Foundation for Intelligent Agents
(FIPA)

Technical Committee C:
Agent Communication

Response to the OMG
ANALYSIS & DESIGN TASK FORCE
UML 2.0
REQUEST FOR INFORMATION

Extending UML for the Specification of Agent Interaction Protcols

OMG Document ad/99-12-03
16 December 1999

Response contact point

Dr. Bernhard Bauer,

Corporate Technology

Intelligent Autonomous Systems

ZT IK 6

D-81730 Munich

Tel: +49 (89) 636 - 5 06 54

Fax: +49 (89) 636 - 4 14 23

Email: bernhard.bauer@mchp.siemens.de

Contents

1 RESPONSE INTRODUCTION 4

1.1 Objectives 4

1.2 Scope 4

1.3 Experiences with UML 1.x 4

1.4 Need for a major revision RFP 5

2 EXTENDING UML FOR THE SPECIFICATION OF PROTOCOLS 6

2.1 Introduction 6

2.2 Extending UML by Protocol Diagrams 7
2.2.1 Protocol Diagrams 7
2.2.2 Agentroles 9
2.2.3 Agent Lifeline 10
2.2.4 Threads of Interaction 12
2.2.5 Messages 13
2.2.6 Complex Messages 15
2.2.7 Nested Protocols 16
2.2.8 Complex Nested Protocols 17
2.2.9 Threads of Interaction and Messages inside and outside Nested Protocols 18
2.2.10 Parameterized Protocol 19
2.2.11 Bound Element 20

2.3 Defined protocols 22
2.3.1 Failure to understand a response during a protocol 22
2.3.2 FIPA-request Protocol 22
2.3.3 FIPA-query Protocol 22
2.3.4 FIPA-request-when Protocol 23
2.3.5 FIPA-contract-net Protocol 24
2.3.6 FIPA-Iterated-Contract-Net Protocol 25
2.3.7 FIPA-Auction-English Protocol 26
2.3.8 FIPA-Auction-Dutch Protocol 27
2.3.9 Brokering / Recruiting protocol .... **Error! Bookmark not defined.**

ACKNOWLEDGEMENT 28

REFERENCES 28

1 Response Introduction

The Foundation for Intelligent Physical Agent (FIPA) standards organization submits this document to the OMG's Analysis and Design Task Force (ADTF) in response to the Request for Information (RFI) entitled "UML 2.0 RFI." The specifications in this document are currently under consideration by FIPA, and are expected to be released as draft specifications in Q1 2000.

1.1 Objectives

The purpose of this RFI response is to suggest possible UML (Unified Modeling Language) extensions to the interaction diagram that will express agent interaction protocols (AIP). *An agent interaction protocol describes a communication pattern as an allowed sequence of messages between agents and the constraints on the content of those messages.* While such communication protocols might also be used between objects, an immediate need for agents and agent-based systems now exists.

This RFI response seeks to provide information that will help the AD PTF determine whether a major revision to UML 1.x is required and, in the event that it is, assist it to specify the requirements for a major revision RFP. In particular, this response maintains the following:

- Based on our user-base modeling and tool implementation experiences, we have assessed the strength and weaknesses of the current UML 1.x specification and found that agent interaction protocols could not be adequately expressed.
- To accommodate the modeling requiements for interaction protocols, we believe a major revision for UML will be required in the near future.

1.2 Scope

While the OMG Policies and Procedures do not provide detailed guidelines regarding what is inside or outside the scope of a major revision, the Foundation for Intelligent Physical Agent standards organization proposes the improvements contained in this response. In doing so, we have kept the following in mind:

- Any proposed changes either maintain or improve the rigor and the integrity of the current specification.
- All proposed changes consider backward incompatibility issues.
- All proposed changes consider the pragmatics of usage and implementation within a reasonable time frame.

1.3 Experiences with UML 1.x

FIPA has recognized the strengths and weaknesses of the UML 1.x specification based on user modeling and tool vendor implementation experiences. In particular, this response addresses the following subset of solicited experiences:

4.1.4 Are there any parts of the specification that need further clarification or expansion?
This response suggests that some further expansion is required to express interaction protocols more clearly.

4.1.6 What are the most difficult issues facing UML modelers and implementers? How should these issues be resolved?
This response suggests that press interaction protocols are both an important as well as difficult issue facing UML modelers and implementors. FIPA suggests that the extensions to the Interaction Diagram

modeling language proposed in this document will alleviate many of the problems encountered in representing interaction protocols.

4.1.7 What UML profiles are you using or do you plan to use?

Our experience indicates that most—if not all—extensions recommended in this RFI response are also required or desired by object-oriented analysis and design. As such, an additional profile might not be required. However, if the UML Revision Task Force determines that one or more of the recommendations in this paper are solely agent-based extension, an Agent UML Profile might be appropriate for UML 2.0.

4.1.8 Are you using UML for any applications outside its original scope.

Agent-based applications were not in the original scope of UML. However, the agent-based approach is now becoming a prominently employed technology. Again, many object-oriented applications have also found—*independently*—that many of the properties of agents (and agent-based systems) are also useful for objects (and OO systems).

1.4 Need for a major revision RFP

Respondents are asked to provide their opinion regarding the need for, and the timing of, one or more major revision RFPs. They are asked to answer the following questions:

4.2.1 Why or why not is a major revision of UML specification required within the next 2 years?

Interaction protocols and agent-based technology are becoming widely used. Without a standard modeling language to support these in a relatively short time frame, users will most probably respond by defining their own interaction protocol modeling languages—resulting in a tour of modeling-language Babel similar to that which existed prior to UML. In our opinion, a two-year time frame is a *maximum*.

4.2.2 If you think a major revision is required, should more than one RFP be issued? If so, how should the requirements be distributed among multiple RFPs?

As mentioned above, our experience indicates that the extensions recommended in this RFI response are also required or desired by object-oriented analysis and design. As such, a single RFP might only be required. However, if the UML Revision Task Force determines that one or more of the recommendations in this paper are solely agent-based extension, a separate agent-based RFP might be appropriate for UML 2.0.

2 Extending UML for the Specification of Protocols

2.1 Introduction

During the seventies, structured programming was the dominant approach to software development. Along with it, software engineering technologies were developed in order to ease and formalize the system development lifecycle: from planning, through analysis and design, and finally to system construction, transition, and maintenance. In the eighties, object-oriented (OO) languages experienced a rise in popularity, bringing with it new concepts such as data encapsulation, inheritance, messaging, and polymorphism. By the end of the eighties and beginning of the nineties, a jungle of modeling approaches grew to support the OO marketplace. To make sense of and unify these various approaches, an Analysis and Design Task Force was established on 29 June 1995 within the OMG. And by November 1997, a de jure standard was adopted by the OMG members called the Unified Modeling Language (UML).

The UML unifies and formalizes the methods of many approaches, including Booch, Rumbaugh (OMT), Jacobson, and Odell. It supports the following kinds of models:

static models- such as class and package diagrams describe the static semantics of data and messages. Within system development, class diagrams are used in two different ways, for two different purposes . First, they can model a problem domain conceptually. Since they are conceptual in nature, they can be presented to the customers. Second, class diagrams can model the implementation of classes—guiding the developers. At a general level, the term class refers to the encapsulated unit. The conceptual level models types and their associations; the implementation level models implementation classes. While both can be more generally thought of as classes, their usage as concepts and implementation notions is important both in purpose and semantics. Package diagrams group classes in conceptual packages for presentation and consideration. (Physical aggregations of classes are called components which are in the implementation model family, mentioned below.)

dynamic models- including interaction diagrams (i.e., sequence and collaboration diagrams), state charts, and activity diagrams.

use cases- the specification of actions that a system or class can perform by interacting with outside actors. They are commonly used to describe how a customer communicates with a software product.

implementation models- such as component models and deployment diagrams describing the component distribution on different platforms.

object constraint language (OCL)- is a simple formal language to express more semantics within an UML specification. It can be used to define constraints on the model, invariant, pre- and post-conditions of operations and navigation paths within an object net.

