

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA Modeling: Interaction Diagrams

Document title	FIPA Modeling: Interaction Diagrams		
Document number	TBA	Document source	FIPA TC Modeling
Document status	Preliminary	Date of this status	2003/mm/dd (TBD)
Supersedes			
Contact	M.P.Huget@csc.liv.ac.uk or auml@jamesodell.com		
Change history			
2003/mm/dd	First proposal		

Working Draft

Version 2003-07-02

© 2003 Foundation for Intelligent Physical Agents - <http://www.fipa.org/>

Geneva, Switzerland

Notice

Use of the technologies described in this specification may infringe patents, copyrights or other intellectual property rights of FIPA Members and non-members. Nothing in this specification should be construed as granting permission to use any of the technologies described. Anyone planning to make use of technology covered by the intellectual property rights of others should first obtain permission from the holder(s) of the rights. FIPA strongly encourages anyone implementing any part of this specification to determine first whether part(s) sought to be implemented are covered by the intellectual property of others, and, if so, to obtain appropriate licenses or other permission from the holder(s) of such intellectual property prior to implementation. This specification is subject to change without notice. Neither FIPA nor any of its Members accept any responsibility whatsoever for damages or liability, direct or consequential, which may result from the use of this specification.

Foreword

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

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

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

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

Contents

1	Scope	1
2	Sequence Diagram	2
2.1	Protocol Frame	Error! Bookmark not defined.
2.1.1	Description	2
2.1.2	Notation	2
2.1.3	Constraints	5
2.1.4	Semantics	6
2.1.5	Examples	6
2.2	Lifelines	6
2.2.1	Description	6
2.2.2	Notation	6
2.2.3	Constraints	8
2.2.4	Semantics	9
2.2.5	Examples	9
2.3	Messages	9
2.3.1	Description	9
2.3.2	Notation	10
2.3.3	Constraints	12
2.3.4	Semantics	12
2.3.5	Examples	12
2.4	Constraints	13
2.4.1	Description	13
2.4.2	Notation	13
2.4.3	Constraints	14
2.4.4	Semantics	14
2.4.5	Examples	14
2.5	Timing Constraints	15
2.5.1	Description	15
2.5.2	Notation	15
2.5.3	Constraints	16
2.5.4	Semantics	16
2.5.5	Examples	16
2.6	Splitting/Merging Paths	16
2.6.1	Description	16
2.6.2	Notation	18
2.6.3	Constraints	20
2.6.4	Semantics	20
2.6.5	Examples	20
2.7	Protocol Interactions	20
2.7.1	Description	20
2.7.2	Notation	20
2.7.3	Constraints	20
2.7.4	Semantics	20
2.7.5	Examples	20
2.8	Interaction Termination	20
2.8.1	Description	20
2.8.2	Notation	21
2.8.3	Constraints	21
2.8.4	Semantics	21

2.8.5	Examples	21
2.9	Protocol Combination	22
2.9.1	Description	22
2.9.2	Notation	22
2.9.3	Constraints	22
2.9.4	Semantics	22
2.9.5	Examples	23
2.10	Actions	23
2.10.1	Description	23
2.10.2	Notation	23
2.10.3	Constraints	23
2.10.4	Semantics	23
2.10.5	Examples	23
2.11	Protocol Templates	23
2.11.1	Description	23
2.11.2	Notation	24
2.11.3	Constraints	24
2.11.4	Semantics	24
2.11.5	Examples	24
2.12	Graphical Notation Summary	24
2.13	Textual Notation Summary	28
2.14	FIPA Interaction Protocols	28
2.14.1	Request Interaction Protocol	28
2.14.2	Query Interaction Protocol	29
2.14.3	Request When Interaction Protocol	30
2.14.4	Contract Net Interaction Protocol	31
2.14.5	Iterated Contract Net Interaction Protocol	32
2.14.6	English Auction Interaction Protocol	34
2.14.7	Subscribe Interaction Protocol	36
2.14.8	Propose Interaction Protocol	36
3	Interaction Overview Diagram	37
4	Communication Diagram	38
4.1	Communication Diagram Expressions	39
5	Timing Diagram	40
6	References	41

1 Scope

This document contains specifications for the FIPA AUML Interaction Diagrams. In UML, Interaction Diagrams are a “family” of diagrams consisting of:

- **Sequence Diagram** - A diagram that depicts an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding event occurrences on the lifelines. Unlike a communication diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible scenarios) and in an instance form (describes one actual scenario). Sequence diagrams and collaboration diagrams express similar information, but show it in different ways.
- **Interaction Overview Diagram** - A diagram that depicts interactions through a variant of activity diagrams in a way that promotes overview of the control flow. It focuses on the overview of the flow of control where each node can be an interaction diagram.
- **Communication Diagram** (formerly called Collaboration Diagram) - A diagram that focuses on object relationships where the message passing is central. The sequencing of messages is given through a sequence numbering scheme. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways
- **Timing Diagram** - An interaction diagram that shows the change in state or condition of a lifeline (representing a Classifier Instance or Classifier Role) over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli.

