FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA Policies and Domains Specification

Document title	FIPA Domains and Policies Specification		
Document number	PC00089D	**Document source**	FIPA TC Architecture
Document status	Preliminary	**Date of this status**	2001/08/10
Supersedes	None		
Contact	arch@fipa.org		
Change history			
2000/12/15	An initial summary of use case scenarios and architectural elements that have been identified to support policy mechanisms in agent platforms		
2000/12/26	Edits and clean ups of initial text		
2001/01/03	Format conversion to FIPA 2000 compliance, edits and clean-up		
2001/01/29	Additional use cases and material concerning conversation policies		
2001/08/10	Line numbering added		

Foreword

The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-based applications. This occurs through open collaboration among its member organizations, which are companies and universities that are active in the field of agents. FIPA makes the results of its activities available to all interested parties and intends to contribute its results to the appropriate formal standards bodies.

The members of FIPA are individually and collectively committed to open competition in the development of agent-based applications, services and equipment. Membership in FIPA is open to any corporation and individual firm, partnership, governmental body or international organization without restriction. In particular, members are not bound to implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their participation in FIPA.

The FIPA specifications are developed through direct involvement of the FIPA membership. The status of a specification can be Preliminary, Experimental, Standard, Deprecated or Obsolete. More detail about the process of specification may be found in the FIPA Procedures for Technical Work. A complete overview of the FIPA specifications and their current status may be found in the FIPA List of Specifications. A list of terms and abbreviations used in the FIPA specifications may be found in the FIPA Glossary.

FIPA is a non-profit association registered in Geneva, Switzerland. As of January 2000, the 56 members of FIPA represented 17 countries worldwide. Further information about FIPA as an organization, membership information, FIPA specifications and upcoming meetings may be found at http://www.fipa.org/.

Contents

1 Introduction

This document gives a set of use cases and abstract architectural elements that can be used to guide the specification of policy mechanisms in concrete Agent Platform architectures. It is based on and derives from the FIPA Abstract Architecture Specification [FIPA00001].

As this specification is defined in terms of generic abstractions rather than specific concrete elements, this specification would require reification in order to derive specifications for particular types of agent platform. However, we anticipate that the architectural elements identified here will be present in all agent platforms that support policy mechanisms.

The basic services offered by an agent platform[1] to an agent are unconstrained. An agent may register any attributes that it chooses through the Agent Directory Service; it may use the Agent Message Transport Service to communicate with any reachable agent using any available transport and messages of any size or encoding, and it may operate on behalf of any principal.

In practice, however, developers and users of multi-agent systems often wish to place strong constraints on the behaviour of agents within agent environments. This especially means being able to apply and enforce these constraints and policies across distributed agents and systems.

Typical constraints that we may wish to enforce include:

Requiring that an agent use a particular encoding for its messages.

Preventing an agent from communicating with non-local agents (agents which lie outside some domain, in the transport addressing sense of the word).

Requiring than an agent select a particular quality of service (e.g. encryption, non-repudiation) when communicating with non-local agents.

Preventing an agent from registering certain attributes with the Agent Directory Service unless it is operating on behalf of a particular principal.

Limiting the total number of agents registered with a platform.

Restricting access to certain host directories or setting ceilings on the amount of system resources that can be used.

All of these constraints may be expressed as constraints over agent platform services. There may be other types of constraint that we wish to apply to an agent *X* for example, requiring the use of a particular conversation policy when interacting with a particular class of agent, or preventing an agent from transmitting confidential data to a non-local agent *X* but these lie outside the scope of this specification.

1.1 Contents

This document is organized into the following sections:

This **Introduction**.

The **Scope and Methodology** section explains the background of this work, its purpose, and the methodology that has been followed.

The **Policies, Domains and Agent Platforms** section is a description of the concepts and considerations necessary to the creation of policy driven agent systems.

