ソフトウェア工学 140−13
（2003．3．7）

サービス指向アーキテクチャにおける相互認可モデル

坂田 祐司, 山岡 正輝, 松田 栄之
（株）NTTデータ 技術開発本部
e-mail: {sakatayu, yamaokam, matsudasg}@nttdata.co.jp

あらまし 現在、組織、企業間の柔軟な連携を迅速に実現するためのアーキテクチャとして、サービス指向アーキテクチャが注目されている．このアーキテクチャでは，サービス利用者の要求に応じてサービス提供者を自動的に発見する仕組み，および，サービス提供者と動的に結合することによってサービスを受けるという仕組みが規定されている．しかし，サービス提供者とサービス利用者が，サービスが正しく実行されることを確信するために必要な相互認可を自動的に行う仕組みは検討が始まった段階である．本稿では，サービス指向アーキテクチャにおける相互認可の実現における課題と要件を整理するとともに，相互認可のモデルを示す．

A mutual authorization model in service-oriented architecture

Yuji Sakata, Masaki Yamaoka, Shigeyuki Matsuda
NTT Data Corporation
e-mail: {sakatayu, yamaokam, matsudasg}@nttdata.co.jp

Abstract Nowadays, service-oriented architecture is highly regarded as an innovative one that easily enables organizations and enterprises to collaborate with others dynamically. The basic reference model of this architecture specifies how a service requestor automatically discovers a service provider who has the service that the requestor needs, and how the requestor dynamically acts with the discovered provider to perform the service. However, not much research has been done on the method of automatic mutual authorization that allows a provider and a requestor to trust each other on the proper execution of a service. In this article, we address the issues of establishing mutual authorization automatically in service-oriented architecture, and present a mutual authorization model.

1. サービス指向アーキテクチャ

1.1. サービス指向アーキテクチャの必要性

近年，企業活動を効率化し，企業の収益を高めることを目的としたe-ビジネスが急速に普及してきた．

e-ビジネスは，初期には，企業内や顧客とのチャネルに対して適用され，次に，固定的な取引関係のある企業間の商取引に適用されるようになった．現在では，さらに動的な企業間連携に対する適用が検討されている[1]．ここで，動的な企業間連携とは，各企業が，商活動の状況に応じて最適な取引相手を発見し，取引関係を締結し，商取引を実行するという形態のビジネスを指す．

このような動的な企業間連携を実現するためには，企業システムには，内部のデータベースやトランザクションシステムなどのアプリケーションをインターネット上に公開し，インターネットを介して他の企業システムと自動的に連携する機能が求められる．特に，システムを自動的に連携するためには，アプリケーションの処理の詳細を隠蔽し，業務単位や利用者にとって意味のある粒度の機能単位でアプリケーションの機能を提供する必要がある．このようなアプリケーションの機能の提供形態をサービスと呼ぶ．

現在，インターネットを介した企業システムの自動的な連携を容易に実現するためのシステムアーキテクチャとして，サービス指向アーキテクチャが注目されている．

1.2. サービス指向アーキテクチャの概要

サービス指向とは，複数のサービスを必要に応じて自動的に組み合わせ，動的な企業間連携を実現するアプリケーションを迅速に構築しようという考えである．また，サービス指向を実現するアーキテクチャがサービス指向アーキテクチャである．

サービス指向アーキテクチャの基本モデルでは，サービスの実行に関与するエンティティの役割を，図 1で示すような3つの役割にモデル化している．[2]

1. サービスリクエスタ: サービス要求を行うエンティティ(以下，リクエスタ)
2. サービスプロバイダ: サービスを提供するエンティティ(以下，プロバイダ)
3. サービスブローカ: サービス記述(サービス実現内容，アクセス方式，サービスを提供するエンティティ，サービス接続点などに関する記述)を公開し，その検索機能を提供するエンティティ（以下，ブローカ）

プロバイダは，サービスの設計時に，提供するサービスのサービス記述を定義してブローカに登録する．一方，リクエスタは，サービスを利用するアプリケーションの設計時に，ブローカから，そのサービスがどのような処理を実現するのかを示したサービス実現内容記述とアクセス方式記述を獲得し，それらの記述を利用してサービスを利用するアプリケーションを実装する．

