ E_2 の値．

B.4 ＜式＞で使われる記号

Z 記法では，＜式＞で使われる記号の多くは Mathematical Tool-kit の中で定義される．Mathematical Tool-kit は数学コンポーネント定義のライブラリであり，Z 記法に語彙を提供する．

1. Natural numbers

   $\mathbb{N}$

   $\mathbb{N}$ は，自然数の集合．

2. Non-membership

   $\notin$

   $x \notin S$ は，x が集合 S の要素でないとき真

3. Empty set

   $\emptyset$

   $\emptyset$ は，空集合を表す．

4. Subset relation

   $\subseteq$

   $S \subseteq T$ は，集合 S の全ての要素が集合 T の要素であるならば真．

5. Set union

   $\cup$

   $S \cup T$ の値は，要素が集合 S または集合 T または S と T の要素である集合．

6. Set intersection

   $\cap$

   $S \cap T$ の値は，要素が集合 S と集合 T の両者の要素である集合．

7. Set difference

   $\setminus$

   $S \setminus T$ の値は，集合 S の要素であるが集合 T の要素ではない要素からなる集合．

8. Generalized union

   $\bigcup$

   A が集合の集合であるとき，$\bigcup A$ の値は A の要素である集合の要素を全て含む集合．

9. Generalized intersection

   $\bigcap$

   A が集合の集合であるとき，$\bigcap A$ の値は A の要素である集合全てに共通に属する要素からなる集合.

10. Binary relations

    $\leftrightarrow$

    X と Y が集合であるとき，$X \leftrightarrow Y$ は X と Y の間の 2 項関係であり，$X \times Y$ の部分集合.

11. Maplet

    $\mapsto$

    $x \mapsto y$ は，順序対 (x, y) を表す.

12. Domain anti-restriction

    $\lhd$

    関係 R によって x が y に関係付けられかつ x が集合 S の要素でないとき，x は関係 $S \lhd R$ によって y に関係付けられる.

13. Reflexive-transitive closure

    *

    R^* は，関係 R の反射推移閉包を表す.

14. Total function

    $\rightarrow$

    $X \rightarrow Y$ は，X から Y への全関数を表す.

15. Numbers of members of a set

    $\#$

    $\#S$ は，有限集合 S の要素の数を表す.

付 録C　エージェントの記法

```
＜エージェント＞  ::=
    "{" エージェント記述子 "|"
    [ "<attribute>" ＜属性＞ ("," ＜属性＞)* ";;" ]
    [ "<method>" ＜メソッド＞ ("," ＜メソッド＞)* ";;" ] "}"
＜エージェント記述子＞  ::=
    エージェント名  |
    "m(" ＜エージェント記述子＞ ")"
＜属性＞  ::=
    変数名 ":" ＜項＞
＜メソッド＞  ::=
    ＜メッセージパターン＞ ":" ＜手続き＞
＜メッセージパターン＞  ::=
    メッセージ識別子 ＜項＞*
＜手続き＞  ::=
    ＜文＞ (";" ＜文＞)*
＜文＞  ::=
    ＜条件文＞      |
    ＜ｃａｓｅ文＞  |
    ＜代入文＞      |
    ＜式＞
＜条件文＞  ::=
    "if" ＜式＞ "then" "(" ＜手続き＞ ")"
    [ "else" "(" ＜手続き＞ ")" ]
＜ｃａｓｅ文＞  ::=
    "case" ＜式＞ "of"
    ( ＜変数名パターン＞ ":" "(" ＜手続き＞ ")" )+
    [ "otherwise" ":" "(" ＜手続き＞ ")" ]
＜代入文＞  ::=
    "let" ＜パターン式＞
＜式＞  ::=
    ＜メッセージ式＞              |
    ＜項＞                        |
    ＜パターン式＞                |
    "forsome" 変数名 "with (" ＜式＞ ")" ＜式＞  |
    "forall" 変数名 "with (" ＜式＞ ")" ＜式＞
```

```
＜メッセージ式＞  ::=
    ＜項＞ "<-" ＜メッセージパターン＞
＜項＞  ::=
    "( " ＜式＞ ")"              |
    ＜エージェント名パターン＞  |
    "execute(" 変数名 ")"       |
    "dequeue"                    |
    変数名
＜パターン式＞  ::=
    ＜パターン＞ "=" ＜パターン＞
＜パターン＞  ::=
    ＜式＞          |
    変数名パターン
＜エージェント名パターン＞  ::=
    エージェント名 "." 場の名前
```