---

[1] The Abstract Architecture specification does not refer to an Agent Platform. However, we use the term here informally to mean the set of inter-related services that are offered to an agent in order for it to discover other agents, and to communicate with those agents.

The **Policy Scenarios** section contains a selection of use cases illustrating the contexts in which policies are applied to agent systems.

The **Architectural Elements** section describes architecture components required.

1.2 Audience

The primary audience for this document is developers of concrete specifications for agent systems – specifications grounded in particularly technologies, representations, and programming models. It may also be read by the users of concrete specifications including implementers of agent platforms, agent systems, and gateways between agent systems.

This document describes abstract architectural elements for guiding the creation of policy mechanisms in intentional multi-agent systems. It assumes that the reader has a good understanding about the basic principles of multi-agent systems. It does not provide the background material to help the reader assess whether multi-agent systems are an appropriate model for their system design, nor does it provide background material on topics such as Agent Communication Languages, BDI systems, or distributed computing platforms.

1.3 Acknowledgements

TBD.

2 Policies, Domains and Agent Platforms

A set of constraints is termed an **Agent Service Policy**. In this specification, we are only concerned with policies that are *public*, i.e., accessible to systems, *machine readable*, i.e., can be processed by computer systems and in particular *declarative*, i.e., amenable to inference.

Policies may be expressed in a variety of languages. At one extreme they may be written in some propositional or constraint language such as SL2, in terms of some kind of agent platform service ontology. There are a wide variety of simpler schemes, each of which gives up some types of expressivity. The choice of language will be affected by (at least) the following considerations:

Composability The ability to combine two or more policies.

Computability The ability to compute the legality of some service request.

Efficiency The resource cost of evaluating the legality of a request.

Consistency Whether it is possible to express Ð or detect Ð inconsistent or contradictory requirements.

Expressivity Whether it is possible to express the required constraints in the language.

Equivalency Whether it is possible to compute the functional equivalence of two policies (and so, for example, reduce "legality of request" to "membership of some class associated with a given policy").

We assume that there are fundamentally two kinds of **policy constraints**: those relating to **permissions** and those relating to **obligations**. Not all platforms require both kinds of policies; however this specification introduces architectural elements that correspond to both forms. These policies are often related: by entering into particular obligations an agent may acquire specific permissions; and vice versa: when an agent is given permission to access a shared resource, it may incur obligations as a result.

Associated with policies and the mechanisms required to support policy application is the concept of a **contract**. A contract is an agreement entered into by agents and services to be constrained by one or more sets of policy constraints. In addition to the architectural elements needed by platforms to support policy mechanisms a given reification of these architectural elements may also require the **promise** communicative act. A promise is a speech act uttered by an agent when it agrees to abide by a set of policy constraints.

Many policies are applied at the point where an agent invokes a service of the agent platform (what about invariants?). The constraints on the use of a service can be of many kinds: constraints on the parameters supplied by the agent (for example on the size or format of a message), and constraints based upon the state of the agent platform, including the history of the interactions between the agent and the platform. In order for a service request to be honored, an inference procedure must be used to verify that the applicability requirements of the service may be satisfied in relation to the policies in force.

In principle, the inference procedure can be performed for every service request performed by every agent on a platform. However this may be prohibitively expensive from a computational standpoint. There are also situations when it is desirable to ask whether or not a request, or set of requests, would be permitted if an agent were to make them. (For example, a mobile agent might wish to know this before deciding whether to move to a particular agent platform.)

It is common to associate **policy mechanisms** with **policy domains**. A policy domain is simply a set of agents that is characterized by a set of policies. However, there are many benefits to constructing explicit domains: as aids to efficiently applying policies for example. The infrastructure needed to support policy domains typically includes constructs such as **domain managers**, etc.