For modeling agents and agent-based systems, UML is insufficient. Compared to objects, agents are active because they act for reasons that emerge from themselves. The activity of agents is based on their internal states, which include goals and conditions that guide the execution of defined tasks. While objects need control from outside to execute their methods, agents know the conditions and intended effects of their actions and hence take responsibility for their needs. Furthermore, agents do not only act solely but together with other agents. Multiagent systems can often resemble a social community of interdependent members that act individually.

However, no sufficient specification formalism exists yet for agent-based system development. To employ agent-based programming, a specification technique must support the whole software engineering process—from planning, through analysis and design, and finally to system construction, transition, and maintenance.

A proposal for a full life-cycle specification of agent-based system development is beyond the scope of this specification. Here, a subset of an agent-based extension to the standard UML (AUML) for the specification of *agent interaction protocols* (AIP) is suggested. Ongoing conversations between agents often fall into typical patterns. In such cases, certain message sequences are expected, and, at any point in the conversation, other messages are expected to follow. These typical patterns of message exchange are called *protocols*. A designer of agent systems has the choice to make the agents sufficiently aware of the meanings of the messages, and the goals, beliefs and other mental attitudes the agent possesses, that the agent's planning process causes such protocols to arise spontaneously from the agents' choices. This, however, places a heavy burden of capability and complexity on the agent implementation, though it is not an uncommon choice in the agent community at large. An alternative, and very pragmatic, view is to pre-specify the protocols, so that a simpler agent implementation can nevertheless engage in meaningful conversation with other agents, simply by carefully following the known protocol.

Before defining interaction protocols and develop an UML extension for that purpose a common understanding on interaction protocols has to be established. The definition of a protocol describes

- a *communication pattern*, with
  - the allowed sequence of messages between agents having different roles,
  - constraints on the content of the messages, and
- the semantics according to the semantics of the communicative acts, i.e. the use of communicative acts within the pattern has to be consistent with their semantics.

It has to be distinguished between *generic (or parameterized) protocols* (and their instantiations) and *domain-specific protocols*.

2.2 Extending UML by Protocol Diagrams

The new diagram type suggested as an extension to UML is called *Protocol Diagram*. Well-defined interactions among agents are shown by protocol diagrams. They are similar to sequence diagrams and collaboration diagrams but combine the ideas of both. The complete used parts of UML are presented in order to obtain a self-explaining presentation.

2.2.1 Protocol Diagrams

Adapted from [UML, v1.3 section 3.59]

2.2.1.1 Semantics

A protocol diagram represents an Interaction, which is a set of messages exchanged among different agent roles within a collaboration to effect a desired behavior of other agentroles.

2.2.1.2 Notation

A protocol diagram has to dimensions: the vertical dimension represents time, the horizontal dimension represents different agentroles. Normally the time proceeds down the page. Usually only time sequences are important, but in real-time applications the time axis could be an actual metric. There is no significance to the horizontal ordering of the agentroles.

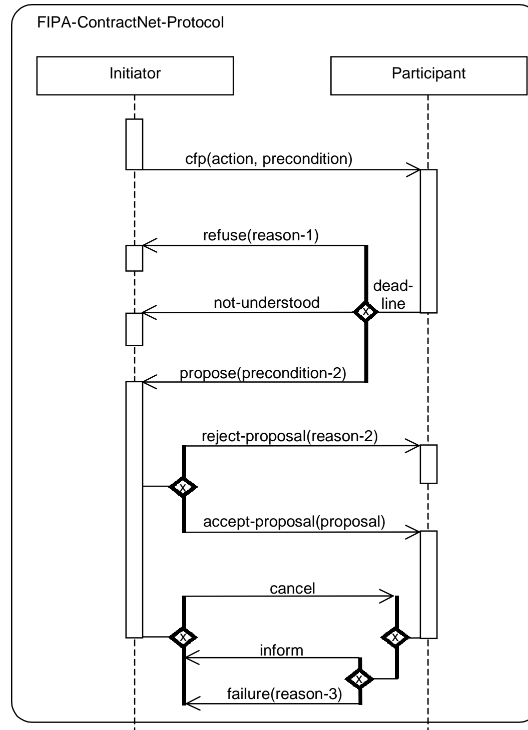
See subsequent sections for details of the contents of a protocol diagram.

2.2.1.3 Presentation Options

The axes can be interchanged, so that time proceeds horizontally to the right and different agentroles are shown as horizontal lines.

Various labels (such as timing marks, generated goals depending on the received message,...) can be shown either in the margin or near the lifelines or messages that they label.

2.2.1.4 Example



2.2.1.5 Mapping

The mapping is analogous defined as for sequence diagrams, see [UML: V1.3].

A protocol diagram maps like a sequence diagram into an Interaction and an underlying Collaboration. An Interaction specifies a sequence of communications; it contains a collection of partially ordered Messages, each specifying a communication between a sender role and a receiver role. Collections of agent roles that conform to the ClassifierRoles in the Collaboration owning the Interaction, communicate by dispatching Stimuli that conform to the Messages in the Interaction. An Agentrole maps into a ClassifierRole. A protocol diagram presents one collection of agentroles and arrows mapping to Agentrole and Stimuli that conform to the ClassifierRoles and Messages in the Interaction and its Collaboration.

In a protocol diagram, each agentrole box with its lifeline maps into an agent role which conforms to a ClassifierRole in the Collaboration. The name fields maps into the name of the agent, the role name into the Classifier's name and the class field maps into the names of the Classifier (in this case AgentClasses being Classes) being the base Classifiers of the ClassifierRole. The splitting of lifelines has an *concurrency* Association defining either AND/OR parallelism or decision Association denoting threads (<<thread>>). The associations among roles are not shown on the sequence diagram. They must be obtained in the model from a complementary collaboration diagram or other means. A message arrow maps into a Stimulus connected to two AgentRoles. the sender and receiver agentrole. The Stimulus conforms to a Message between the ClassifierRoles corresponding to the two agentroles' lifelines that the arrow connects. The Link used for the communication of the Stimulus plays the role specified by the AssociationRole connected to the Message. Unless the correct Link can be determined from a complementary collaboration diagram or other means, the Stimulus is either not attached to a Link (not a complete model), or it is attached to an arbitrary Link or to a dummy Link between the Instances

conforming to the AssociationRole implied by the two ClassifierRoles due to the lack of complete information. The name of the communicative act is mapped onto the behavior associated by the action performing, requested information, information passing, negotiation or error handling connected to the Message. Different alternatives exist for showing the arguments of the Stimulus. If references to the actual Instances being passed as arguments are shown, these are mapped onto the arguments of the Stimulus. If the argument expressions are shown instead, these are mapped onto the Arguments of the action performing, requested information, information passing, negotiation or error handling connected to the dispatching communicative act. Finally, if the types of the arguments are shown together with the name of the communicative act, these are mapped onto the parameter types of the communicative act. A timing label placed on the level of an arrow endpoint maps into the name of the corresponding Message. A constraint or guard placed on the diagrams maps into a Constraint on the entire Interaction. The cardinality label restricts the number of sending and receiving instances of agent roles accordingly to the numbers denoted at the beginning (sender) and end (receiver) of the message.

An arrow with the arrowhead pointing to an agentrole symbol within the frame of the diagram maps into a Stimulus dispatched by a CreateAction, i.e. the Stimulus conforms to a Message in the Interaction which is connected to the CreateAction. The interpretation is that the agentrole instance (not an arbitrary agent role, or a set of agentrole instances) is created by dispatching the Stimulus, and the agentrole instance conforms to the receiver role specified in the Message. After the creation of the agentrole instance, it may immediately start interacting with other agentroles. This implies that the creation of the agentrole dispatches these Stimuli. If an agentrole instance termination symbol (“X”) is the target of an arrow, the arrow maps into a Stimulus which will cause the receiving agent role instance to be removed. The Stimulus conforms to a Message in the Interaction with a DestroyAction attached to the Message or the agent instance terminates itself.