付 録D　メッセージ送受信のモデル

D.1　同一場内のメッセージ送受信

```
/* エージェントのメタレベル記述
{ m(AGENT) |
 <method>
/* メッセージ送受信を行なうメソッド
  [] :
   if not (msg_queue <- is_empty)
   then (
    let  <Tag, Packet, Cont> =
         (msg_queue <- dequeue);
    case Tag of
/* メッセージ送信の場合
    send: (
     if Packet =
        <Receiver, Sender, <'return', Args>>
/* 返信メッセージならば
/* 継続から返信先の局所環境を回復した
/* メッセージ閉包を伝達する
     then (
      let <Env, RtnEnv:History> = Cont;
      let Result = (
           self
           <- send_closure <Tag,
                            Packet,
                            <RtnEnv, History>>)
     )
/* 新規メッセージならば
/* 自身の局所環境を履歴に退避した
/* メッセージ閉包を伝達する
     else (
      let <Env, History> = Cont;
      let Result = (
           self
           <- send_closure <Tag,
```

```
                              Packet,
                              <[], Env:History>>)
     );
     if Result = 'failure'
/* 伝達に失敗した時には
/* エラー回復処理を行なう
     then (
      self
      <- error_handler <Tag,
                        Packet,
                        Cont>)
     )
/* メッセージ受信の場合
    receive: (
/* メッセージを評価する
     let MsgClosure = (self <- execute Packet Cont);
/* 評価の結果得られるメッセージ閉包を
/* メッセージ・キューに格納する
     msg_queue <- enqueue MsgClosure
    )
  ),

/* メッセージ閉包を伝達するメソッド
  send_closure MsgClosure :
   let <Tag,
        <Receiver, Sender, MsgBody>,
        Cont> =
       MsgClosure;
   case Receiver of
/* 伝達先エージェントがエージェント名と場の名前で指定されている場合
   AgnName.FldName: (
/* 自身がその場に属していることを確認し
    if (field <- member FldName)
    then (
/* 伝達先エージェントにメッセージ閉包を伝達する
     let Result = (
         AgnName.FldName
         <- receive_closure MsgClosure);
     sender <- return Result
    )
    else ( sender <- return 'failure')
   )
```

```
   otherwise:
    (sender <- return 'failure'),

/* 伝達されたメッセージ閉包を
/* メッセージ・キューに格納するメソッド
  receive_closure <Tag, Packet, Cont> :
   if (self <- valid_message Packet Cont)
/* 受け取ったメッセージが継承の下で
/* 評価可能ならば
   then (
    let <Receiver, Sender, MsgBody> =
        Packet;
/* メッセージ閉包をメッセージ・キューに格納し
    msg_queue
    <- enqueue
        <receive,
         <Receiver, Sender, MsgBody>,
         Cont>;
/* メッセージ伝達が成功したことを通知する
    sender <- return 'success')
   else (sender <- return 'failure'),

/* エラー処理を行なうメソッド
  error_handler MsgClosure :
   ... ;; }
```

