Session 5 : FIPA: Foundation for Intelligent Physical Agents

Behrouz Far
Schulich School of Engineering, University of Calgary
far@ucalgary.ca
http://www.enel.ucalgary.ca/People/far/
Course Curriculum

Overview of agent-based SE

Methodologies for agent-based analysis and design

Agent communication & knowledge sharing

Agent-based System Architecture & Organization

FIPA: Foundation for Intelligent Physical Agents

Principles of Object Technology

Other topics: Agent Interaction, Infrastructure, APIs, Performance metrics, Learning, Self-organizing systems etc.

far@ucalgary.ca
Contents

- **Session 5 : FIPA: Foundation for Intelligent Physical Agents**
  - Overview of the FIPA specification
  - Abstract architecture specification (final)
  - Domains and policies specification (experimental)
Agent Technologies Today

- Agents are a technology field gaining popularity and approaching maturity.
- Some products that claim to (and sometimes do) use agent technologies have begun to appear increasingly.
- In many cases standardization is the factor enabling wide deployment of agent-based products, services and applications.
Agent technology provides solutions for
- Open and interoperable systems
- Cooperation in system development
- Dynamic integration of new SW/HW components

Before FIPA (before 1996)
- About 60 proprietary agent systems were in competition
- Most of which were ‘closed’ systems
- Most of which were incompatible

The question was (and still is) how to develop software agents using Agent Technology?
Standards Are Good, But ...

- Application and product developers agree on the benefits of standardization, however, there is reluctance on the part of many to commit resources in processes perceived as unreliable because they are
  - Bureaucratic
  - Passive
  - Driven by extreme search of consensus
  - Sometimes poorly managed
Closed Agreements Are Good, But ...

- Companies then consider local agreements between individual business players.
- These are antagonized by other players who fear that an advantage is gained by others because a particular technology is:
  - Unfamiliar or poorly documented.
  - Owned by the others.
  - Based on business models not shared by others.
Agents Are Special Case, Why...

- Agents are a *common technology* that can potentially be utilized by various and numerous application fields.
- If application fields tend to develop and use technologies that solve just their own problems the results will be a waste of development efforts, resources and lack of *interoperability*. 
About FIPA

- FIPA is an IEEE Computer Society standards organization of parties which agree to share efforts to produce specifications of generic agent technologies:
  - in a timely fashion
  - internationally agreed
  - usable across a large number of applications
  - so that a high level of interoperability across agent based applications, services and equipment is achieved

- FIPA has been formally established, in 1996. Founding members were:
  - BT, CSELT, IBM, SUN, Matsushita, etc.

http://www.fipa.org/
FIPA Members

- FIPA Members are
  - Companies
  - Organizations
  - Universities
  - Governmental institutions

- They pay yearly membership fees, have right to vote on matters that require voting and are allowed to join the technical committees developing technical specifications.
FIPA Mission

The promotion of technologies and interoperability, specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings.

- In short:
  
  **Interoperability among autonomous software/hardware systems**
FIPA Principles

- Specifications are produced by identifying, selecting, augmenting and developing specifications of *generic agent technologies* that are usable across a large number of instances and provide a *high level of interoperability* with other applications.

- The goals are realized through the open collaboration of all players in the field.

- Specifications must be produced before industries make commitment.

- Specifications should cover only the minimum that is needed for interoperability.
FIPA Spec: Orientation

- **Application-oriented**
  - Personal Assistant
  - Personal Travel Assistance
  - Audio/Visual Entertainment and Broadcasting
  - Network Management
  - Nomadic Application Support

- **Technology-oriented**
  - Message transport
  - Agent communication languages
  - Semantic content languages
  - Interaction protocols (dialogues, conversations)
  - Platform management (white and yellow pages)
FIPA Spec: Subject Categories

Application

Abstract Agent Architecture

Agent Communication
Agent Management
Agent Message Transport
FIPA Spec: Subject Categories

- Application
- Abstract Agent Architecture
  - Agent Communication
  - Agent Management
  - Agent Message Transport
  - Interaction Protocols
  - Communicative Acts
  - Content Languages
FIPA specifications are classified according to their state in the specification life cycle.

Most of the specifications are either in the *preliminary* or *experimental* state. A few *standards* have been emerged since 2002.

Almost all of the FIPA’97 and ’98 specifications are obsolete by 2003!