The order of the arrows in the diagram map onto a pair of associations between the Messages that correspond to the Stimuli the arrows maps onto. A predecessor association is established between Messages corresponding to successive arrow ends in the vertical sequence. In the definition of repetition the arrow can timely end before the actual message was sent. In this case the messages following this message are after the actual one. In case of concurrent arrows preceding an arrow, the corresponding Message has a collection of predecessors. In case of exclusive-or and inclusive-or arrows preceding an arrow the corresponding message has one and at least one out of the collection of possible predecessors, respectively. Moreover, each Message has an activator (thread of interaction) association to the Message corresponding to the incoming arrow of the activation or pro-active sending of a message.

A nested protocol maps into a protocol diagram. The name compartment of a nested protocol maps into the name of the Collaboration. The guard and constraint compartment maps into a constraint on the complete Interaction.

A complex nested protocol maps into a protocol diagram. The order of the messages within the protocol are defined according to the combination of the complex nested protocol. The ordering of the messages in the nested protocol is the ordering of theses protocols. Depending on the combination the messages are sent in AND/OR-parallelism or decision ordering.

2.2.2 Agentroles

In the framework of agent oriented programming an agent satisfying a distinguished role behaves in a special way. In contrast to this semantics “role” in UML is an instance focused term. Moreover the term “multi-object” does not fit to describe agentroles. “multi-objects” are used to show operations that address the entire set, rather than a single object in it. But it should be expressed that there is a communication with one instance of this multi-object. By “agentrole” a set of agents satisfying distinguished properties, interfaces or having a distinguished behavior are meant.

UML distinguishes between

- *multiple classifications* like in the retailer example, where the retailer agent act as well as a buyer as well as a seller agent, ande
- *dynamic classification*, where an agent can change its classification during its existence.

Agents can perform various roles within one interaction protocol. Using a contract-net protocol, e.g. between a buyer and a seller of a product, the initiator of the protocol has the role of a buyer and the participant has the role of a seller. But the seller can as well be a retailer agent, which acts as a seller in one case and as a buyer in another case, i.e. agents satisfying a distinguished role can support multiple classification and dynamic classification. Another example can be found in the agent management specification of FIPA 97, it defines the functionality of the directory facilitator (DF), the agent management system (AMS) and the agent communication channel (ACC). These functionalities can be implemented by different agents, but for example the DF, AMS and ACC functionality can also be integrated into one agent.

An agentrole can be seen as a set of agents satisfying a distinguished interface, service description or behavior. Therefor the implementation of an agent can satisfy different roles.

See also [UML].

2.2.2.1 Semantics

An agentrole describes two different variations which can apply within a protocol definition. A protocol can be defined between different concrete agent instances or a set of agents satisfying a distinguished role and/or class. An agent satisfying a distinguished agentrole and class is called *agent of a given agentrole* and *class*, respectively.

2.2.2.2 Notation

An agentrole is shown as a rectangle. The rectangle is filled with some string of one of the following forms:

- “role” denoting arbitrary agents satisfying the distinguished agentrole “role”
- “instance / role-1 ... role-n” denoting a distinguished agent instance “instance” satisfying the agentroles “role-1”,..., “role-n” with $n \geq 0$.
- “instance / role-1 ... role-n : class” denoting a distinguished agent instance “instance” satisfying the agentroles “role-1”,..., “role-n” with $n \geq 0$ and class it belongs to.

2.2.2.3 Presentation Options

The second case can be abbreviated as “instance” if n equals zero, i.e. a concrete agent is meant independent of the role(s) it satisfies and class it belongs to.

2.2.2.4 Example

2.2.2.5 Mapping

see Mapping 2.2.1.5

2.2.3 Agent Lifeline

The agent lifeline defines the time period when an agent exists. For example a user agent is created when a user logs on to the system and the user agent is destroyed when the user logs off. Another example is when an agent migrates from one machine to another.

See also [UML, v1.1 section 7.3]

2.2.3.1 Semantics

A protocol diagram defines the pattern of communication, i.e. the steps in which the communicative acts are send between agents of different agentroles. The agent lifeline describes the time period in which an agent of a given agentrole exists. Only during this time period an agent can participate on a protocol.

The lifeline starts when the agent of a given agentrole is created and ends when it is destroyed.

The lifeline can be split in order to describe AND and OR parallelism and decisions and may merge together at some subsequent point.

2.2.3.2 Notation

An agent lifeline is shown as a vertical dashed line. The lifeline represents the existence of an agent of a given agentrole at a particular time. If the agent is created or destroyed during the period of time shown on the protocol diagram, then its lifeline starts or stops at the appropriate point; otherwise it goes from the top of the diagram to the bottom. An agentrole is drawn at the head of the lifeline. If an agent of a given agentrole is created during the protocol diagram, then the message that creates it is drawn with its arrowhead on the agentrole. Note, that the agentrole (see Example 2.2.3.4) that receives the message is responsible for the creation of the agentrole, i.e. the arrowhead ends at the dashed line of the agentrole receiving the message and the agentrole is fixed at the left-hand or right-hand side of the lifeline or the thread of interaction. If an agentrole is destroyed during the protocol diagram, then its destruction is marked by a large “X”, either at the message that causes the destruction or (in the case of self destruction) at the final action of the agentrole. The termination is restricted to concrete instances of an agent role.

Agentroles that exist when a protocol starts is shown at the top of the diagram (above the first message arrow). An agentrole that exists when the protocol finishes has its lifeline continued beyond the final arrow of the diagram.

The lifeline may split into two or more lifelines to show AND and OR parallelism and decisions. Each separate track corresponds to a branch in the message flow. The lifelines may merge together at some subsequent point.

The splitting of the lifeline for

- AND parallelism starts at a horizontal heavy bar,
- OR parallelism (inclusive-or) starts at a horizontal heavy bar with a non-filled diamond and
- decision (exclusive-or) starts at a horizontal heavy bar with a non-filled diamond with “x” inside the diamond.

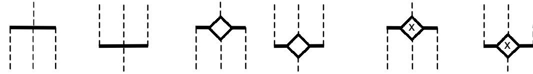
and is continued with a set of parallel vertical lifelines connected to the heavy bar.

The merging is done the analogous way, i.e. the parallel vertical lifelines stop at some of the horizontal heavy bars and one lifeline is continued at the heavy bar.

2.2.3.3 Presentation Options

none.

2.2.3.4 Example



see also Example 2.2.1.4.

2.2.3.5 Mapping

see Mapping 2.2.1.5

2.2.4 Threads of Interaction

The sending of messages can be done either in parallel or as a decision between different communicative acts. Receiving different communicative acts usually results in different behavior and different answers, i.e. the behavior of an agentrole depends on the received message.

Adapted from [UML, v1.1 section 7.4].

2.2.4.1 Semantics

Since the behavior of an agentrole depends on the incoming message and different communicative acts are allowed as an answer to a communicative act, the thread of interaction, i.e. the processing of incoming messages, has to be split up into different threads of interaction. The lifeline of an agentrole is split and the thread of interaction defines the reaction to received messages.

The thread of interaction shows the period during which an agentrole is performing some task as a reaction to an incoming message. It represents only the duration of the action in time, but not the control relationship between the sender of the message and the receiver. A thread of interaction is always associated with the lifeline of an agentrole.

2.2.4.2 Notation

A thread of interaction is shown as a tall thin rectangle whose top is aligned with its initiation time and whose bottom is aligned with its completion time. It is drawn over the lifeline of an agentrole. The task being performed may be labeled in text next to the thread of interaction or in the left margin, depending on the style; alternately the incoming message may indicate the task, in which case it may be omitted on the thread of interaction itself.

If the distinction between the reaction to different incoming communicative acts can be neglected, the entire lifeline may be shown as one thread of interaction.