D.2 場を越えたメッセージ送受信

```
/* ファシリテータを除くエージェントのメタレベル記述
{ m(AGENT) |
 <method>
/* メッセージ閉包を伝達するメソッド
  send_closure MsgClosure :
   let <Tag,
        <Receiver, Sender, MsgBody>,
        Cont> =
       MsgClosure;
   let AgnName.FldName = Receiver;
/* 自身が属する場の中に目的エージェントが存在する
/* ならばメッセージ閉包を伝達する
   if (
```

```
    field <- member FldName)
   then (
    let Result =
     (AgnName.FldName <- receive_closure MsdClosure))
   else (
    if (
/* 自身が属する場の中のファシリテータに
/* メッセージ閉包の転送を依頼する
     forsome Fld
     with (field <- member Fld)
      forsome Facilitator
      with (Facilitator <- is_in Fld)
       (Facilitator.Fld
        <- forward_message MsgClosure field) =
        'success')
      then (Result = 'success')
      else (Result = 'failure')
    )
       ...
   otherwise: (sender <- return 'failure'),
  ...;; }

/* ファシリテータのメタレベル記述
{ m(FACILITATOR) |
 <method>
/* メッセージ閉包を転送するメソッド
  forward_message AgnName MsgClosure
             SearchedFields :
   let SetOfFields =
    field - SearchedFields;
   if SetOfFields = {}
/* 自身が属する場が全て探索済ならば
/* 失敗を通知する
   then (sender <- return 'failure')
   else (
    let <Tag,
        <Receiver, Sender, MsgBody>,
        Cont> = MsgClosure ;
    let AgnName.FldName = Receiver;
    if (
/* 場内に目的エージェントが存在するならば
/* そのエージェントにメッセージ閉包を伝達する
```

```
     if (
      field <- member FldName)
     then (
      let Result =
       (AgnName.FldName <- receive_closure MsgClosure);
      sender <- return Result)
/* それ以外の場合には
/* 探索済の場を更新し
/* 場内のファシリテータに
/* メッセージ閉包の転送を依頼する
     else (
      if (
       forsome Fld
       with (SetOfFields
             <- member Fld)
        forsome Facilitator
        with (Facilitator <- is_in Fld)
         (Facilitator.Fld
          <- forward_message AgnName
                             MsgClosure
                             SearchedFields +
                             SetOfFields) =
         'success')
      then (sender <- return 'success')
      else (sender <- return 'failure')
   )),
  ...;; }
```

付 録E　COACフレームワーク

コミュニティ・モデルを構成する各エージェントが，メッセージ送受信のために共通に持つメタレベル記述を以下に示す．ここでは，複数の場に属してメッセージを転送するエージェントをファシリテータと呼び，メッセージを転送する機能を持たせる．

E.1　エージェント

```
{ m(AGENT) |
  ...
  <method>
/* メッセージ・キューの操作
    [] :
      if not (msg_queue <- is_empty)
      then (
        let <Tag, Packet, Cont> = (msg_queue <- dequeue);
        case Tag of
        send: (
          if Packet = <Receiver, Sender, <'return', Args>>
          then (
            let <Env, RtnEnv:History> = Cont;
            let Result = (
                self <- send_closure <Tag, Packet, <RtnEnv, History>>
            )
          )
          else (
            let <Env, History> = Cont;
            let Result = (
              self <- send_closure <Tag, Packet, <[], Env:History>>
            )
          );
          case Result of
          'failure': (
            self <- transfer_error <Tag, Packet, Cont>
          )
          'unknown_message': (
            self <- message_error <Tag, Packet, Cont>
```

```
          )
        )
        receive: (
          let MsgClosure = (
            self <- execute Packet Cont
          );
          msg_queue <- enqueue MsgClosure
        )
      ),

/* 下位レベルメッセージの送信
    send_closure MsgClosure :
      let <Tag, <Receiver, Sender, MsgBody>, Cont> = MsgClosure;
      let AgnName.FldName = Receiver;
      let <Msg, Args> = MsgBody;
      if (
        Msg = 'return' or
/* ポリシチェックポイント1
/*      個々のエージェントについて，送信メッセージがポリシに
/*      したがうことをチェックする
        true
      )
      then (
        if (field <- member FldName)
        then (
/*      受信者が場内にある時、メッセージを伝達
          let Result =
            (AgnName.FldName <- receive_closure MsgClosure)
        )
        else (
/*      受信者が場内にない時、ファシリテータに転送を依頼
          if (
            forsome Fld
              with (field <- member Fld)
              forsome Facilitator
                with (Facilitator <- is_in Fld)
                (Facilitator.Fld <- forward_message MsgClosure field)
            = 'success'
          )
          then (let Result = 'success')
          else (
            if (
```

```
              forsome Fld
                with (field <- member Fld)
                forsome Facilitator
                  with (Facilitator <- is_in Fld)
                  (Facilitator.Fld <- forward_message MsgClosure field)
              = 'unknown_message'
            )
            then (let Result = 'unknown_message')
            else (let Result = 'failure')
          )
        )
      )
      else (let Result = 'policy_violation');
    sender <- return Result,

/* 下位レベルメッセージの受信
    receive_closure <Tag, Packet, Cont> :
      if (self <- valid_message Packet Cont)
      then (
        let <Receiver, Sender, MsgBody> = Packet;
        msg_queue
        <- enqueue <receive, <Receiver, Sender, MsgBody>, Cont>;
        sender <- return 'success')
      else (sender <- return 'unknown_message'),

/* メッセージが伝達できなかった時のネゴシエーション
    transfer_error MsgClosure :
/*    この例では何もしない
      ,

/* メッセージを実行できなかった時のネゴシエーション
    message_error MsgClosure :
/*    この例では何もしない
  ;;
}
```