Note: In previous Agent UML, protocols are rendered through protocol diagrams, which are now called sequence diagrams in this new version in order to be conformed to UML 2 Interaction specifications.

The primary author of this document is Marc-Philippe Huget. Supporting contributors include James Odell, Øystein Haugen, Mariam “Misty” Nodine, Stephen Cranefield, Renato Levy, and Lin Padgham.

2 Sequence Diagram

Agent UML Sequence Diagrams were initially adopted by FIPA to express agent interaction protocols. Sequence diagrams are defined as follows in UML: "A diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in the interaction and the sequence of messages exchanged. Unlike a collaboration diagram, a sequence diagram includes time sequences but does not include object relationships." Since Agent UML considers agents and not objects, one must read agents instead of objects in the previous definition. Sequence diagrams in multiagent systems are diagrams that express the exchange of messages through protocols.

Two parts can be considered in sequence diagrams: a frame, which delimits the sequence diagram (see Section **Error! Reference source not found.**) and the message flow between roles through a set of lifelines and messages (see Sections 2.2 and 2.3).

Sequence diagrams have two dimensions: 1) the vertical dimension represents the time ordering and 2) the horizontal dimension represents different roles or agents playing specific roles. Messages in sequence diagrams are ordered according to a time axis. This time axis is usually not rendered on diagrams but it goes according to the vertical dimension from top to bottom. Sequence diagrams do not use sequence numbers like collaboration diagrams to represent the message ordering. Message ordering is performed by the time axis. This time axis is followed as long as designers do not use Continuations (see Section 2.6) that allow interactions to leap backward or forward in the diagram.

As stated in [Odell2000] and [Bauer 2001], agents are described through their roles in the interaction. Sequence diagrams mainly describe roles in the interaction but it is also possible to depict a specific agent playing a given role as stated in Section 2.2.

2.1 Interaction

2.1.1 Description

The frame in UML 2.0 Interactions [UML2, p. 374] allows bringing together all the elements that intervene in an interaction protocol. It means the message flow as well as the roles involved in the interaction, the timing constraints, and the constraints on this interaction.

The purpose of this frame is to encapsulate all the elements used in the interaction protocol as a unit. As a consequence, elements located outside the frame are external to this protocol. Even if protocols are seen as a closed unit, they are able to interact with the environment (agents and other protocols) through the notion of Gates proposed in UML 2 [UML2, p. 377] as depicted in Section 2.7. It is then possible to send or receive messages from the environment. This environment can be other protocols allowing interaction between protocols.

2.1.2 Notation

An interaction is rendered as a solid-outline rectangle as shown on Figure 1.

The keyword **sd** followed by the interaction protocol name is in a "snipped-corner" pentagon in the upper-left corner of the rectangle. This keyword stands for sequence diagram and comes from UML 2. Contrary to UML 2.0, the parameters are not enclosed in the "snipped-corner" pentagon but outside the sequence diagram in a comment linked to the sequence diagram. The parameters are prefixed by the stereotype `<<parameters>>`. Such approach is to reduce the size of the sequence diagram in case of numerous parameters. Moreover, parameters in AUML do **not** refer to parameters in messages as shown in Figure 2 for UML Sequence Diagrams, but to parameters that correspond to the instantiation of a template as proposed in [Odell2000] (see Figure 3). On Figure 3, there are three parameters for the *English auction* protocol. These three parameters correspond to the instantiation of the *English auction* protocol to a specific case: selling a car at a certain price and before a certain date. A protocol template is distinguished by the use of the stereotype `<<template>>` before the protocol name as shown on Figure 4. Discussions about the content of protocol templates are given in Section 2.11. Parameters in templates instantiations are given in a comment linked to the Interaction. In case, some confusion may arise from several instantiation values (parameters) that could be applied to the same template parameters,

designers must prefix the instantiation value (parameter) by the template parameter in the comment. The format is
template parameter = instantiation value.