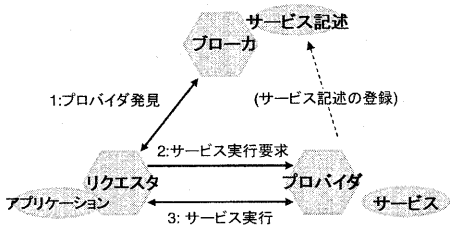

図 1 サービスの役割と実行時の相互作用

実装されたリクエスタのアプリケーションは，その実行時に下記のプロセスで動的にプロバイダを発見し，サービスの実行を要求する．

1. プロバイダ発見: リクエスタは，ブローカに対して利用したいサービス実現内容の記述を示し，そのサービスの提供が可能なプロバイダのサービス接続点の記述の一覧を受け取る．
2. サービス実行要求: リクエスタは，一覧からサービス提供を受けるプロバイダを決定する．そして，そのプロバイダのサービス接続点の記述を利用し，プロバイダに対してサービス実行要求を行う．
3. サービス実行: プロバイダは，要求されたサービスを実行する．

以上のエンティティの役割モデルと設計時および実行時のプロセスモデルから，サービス指向アーキテクチャは，次の 2 点の特徴を有している．

・疎結合性の実現: リクエスタは，アプリケーションの設計時に，サービスの実現内容とアクセス方式の記述のみを獲得していればよい．設計時には，これらの記述のみを獲得していれば，実行時に初めて特定されるプロバイダのサービスも，実行可能である．よって，実行時に連携相手を選択するための条件が決定される場合においても，連携が可能である．

・プロバイダの自動的な発見:リクエスタは，実行時に決定した条件を満たすサービスを提供しているプロバイダを，ブローカの検索機能により，自動的に発見することができる．

以上の特徴をもつサービス指向アーキテクチャを導入することにより，動的な企業間連携が実現できる．さらに，ビジネス・ユーザの立場から見ると，詳細で多様なソフトウェアの実装を意識する必要がなくなる．また，ソフトウェアの立場から見ると，ビジネス・ユーザの多様な振る舞いを意識しなくてよいため，ソフトウェアの再利用性や相互運用性を高めることが期待できる．青山らは，この点を図 2で示す 3 階層参照モデルによって説明している[3].

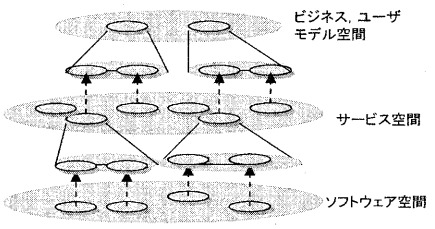

図 2 ビジネス・サービス・ソフトウェア

1999 年に Sun Microsystems から発表された Jini[4]や次節で示す Web サービスは，サービス指向アーキテクチャを具体化した代表的な技術である．

1.3. Web サービス

Web サービスは，サービスを実現するための基盤技術の一つの体系である．現在 W3C(The World Wide Web Consortium)によって，Web サービス技術を用いたサービス指向アーキテクチャの標準化が進められている[5].

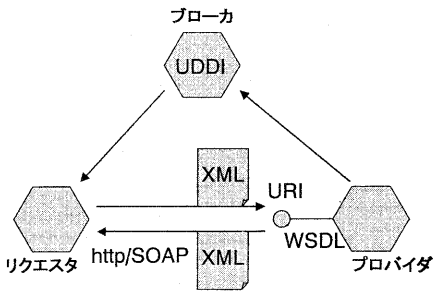

図 3 Web サービスのアーキテクチャ

図 3は，図 1で示されるアーキテクチャに対して Web サービスにおける実現技術を対応させたものである．サービスは URI[6]で一意に識別される．また，XML で定義された WSDL[7]によって，サービスのアクセス方式と接続点が記述される．リクエスタは http などのインターネット標準プロトコル上で SOAP[8]によってカプセル化された XML 文書をメッセージとして送信し，その返信を受信することによってサービスを利用する．また，ブローカの保持すべき情報のスキーマとブローカに対するアクセスプロトコルを XML によって規定している技術が UDDI[9]である．

上記で示すように，Web サービスでは，XML や http などの既存のインターネット技術を用い，サービス指向アーキテクチャを実現している．そのため，インターネット上で利用する場合の親和性が高く，プラットフォームや言語によらない相互運用性を実現している．よって，インターネットを介する企業間取引に用いられるサービス指向アーキテクチャとして，今後の利用が拡大するものと考える．