2.2.4.3 Presentation Options

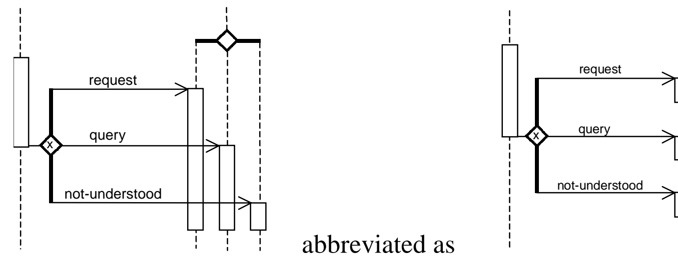
Variation:

A thread of interaction can take only a short period of time. For compactification reasons of the diagram the parallelism and the decision can be abbreviation omitting the splitting / merging and putting the different threads of interaction one after another on the lifeline.

Variation:

A break of the rectangle describes a change in the thread of interaction.

2.2.4.4 Example



2.2.4.5 Mapping

see Mapping 2.2.1.5

2.2.5 Messages

The main issue of protocols is the definition of communicative patterns, especially the sending of messages from one agentrole to another. This sending can be done in different ways, e.g. with different cardinalities, depending on some constraints or using AND / OR parallelism and decisions.

Adapted from [UML, v1.1 section 8.9, v1.1 section 7.5]

2.2.5.1 Semantics

A message or sending of a communicative act is a communication from one agentrole to another that conveys information with the expectation that the receiving agentrole would react according to the semantics of the communicative act. The specification of the protocol says nothing about the implementation of the processing of the communicative act.

2.2.5.2 Notation

A message sending is shown as a horizontal solid arrow from a thread of interaction of an agentrole to another thread of interaction of another agentrole. In case of a message is sent from an agentrole to itself, the arrow may start and end on the same lifeline or thread of interaction. Such a nested thread of interaction is denoted by a thread of interaction which is shifted a little bit to the right side in the actual thread of interaction.

The starting and termination of a protocol for a distinguished thread of interaction is shown by an arrow starting with a small solid filled circle and an arrow ending at a circle surrounding a small solid filled circle (a bull's eye).

Each arrow is labeled with a message label[1]. The message label has the following syntax

predecessor guard-condition sequence-expression communicative-act argument-list

predecessor. The predecessor consists of at most one natural number followed by a slash ('/') defining the sequencing of a parallel construct or the number of the input and output parameter in the context of section 2.2.9. The clause is omitted if the list is empty.

guard-condition. The guard-condition is a usual UML guard-condition, with the semantics, that the message is only sent iff the guard is true.

---

[1] The message label is a special case of the message label presented in the UML 1.1 specification section 8.9.2.

sequence-expression. The sequence-expression is a constraint. Especially “n..m” denotes that the message is sent n up to m times with $n \in N$, $m \in N \cup \{ * \}$, the asterix denotes arbitrary times. “broadcast “ denotes a broadcast sending of a message and “deadline”, a string coded according to ISO 8601 time coding, denotes the deadline until which a message has to be received.

communicative-act. The communicative-act is either the name, i.e. string representation with underlined name, of a concrete communicative act instance, the name of a concrete communicative act instance and its associated communicative act, written as “name:communicative-act” or only the name of the communicative act, e.g. “inform”.

argument-list. The argument-list is a comma-separated list of arguments enclosed in parentheses. The parentheses can be omitted of the list is empty. Each argument is an expression in pseudocode or an appropriate programming language or an OCL expression.

2.2.5.3 Presentation Options

Variation: Cardinality

The cardinality of a message, i.e. there are n sender and m receiver of a message is shown writing natural numbers at the beginning and at the end of the arrow. This variation is only allowed if the sender and/or receiver is not an instance of an agent.

Variation: Asynchronous Message Passing

An asynchronous message is drawn with a stick arrowhead. It shows the sending of the message without yielding control.

Variation: Synchronous Message Passing

A synchronous message is drawn with a filled solid arrowhead. It shows the yielding of the thread of control (wait semantics), i.e. the agentrole waits until an answer message is received and nothing else can be processed.

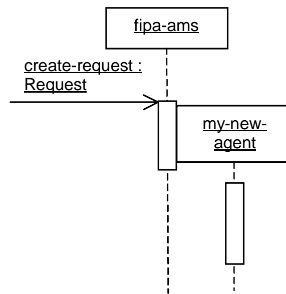
Variation: Time intensive Message Passing

Normally message arrows are drawn horizontally. This indicates the duration required to send the message is “atomic”, i.e. it is brief compared to the granularity of the interaction and that nothing else can “happen” during the message transmission. That is the correct assumption within many computers. If the messages requires some time to arrive, e.g. for mobile communication, during which something else can occur then the message arrow may be slanted downward so that the arrowhead is below the arrow tail.

Variation: Repetition

The repetition of parts of a protocol diagram is represented by an arrow or one of its variations usually marked by some guards or constraints ending at a thread of interaction which is according to the time axis before or after the actual time point.

2.2.5.4 Example



2.2.5.5 Mapping

see Mapping 2.2.1.5

2.2.6 Complex Messages

2.2.6.1 Semantics

A complex message may be the parallel sending of messages or exclusively sending of exactly one message out of a set of different messages.

2.2.6.2 Notation

Three kinds of complex messages are distinguished. All three complex messages substitute an arrow from one thread of interaction to another one by an arrow starting at one thread of interaction ending either

- at a heavy bar (for AND parallelism),
- at a heavy bar with a non-filled diamond (for OR parallelism; inclusive-or) or
- at a heavy bar with a non-filled diamond (for decisions; exclusive-or) with a “x” inside the diamond.

From these different kinds of heavy bars new arrows start in a right angle at the bar and end at possibly different threads of interaction which are possibly combined in a parallel or decisional way.

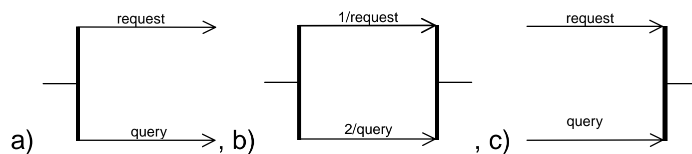
The merging of different messages is done in the analogous way, i.e. the parallel horizontal message arrow stop at some of the vertical bars and one message arrow is continued at the heavy bar.

2.2.6.3 Presentation Options

none.

2.2.6.4 Example

AND-parallelism of different messages, with sequencing b)



2.2.6.5 Mapping

see Mapping 2.2.1.5

2.2.7 Nested Protocols

To specify complex systems in a modular way nested protocols are introduced. Moreover the reuse of parts of a specification increases the readability of them. Additionally nested protocols are used for the definition of repetition of a nested protocol according to guards and constraints.

2.2.7.1 Semantics

The semantics of a nested protocol is the semantics of the protocol.

If the nested protocol is marked with some guard then the semantics of the nested protocol is the semantics of the protocol under the assumption that the guard evaluates to true, otherwise the semantics is the semantics of an empty protocol, i.e. nothing is specified.

If the nested protocol is marked with some constraints the nested protocol is repeated as long as the constraints evaluate to true.

2.2.7.2 Notation

A nested protocol is shown as a rectangle with rounded corners. It may have one or more compartments. The compartments are optional. They are as follows:

- *Name compartment*. Holds the (optional) name of the nested protocol as a string. Nested protocols without names are “anonymous” and are distinct up to isomorphism. It is undesirable to show the same named nested protocol twice in the same diagram except they define the same nested protocol. The compartment is written in the upper left-hand corner of the rectangle.

- *Guard compartment*. Holds the (optional) guard of the nested protocol in the usual guard notation as ‘[‘ guard-condition ‘]’. Nested protocols without guards are equivalent with nested protocols with guard ‘[true]’. The guard compartment is written together with the constraint compartment in the lower left-hand corner of the rectangle.

- *Constraint compartment*. Holds the (optional) constraint of the nested protocol in the usual constraint notation as ‘{‘ constraint-condition ‘}’. Nested protocols without constraints are equivalent with nested protocols with constraint ‘{ 1 }’. The constraint compartment is written together with the guard compartment in the lower left-hand corner of the rectangle. In addition to the constraint-condition used in UML the description n..m, denoting that the nested protocol is repeated n up to m times with $n \in \mathrm{N}$, $m \in \mathrm{N} \cup \{ * \}$, the asterix denotes arbitrary times, is used as a constraint condition.