A set of 25 specifications has made it to Standard (S) stage.
FIPA: Process

- An idea for FIPA work is formed
  - Developed further, possibly within a Special Interest Group (SIG)
- Submitted as work plan to FIPA Architecture Board (FAB)
  - Timeline
  - Committed participants
- Work plan carried out by
  - Technical Committee (TC) for normative specifications
  - Working Group (WG) for informative specs, applications, field trials, …
- Specifications are created (approved by…)
  - Preliminary (P): Draft under discussion (TC)
  - Experimental (X): Stable, suitable for implementation (FAB)
  - Standard (S): Stable, successfully implemented (FAB, Membership)
  - Deprecated (D): Potentially unnecessary (FAB, Membership)
  - Obsolete (O): Rendered unnecessary
FIPA: Technical Committees

- **Architecture**
  - Works on service and agent – description and location-as well as on policies (permissions and obligations)

- **Compliance**
  - Created to generate conformance profiles for FIPA specs and conformance methodology

- **Gateways**
  - Developed Nomadic applications support specs, and interoperability between FIPA agents operating in wireless and wireline network domains

- **Ontologies**
  - Develops and adapts existing ontologies to be used by FIPA agents

- **Semantics**
  - Develops a semantic framework for contracts, conversations and social behaviour
FIPA: Workgroups

- **Interoperability**
  - Created to run interoperability trials providing feedback on specifications

- **Security**
  - Develop a security story for FIPA and agents in general

- **Product Design & Manufacturing**
  - Undergoing reorganization, the input of the three groups should feed into a SIG proposal and liaison activities with the Holonic Manufacturing Services.
FIPA: Special Interest Groups

- Agentcities
  - Coordinates with Agentcities activities to enable a feedback from large scale deployment of Agent Technology to FIPA

- FIPA for Business Applications
  - Created to ensure and demonstrate the business relevance of FIPA

- Liaison
  - Makes liaison to projects, implementations, universities, etc.
FIPA: Subject Category
FIPA Spec: Subject Categories

- Application
- Abstract Agent Architecture
- Agent Communication
- Agent Management
- Agent Message Transport
1. Applications

- SI00014 Nomadic Application Support Specification (2002/12/03: Standard)
- XC00079 Agent Software Integration Specification (2001/08/10: Experimental)
- XI00080 Personal Travel Assistance Specification (2001/08/10: Experimental)
- XI00083 Personal Assistant Specification (2001/08/10: Experimental)
- XC00092 Message Buffering Service Specification (2002/05/10: Experimental)
- SC00094 Quality of Service Specification (2002/12/03: Standard)
2. Abstract Architecture

Abstract Agent Architecture

- **SC00001**  Abstract Architecture Specification  
  (2002/12/03: Standard)

- **PC00089**  Domains and Policies Specification  
  (2001/08/15: Preliminary)
3. Agent Communication

- SC00061 ACL Message Structure Specification (2002/12/03: Standard)
- XC00086 Ontology Service Specification (2001/08/10: Experimental)
- SC00091 Device Ontology Specification (2002/12/03: Standard)
- OC00018 FIPA 97: Agent Communication Language (Obsolete)
- OC00003 FIPA 97: Agent Communication Language, ver.2 (Obsolete)
Interaction Protocols

- SC00028 Request When Interaction Protocol Specification (2002/12/03: Standard)
Communicative Acts

- SC00037 Communicative Act Library Specification (2002/12/03: Standard)
Content Languages

- DC00007  Content Languages Specification  
  (2003/02/27: Deprecated)
- SC00008  SL Content Language Specification  
  (2002/12/06: Standard)
- XC00009  CCL Content Language Specification  
  (2001/08/15: Experimental)
- XC00010  KIF Content Language Specification  
  (2003/01/29: Experimental)
- XC00011  RDF Content Language Specification  
  (2001/08/15: Experimental)
4. Agent Management

- SC00023  Agent Management Specification (2002/12/06: Standard)
- OC00002  FIPA 98: Agent Management (Obsolete)
- OC00005  FIPA 98: Agent Management Support for Mobility (Obsolete)
- OC000017 FIPA 97: Agent Management (Obsolete)
- OC000019 FIPA 97: Agent Management ver. 2 (Obsolete)
- DC000087 Agent Management Support for Mobility Specification (2002/05/14: Deprecated)
- DC000090 Agent Configuration Management Specification (2001/08/10: Deprecated)
5. Agent Message Transport

- **SC00067**  
  Agent Message Transport Service Specification  
  (2002/12/06: Standard)

- **XC00093**  
  Messaging Interoperability Service Specification  
  (2002/04/22: Experimental)