Policy domains enable agent users to be assured of policy uniformity across multiple platforms and hosts, as long as semantically equivalent monitoring and enforcement mechanisms are available across those platforms and hosts. Under these conditions, it follows that a given domain could extend across host boundaries and, conversely, multiple domains could exist concurrently on the same host. With respect to platform independence, it should be possible for agents running on the same platform to be in different domains (for example, a resident and a visiting mobile agent running on the same platform may belong to different domains having more or less restrictive security privileges).

It is easy to imagine that agents might want to simultaneously belong to multiple domains. For example, it might be useful to structure an agent application as a series of hierarchically nested sub-domains. It might also be useful in some instances to specify a policy that precludes an agent from simultaneously belonging to more than one domain (e.g., if two domains are governed by incompatible security policies). Simultaneous membership of agents in multiple domains raises a number of currently unsolved technical issues.

3 Policy Scenarios

In this section, we identify a number of use case scenarios that illustrate many of the classical situations where policies and policy enforcement are relevant. They cover a range of situations that we may expect to encounter in policy application.

The scenarios are abstract in nature, rather than examples of concrete situations. Their emphasis is on illustrating the many different situations that policies may be applied and the kind of architectural support that would be required on Agent Platforms in order to support them. For the sake of continuity and comparison each of the use case scenarios is expressed using aspects of lawyer-client interaction.

3.1 Access Use Case

3.1.1 Description

Many policies relate to the provision of shared resources to agents. Shared resources are often constrained by quality of service constraints, access constraints and availability constraints. A key aspect of this class of policy scenarios is that an *owner* of each resource must be identifiable (which may or may not be an agent) and that an *owner* be responsible for applying any policy constraints to the resource.

This scenario is characterized by a set of resources, methods for accessing those resources, ownership of the resources and quality of service constraints upon the resources.

3.1.2 Scenario

The resource may be viewed as an entity offering a selection of legal services: a lawyer agent. To apply for access to the resource, an agent must present its credentials and requirements to the lawyer.

The lawyer agent applies policy constraints to the request, relating to its contractual requirements of the client agent.

The result is a 'quality of service' specification that constrains the set of actions that the client agent may perform on the lawyer resource.

3.2 Social Grouping Use Case

3.2.1 Description

There may be policy constraints on the permissible communication between agents based on external attributes of those agents.

In many situations agents with access to one set of resources are not permitted to communicate with agents that have access to other resources. For example, in a merchant bank, agents (typically human agents) who have access to the stock market - i.e., are able to buy and sell stocks and shares, are not permitted access to financial services such as loan arrangements. This is the so-called 'Chinese Wall' encountered in larger merchant banks and represents the conditions that legislation imposes on merchant banks to allow them to do business in multiple sectors.

These policy constraints are therefore strongly connected to groups of agents rather than the ability of individual agents to access resources.

3.2.2 Scenario

The different groups of agents in a merchant bank are divided into disjoint domains. An agent is required to register with a domain, either the stock domain or the mortgage domain (say), in order to communicate with agents in those domains.

An agent enters a domain by registering with the domain manager of that domain. Once registered, the agent is permitted to send and receive messages from agents in the same domain. In general, an agent may be permitted to be a member of several domains; depending on the policy constraints of the various domain managers.

This policy is enforced by preventing agents in one domain from communicating with agents in another domain.

In addition to preventing communication, other restrictions may include *hiding* agent descriptions: a directory service can hide information about agents to non-member agents.

3.3 Obligation Use Case

3.3.1 Description

Agents may enter into agreements that oblige them into a certain future behaviour. Obligation constraints cannot be enforced *a priori*, however sanctions can be applied to agents that fail to meet their obligations.

There are many situations where an agent may be obliged to perform a task: for example, a clock agent will enter into an agreement to send a message at specific intervals, a database update agent will agree to inform the requester that an update has taken place within the database and a file printing agent will agree to print a file within some interval or at an agreed time.

An important service that can support obligations is the reputation service (see *Section 4.3.4,Reputation Service*). Such services provide a means for agents to 'complain' about other agents' failure to meet obligations and for agents to verify the reliability of other agents before entering into agreements.