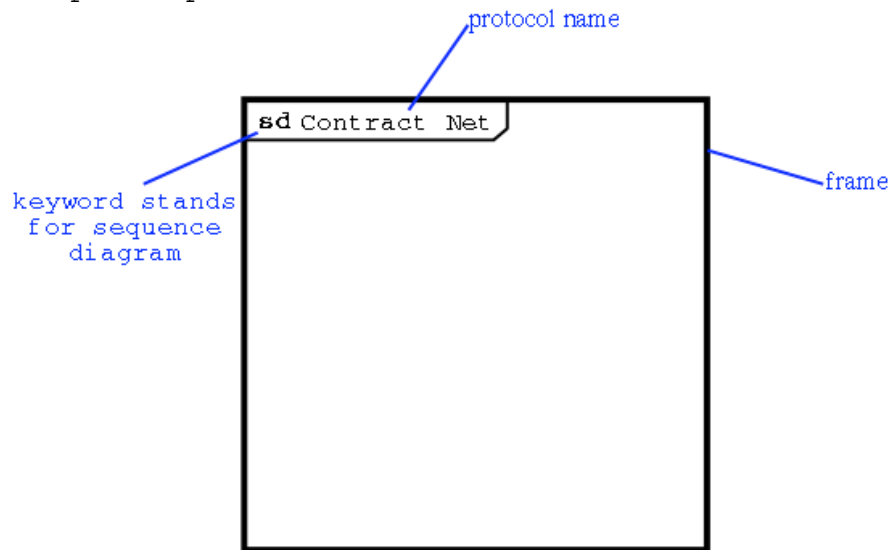


Figure 1: Agent UML Frame Notation

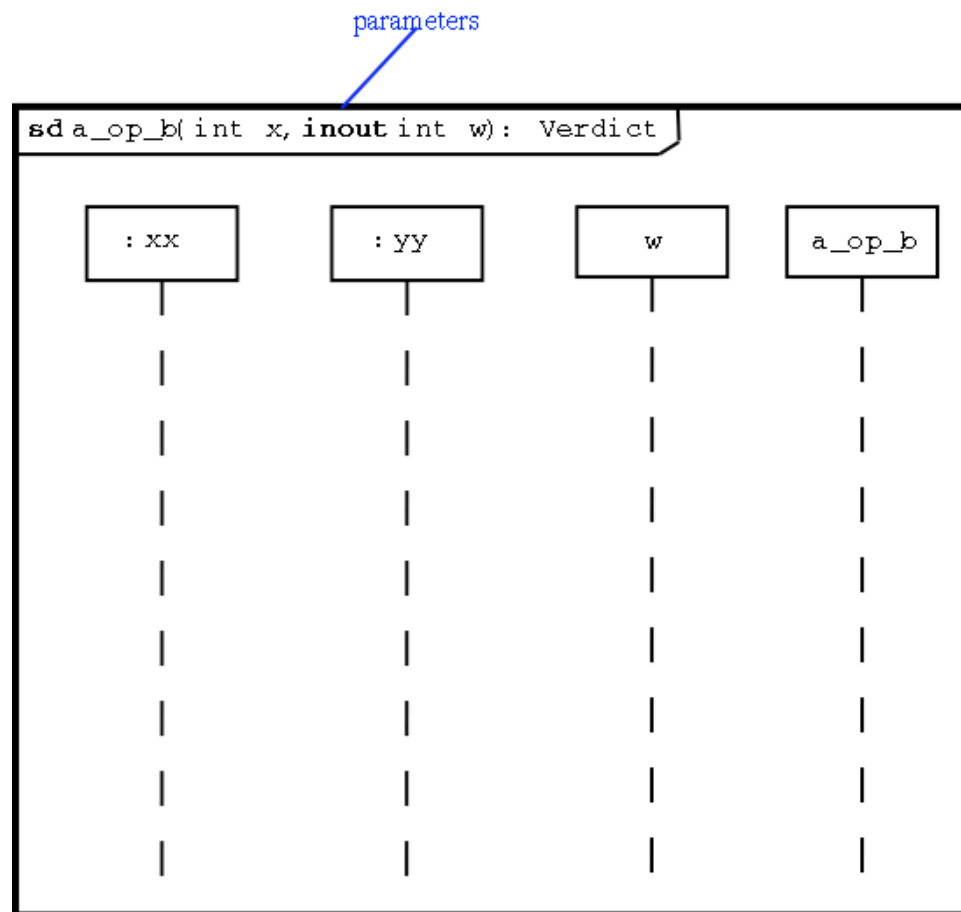


Figure 2: UML 2.0 Frame Notation

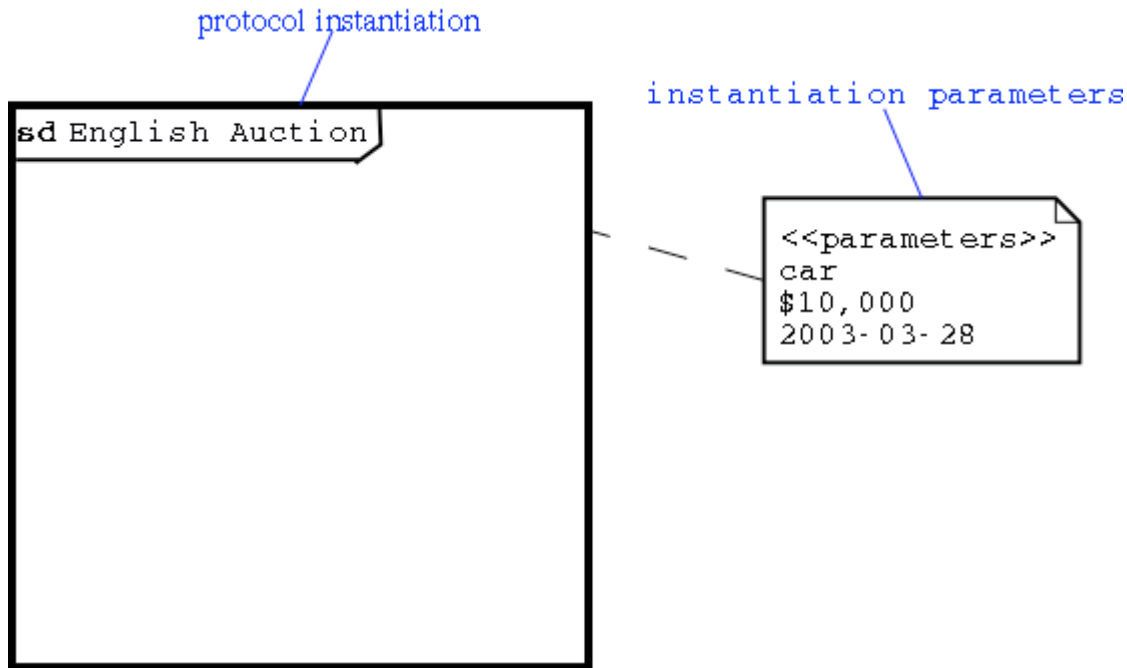


Figure 3: Agent UML Frame Notation with Parameters

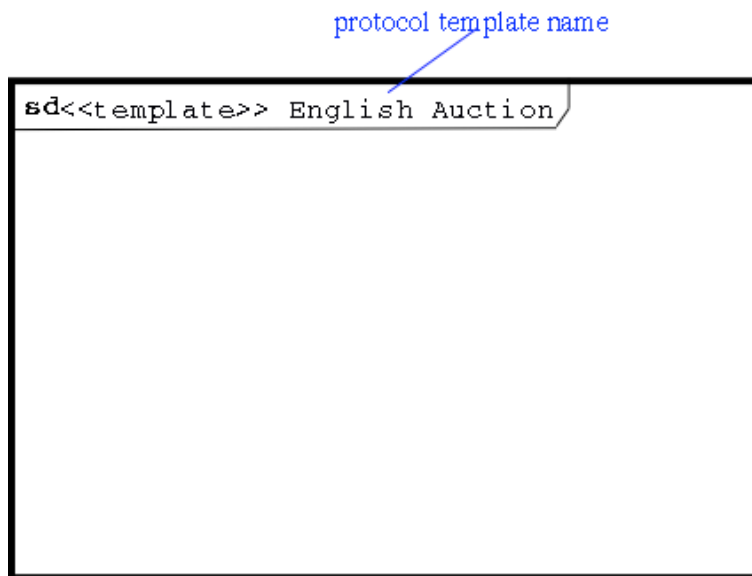


Figure 4: Agent UML Protocol Template Notation

Protocol designers need to know as well which ontologies, content language and agent communication language are used in this protocol. The ontology parameter is a text field that may be the name of an ontology or an expression that represents a set of ontologies together with other information such as namespace prefix definitions. This information is stored as parameters as shown in Figure 5. Three reserved keywords are used for parameters: `ontology`, `ACL` and `CL`. They refer respectively to the set of ontologies, the agent communication language and the content language for the message used in the sequence diagram. These keywords are mandatory when writing these values in a comment as shown on Figure 5. Designers can omit this information as long as there is no misunderstanding. We deliberately omit this information in the next figures to lighten them.