E.2 ファシリテータ

```
{ m(FACILITATOR) |
  ...
  <method>
/* 下位レベルメッセージの転送
```

```
    forward_message MsgClosure SearchedFields :
      let SetOfFields = (field <- origin) - SearchedFields;
      if SetOfFields = {}
      then (let Result =  'failure')
      else (
        let <Tag, <Receiver, Sender, MsgBody>, Cont> =
          MsgClosure ;
        let AgnName.FldName = Receiver;
        let <Msg, Args> = MsgBody;
/*      管理する場内に受信者がある時，メッセージを伝達
        if ((field <- origin) <- member FldName)
        then (
          if (
            Msg = 'return' or
/* ポリシチェックポイント 2
/*          メッセージがポリシにしたがうことを確認
            false
          )
          then (
            let Result =
              (AgnName.FldName <- receive_closure MsgClosure)
          )
          else (
            let Result = 'policy_violation'
          )
        )
        else (
/*      管理する場内に受信者がない時，メッセージを他のファシリテータに転送
          if(
            forsome Fld
              with (field <- member Fld)
              (Facilitator.Fld <- forward_message MsgClosure
              SearchedFields + SetOfFields)
            = 'success'
          )
          then (let Result = 'success')
          else (
            if(
              forsome Fld
                with (field <- affiliate_member Fld)
                (Facilitator.Fld <- forward_message MsgClosure
                SearchedFields + SetOfFields)
```

```
              = 'unknown_message'
            )
            then (let Result = 'unknown_message')
            else (let Result = 'failure')
          )
        )
      );
      if (Result = 'success')
      then (sender <- return Result)
      else (
        if (Result = 'unknown_message')
        then (
          let Res =
            self <- alternate_message MsgClosure
        )
        else (
          let Res =
            self <- alternate_transfer MsgClosure
        );
        sender <- return Res
      ),

/* メッセージが伝達できなかった時のネゴシエーション
    alternate_transfer MsgClosure :
      /* この例では'failure' を返すだけ
      sender <- return 'failure',

/* メッセージを実行できなかった時のネゴシエーション
    alternate_message MsgClosure :
      /* この例では'unknown_message' を返すだけ
      sender <- return 'unknown_message'
  ;;
}
```

付 録F　情報サービス・コミュニティ

F.1　場Aのエージェント

F.1.1　AR

- ベースレベル記述

```
{ AR |
  ...
  method
    some_method :
      ...
      AP.A <- get_information ARGUMENTS;
      ...
  ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更し，場Aのエージェント APへの get_infromation メッセージの送信のみを許可する．

```
Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A'
```

F.1.2　AP

- ベースレベル記述

```
{ AP |
  ...
    method
      get_information ARGUMENTS :
      ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

F.1.3 AC

付録 E.2 のまま変更なし.

F.2 場 B のエージェント

F.2.1 BR

- ベースレベル記述

```
{ BR |
  ...
    method
      some_method :
        ...
        BP.B <- get_information ARGUMENTS;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B')
```

F.2.2 BP

- ベースレベル記述

```
{ BP |
  ...
    method
      get_information ARGUMENTS :
        ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

F.2.3 BC

付録 E.2 のまま変更なし.

F.3 コミュニティAとBの連合

F.3.1 AR

- ベースレベル記述

```
{ AR |
  ...
      some_method :
        ...
        AP.A <- get_information ARGUMENTS;
        ... ,
      another_method
        ...
        BP.B <- get_information ARGUMENTS2;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更する.

```
(Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B')
```

F.3.2 AP

変更なし.