2. サービス指向アーキテクチャにおける相互認可

2.1. 自動的な相互認可の必要性

リクエスタとプロバイダ間で，間接的であれ，何らかの資金の流れを伴うサービスが実行される場合を考える．この場合，リクエスタは，プロバイダが正しくサービスを実行するか判断し，また，プロバイダは，リクエスタにサービスを提供してよいか判断することが通常である．

一方，既存にサービスの実行関係がない場合，1.2で示すように，リクエスタは，プロバイダを発見する際に初めてプロバイダに関する記述を獲得できる．また，プロバイダは，リクエスタのサービス実行要求の際に初めてリクエスタに関する情報を獲得できる．

よって，リクエスタとプロバイダは，実行時に知りえた相手エンティティに対して，自動的にサービス実行の合意を行うことができなければならない．

本稿では，このようにリクエスタとプロバイダの両者が，相手エンティティとのサービス実行の合意を確認するプロセスを相互認可と呼び，その方式について検討する．

2.2. 相互認可の方式

図 4は，一般的なアクセスコントロールリスト(ACL)[10] を用いた認可方式を示している．客体がリソースを保持するエンティティ，主体がリソースを要求するエンティティである．この方式は，クライアント/サーバ型のシステムやブラウザでアクセスする WWW サイトなどで広範に用いられている方式である．この方式では，最初に，客体は主体を認証(Authentication) [10]し，主体を一意に識別するアカウントを特定する．次に，定義済みの ACL と，主体のアカウントに関連する情報から,リソース要求の可否を判断する．

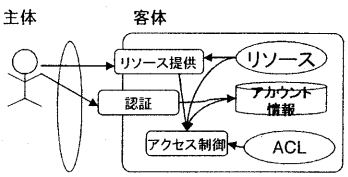

図 4 ACL ベースの認可モデル

相互認可では，図 5で示すように，両エンティティが，認可アーキテクチャにおける主体と客体の両方の役割となることが可能であればよいと仮定する．

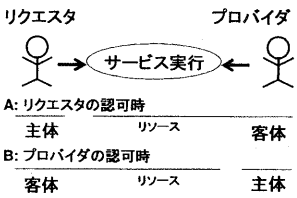

図 5 既存の認可方式の相互認可への適用

例えば，ACL 方式で相互認可を行う場合，プロバイダがリクエスタを認証し，リソースであるサービス実行の可否をプロバイダの保持する ACL に基づき認可を行う．また，リクエスタがプロバイダを認証し，リクエスタの保持する ACL に基づき認可を行う．両者の認可が成功すれば，サービスは実行されることになる．

このように仮定し，既存の認可アーキテクチャをサービス指向アーキテクチャの相互認可に適応しようとすると次章で示す課題が存在する．

3. 相互認可の課題

サービス指向アーキテクチャにおける相互認可の課題を，1.2 で述べたサービス指向アーキテクチャの持つ特徴と，サービスのもつ機能の抽象性とから検討する．

3.1. リクエスタとプロバイダが互いに未知である点

サービス指向アーキテクチャにおいては，リクエスタがプロバイダを動的に発見し，サービスを実行することが可能である．よって，リクエスタとプロバイダは，既存に取引関係が存在しない場合がある．この点に起因する課題を検討する．

3.1.1. 直接的に信頼できない相手エンティティ

エンティティ同士が未知であるため，相手エンティティが示す情報を直接的には信頼できない．よって，ACL ベースの認可モデルを考える場合，客体となるエンティティが主体となるエンティティを直接認証することは出来ない．

3.1.2. エンティティ間の情報の記述形式の不一致

エンティティ間で交換される情報は，相手エンティティと同じ方式で記述されているとは限らない．

3.1.3. 相手の認可規則が未知である点

各エンティティは，サービス実行に対して相手エンティティが，どのような認可規則を持っているかに関する情報を保持していない場合がある．よって，その認可規則を満たすために必要な情報は何かという情報を自身で獲得しなければならない．

3.1.4. 無条件に相手エンティティに情報を渡せない点

相互認可の際に，相手エンティティに要求されて渡す情報が，その相手エンティティに本当に渡して良い情報かを判断する必要がある．

例えば，非常に価値の高いサービスの場合，そのサービス実行の認可規則を，プロバイダがリクエスタの要求に対して，無条件に示すことは，サービスを不正に利用しようとするリクエスタに対して，どのような情報を偽