3.3.2 Scenario

A lawyer agent agrees the terms of legal contract with a client. After contractual negotiation between the lawyer and client the terms are submitted to a recognised reputation service.

The client agent notices that an action required of the lawyer agent has not taken place and files a complaint with the reputation service.

A subsequent query to the reputation service reveals that the lawyer agent failed to complete on a contractual obligation, thereby potentially affecting future agreements clients.

3.4 Compositional Use Case

3.4.1 Description

An agent may require or be required to enter into several conjunctive policies. The relationship between the individual policy expressions may vary in strength, from weak influence to a strong propositional binding. Compositions may be changed dynamically (in agreement with a policy authority) through the addition, modification or removal of constraint clauses.

Constraint clauses may be: directly conjunctive to those describing the policy expression, at an upper-level changing the context of the policy, or at a sub-level modifying an individual constraint by adding a conditional factor.

3.4.2 Scenario

An agent that has an active contract with a lawyer agent may wish to augment the contract policies with respect to a specific legal scenario.

This implies retention of the original contract policy with an extension for the additional requirements, resulting in a new, composed policy expression.

3.5 Refrain Use Case

3.5.1 Description

An agent may be required to subjectively refrain from a particular action or set of actions according to the policy constraints governing its interactions with other agents.

This implies that no direct intervention is required on behalf of another agent or policy authority. Rather the agent knows that it must refrain from an action, perhaps one requested by another agent, in accordance with its policy constraints.

3.5.2 Scenario

A client agent wishes to express to a Lawyer agent that legal action should always be taken autonomously in regard to a specific case instance, with the exception that on the satisfaction of certain constraints the Lawyer should refrain from action.

The lawyer agent may be authorized to proceed with legal action with the constraint that no contact is to be made with agent X at any time. The lawyer agent will exert a refrain if such an instance arises.

3.6 Content Use Case

3.6.1 Description

Agents often apply policy constraints to their interactions with other agents. Policy driven agents such as these may publish public policies to guide interactions with other agents.

For example, an agent may choose to constrain the form of messages it receives from other agents, and publish those policies in a way that is revealed to certain other agents. This may perhaps include requirements that messages are signed or have specific content attached.

3.6.2 Scenario

A Lawyer agent may only interact with a client agent if the messages contain a form of payment.

The client agent must therefore ensure that, in addition to any of its own requirements, any messages it sends to the Lawyer agent contain some form of payment.

A third party, such as a bank service, may be involved to provide the client agent with appropriate modification to its messages thereby ensuring the Lawyer agent recognize the payment portion of the message content.

3.7 System Configuration Use Case

3.7.1 Description

In addition to individual agents entering into individual obligations, a group or system of agents (and services) may enter into coordinated performance related obligations. For example, a group of agents may guarantee to provide high availability for an explicit period.

Such obligations may not, in fact, be honoured by individual agents but by the agent system as a whole; and therefore will typically require monitoring and maintenance services.

3.7.2 Scenario

A group of agents is required to offer continuous high availability, with automatic reconfiguration as necessary.

A monitoring agent is used to observe the health of this group of agents and exert control if necessary. For example, if it observes that one or more agents are not performing as expected, it can compensate by adjusting the properties of the offending agents or by launching additional agents to offset the performance deficit.

Such a group service may be governed by service level agreements established between the agents and the monitoring agent.

3.8 Cooperation use case

3.8.1 Description

This is an agreement to agree between agents. For example, an agent may enter a non-antagonistic posture agreement with other agents incorporating guarantees and obligations on future behavior. This amounts to a sharing of goals between agents.

3.8.2 Scenario

In the Lawyer-client scenario, the client can pay a retainer to the Lawyer agent thereby creating a co-operational stance between the two.

The client can then make requests without submitting further payment for the duration of the contract.

This may require use of a reputation service to which a client agent may submit a complaint if the Lawyer agent refuses a request covered by the cooperation agreement.