Another nested protocol can completely be drawn within the actual nested protocol denoting that the inner one is part of the outer one.

2.2.7.3 Presentation Options

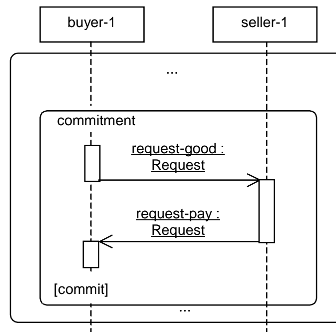
The abbreviations n and * can be applied to denote n..n and 0..*, respectively.

Beyond the above usage of nested protocols for simple protocols, nested protocols can also be used applying parameterized protocols or instantiated parameterized protocols.

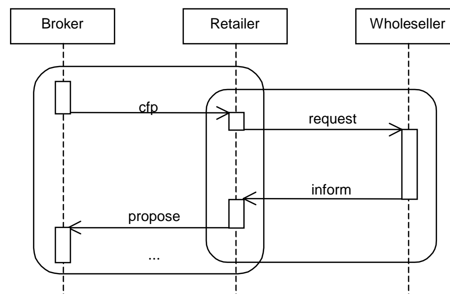
Another presentation option is the definition of *interleaved protocols*. For a nested protocol being part of another protocol the rectangle representing it has to be completely drawn within the other one. If interleaved protocols are defined, i.e. during performing one interaction protocol another protocol has to be processed, the rectangles are not drawn within each other.

2.2.7.4 Example

nested protocol



interleaved protocols



2.2.7.5 Mapping

see Mapping 2.2.1.5

2.2.8 Complex Nested Protocols

2.2.8.1 Semantics

A complex nested protocol defines the parallel or decisional combination of nested protocols. It has to take into consideration the thread of interaction at the beginning and at the end of the complex nested protocol. Furthermore the starting and ending point within the nested protocols have to be considered.

2.2.8.2 Notation

Three kinds of complex nested protocols are distinguished. All three complex nested protocols are drawn over the lifeline and threads of interaction within a protocol diagram. The nested protocols are combined lines starting in a right angle at the rectangle of a nested protocol. These lines are connected either by

- a heavy bar defining AND parallelism,
- a heavy bar with a non-filled diamond defining OR parallelism (inclusive-or) or
- a heavy bar with a non-filled diamond defining decisions (exclusive-or) with in “x” inside the diamond.

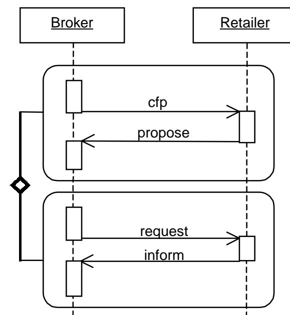
of nested protocols parallel to the rectangle edge where the lines are starting.

The threads of interaction which are continued in the different nested protocols are drawn accordingly.

2.2.8.3 Presentation Options

none.

2.2.8.4 Example



2.2.8.5 Mapping

see Mapping 2.2.1.5

2.2.9 Threads of Interaction and Messages inside and outside Nested Protocols

2.2.9.1 Semantics

Nested Protocols are defined in detail either within a protocol diagram where it is used or outside another protocol diagram. Threads of interaction and messages inside and outside nested protocols define the input and output parameter for nested protocols.

The input parameters are on the one side the threads of interaction which are carried on in the nested protocol and on the other side the messages which are received from other protocols.

The output parameters are on the one side the threads of interaction which are started within the nested protocol and are carried on outside the nested protocol and the messages which are sent from inside the nested protocol to agentroles not involved in the actual nested protocol. A message or thread of interaction ending at an input or starting at an output parameter of a nested protocol describes the connection of a whole protocol diagram with the embedded nested protocol.

2.2.9.2 Notation

The input and output parameters for the threads of interaction of a nested protocol are shown as a tall thin rectangle (like a thread of interaction) which is drawn over the top line and bottom line of the nested protocol rectangle, respectively.

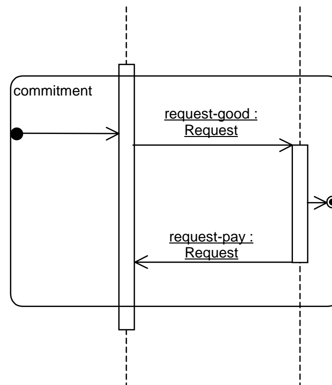
The input and output message parameters are shown by arrows starting with a small solid filled circle and arrows ending at a circle surrounding a small solid filled circle (a bull’s eye).

2.2.9.3 Presentation Options

The message arrows can be marked like usual messages. In this context the predecessor denotes the number of the input / output parameter.

The input / output thread of interaction can be marked with natural numbers to define the exact number of the parameter.

2.2.9.4 Example



2.2.9.5 Mapping

see Mapping 2.2.1.5

2.2.10 Parameterized Protocol

Adapted from [UML, v1.1 section 5.11]

2.2.10.1 Semantics

A parameterized protocol is the description for a protocol with one or more unbound formal parameters. It therefore defines a family of protocols, each protocol specified by binding the parameters to actual values. Typically the parameters represent agent roles, constraints, instances of communicative acts and nested protocols. The parameters used within the parameterized protocol are defined in terms of the formal parameters so they are become bound when the parameterized protocol itself is bound to the actual values.

A parameterized protocol is not a directly-usable protocol because it has unbound parameters. Its parameters must be bound to actual values to create a bound form that is a protocol.

2.2.10.2 Notation

A small dashed rectangle is superimposed on the upper right-hand corner of the rectangle with rounded corners like defining a nested protocol. The dashed rectangle contains a parameter list of formal parameters for the protocol. The list must not be empty, although it might be suppressed in the presentation. The name of the parameterized protocol is written as a string in the upper left-hand corner.

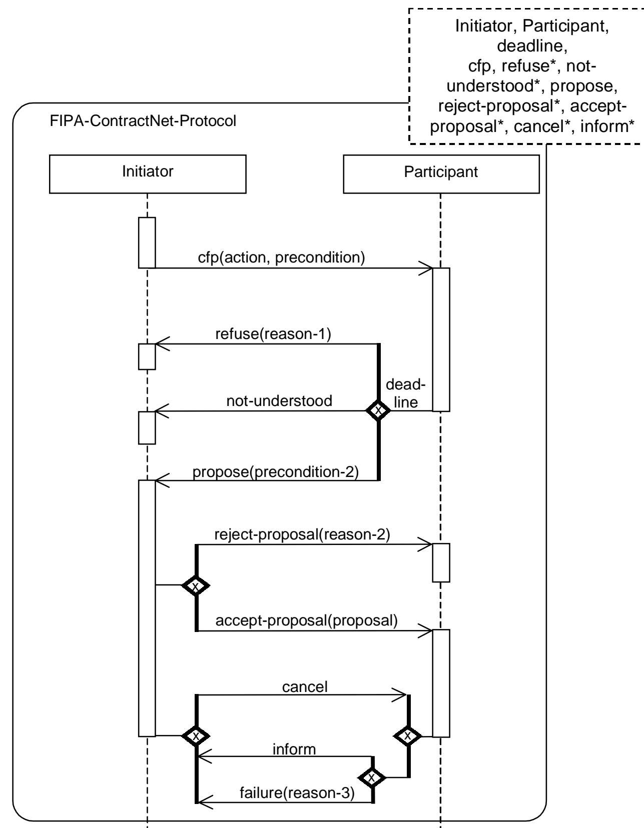
The parameter list is a comma-separated list of arguments (formal parameters) defined by identifier, like names for agentroles, constraint expressions, communicative acts or nested protocol names.

2.2.10.3 Presentation Options

The input / output parameters like messages and threads of interactions can be used and defined as for nested protocols.