- **OC00024**  
  Agent Message Transport Specification  
  (2001/08/10: Obsolete)
ACL Message Representation

- **SC00069**  ACL Message Representation in Bit-Efficient Specification (2002/12/03: Standard)
- **SC00070**  ACL Message Representation in String Specification (2002/12/03: Standard)
- **SC00071**  ACL Message Representation in XML Specification (2002/12/03: Standard)
Envelope Representation

- **SC00085**  Agent Message Transport Envelope Representation in XML Specification
  (2002/12/03: Standard)
- **SC00088**  Agent Message Transport Envelope Representation in Bit Efficient Specification
  (2002/12/03: Standard)
Transport Protocols

- **SC00075**  Agent Message Transport Protocol for IIOP Specification (2002/12/03: Standard)
- **SC00084**  Agent Message Transport Protocol for HTTP Specification (2002/12/03: Standard)
FIPA Abstract Architecture Specification

Revision SC00001L
(2002/12/06)
Contents

- Introduction
- Scope and Methodology
- Themes of The Abstract Architecture
- Architectural Overview
- Architectural Elements (AE)
- Agent Information Model (AIM)
To create agent systems, which could be deployed in commercial settings, it is important to understand and to use existing software environments.

This includes elements such as:

- Distributed computing platforms or programming languages
- Messaging platforms
- Security services
- Directory services
- Intermittent connectivity technologies
Coverage

The FIPA abstract architecture include:

- A specification that defines *architectural elements* and their relationships.
- Guidelines for the specification of agent systems in terms of particular *software* and *communications* technologies.
- Specifications governing the *interoperability* and conformance of agents and agent systems.
Model for Communication

- At the heart of FIPA’s model for agent systems is *agent communication*, where agents can pass semantically meaningful messages to one another in order to accomplish the tasks required by the application.
  - How those messages are *transferred* (i.e., the transport).
  - How those messages are *represented* (i.e., strings, objects, XML).
  - Optional *attributes* of those messages, such as how to authenticate or encrypt them.
Why Abstract Architecture?

- By describing systems abstractly, one can explore the relationships between fundamental elements of these agent systems.
- By describing the relationships between these elements, it becomes clearer how agent systems can be created so that they are interoperable.
- From these one can derive a broad set of possible concrete architectures, which will interoperate because they share a common abstract design.
Why Abstract Architecture?

- Because the abstract architecture permits the creation of multiple concrete realizations, it must provide mechanisms to permit them to interoperate.

- For example, one agent system may transmit ACL messages using the OMG IIOP protocol. A second may use IBM’s MQ-series enterprise messaging system. An analysis of these two systems (how senders and receivers are identified, and how messages are encoded and transferred) allows us to arrive at a series of architectural abstractions involving messages, encodings, and addresses.
Scope, Methodology & Themes of The Abstract Architecture
The primary focus is to create semantically meaningful message exchange between agents which may be using different messaging transports, different Agent Communication Languages (ACL), or different content languages. This requires numerous points of potential interoperability.

The scope of this architecture includes:

- Message transport interoperability.
- Supporting various forms of ACL representations.
- Supporting various forms of content language.
- Supporting multiple directory services representations.
Scope /2

- Some aspects of potential standardization are outside of the scope of this architecture. There are three different reasons why things are out of scope:
  - The area cannot be described abstractly.
  - The area is not yet ready for standardization, or there was not yet sufficient agreement about how to standardize it.
  - The area is sufficiently specialized that it does not need standardization.

- Some of the key areas that are **not** included are:
  - Agent lifecycle and management; Agent mobility; Domains; Conversational policy; Agent identity.
Abstract architecture is not directly implemented, but instead it forms the basis for the development of concrete architectural specification that can be realized. Abstract architecture may be mapped to various concrete realizations.
There may be a single element that can be defined concretely, and then used in a number of different systems.

**Example:** if a concrete specification were created for the directory-service element that describes the schemas to use when implemented in LDAP, that particular element might appear in a number of different agent systems.

Concrete realizations using a shared element realization: you can implement them in any way you want.
Themes

- Hierarchical relationships between the abstraction and the elements of a specific instantiation
- It lacks a number of elements such as agent-platform, gateways between agent systems, agent configuration, etc.
- These are not included because they are inherently coupled with specific implementations of the architecture, rather than across all implementations.
Architectural Overview

- The FIPA architecture defines at an abstract level how two agents can *locate* and *communicate* with each other by *registering* themselves and *exchanging* messages.

- To do this, a set of architectural elements and their relationships are described.