3.9 Delegation Use Case

3.9.1 Description

An agreement where an agent delegates authority or obligation. For example, an agent may choose or be forced to defer authority on a particular stance, to another agent or group of agents. In the case where an agent segments a policy governed task and delegates it across a number of other agents, the policy should be transposed according to the actions of each delegated task segment.

In terms of contractual obligations, an agent may delegate only if the authority governing the obligation is aware of and accepts the action.

3.9.2 Scenario

The Lawyer agent may delegate a contractual obligation to a 'legal specialist' agent, perhaps operating within the same legal entity.

The delegatee is then required to meet the contractual obligations (or agreed subset thereof) specified by the delegator and agreed with the policy authority.

This requires the policy authority managing the contract, say a guard mechanism, to accept the delegation and make appropriate changes to the contract terms.

3.10 Meta Order Use Case

3.10.1 Description

A meta order policy governs the nature of other policies. For example, it may specify that all agreements between agents must involve a 'consideration' on both sides. (In Anglo-Saxon law, it is not possible to have a contract without something of value being exchanged between all participating parties.)

3.10.2 Scenario

In the Lawyer-client scenario; the exchange of information between the two parties may be governed by the mutual-consideration meta order policy.

In such a case, the reputation service must ensure that the client agent receives information from the Lawyer agent sufficient to represent any payment made.

Therefore, when a reputation service is asked to validate an agreement, it must verify that it contains an co-exchange of appropriate value. It will refuse to validate non-conforming agreements.

3.11 High Order Use Case

3.11.1 Description

A higher order constraint is parameterized by other constraints. This is a form of dependency amongst constraints; however, it is different to normal conjunction (which is implied by policy inheritance for example), in that a higher-order policy refers explicitly to a `policy variable'.

3.11.2 Scenario

A contract specifies that in the event of a dispute, the conflict resolution procedure associated with the domain that a particular agent is in should be used.

3.12 Trust Use Case

3.12.1 Description

Multiple levels of security may govern the relationships between agents and establishing a level of trust constrains the type of agreement relationships agents can enter into. A particular trust level, indicated by a label or directly by a set of policies, defines the constraints applicable to a given relationship.

3.12.2 Scenario

A new agent registers with a domain manager in order to interact with other agents within the domain.

The manager determines an appropriate trust level to assign the new agent and thereby a set of policies governing its interaction with other agents within the domain.

4 Architectural Elements Needed to Support Policies

The elements of the policies and domains framework are defined here. For each element, the semantics are described informally followed by the relationships between the element and others.

4.1 Introduction

4.2 Policy Structures

4.2.1 Policy

A policy is a constraint or set of constraints on the behaviour of agents and services.

We are concerned with policies that are both public, i.e., available for inspection by third parties (although with obvious caveats regarding access control) and machine readable, i.e., a software system should be able to interpret a policy statement and determine legal courses of action.

Types of constraints are defined as:

Structural constraints specify policies about agents, their states, relationships, and communications that should not be violated. For example, a purchasing agent that is on probation may not place more than three orders. Or, there may never be more than seven agents bidding for a given item. The requirement that all messages must be encoded in a particular manner is another example of a structural constraint.

Operational constraints specify policies about agent behaviour that should not be violated. For example, an Order agent may not close a particular order unless it has been shipped and paid for. Or, an Order may only be cancelled in it has not yet been shipped. Interaction protocols are also an example of operation constraints.

4.2.2 Policy Language

The language used to express policy statements and contracts. Semantically, a policy statement and contract are equivalent: they express an agreement between an agent and other agents and/or services that constrain the behaviour of both.

We are assuming that policy languages are *declarative*. This fits with the overall FIPA methodology as well as providing a number of substantial benefits to policy developers and policy mechanism implementers.

Logically, a policy statement takes the form of a conjunction of implications: when a condition holds then an action is permitted, prohibited or whatever. In fact, the consequence of a policy rule need not be limited to single actions: it may also denote an enabling condition which allows other policy rules to trigger.