F.3.3 AC

他の場からのアクセスを受け入れるため，ポリシチェックポイント2を以下のように変更.

```
(Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A')
```

F.3.4 BR

- ベースレベル記述

```
{ BR |
  ...
    method
      some_method :
        ...
        BP.B <- get_information ARGUMENTS;
        ... ,
      another_method :
        ...
        AP.A <- get_information ARGUMENTS2;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B') or
(Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A')
```

F.3.5 BP

変更なし.

F.3.6 BC

他の場からのアクセスを受け入れるため, ポリシチェックポイント 2 を以下のように変更.

```
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B')
```

F.4 コミュニティA，B，Cの連合

F.4.1 AR

- ベースレベル記述

```
{ AR |
  ...
      some_method :
        ...
        AP.A <- get_information ARGUMENTS;
        ... ,
      another_method
        ...
        BP.B <- get_information ARGUMENTS2;
        ... ,
      yet_another_method :
        ...
        CP.C <- get_information ARGUMENTS3;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更する.

```
(Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B') or
(Msg = 'get_information' and
  AgnName = 'CP' and FldName = 'C')
```

F.4.2 AP

変更なし.

F.4.3 AC

コミュニティAとBの連合の場合から変更なし.

F.4.4 BR

- ベースレベル記述

```
{ BR |
  ...
    method
```

```
        some_method :
          ...
          BP.B <- get_information ARGUMENTS;
          ... ,
        another_method :
          ...
          AP.A <- get_information ARGUMENTS2;
          ... ,
        yet_another_method :
          ...
          CP.C <- get_information ARGUMENTS3;
          ...
      ;;
  }
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B') or
(Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'get_information' and
  AgnName = 'CP' and FldName = 'C')
```

F.4.5 BP

変更なし.

F.4.6 BC

コミュニティ A と B の連合の場合から変更なし.

F.4.7 CR

- ベースレベル記述

```
{ CR |
  ...
    method
      some_method :
        ...
```

```
        CP.C <- get_information ARGUMENTS;
        ... ,
      another_method :
        ...
        AP.A <- get_information ARGUMENTS2;
        ... ,
      yet_another_method :
        ...
        BP.B <- get_information ARGUMENTS3;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更する.

```
(Msg = 'get_information' and
  AgnName = 'CP' and FldName = 'C') or
(Msg = 'get_information' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'get_information' and
  AgnName = 'BP' and FldName = 'B')
```

F.4.8 CP

- ベースレベル記述

```
{ CP |
  ...
    method
      get_information ARGUMENTS :
      ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

F.4.9 CC

他の場からのアクセスを受け入れるため，ポリシチェックポイント2を以下のように変更.

```
(Msg = 'get_information' and
  AgnName = 'CP' and FldName = 'C')
```

付 録G　仲介サービス・コミュニティ

G.1　場Aのエージェント

G.1.1　AR

- ベースレベル記述

```
{ AR |
  ...
  method
    some_method :
      ...
      AB.A <- broker ARGUMENTS;
      ...
  ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更し，場Aのエージェント ABへの broker メッセージの送信のみを許可する．

```
Msg = 'broker' and
  AgnName = 'AB' and FldName = 'A'
```

G.1.2　AB

- ベースレベル記述

```
{ AB |
  ...
    method
      broker ARGUMENTS :
        ...
        let Res = AM.A <- h_recommend ARGUMENTS1;
        ...
        AP.A <- h_service_request Res ARGUMENTS2;
        ...
```

```
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'h_recommend' and
  AgnName = 'AR' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A')
```

G.1.3 AM

- ベースレベル記述

```
{ AM |
  ...
    method
      h_recommend ARGUMENTS :
        ...
        sender <- return RESULT;
        ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

G.1.4 AP

- ベースレベル記述

```
{ AP |
  ...
    method
      h_service_request ARGUMENTS :
      ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

G.1.5 AC

付録E.2のまま変更なし.