There are several ways to do so!
Agents & Services

- **Agents** communicate by exchanging messages which represent speech acts, and which are encoded in an agent communication language (ACL)
- **Services** provide support services for agents
- There are two support services:
  - Directory-services
  - Message-transport-services
Agents & Services /2

- **Services** may be implemented either as **agents** or as **software** that is accessed via method invocation, using programming interfaces such as those provided in Java, C++, or IDL.

- An **agent** providing a **service** is more constrained in its behavior than a general-purpose agent. In particular, these agents are required to preserve the semantics of the service.

- **Service agents** shall not have the degree of autonomy normally attributed to agents, e.g., they may not arbitrarily refuse to provide the service.
Agent Start Up

- On start-up an agent must be provided with a service-root.

- Typically the provider of the service-root will be a service-directory-service which will supply a set of service-locators for available agent lifecycle support services, such as message-transport-services, agent-directory-services and service-directory-services.

- In general, a service-root will provide sufficient entries to either describe all of the services available within the environment directly, or it will provide pointers to further services which will describe these services.

Remember Jade or Fipa-OS
The role of the **directory-service** is to provide a location where **agents** register **directory-entries**.

Other agents can search the **directory-entries** to find agents with which they wish to interact.

The **directory-entry** is a **key-value-tuple** consisting of at least two **key-value-pairs**:

- **Agent-name**: A globally unique name for the agent
- **Locator**: One or more transport-descriptors that describe the transport-type and the transport-specific address to use to communicate with that agent

**Directory-entry** may contain other descriptive attributes, such as the services offered, cost, restrictions on using the agent, etc.
Registering an agent with a Directory Service:

- Agent A wishes to advertise itself as a provider of some service.
- It first binds itself to one or more transports.
- It must advertise its presence by constructing a directory-entry and registering it with the directory-service.
Finding an Agent:

- Agents can use the **directory-service** to locate other agents with which to communicate.

Example:

- If agent B is seeking stock quotes, it may search for an agent that advertises use of the stock quote ontology.
- If it succeeds it will retrieve the **directory-entry** for agent A.

The **directory entry** includes the **agent-name**, the **locator**, which contains information related to how to communicate with the agent, and other optional attributes.
Agent Messages

Message Structure:
- The structure of a message is a key-value-tuple and is written in an agent-communication-language (ACL).
- The content of the message is expressed in a content-language (such as KIF, XML, etc.).
- The content-language may reference an ontology, which grounds the concepts being discussed in the content.
- The message also contains the sender and receiver names, expressed as agent-names.

Agent-names are globally unique unique name identifiers for an agent.
Messages can recursively contain other messages.

Remember agent-name is different from transport name.
Agent Messages /2

Message Transport:

- When a message is sent it is transformed into a payload, and included in a transport-message.
- The payload is encoded using the encoding-representation appropriate for the transport.
- The transport-message is the payload plus the envelope.
- The envelope includes the sender and receiver transport-descriptions.
- The transport-descriptions contain the information about how to send the message (via what transport, to what address, etc).
- The envelope can also contain additional information, such as the encoding-representation, data related security, and other data that needs to be visible to the transport or recipient.
**Agent Messages /3**

**Message Transport:**
- A message is transformed into a **payload** suitable for transport over the selected **message-transport**. An appropriate **envelope** is created that has sender and receiver information that uses the **transport-description** data appropriate to the transport selected. The combination of the payload and envelope is termed as a **transport-message**.
Agent 1234 can communicate with Agent ABC using either an SMTP transport or an HTTP transport.

If Agent 1234 is doing any reasoning about agents that it communicates with, it can use the agent-name “ABC” to record which agent it is communicating with, rather than the transport description.

Thus, if Agent 1234 changes transports, it would still have continuity of reasoning.

Remember locator in registry service
Agent Messages /5

- Same messages on two different transports:
  - The **transport-description** is different, depending on the transport that is going to be used.
  - The **message-encoding** of the **payload** may also be different.
  - However, the **agent-names** remain consistent across the two message representations.
Encrypting a Message Payload:

- The payload is encrypted, and additional attributes are added to the envelope to support the encryption.
- These attributes must remain unencrypted in order that the receiving party be able to use them.
Providing Interoperability

- Two ways in which the abstract architecture makes provision for interoperability: *transport interoperability* and *message representation interoperability*.

- To provide interoperability, there are certain elements that must be included throughout the architecture to permit multiple implementations. For example, an *agent* has both an *agent-name* and a *locator*. The *locator* contains *transport-descriptions*, each of which contains information necessary for a particular transport to send a message to the corresponding agent.