Communicative act can be marked with an asterix, denoting different kinds of messages which can alternatively be sent in this context.

2.2.10.4 Example



2.2.10.5 Mapping

The addition of the parameterized dashed box to a protocol causes the addition of the parameter names in the parameter list as ModelElement within the Namespace of the ModelElement corresponding to the protocol. Each of the parameter ModelElements has to parameterized associated to the Namespace.

2.2.11 Bound Element

Adapted from [UML, v1.1 section 5.12]

2.2.11.1 Semantics

A parameterized protocol diagram cannot be used directly in an ordinary interaction description, because it has free parameters that are not meaningful outside of a scope that declares the parameter. To be used, a formal parameter of a parameterized protocol must be bound to actual values. The actual value for each parameter is an expression defined within the scope of use. If the referencing scope is itself a parameterized protocol, then the parameters of the referencing parameterized protocol can be used as actual values in binding the referenced parameterized protocol, but the parameter names in the two templates cannot be assumed to correspond, because they have no scope outside of their respective templates. We can assume without less of generality that the parameter names of the different parameterized protocols are different.

2.2.11.2 Notation

A bound element is indicated by a text syntax in the name string of an element, as follows:

parameterized-protocol-name ‘<’ value-list ‘>’

where value-list is a comma-delimited non-empty list of value expressions and parametrized-protocol-name is identical to the name of the parameterized protocol.

The number and types of the values must match the number and types of the parameterized protocol formal parameters for the parameterized protocol of the given name.

The bound element name may be used anywhere that protocol of the parameterized kind could be used.

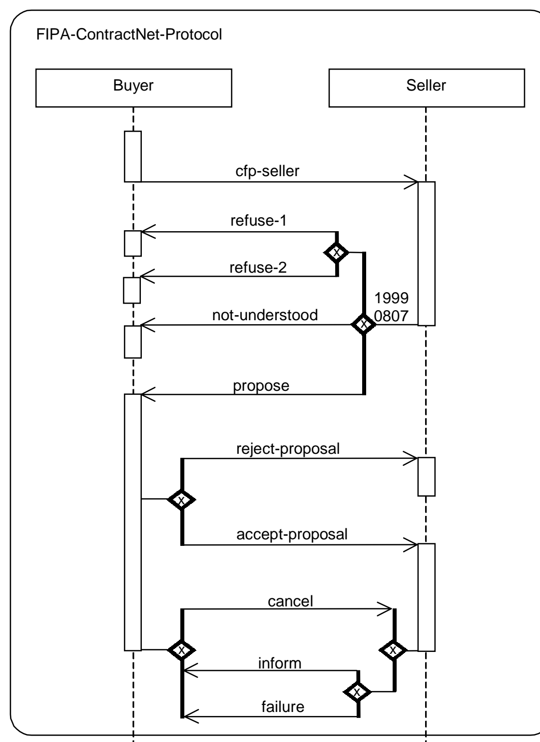
2.2.11.3 Presentation Options

none.

2.2.11.4 Example

```
FIPA-ContractNet-Protocol <
Buyer, Seller
19990807120000
cfp-seller, refuse-1, refuse-2, not-understood, propose, reject-proposal,
accept-proposal, cancel, inform, failure
>
```

The instantiated protocol looks like



2.2.11.5 Mapping

The use of the bound element syntax for the name of a symbol maps into a Binding dependency between the dependent ModelElement corresponding to the bound element symbol and the provider ModelElement whose name matches the name part of the bound element without the arguments. If the name does not match a parameterized protocol or if the number of arguments in the bound element does not match the number of formal parameters in the parameterized protocol, then the model is ill formed. Each argument in the bound element maps into a ModelElement bearing a templateArgument association to the Namespace of the bound element. The Binding relationship bears the list of actual argument values.

2.3 Defined protocols

2.3.1 Failure to understand a response during a protocol

Whilst not, strictly speaking, a protocol, by convention an agent which is expecting a certain set of responses in a protocol, and which receives another message not in that set, should respond with a *not-understood* message.

To guard against the possibility of infinite message loops, it is not permissible to respond to a *not-understood* message with another *not-understood* message!

2.3.2 FIPA-request Protocol

The FIPA-request protocol simply allows one agent to request another to perform some action, and the receiving agent to perform the action or reply, in some way, that it cannot.

The representation of this protocol would be as follows:

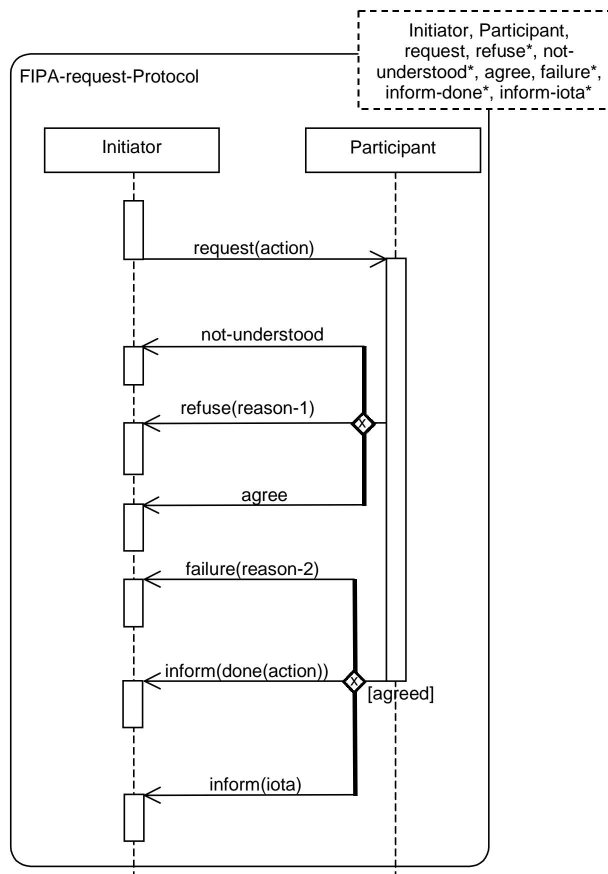


Figure 1 — FIPA-Request Protocol

2.3.3 FIPA-query Protocol

In the FIPA-query protocol, the receiving agent is requested to perform some kind of inform action. Requesting to inform is a query, and there are two query-acts: query-if and query-ref. Either act may be used to initiate this protocol. If the protocol is initiated by a query-if act, it the responder will plan to return the answer to the query with a normal inform act. If initiated by query-ref, it will instead be an inform-ref

that is planned. Note that, since inform-ref is a macro act, it will in fact be an inform act that is in fact carried out by the responder.

The representation of this protocol would be as follows:

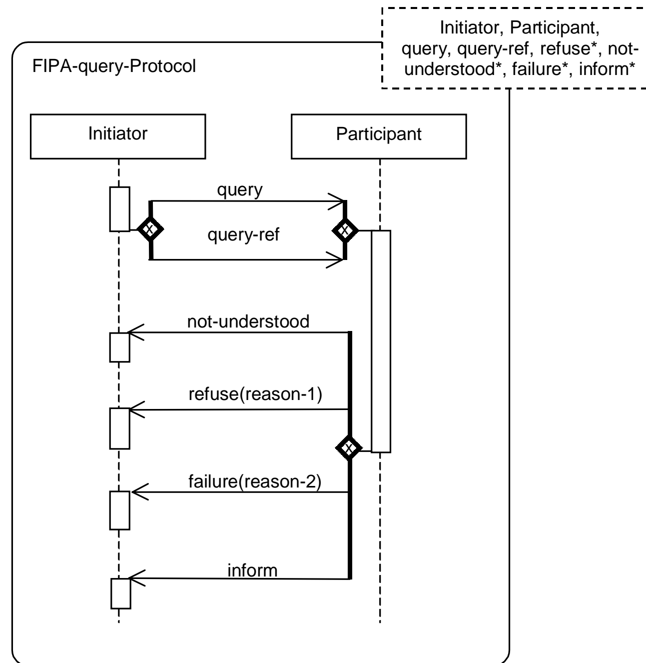


Figure 2 — FIPA-Query Protocol

2.3.4 FIPA-request-when Protocol

The FIPA-request-when protocol is simply an expression of the full intended meaning of the request-when action. The requesting agent uses the *request-when* action to seek from the requested agent that it performs some action in the future once a given precondition becomes true. If the requested agent understands the request and does not refuse, it will wait until the precondition occurs then perform the action, after which it will notify the requester that the action has been performed. Note that this protocol is somewhat redundant in the case that the action requested involves notifying the requesting agent anyway. If it subsequently becomes impossible for the requested agent to perform the action, it will send a refuse request to the original requestor.