装すればよいかを示していることになり，セキュリティ上の脆弱性となる．また，自エンティティに関する情報を相手エンティティが認可判断のために必要な情報として要求する場合も，その情報を未知のエンティティに渡してよいか判断する必要がある．

3.2. 非集中化アーキテクチャ

サービス指向アーキテクチャにおいては，リクエスタとプロバイダは疎結合で連携できるため，何らかの処理を行う際に，必要に応じて複数のエンティティが自動的に連携するという場合がある．この点に起因する課題を検討する．

3.2.1. 合成されたサービスに対する相互認可

サービスは，動的に，エンティティ内部もしくはエンティティ外部のサービスと合成され，新しいサービスとして提供される場合がある．例えば，Web サービスにおいて原子的なサービスの単位は，一つの WSDL 記述で表現されるが，WSCI[12]や BPEL4WS[13]などの技術を用いて，合成される．このように，定められた規則に従い合成され，提供されるサービスの認可規則は，原子的なサービスの持つ認可規則と合成規則から動的に生成されることが望まれる．

また，合成されたサービスにおいて，どのエンティティ間において相互認可が行われるか，二種類のモデルを考えることが出来る．図 6はリクエスタがプロバイダ A のサービスを利用する際に，プロバイダ A がプロバイダ B のサービスを合成して，サービスを実行するケースである．

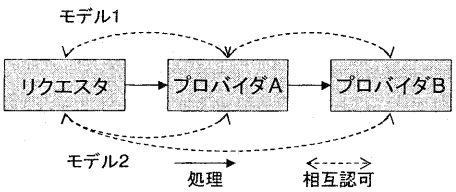

図 6 二者間以上の処理の場合の相互認可モデル

モデル 1 は，リクエスタとプロバイダ A 間と，プロバイダ A とプロバイダ B 間で相互認可がなされる場合であり，モデル 2 はリクエスタとプロバイダ A,B 間で相互認可がなされる場合である．どちらのモデルが適切であるかはサービスに依存するため，この両モデルの相互認可の実現が必要である．

3.2.2. 情報や認可規則の存在が非集中化している点

ACL ベースの認可モデルにおいては，ACL は認可するエンティティ自身が保持している．しかし，3.1.3で示したように，認可されるエンティティは，認可されるためにどのような情報が必要か相互認可の実行時まで知りえない．よって，エンティティは，認可のために相手エンティティに渡すことの出来る情報を，自エンティティ内部ですべて静的に保持するより，その情報を，実行時に外部エンティティから獲得するほうが効率的である．

また，認可の条件に関する規則も，認可するエンティティのみが保持している必要は無い．例えば，効率を考慮し，プロバイダの認可規則も静的な規則はブローカが保持し，動的に変化する規則はプロバイダが保持するという形態も想定すべきである．

3.3. サービスの提供する機能が粗粒度である点

図 2で示すように，サービスは，ビジネス・ユーザの要求をサービスの組み合わせとして実現できる程度に高い抽象化が行われた粗粒度の機能である．この点に起因する課題をする．

3.3.1. 認可に関するルールや情報の形式化

認可規則は，サービス実行に関する合意ができるかどうかの判断基準を規定するものである．サービスが企業の業務に相応するような処理を行うことを考慮すると，サービス実行の合意の判断基準を表す情報の構造を規定し，形式化することは，困難であると考えられている[14].

3.3.2. サービスの機能粒度に適した情報による解釈

既存の認可モデルにおける記述形式は，高い抽象的機能を提供するサービスに対する認可を想定していない．

例えば，OS のファイルシステムやブラウザベースの WWW サーバのコンテンツなど，ソフトウェア空間に閉じたリソースに対する認可を考慮すると，利用者の所属グループやファイルが所属しているディクレトリ，ファイル名など OS が管理している情報が認可規則に利用される情報となる．よって，それらの情報を用いた認可規則が記述できればよい．

一方，業務に相当するサービスの場合は，「提示する格付け機関の値が AA＋を超えることを示しなさい」,「提示する企業リストの中から二つ以上の企業とのあなたの現在までの取引実績を示しなさい」「あなたの配送処理における契約違反率を示しなさい」というような業務に近い条件が認可規則となると考えられる．しかし，プログラム言語や OS に依存する情報や記述形式を用いて，サービスの認可規則を表現することは，サービスによるソフトウェアの隠蔽という観点から好ましくない．