- The semantics of agent communication require that an agent’s name be preserved throughout its lifetime, regardless of what transports may be used to communicate with it.
WANT A BREAK?
You need this if:
Want to build your own agents; and
Want to comply with the FIPA requirements.
Architectural Elements (AE)

- An **element** is an item or entity that is part of the architecture, and participates in relationships with other elements of the architecture.
- The architectural elements are classified as either **mandatory** or **optional**.
- **Mandatory** elements must appear in all instantiations of the abstract architecture. They describe the fundamental services, such as agent registration and communications.
- **Optional** elements represent architecturally useful features that may be shared by some, but not all, concrete instantiations.

Any implementation “must” provide the mandatory architectural elements
AE: Service

- A service is defined in terms of a set of actions that it supports.
- Each action defines an interaction between the service and the agent using the service.
- The semantics of these actions are described informally, to minimize assumptions about how they might be reified in a concrete specification.
An Agent is a computational process that implements the autonomous, communicating functionality of an application.

Typically, agents communicate using an Agent Communication Language (ACL).

A concrete instantiation of agent is a mandatory element of every concrete instantiation of the abstract architecture.

- Agent is an instance of agent
- Agent has an agent-name
- Agent may have agent-attributes
- Agent has a locator, which lists the transport-descriptions for that agent
Agent may send messages via a transport-description, using the transport corresponding to the transport-description.

Agent may send a transport-message to one or more agents.

Agent may register with one or more directory-services.

Agent may have a directory-entry, which is registered with a directory-service.

Agent may modify its directory-entry as registered by a directory-service.

Agent may delete its directory-entry from a directory-service.

Agent may query for a directory-entry registered within a directory-service.

Agent is addressable by the mechanisms described in its transport-descriptions in its directory-entry.
AE: Agent Attributes

- The **agent-attributes** are optional attributes that are part of the **directory-entry** for an **agent**.
- They are represented as **key-value-pairs** within the **key-value-tuple** that makes up the **directory-entry**.
- The purpose of the attributes is to allow searching for **directory-entries** that match **agents** of interest.
  - A **directory-entry** may have zero or more **agent-attributes**
  - **Agent-attributes** describe aspects of an **agent**
An agent-communication-language (ACL) is a language in which communicative acts can be expressed. The FIPA architecture is defined in terms of an Abstract ACL.

An abstract syntax is a syntax in which the underlying operators and objects of a language are exposed, together with a set of precise semantics for those entities.

The primary role of an abstract syntax is to highlight the semantic meaning of constructs in the language at the possible expense of legibility and convenience of expression.
AE: Agent Name

- An agent-name is a means to identify an agent to other agents and services. It is expressed as a **key-value-pair**, is unchanging (i.e., it is immutable), and unique under normal circumstances of operation.
  - Agent has one agent-name.
  - Message must contain the agent-names of the sending and receiving agents.
  - Directory-entry must contain the agent-name of the agent to which it refers.
  - The agent-name should have no semantics.
**AE: Content**

- **Content** is that part of a communicative act that represents the component of the communication that refers to a domain or topic area.

- Note that, “the content of a message” does not refer to “everything within the message, including the delimiters”, as it does in some languages, but rather specifically to the domain specific component.

- In the ACL semantic model, a content expression may be composed from propositions, actions or terms.
  - **Content** is expressed in a **content-language**
  - **Content** may reference one or more **ontologies**
  - **Content** is part of a **message**
A content-language is a language used to express the content of a communication between agents.

FIPA allows considerable flexibility in the choice, form and encoding of a content language. However, content languages are required to be able to represent propositions, actions and terms (names of individual entities).

- Content is expressed in a content-language
- FIPA-SL, FIPA-RDF, FIPA-KIF, and FIPA-CCL are examples of content-language
A directory-entry is a key-value tuple consisting of the agent-name, a locator, and zero or more agent-attributes.

A directory-entry refers to an agent; in some implementations this agent will provide a service.

- Directory-entry contains the agent-name of the agent to which it refers
- Directory-entry contains one locator of the agent to which it refers. The locator contains one or more transport descriptions Directory-entry is managed by and available from a directory-service
- Directory-entry may contain agent-attributes
AE: Directory Service

- A directory-service is a shared information repository in which agents may publish their directory-entries and in which they may search for directory-entries of interest.
  - Agent may register, modify and delete its directory-entry with/from a directory-service
  - Agent may search for a directory-entry registered within a directory-service
  - A directory-service must accept valid, authorized requests to register, de-register, delete, modify and identify agent descriptions
  - A directory-service must accept valid, authorized requests for searching
  - A directory-service supports the following actions: register, modify, delete, query
An encoding-representation is a way of representing an abstract syntax in a particular concrete syntax.