The representation of this protocol would be as follows:

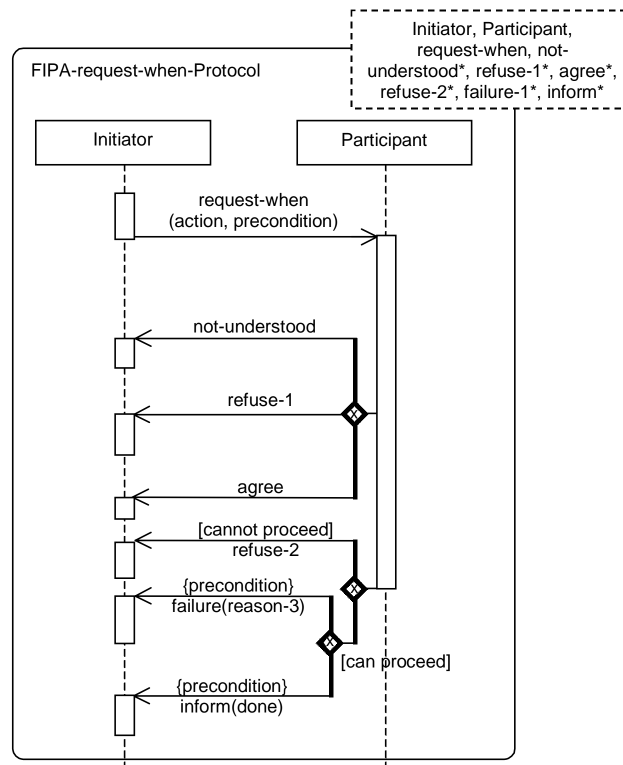


Figure 3 — FIPA-request-when protocol

2.3.5 FIPA-contract-net Protocol

This section presents a version of the widely used *Contract Net Protocol*, originally developed by Smith and Davis [Smith & Davis 80]. FIPA-Contract-Net is a minor modification of the original contract net protocol in that it adds *rejection* and *confirmation* communicative acts. In the contract net protocol, one agent takes the role of *manager*. The manager wishes to have some task performed by one or more other agents, and further wishes to optimize a function that characterizes the task. This characteristic is commonly expressed as the *price*, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc.

The manager solicits *proposals* from other agents by issuing a *call for proposals*, which specifies the task and any conditions the manager is placing upon the execution of the task. Agents receiving the call for proposals are viewed as potential *contractors*, and are able to generate proposals to perform the task as *propose* acts. The contractor's proposal includes the preconditions that the contractor is setting out for the task, which may be the price, time when the task will be done, etc. Alternatively, the contractor may *refuse* to propose. Once the manager receives back replies from all of the contractors, it evaluates the proposals and makes its choice of which agents will perform the task. One, several, or no agents may be chosen. The agents of the selected proposal(s) will be sent an acceptance message, the others will receive a notice of rejection. The proposals are assumed to be binding on the contractor, so that once the manager accepts the proposal the contractor acquires a commitment to perform the task. Once the contractor has completed the task, it sends a completion message to the manager.

Note that the protocol requires the manager to know when it has received all replies. In the case that a contractor fails to reply with either a *propose* or a *refuse*, the manager may potentially be left waiting indefinitely. To guard against this, the *cfp* includes a deadline by which replies should be received by the manager. Proposals received after the deadline are automatically rejected, with the given reason that the proposal was late.

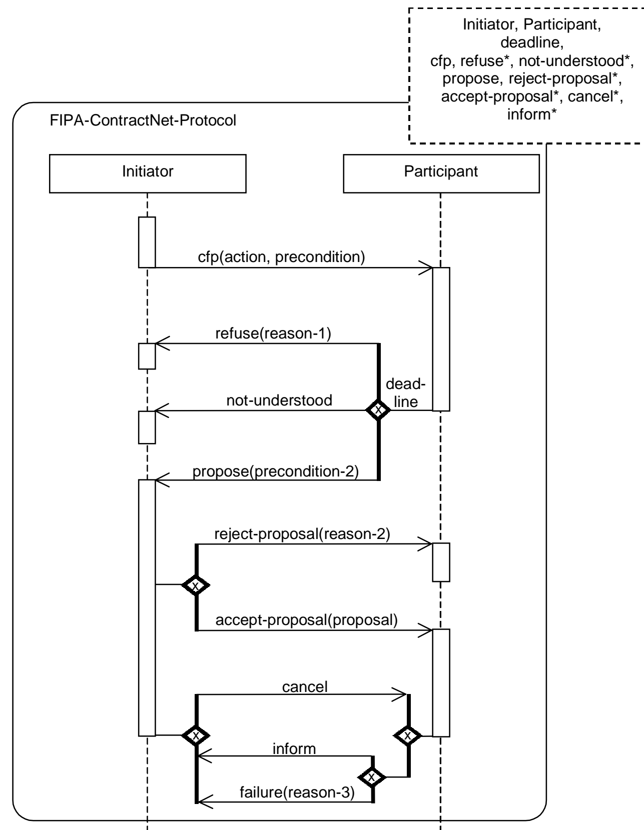


Figure 4 — FIPA-Contract-Net

2.3.6 FIPA-Iterated-Contract-Net Protocol

The *iterated contract net* protocol is an extension of the basic contract net as described above. It differs from the basic version of the contract net by allowing multi-round iterative bidding. As above, the manager issues the initial call for proposals with the *cfp* act. The contractors then answer with their bids as *propose* acts. The manager may then accept one or more of the bids, rejecting the others, or may iterate the process by issuing a revised cfp. The intent is that the manager seeks to get better bids from the contractors by modifying the call and requesting new (equivalently, revised) bids. The process terminates when the manager refuses all proposals and does not issue a new call, accepts one or more of the bids, or the contractors all refuse to bid.

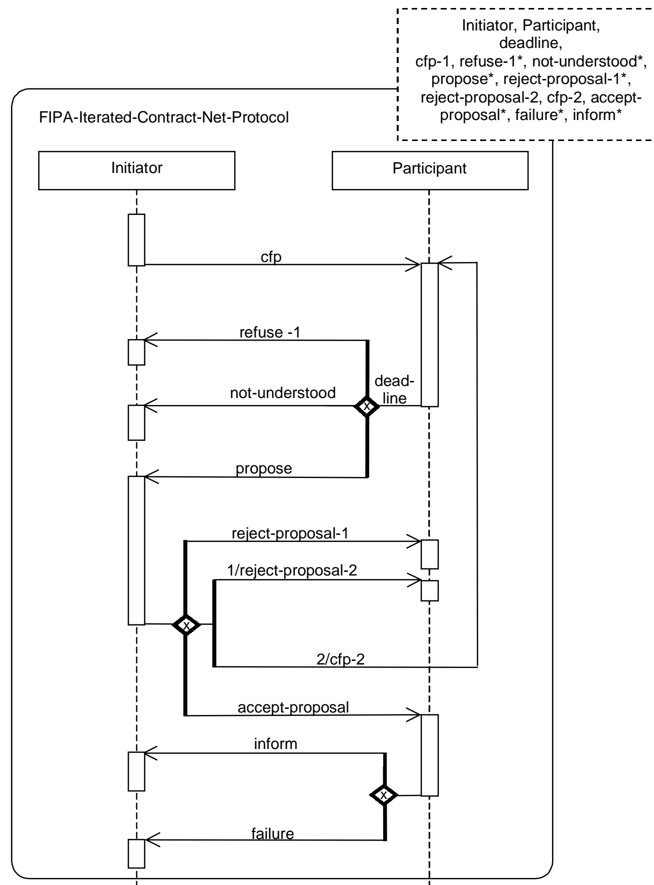


Figure 5 — FIPA-iterated-contract-net protocol

2.3.7 FIPA-Auction-English Protocol

In the English Auction, the auctioneer seeks to find the market price of a good by initially proposing a price below that of the supposed market value, and then gradually raising the price. Each time the price is announced, the auctioneer waits to see if any buyers will signal their willingness to pay the proposed price. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with an incremented price. The auction continues until no buyers are prepared to pay the proposed price, at which point the auction ends. If the last price that was accepted by a buyer exceeds the auctioneer's (privately known) reservation price, the good is sold to that buyer for the agreed price. If the last accepted price is less than the reservation price, the good is not sold.