3.3.3. 複数の業務アプリケーションなどとの統合

サービスが企業の業務に密接した形態であるため，サービス実行の認可規則の判断には，企業が既存に保持する業務情報を利用すると考えられる．よって，既存の業務情報を持つデータベースやアプリケーションと連携し，それらの保持する情報を自動的に獲得できることが望まれる．

4. 相互認可の要件

3で示した課題から相互認可における要件を検討する．

4.1. 第三者による情報保証

相互認可を行う際，リクエスタおよびプロバイダは，そのサービス実行に対する自身の認可規則から相手エンティティへの要求を算出し，その要求を相手エンティティに提示する．そして，その要求を提示されたエンティティは，その回答となる情報を返却する．しかし，エンティティは，相手エンティティが示す回答を，信頼して

よいかの判断ができない．

よって，相手が示した回答が，信頼できる回答であることを保証する第三者が必要となる．

関連課題: 3.1.1

4.2. 情報の完全性の保証

相互認可で用いられる情報は，インターネットを介し各エンティティ間で交換される．また，その情報は，その情報の発行エンティティではないエンティティにおいて，保存される可能性がある． そのため，情報の秘匿性と完全性の保証が必要である．

関連課題: 3.2.2

4.3. エンティティへの判断規則処理モジュールの導入

各エンティティは，以下の二つの処理を行うことが必要である．

一つ目は，相手エンティティからの要求に対して，その要求が認可されるか，認可されないか，もしくはある条件下で認可が可能かを判断する処理である．相手エンティティからの要求は，サービス実行の要求の場合と自エンティティに関する情報の取得要求の場合がある．

二つ目は，相手エンティティに要求される自エンティティに関する情報を外部エンティティから獲得する処理である．

これらの処理を実現するモジュールは，業務における認可規則を処理できる拡張性の高いモジュールでなければならない．

関連課題: 3.1.3，3.1.4，3.3.2

4.4. 複数認可規則の合成の自動化の実現

ある記述済みの合成規則に基づき合成され，一つのサービスとして提供されているサービスの認可規則は，原子的なサービスの持つ認可規則と合成規則から動的に生成することが可能であることが望まれる．

関連課題: 3.2.1

4.5. 分散環境下での情報の取得の自動化の実現

各エンティティは，相手エンティティから認可のために必要な自エンティティに関する情報を要求される． そして，その情報を適切な外部エンティティから取得しなければならない．よって，各エンティティは，相手エンティティの要求から，要求に対応する自エンティティに関する情報を，どの外部エンティティから獲得できるのかを把握できる仕組みが必要である．

関連課題: 3.2.2

4.6. 情報，認可規則の記述形式の共通化

各エンティティは，他エンティティから渡される情報や自エンティティが持つ様々なリソースからの情報を理解できるようにしなければならない．また，4.4で示した認可規則の合成の実現も考慮すると，情報および認可規則は共通化された記述形式で記述されることが望まれる．

さらに，記述形式は，採用しているサービス指向アーキテクチャにおいて用いられる，エンティティ間のデータの送信手法やデータの記述形式に適合していることが望まれる．

関連課題: 3.1.2，3.3.3，3.2.1

4.7. 認可規則の決定と妥当性評価の仕組みの実現

あるサービスを認可するために，実際に相手にどのような条件を提示し，確認すべきかいう点を決定し，その妥当性を検証する仕組みを導入する必要がある．

関連課題: 3.3.1

5. 相互認可の概念モデル

本章では，4で示した要件を考慮し，サービス指向アーキテクチャに適した相互認可の概念モデルを示す．

5.1. 概念モデルの概要

リクエスタとプロバイダは，サービスの実行に関して，自エンティティのサービス認可規則から相手エンティティが満たすべき条件を算出する．そして，その条件に合致しているかを判断する際に必要となる情報を相手エンティティに要求する．相手エンティティはその要求に回答する．そして，回答を受けたエンティティは，その回答からサービス実行を合意すべきか判断する．これらの処理を互いに行うことにより，相互認可を実現する．

以下で，その詳細を示す．

5.2. エンティティモデル

第三者による情報保証を実現するために，図 7で示すように，権威のある第三者をエンティティの役割として導入する．

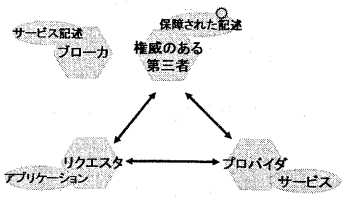

図 7 信頼できる第三者を加えたモデル

権威のある第三者は次節で示す*記述*の確からしさを保証する役割を持つ．また，公開鍵基盤によって，*記述*の保証と，エンティティ間を接続するネットワークにおける秘匿性を実現する．