Examples of possible representations are XML, FIPA Strings, and serialized Java objects.

In principle, nested elements of the architecture may use different encodings, for example, a message may be encoded in XML and the resulting string used as the payload of a transport-message encoded as a CORBA object.

- Payload, Message, Content and Transport-message are encoded according to an encoding-representation.
AE: Encoding Service

- An encoding-transform-service is a service. It provides the facility to transform a transport-message, payload, message or content from one encoding-representation to another.
  - Encoding-transform-service converts one encoding-representation to another encoding-representation
  - Encoding-transform-service can transform the encoding-representation of a transport-message, a payload, a message
  - Encoding-transform-service can transform the encoding-representation of message content
  - Encoding-transform-service supports the following actions: Transform encoding, Query encoding representation, Query available encodings
AE: Envelope

- An **envelope** is a **key-value tuple** that contains message delivery and encoding information. It is included in the **transport-message**, and includes elements such as the sender and receiver(s) **transport-descriptions**.

- It also contains the **encoding-representation** for the **message** and optionally, other message information such as validation and security data, or additional routing data.
  - **Envelope** contains **transport-descriptions**
  - **Envelope** optionally contains validity data (such as security keys for message validation), security data (such as security keys for message encryption or decryption) and routing data
  - **Envelope** contains an **encoding-representation** for the **payload** being transported
  - **Envelope** is contained in **transport-message**
AE: Explanation

- An encoding of the reason for a particular action-status.

- When an action exerted by a service leads to a failure response, the explanation is an optional descriptor giving the reason why the particular action failed.
  - Explanation qualifies an action-status.
A **locator** consists of the set of **transport-descriptions**, which can be used to communicate with an **agent**.

A **locator** may be used by a **message-transport-service** to select a **transport** for communicating with the **agent**, such as an **agent** or a **service**.

**Locators** can also contain references to software interfaces. This can be used when a **service** can be accessed programmatically, rather than via a messaging model.

- **Locator** is a member of **directory-entry**, which is registered with a **directory-service**
- A **locator** contains one or more **transport-descriptions**
- A **locator** is used by **message-transport-service** to select a **transport**
A **message** is an individual unit of communication between two or more **agents**.

A **message** encodes the communicative act.

Communicative acts can be recursively composed, so while the outermost act is directly encoded by the **message**, taken as a whole a given **message** may represent multiple individual communicative acts.

**Messages** are encoded using an **encoding-representation** and transmitted between **agents** over a **transport**.

A **message** includes an indication of the type of communicative act (for example, INFORM, REQUEST), the **agent-names** of the sender and receiver **agents**, the **ontology** to be used in interpreting the **content**, and the **content** of the **message** itself.
A message does not include any transport or addressing information. It is transmitted from sender to receiver by being encoded as the payload of a transport-message, which includes this information.

- Message is written in an agent-communication-language
- Message has content
- Message has an ontology
- Message includes an agent-name corresponding to the sender of the message and one or more agent-name corresponding to the receiver or receivers of the message
- Message is sent by an agent and is received by one or more agents
- Message is transmitted as the payload of a transport-message
- Message is encoded according to an encoding-representation
- Message is encoded by an encoding-transport-service
AE: Message Transport Service

- A message-transport-service is a service that supports the sending and receiving of transport-messages between agents.
  - Message-transport-service may be invoked to send a transport-message to an agent
  - Message-transport-service selects a transport based on the recipient’s transport-description
  - Message-transport-service supports the following actions: Bind transport, Unbind transport, Send message, Deliver message
Ontologies provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary.

A concrete instantiation of **ontology** is an optional element of concrete instantiations of the abstract architecture.

- **Message** has an **ontology**
- **Content** has one or more **ontologies**
- **Ontologies** must be nameable, findable and manageable
A payload is a message encoded in a manner suitable for inclusion in a transport-message.

- Payload is an encoded message
- Transport-message contains a payload
- Payload is encoded according to an encoding-representation
A service is a functional coherent set of mechanisms that support the operation of agents, and other services. These are services used in the provisioning of agent environments and may be used as the basis for interoperation.