In the following protocol diagram, the auctioneer's calls, expressed as the general *cfp* act, are multicast to all participants in the auction. For simplicity, only one instance of the message is portrayed. Note also that in a physical auction, the presence of the auction participants in one room effectively means that each acceptance of a bid is simultaneously broadcast to all participants, not just the auctioneer. This may not be true in an agent marketplace, in which case it is possible for more than one agent to attempt to bid for the suggested price. Even though the auction will continue for as long as there is at least one bidder, the agents will need to know whether their bid (represented by the *propose* act) has been accepted. Hence the appearance in the protocol of *accept-proposal* and *reject-proposal* messages, despite this being implicit in the English Auction process that is being modelled.

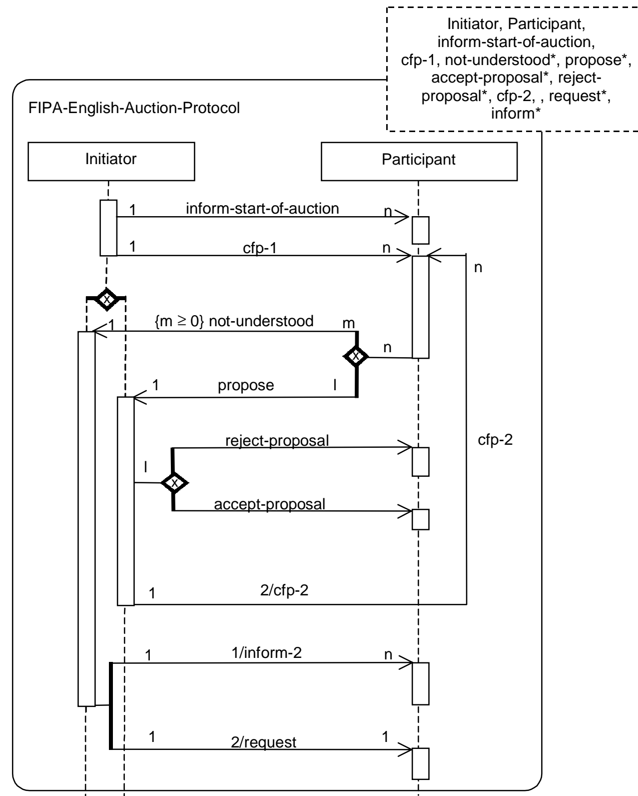


Figure 6 — FIPA-auction-english protocol

2.3.8 FIPA-Auction-Dutch Protocol

In what is commonly called the *Dutch Auction*, the auctioneer attempts to find the market price for a good by starting bidding at a price much higher than the expected market value, then progressively reducing the price until one of the buyers accepts the price. The rate of reduction of the price is up to the auctioneer, and the auctioneer usually has a *reserve price* below which it will not go. If the auction reduces the price to the reserve price with no buyers, the auction terminates.

The term "Dutch Auction" derives from the flower markets in Holland, where this is the dominant means of determining the market value of quantities of (typically) cut flowers. In modelling the actual Dutch flower auction (and indeed in some other markets), some additional complexities occur. First, the good may be split: for example the auctioneer may be selling five boxes of tulips at price *x*, and a buyer may step in and purchase only three of the boxes. The auction then continues, with a price at the next increment below *x*, until the rest of the good is sold or the reserve price met. Such partial sales of goods are only present in some markets; in others the purchaser must bid to buy the entire good. Secondly, the flower market mechanism is set up to ensure that there is no contention amongst buyers, by preventing any other bids once a single bid has been made for a good. Offers and bids are binding, so there is no protocol for accepting or rejecting a bid. In the agent case, it is not possible to assume, and too restrictive to require, that such conditions apply. Thus it is quite possible that two or more bids are received by the auctioneer for the same good. The protocol below thus allows for a bid to be rejected. This is intended only to be used in the case of multiple, competing, simultaneous bids. It is outside the scope of this specification to pre-specify any particular mechanism for resolving this conflict. In the general case, the agents should make no assumptions beyond "first come, first served". In any given domain, other rules may apply.

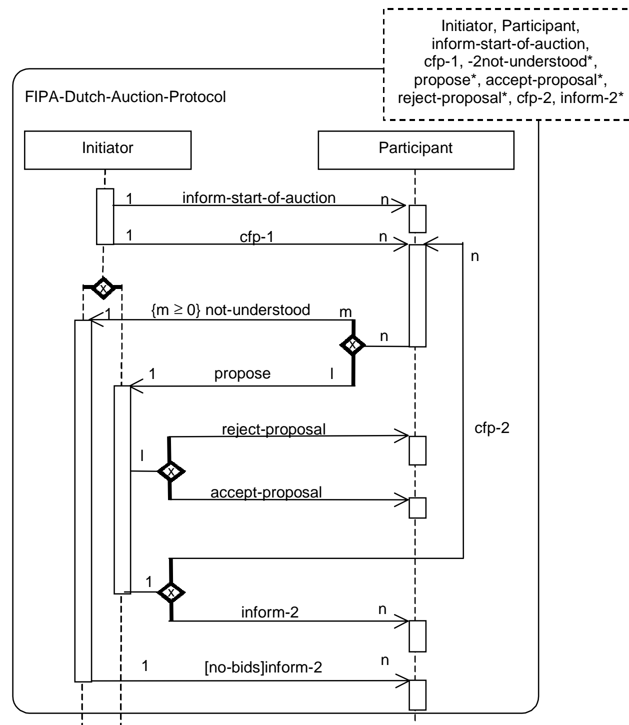


Figure 7 — FIPA-auction-dutch protocol

Acknowledgement

Thanks to James J. Odell, OMG member and UML guru, and the FIPA TCC for very helpful suggestions and discussions on the representation and formalism on protocol diagrams.

References

[Booch 94] Booch, G.: *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, 1994.

[Booch 95] Booch, G.: *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1995.

[UML] http://www.rational.com/uml and http://www.omg.org/

[Coad, Yourdon 91] Coad, P. Yourdon, E. : *Object-Oriented Analysis*, Yourdon, 1991 and *Object-Oriented Design*, Yourdon, 1991.

[Jacobson et al. 94] Jacobson, I., Christerson M., Jonsson, P., Övergaard, G.: Object-oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1994.

[Martin, Odell 98] Martin, J., Odell, J. J.,: *Object-Oriented Methods: A Foundation (UML)*. Prentice Hall, Englewood Cliffs, NJ, 1998.

[Martin, Odell 94] Martin, J., Odell, J. J.,: *Object-Oriented Methods: Pragmatic Considerations*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[Rumbaugh et al. 91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F.,Lorenzen, W.: *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[Shlaer, Mellor 89] Shlaer, S., Mellor, S. J.: *Object-Oriented System Analysis: Modeling the World in Data*. Yourdon, 1989.

[Shlaer, Mellor 91] Shlaer, S., Mellor, S. J.: *Object-Lifecycles: Modeling the World in States*. Yourdon, 1989.