5.3. データモデル

図 8に本モデルで使用するデータの抽象的な分類を示す．

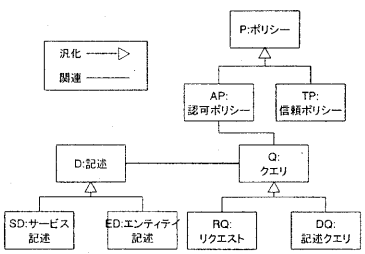

図 8 データモデル

*記述*はサービスおよびエンティティの性質を表現するデータである．サービスに関する記述は*サービス記述*であり，エンティティに対する記述は*エンティティ記述*と呼ぶ．

*クエリ*とは，他のエンティティに対してなされる要求である．サービスの実行のためのクエリを*リクエスト*，*記述*の取得要求のためのクエリを*記述クエリ*と呼ぶ．

*ポリシー*とは，各エンティティの管理者が，エンティティの振る舞いに関して定める規則である．*認可ポリシー*とは，他エンティティの*クエリ*に対する認可規則を定めたものである．また，*信頼ポリシー*とは，エンティティがどの権威のある第三者を信頼するかを定める規則である．

また，*記述*と*ポリシー*は，分散環境において，どのエンティティに存在するかを特定できるものとする．

さらに，図 8で示すように，各データタイプは関連しており，処理モジュールにおける処理効率を考慮し，同じ記述形式でデータを表現するものとする．

5.4. 処理モジュール

各リクエスタとプロバイダに，図 9で示す処理モジュールを導入する．

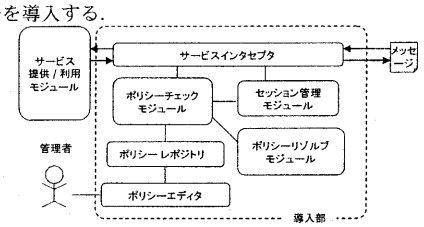

図 9 相互認可処理モジュール概要

図 9で示すように，処理モジュールは，プロバイダのサービスモジュールやリクエスタのサービスを利用するアプリケーションモジュールの前に配置される．

サービスインタセプタは，サービス提供/利用モジュールに入出力されるメッセージを取得し，相互認可の処理が必要な場合のメッセージの処理を行う．

セッション管理モジュールは，リクエスタの*リクエスト*からプロバイダの*リクエスト*の実行までの一連のメッセージ交換を一つのセッションとして認識し，管理する機能を持つ．

ポリシーレポジトリは永続性のある*ポリシー*が保持されており，管理者はポリシーエディタによって内容を管理する．ポリシーチェックモジュールとポリシーリゾルブモジュールの詳細を以下に示す．

5.4.1. ポリシーチェックモジュール

ポリシーチェックモジュールは，図 10で示すように，*クエリ*と他エンティティの*エンティティ記述*を入力として，自エンティティの*認可ポリシー*と*信頼ポリシー*から，その*クエリ*の実行の可否および，特定の条件下で実行が可能である場合は，その条件を相手エンティティに対する*記述クエリ*として返却する機能を持つモジュールである．

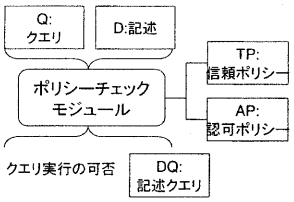

図 10 ポリシーチェックモジュール

5.4.2. ポリシーリゾルブモジュール

ポリシーリゾルブモジュールは，図 11で示すように，相手エンティティの*記述クエリ*を入力として,その*記述クエリ*に対応する*エンティティ記述*を権威のある第三者もしくはローカルのデータベースから取得し，返却する機能を持つモジュールである．