In addition to standard predicate logic, we envisage a policy language having built-in ontologies for the concepts of *action*, *permission*, and *obligation*. For example, SL can be straightforwardly extended to include permission and obligation in a manner similar to its model for action.

4.2.3 Policy Library

A set of rules that form coherent collections of policy statements. A policy library may introduce higher-level policy concepts (for example, National Security Classification) to simplify the task of generating specific policy rules for agents and services.

4.2.4 Interpretation Engine

An interpretation engine is a mechanism for interpreting a set of policy rules and a proposed action to determine if the action is legal according to the policy rules. It is possible that for certain classes of policy languages an interpretation engine could also determine that a particular action is *required* at a given situation. However, in general this is a hard problem.

The interpretation mechanism uses:

Inference rules that state if a certain facts are true, a conclusion can be stated of inferred.

Computation policies that define how to derive results via algorithms. For example, the net price of a product can be computed as follows: (product price * (1 + tax percentage / 100)). Or, the set of all Managing Salesperson agents can be computed as the intersection of all Manager agents and Salesperson agents.

4.2.5 Distribution Mechanism

A distribution mechanism is a means for distributing policy rules from originating authorities to mechanisms that have the ability and responsibility of applying policies.

4.2.6 Conversation Policy

A number of research groups are working to extend the concept of FIPA *interaction protocols* to accommodate recent research in agent conversation policies.

Conversations are sequences of messages involving two or more agents intended to bring about a particular set of (perhaps jointly held) goals. In contrast to early agent communication research, agent researchers now acknowledge that agent communication is better modelled when conversations rather than isolated messages are taken as the primary unit of analysis.

Conversation policies are declarative specifications that govern specific instances of communications between agents using an agent communication language. Contrary to current transition net approaches to specifying FIPA interaction protocols, recent research suggests conversation policies are best represented as sets of fine-grained constraints on ACL usage. These constraints define the computational process models that are implemented in agents. The key notions here are:

Conversation policies provide a level of analysis that abstracts from the actual propositional content, agent communication language, and implementation of individual conversations.

Conversation policies help ensure reliable communication between agents whilst simplifying any inference required in determining which communicative act or other action should be made in response to a message.

The abstract architecture specifies a set of abstract objects that allow for the explicit representation of "a conversation", i.e. a related set of messages between interlocutors that are logically related by some interaction pattern. It is desirable that this property be achieved by the minimum of overhead at the infrastructure or message level; in particular, it is important that interoperability remain un-compromised. For example, a concrete implementation may deliver messages to conversation-specific queues based on an interpretation of the message envelope. To achieve interoperability with an agent that does not support explicit conversations (i.e. which does not allow individual messages to be automatically associated with a particular higher-level interaction pattern), it is necessary to specify the way in which the message envelope must be processed in order to preserve conversational semantics.

4.3 Enforcement mechanisms

The two classes of constraints, corresponding to prohibitions and obligations, require different kinds of enforcement mechanisms. The former can be supported with policy domains and the latter with reputation services.

4.3.1 Guards

An active computational element that interprets high level policies and ensures their enforcement in a platform-specific way. Permissions are necessarily enforced in a different fashion than obligations. Permissions are granted or not before an action is taken; whereas one can only monitor an agent's performance on its obligations and apply necessary remedies after the fact. (Include examples of exception handling here, e.g., rollback.)

4.3.2 Sanctions

Violations of policy can result in remedies being applied to the offending agent; e.g., restrictions on the future behavior of an agent, price controls, reduction in access. An indirect consequence of policy violation can also be that other agents choose not to communicate with an offending agent. The most extreme form of sanction could be loss of domain membership and even termination.

4.3.3 Policy Exception

An event raised as a consequence of a policy violation.

4.3.4 Reputation Service

Is a service that allows agents and services to monitor the public performance of agents and services in terms of their compliance to publicly entered-into policy agreements.