G.2 場Bのエージェント

G.2.1 BR

- ベースレベル記述

```
{ BR |
  ...
    method
      some_method :
        ...
        let Res = BM.B <- f_recommend ARGUMENTS1;
        ...
        BP.B <- f_service_request Res ARGUMENTS2;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更する.

```
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B')
```

G.2.2 BM

- ベースレベル記述

```
{ BM |
  ...
    method
      f_recommend ARGUMENTS :
        ...
        sender <- return RESULT;
        ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

G.2.3 BP

- ベースレベル記述

```
{ BP |
  ...
    method
      f_service_request ARGUMENTS :
        ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

G.2.4 BC

付録 E.2 のまま変更なし.

G.3 コミュニティ A と B の連合

G.3.1 AR

変更なし.

G.3.2 AB

- ベースレベル記述

```
{ AM |
  ...
    method
      broker ARGUMENTS :
          ...
          if ...
          then (
            ...
            let Res = AM.A <- h_recommend ARGUMENTS1;
            ...
```

```
            AP.A <- h_service_request ARGUMENTS2;
            ...
          )
          else (
            ...
            let Res = BM.B <- f_recommend ARGUMENTS3;
            ...
            BP.B <- f_service_request Res ARGUMENTS4;
            ...
          )
          ...
      ;;
  }
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更する.

```
(Msg = 'h_recommend' and
  AgnName = 'AM' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B')
```

G.3.3 AM

変更なし.

G.3.4 AP

変更なし.

G.3.5 AC

他の場からのアクセスを受け入れるため，ポリシチェックポイント2を以下のように変更.

```
(Msg = 'h_recommend' and
  AgnName = 'AM' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A')
```

G.3.6 BR

- ベースレベル記述

```
{ BR |
  ...
    method
      some_method :
        ...
        let Res = BM.B <- f_recommend ARGUMENTS1;
        ...
        BP.B <- f_service_request Res ARGUMENTS2;
        ... ,
      another_method :
        ...
        let Res = AM.A <- h_recommend ARGUMENTS3;
        ...
        AP.A <- h_service_request Res ARGUMENTS4;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B') or
(Msg = 'h_recommend' and
  AgnName = 'AM' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A')
```

G.3.7 BM

変更なし.

G.3.8 BP

変更なし.

G.3.9 BC

他の場からのアクセスを受け入れるため，ポリシチェックポイント2を以下のように変更.

```
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B')
```

G.4 コミュニティA，B，Cの連合

G.4.1 AR

変更なし.

G.4.2 AB

- ベースレベル記述

```
{ AM |
  ...
    method
      broker ARGUMENTS :
          ...
          if ...
          then (
            ...
            let Res = AM.A <- h_recommend ARGUMENTS1;
            ...
            AP.A <- h_service_request ARGUMENTS2;
            ...
          )
          else (
            if ...
            then (
              ...
              let Res = BM.B <- f_recommend ARGUMENTS3;
              ...
              BP.B <- f_service_request Res ARGUMENTS4;
              ...
            )
            else (
```

```
                ...
                let Res = CM.C <- l_recommend ARGUMENTS5;
                ...
                CP.P <- l_service_request Res ARGUMENTS6;
                ...
              )
            )
            ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'h_recommend' and
  AgnName = 'AM' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B') or
(Msg = 'l_recommend' and
  AgnName = 'CM' and FldName = 'C') or
(Msg = 'l_service_request' and
  AgnName = 'CP' and FldName = 'C')
```

G.4.3 AM

変更なし.

G.4.4 AP

変更なし.

G.4.5 AC

コミュニティ A と B の連合の場合から変更なし.

G.4.6 BR

- ベースレベル記述

```
{ BR |
  ...
    method
      some_method :
        ...
        let Res = BM.B <- f_recommend ARGUMENTS1;
        ...
        BP.B <- f_service_request Res ARGUMENTS2;
        ... ,
      another_method :
        ...
        let Res = AM.A <- h_recommend ARGUMENTS3;
        ...
        AP.A <- h_service_request Res ARGUMENTS4;
        ... ,
      yet_another_method :
        ...
        let Res = CM.C <- l_recommend ARGUMENTS5;
        ...
        CP.C <- l_service_request Res ARGUMENTS6;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント1を以下のように変更する.