返却すべき*エンティティ記述*を，どの権威のある第三者が生成もしくは保持しているかについて，このモジュールは管理していなければならない．

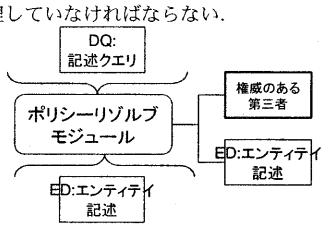

図 11 ポリシーリゾルブモジュール

5.5. 処理例

ポリシーチェックモジュールとポリシーリゾルブモジュールの処理シーケンスを例示する．この例では，下記のようにポリシーを規定しているとする．

1. プロバイダのサービス α に関して，プロバイダはリクエスタが γ と言う条件を満足する場合に，サービス実行に合意する．
2. リクエスタはサービス α に関しては，その α を提供するプロバイダが β という条件を満足する場合にサービス実行に合意する．
3. リクエスタは，γ という自エンティティに関する記述を提供するためには，プロバイダは δ という条件を満たしていなければ成らない．

これらのポリシーをもとに，リクエスタがサービス α をプロバイダに要求する際の相互認可の処理シーケンスを図 12に示す．

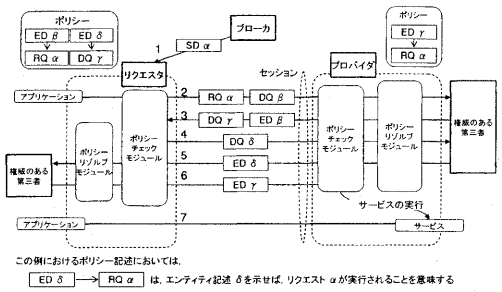

図 12 相互認可のメッセージシーケンス例
(記号の意味は図 8に示す)

1. リクエスタのアプリケーションは，ブローカからプロバイダのサービス記述 α を取得する．
2. リクエスタのアプリケーションがリクエスト α を実行する．リクエスタのポリシーチェックモジュールはポリシーに基づき，プロバイダに記述クエリ β を*リクエスト* α とともに送る．
3. プロバイダのポリシーリゾルブモジュールはエンティティ記述 β を生成することのできる権威のある第三者からエンティティ記述 β を獲得する．また，プロバイダのポリシーチェックモジュールは，リクエスト α に対する条件として記述クエリ γ を算出する．そして，記述クエリ γ とエンティティ記述 β を返却する
4. リクエスタのポリシーチェックモジュールは，エンティティ記述 β が信頼できる権威のある第三者によって保証されていることを確認する．この時点で，リクエスタはサービス実行に合意している．そして，記述クエリ γ に対する条件として，ポリシーに基づき記述クエリ δ を要求する
5. プロバイダのポリシーリゾルブモジュールは権威のある第三者からエンティティ記述 δ を獲得し，リクエスタに返却する
6. リクエスタのポリシーチェックモジュールはエンティティ記述 δ を確認する．そして，ポリシーに基づき，エンティティ記述 β をプロバイダに渡すことを認可する．そして，エンティティ記述 γ をプロバイダに返す
7. プロバイダのポリシーチェックモジュールは6におけるリクエスタのポリシーチェックモジュールと同様の処理によって，リクエスト α の実行を認可する．この時点でサービス α は相互認可されていることになる．よって，プロバイダのサービスはリクエスタに対してサービス α を実行する．

このような処理シーケンスによって，相互認可が実現し，サービスが実行される．

6. 関連研究

5で示したサービス指向アーキテクチャでの相互認可モデルの関連研究を，2つの観点から示す．

まず，サービス指向アーキテクチャの代表的な実装技術である Web サービスにおけるセキュリティの関連研究を示す．

WS-Security とその拡張仕様は Web サービスアーキテクチャにおいて，認証，認可，信頼構築などのセキュリティ機能に関する既存の技術の統合抽象概念とその記述形式を規定している[16]. Web サービスにおける相互認可を考慮すると，本稿で示す提案の具体化において，この仕様を活用する必要がある．

また，OASIS[17]においても関連する技術の標準化が行われている．XACML[18]は，本モデルにおける認可ポリシーに相当する. XACML の想定するモデルでは, XACML における認可ポリシーのプロセッサ(Policy Decision Point)は，5.4.1で示すポリシーチェックモジュールと同様に*クエリとエンティティ記述*から*クエリ*の実行可否を判断する．しかし，入力される*エンティティ記述*が，信頼できるものかを判断する機能，及び*クエリ*実行が条件付で可能である場合に，その条件を*記述クエリ*として示す機能については考慮されていない．SAML[19]は，本モデルにおける*記述クエリ*と*エンティティ記述*に相当する仕様であり，XML をベースとしている．

さらに，[20]は，Web サービスにおけるプロバイダとサービスをモデル化し，認可処理自身を Web サービスとして設計することにより，導入が簡単で，かつモジュール性の高い認可処理モジュールの実現が可能であるということを示す研究を行っている．