A reputation service takes the role of a trusted third party that agents and service providers may use to monitor compliance with agreements. Reputation services are one of the few mechanisms that are able to enforce obligations; since obligations cannot be prevented but only required.

A typical use of a reputation service is for all parties to an agreement to `escrow' their agreement with the reputation service. If one of the parties determines that another party has defaulted on an obligation it may lodge a complaint with the reputation service.

In software systems the concept of a legal remedy may seem moot; however, simply recording instances of default and offering that information to others querying the service may be a powerful deterrence mechanism. If an agent defaults on an obligation, other agents and services may become more reluctant to offer it facilities if they are able to query a reputation service.

4.3.5 Policy Domain

A set of agents to which a given set of policies apply. In certain cases it may be possible to use domain membership as a shorthand for applying the policy constraint inference procedures. In other words, the inference that a particular service request is consistent with the policies in force in a given context may be reduced to the tests that (1) the domain policies are consistent with the agent platform and (2) that the agent is a member of the domain.

A major purpose of Policy Domains is to ensure consistency of policy across a set of agents potentially running on different agent platforms and hosts. This can be accomplished as long as semantically equivalent monitoring and enforcement mechanisms are available across those platforms and hosts. Under these conditions, it follows that a given domain could extend across host boundaries and, conversely, multiple domains could exist concurrently on the same host. With respect to platform independence, it should be possible for agents running on the same platform to be in different domains (for example, a resident and a visiting mobile agent running on the same platform may belong to different domains having more or less restrictive security privileges).

4.3.6 Domain Manager

An agent domain consists of a unique instance of a domain manager along with any agents that are registered to it. The function of a domain manager is to serve as a single point of administration for policy management, i.e., configure, re-configure, store, publish and enforce where possible the set of policies declared for that domain.

4.4 Domain Management

It is possible to define domains that explicitly require registration, as well as domains that require no registration, or subordinate registration to other elements of the FIPA environment, such as physical agent platforms. In this specification, we are only concerned with policy domains that are active, and have explicit notions of domain membership (i.e., there is an explicit list of agents that are members of a given domain).

4.4.1 Directory Functions

Provides services for agents register directory-entries. Other agents can search the directory-entries to find agents with which they wish to interact. In other words, it provides services for registration, lookup, discovery, authentication, etc.

4.4.2 Conflict Resolution

Resolves conflicts that may arise between any combination of: agent, domain, host, and computational environment.

4.4.3 Policy Derivation

(rule generation)

4.4.4 Policy Change Notification

(broadcast/sub-domain broadcast).

4.4.5 Querying of Domain Policies

5 Elements for Testing Conformance

6 Communicative Acts

6.1 Promise

Summary	The action of agreeing to a contract; equivalent to signing the contract. The normal effect of promising is that the agent is bound by the terms of the contract.
Message Content	A proposition outlining the terms of the contract. Logically, a contract is a proposition, however it is normally in the form of conditional rules: if A is true then Y is true; where the various conditions of the contract and their consequences are outlined.
Description	Promise is a general purpose agreement that an agent will abide by the terms of a contract. Normally, the promise does not directly imply that the agent will perform some action(s); but that if called upon under the right circumstances will agree to perform the actions. The agent sending the promise informs the receiver that it does intend to abide by the terms of the contract.
Pragmatic Note	The form of the proposition should be of the form of a conjunction of conditional rules. Each rule may be of the form: `if Condition then feasible (`*`A`*`, `*`Action`*`)` where A is one of the agents party to the contract. Essentially, a promise is a promise to agree to act on future requests - provided that the preconditions of the rules apply.
Formal Model	<*i*, promise (*j*, *Phi*,))> <*i*, inform (*j*, *Ii Phi*,))> FP: RE: *Bj Ii Phi*
Examples	Fix me

7 References

[FIPA00001] FIPA Abstract Architecture Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00001/`