```
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B') or
(Msg = 'h_recommend' and
  AgnName = 'AM' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'l_recommend' and
  AgnName = 'CM' and FldName = 'C') or
(Msg = 'l_service_request' and
  AgnName = 'CP' and FldName = 'C')
```

G.4.7 BM

変更なし.

G.4.8 BP

変更なし.

G.4.9 BC

コミュニティ A と B の連合の場合から変更なし.

G.4.10 CR

- ベースレベル記述

```
{ CR |
  ...
    method
      some_method :
        ...
        let Res = CM.C <- l_recommend ARGUMENTS1;
        ...
        CP.C <- l_service_request Res ARGUMENTS2;
        ... ,
      another_method :
        ...
        let Res = AM.A <- h_recommend ARGUMENTS3;
        ...
        AP.A <- h_service_request Res ARGUMENTS4;
        ... ,
      yet_another_method :
        ...
        let Res = BM.B <- f_recommend ARGUMENTS5;
        ...
        BP.B <- f_service_request Res ARGUMENTS6;
        ...
    ;;
}
```

- メタレベル記述
  ポリシチェックポイント 1 を以下のように変更する.

```
(Msg = 'l_recommend' and
  AgnName = 'CM' and FldName = 'C') or
(Msg = 'l_service_request' and
  AgnName = 'CP' and FldName = 'C') or
(Msg = 'h_recommend' and
  AgnName = 'AM' and FldName = 'A') or
(Msg = 'h_service_request' and
  AgnName = 'AP' and FldName = 'A') or
(Msg = 'f_recommend' and
  AgnName = 'BM' and FldName = 'B') or
(Msg = 'f_service_request' and
  AgnName = 'BP' and FldName = 'B')
```

G.4.11 CM

変更なし.

G.4.12 CP

- ベースレベル記述

```
{ CP |
  ...
    method
      l_service_request ARGUMENTS :
        ...
    ;;
}
```

- メタレベル記述
  共通の記述のままで変更なし

G.4.13 CC

他の場からのアクセスを受け入れるため，ポリシチェックポイント2を以下のように変更.

```
(Msg = 'l_recommend' and
  AgnName = 'CM' and FldName = 'C') or
(Msg = 'l_service_request' and
  AgnName = 'CP' and FldName = 'C')
```

付 録H　階層関係をともなうコミュニティのアクセス制御ポリシ・モデル

多くのアクセス制御ポリシ・モデルは，組織内の権限の序列を表現する階層関係の概念を持つ．階層関係をどのように定義するかについては様々な議論がある [15] が，多くのポリシ・モデルでは，上位の階層が下位の階層が持つ許可を継承することによって階層関係が定義される．ここでは，コミュニティ内のパートの間に階層関係がある場合のポリシ・モデルについて述べる．

H.1　パート間の階層関係

コミュニティのパート間の階層関係を $\succeq$ とし，反射的推移的であることを仮定する．すなわち，以下が成り立つとする．

- $\forall X \in R_c \mid X \succeq X$
- $\forall X, Y \in R_c \mid X \succeq Y \wedge Y \succeq X \Longrightarrow X = Y$
- $\forall X, Y, Z \in R_c \mid X \succeq Y \wedge Y \succeq Z \Longrightarrow X \succeq Z$

H.2　階層関係をともなうコミュニティのアクセス制御ポリシ

パート間に階層関係があるとき，以下の関係が成り立つならば，コミュニティのアクセス制御ポリシは階層関係を満たすと言う．

$$\forall X, Y \in R_c;\ \forall prm \in PRM \mid X \succeq Y \wedge (Y, prm) \in p_c \Longrightarrow (X, prm) \in p_c$$

H.3　階層関係を損なわない連合のポリシ

連合のポリシ $P^{\dagger}$ について次が成り立つとき，$P^{\dagger}$ は階層関係を損なわない連合のポリシであると言う．

$$\forall X, Y \in R_i;\ \forall X', Y' \in R_j \mid$$
$$X \succeq Y \wedge (X, X') \in P^{\dagger} \wedge (Y, Y') \in P^{\dagger} \Longrightarrow X' \succeq Y'$$