次に，認可モデルについての関連研究を示す．

“信頼管理”と呼ばれる認可モデルは, 認可モデルとしては，比較的最近提唱されているモデルである[21]．この”信頼管理”とは，エンティティの定めた信頼に関するポリシー (policy) とエンティティの信頼に関係する情報 (credential) そして，既存のエンティティ間の信頼関係 (trust relationship) を管理し，信頼関係の締結，解消などを自動的に行うための仕組みである．現在，様々な状況に適応できるような高い表現能力を持つポリシー記述 (たとえば[22]) や分散して存在している情報の取得方式などが研究されている．また，[23]は，認可規則から，認可の要求者に求める認可条件を算出する汎用的な手法を研究している．

また，青山は，Web サービスにおける信用創造・信用仲介での問題を提起している[14]. 本稿で述べた相互認可における課題とあわせて，安心できる e-ビジネスを実現するために，今後，研究が必要であると思われる．

7. 今後の取り組み

本稿では，サービス指向アーキテクチャにおける相互認可について，その課題と課題から導き出される要件を整理し，相互認可のための一抽象モデルを示した．

今後，Web サービスなど，具体的なサービス指向アーキテクチャを実現する技術を対象に，データモデルの具体化と処理モジュールの検討を行っていく予定である．

参考文献

[1] Anders Grangard: ebXML Technical Architecture Specification v1.0.4 Chapter 5 (2001) http://www.ebxml.org/specs/ebTA.pdf

[2] Steve Burbeck : The Tao of e-business services (2000) http://www-106.ibm.com/developerworks/webservices/library/ws-tao/index.html

[3] 青山 幹雄: ソフトウェアサービス技術へのいざない. 情報処理, Vol42,No.9,p 857-862 (2001)

[4] Sun Microsystems: JiniTM Architecture Specification(2001) http://wwws.sun.com/software/jini/specs/jini1.2html/jini-title.html

[5] Web Services Architecture http://www.w3.org/TR/2002/WD-ws-arch-20021114/

[6] Uniform Resource Identifiers (URI): Generic Syntax(1998) http://www.ietf.org/rfc/rfc2396.txt

[7] Web Services Description Language 1.1 (2001) http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[8] SOAP Version 1.2 Part 1: Messaging Framework(2002) http://www.w3.org/TR/soap12-part1

[9] UDDI Version 3 (2002) http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf

[10] Security Taxonomy and Glossary. http://www.garlic.com/~ lynn/secgloss.htm

[11] The Kerberos Network Authentication Service (V5) http://www.ietf.org/rfc/rfc1510.txt

[12] WSCI,, Web Services Choreography Interface (2002) http://www.w3.org/TR/wsci/

[13] BPEL4WS: Business Process Execution Language for Web Services (2002) http://www-106.ibm.com/developerworks/library/ws-bpel/

[14] 青山 幹雄: Webサービスの信用創造・信用仲介モデル(2002). 情処研報 Vol.2002,No.92,p63-68

[15] 日科技連 「セキュリティハンドブック II」 3.1.3

[16] Web サービスのセキュリティ: アーキテクチャとロードマップの提案 (2002) http://www-6.ibm.com/jp/developerworks/webservices/020607/j_ws-secmap.html

[17] OASIS: Organization for the Advancement of Structured Information Standards http://www.oasis-open.org/

[18] XACML: eXtensible Access Control Markup Language (2002) http://www.oasis-open.org/committees/xacml/

[19] SAML: Security Assertion Markup Language (2002) http://www.oasis-open.org/committees/security/

[20] Reiner Kraft: Designing a Distributed Access Control Processor for Network Services on the Web (2002), *ACM Workshop on XML Security 2002*

[21] M Blaze, J Feigenbaum, and Jack Lacy: Decentralized Trust Management (1996). *In Proceedings of the IEEE Symposium on Research in Security and Privacy*

[22] Ninghui Li, John C. Mitchell, and William H. Winsborough:Design of A Role-based Trust-management Framework (2002), *Proceedings of the 2002 IEEE Symposium on Security and Privacy* http://crypto.stanford.edu/~ ninghui/abstracts/rt_oakland02.html

[23] Sushil Jajodia, Michiharu Kudo, and V.S. Subrahmanian: Provisional Authorizations(2000), *Proc. 1st Workshop on Security and Privacy in E-Commerce*