- Service has a public set of behaviors and actions
- Service has a service description
- Service can be accessed by agents
- Directory-service is an instance of service, and is mandatory
- Message-transport-service is an instance of service, and is mandatory
AE: Transport

- A **transport** is a particular data delivery service, such as a message transfer system, a datagram service, a byte stream, or a shared scratchboard.

- A **transport** is a delivery system selected by virtue of the **transport-description** used to deliver messages to an agent.
  - **Transport-description** can be mapped onto a **transport**
  - **Message-transport-service** may use one or more **transports** to effect message delivery
  - A **transport** may support one or more **transport-encodings**
AE: Transport Description

- A transport-description is a key-value tuple containing a transport-type, a transport-specific-address and zero or more transport-specific-properties.
  - Transport-description has a transport-type
  - Transport-description has a set of transport-specific-properties and a transport-specific-address
  - Directory-entries include one or more transport-descriptions
  - Envelopes contain one or more transport-descriptions
AE: Transport Message

- A transport-message is the object conveyed from agent to agent. It contains the transport-description for the sender and receiver together with a payload containing the message.
  - Transport-message contains one or more transport-descriptions for the receiving agents
  - Transport-message contains a payload and an envelope
  - Transport-message is encoded according to an encoding-representation
AE: Transport Specific Properties

- A transport-specific-property is a property associated with a transport-type. These properties are used by the transport-service to help it in constructing transport connections, based on the properties specified.

- Transport-description includes zero or more transport-specific-properties
AE: Transport Type

- A transport-type describes the type of transport associated with a transport-specific-address.
  - Transport-description includes a transport-type
Agent Information Model (AIM)
AIM: Agent Relationships

- Basic relationships between an *agent* and other key elements of the FIPA abstract architecture.
AIM: Transport Message Relationships

- **Transport-message** is the object conveyed from **agent** to **agent**.
- It contains the **transport-description** for the sender and receiver or receivers, together with a **payload** containing the message.
AIM: Agent Directory Entry Relationships

- **Agent-directory-entry** contains the agent-name, agent-locator and agent-attributes.

- **Agent-locator** provides ways to address messages to an agent. It is also used in modifying transport requests.
AIM: Service Directory Entry Relationships

- Service-directory-entry contains service-name, service-type and service-locator.

- Service-locator provides the means to contact and make use of a service and contains one or more service-location-descriptions which in turn each contain a service-signature, the signature-type and the service-address.
AIM: Message Elements

- A Message is contained in a transport-message when messages are sent.
The **message-transport-service** is an option service that can send **transport-messages** between agents.
The purpose of Abstract Architecture: **interoperability** and **reusability**.

The elements of the abstract architecture are identified and codified. These can be formally related to every valid implementation.

If two or more systems use different technologies to achieve some functional purpose, abstract architecture helps to identify the common characteristics of the various approaches.
Conclusions
Some Projects …

- Major projects working around FIPA Agent Technology
  - AGENTCITIES.RTD
  - AGENTCITIES.NET
  - ALIVE
  - COMMA
  - CRUMPET
  - FACTS
  - LEAP
  - SHUFFLE
  - SONG
A Few Implementations …

Lists major publicly available implementations

- Agent Development Kit
- April Agent Platform (AAP)
- Comtec Agent Platform
- FIPA-OS
- Grasshopper
- JACK Intelligent Agents
- Java Agent Development Environment (JADE)
- Lightweight Extensible Agent Platform (LEAP)
- ZEUS
Advantages of Standard Agent Technology

- **Enabling**
  because some products and applications become possible only when common standards exist

- **Technological**
  because economies of scale accelerate the deployment of products and applications

- **Synergetic**
  because different applications have a higher level of interoperability
Disadvantages of Standard Agent Technology

- Toooooooo much overhead!

Don’t bother if you don’t want your system evolve or if you don’t want to tie it to external services and MAS developed by the others!

Don’t bother if you don’t want to sell your system!
Conclusion

- The next few years will see agent technology to add to the momentum of revising the way we assemble our software.
- We are now moving away from software that is pre-engineered, pre-compiled, and pre-optimized toward software that is self-assembling, autonomous, adaptive and learns from its experiences.
- The cost of providing software services due to multiple and incompatible technologies will severely limit the growth of the software industry.
- Standardization of agent technology is the first and crucial step towards this goal.
WANT A BREAK?
FIPA Domains & Policies
Specification

Revision PC00089D
(Experimental: 2001/08/10)
Domain & Policy Specification

- A set of use cases and abstract architectural elements that can be used to guide the specification of policy mechanisms in Agent Platform architectures.
- It is based on and derives from the FIPA Abstract Architecture Specification.
Typical Constraints

In practice, developers of multi-agent systems often place *constraints* on the behaviour of agents within agent environments in order to enforce these constraints and policies across distributed agents and systems.

**Example:**

- 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).
- Requiring than an agent select a particular quality of service (e.g. encryption) 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 *policy constraints* over agent platform services.
Types of Policy Constraints

- There are two kinds of policy constraints: permissions and obligations.

- 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 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.

- 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.
Architectural Elements Needed to Support Policies
Policy Structures

- A *policy* is a set of constraints on the behaviour of agents and services.
- Policies are both *public*, i.e., available for inspection by third parties and *machine readable*, i.e., a software agent should be able to interpret a policy statement and determine (legal) courses of action.
Policy Structures (cont’d)

Types of constraints are defined as:

- **Structural constraints** specify policies about agents, their states, relationships, and communications that should not be violated.
  - **Examples:**
    - A purchasing agent that is on probation may not place more than three orders.
    - There may never be more than seven agents bidding for a given item.
    - All messages must be encoded in a particular manner.

- **Operational constraints** specify policies about agent behaviour that should not be violated.
  - **Examples:**
    - An Order agent may not close a particular order unless it has been shipped and paid for.
    - An Order may only be cancelled if it has not yet been shipped.
    - Interaction protocols.
Policy Structures (cont’d)

Policy Language

- The language used to express policy statements and contracts is assumed to be *declarative*.
- A policy statement takes the form of a conjunction of implications: when a condition holds then an action is permitted or prohibited.
- The consequence of a policy rule need not be limited to a single action: it may also denote an enabling condition which allows other policy rules to trigger.
- In addition to standard predicate logic, a policy language may have built-in ontologies for the concepts of *action*, *permission*, and *obligation*. 
Policy Structures (cont’d)

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 or to determine that a particular action is *required* at a given situation.

- The interpretation mechanism uses:
  - **Inference rules** that state if a certain facts are true, a conclusion can be stated or inferred.
  - **Model-driven computation rules** that define how to derive results via algorithms. E.g., the price of a product can be computed as:
    \[(\text{product price} \times (1 + \text{tax percentage} / 100))\].
Policy Structures (cont’d)

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.

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.
Enforcement Mechanisms

- 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.

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.
Enforcement Mechanisms (cont’d)

Sanctions

- Violations of policy can result in remedies being applied to the offending agent; e.g., restrictions on the future behaviour 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.
- Different forms of sanction, such as loss of domain membership and even termination may be imposed.

Policy Exception

- An event raised as a consequence of a policy violation.
Enforcement Mechanisms (cont’d)

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.
Enforcement Mechanisms (cont’d)

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.
Enforcement Mechanisms (cont’d)

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.
Policy Scenarios
1. Access Use Case

Description

- Many policies relate to the provision of shared resources to agents.
- Shared resources are 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.
Access Use Case (cont’d)

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 QoS specification that constrains the set of actions that the client agent may perform on the lawyer resource.
2. Social Grouping Use Case

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.
- **Example:** in a 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 banks and represents the conditions that legislation imposes on banks to allow them to do business in multiple sectors.
- These policy constraints are connected to groups of agents rather than the ability of individual agents to access resources.
Social Grouping Use Case (cont’d)

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, 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. Obligation Use Case

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.
- A service that can support obligations is the reputation service that 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.
Obligation Use Case (cont’d)

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.
4. Compositional Use Case

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.
Compositional Use Case (cont’d)

Scenario

- An agent that has an active contract with a lawyer 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.
5. Refrain Use Case

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.
Refrain Use Case (cont’d)

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.
6. Content Use Case

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.
Content Use Case (cont’d)

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.
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.

- 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.
System Config Use Case (cont’d)

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.
8. Cooperation Use Case

Description

- This is an agreement to agree between agents.
- For example, an agent may enter an agreement with other agents incorporating guarantees and obligations on future behaviour.
- This amounts to a sharing of goals and/or knowledge between the agents.
Cooperation Use Case (cont’d)

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.
9. Delegation Use Case

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.
Delegation Use Case (cont’d)

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.
10. Meta-Order Use Case

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 all sides. (In Anglo-Saxon law, it is not possible to have a contract without something of value being exchanged between all participating parties.)
Meta-Order Use Case (cont’d)

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.
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.
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.
12. Trust Use Case

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.
Trust Use Case (cont’d)

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.
I'm done!