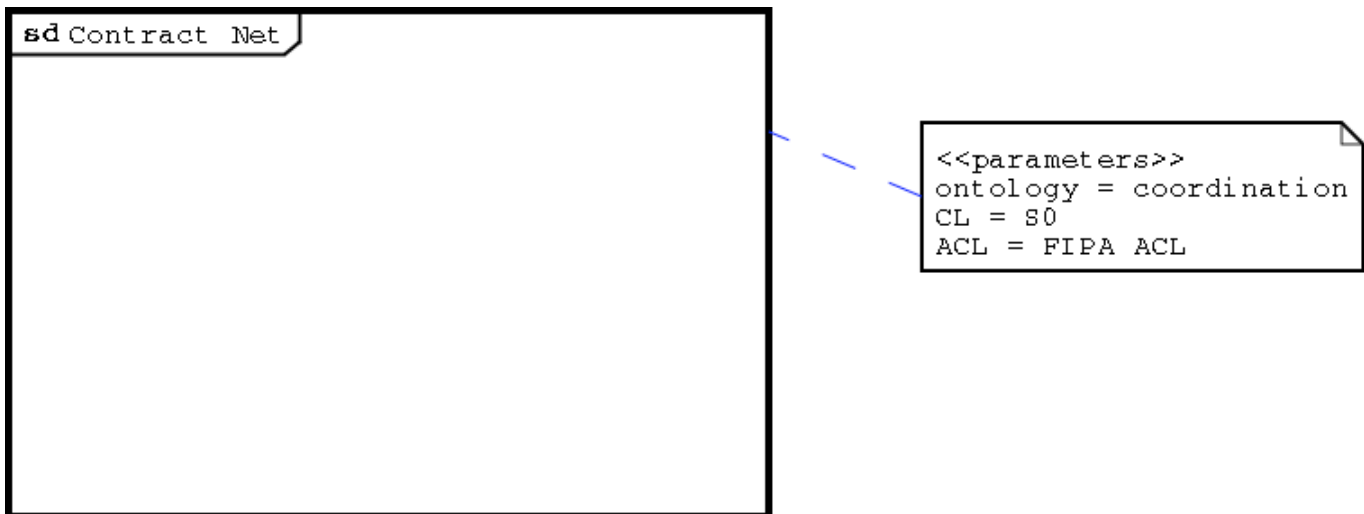


Figure 5: Agent UML Frame Notation with Additional Information

Parameters for the sequence diagrams are declared near the diagram top before the lifelines as depicted on Figure 6. Parameters correspond to local attributes to a sequence diagram. The local attribute on Figure 6 gives the name of the Participant awarded for the task.

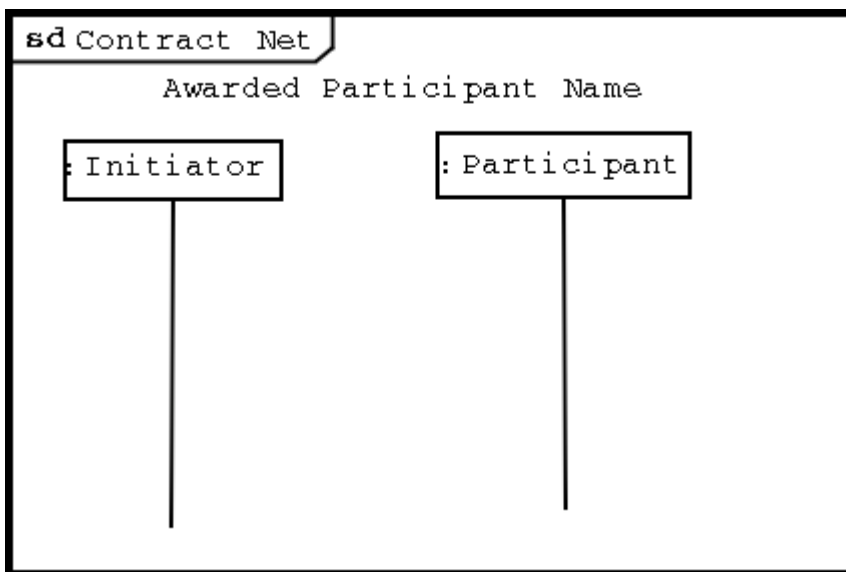


Figure 6: Agent UML Sequence Diagram with Local Attributes

2.1.3 Constraints

- Protocol names must be unique.
- A protocol name should not be used for a protocol and a template protocol.
- When a protocol is an instantiation of a protocol template, the number of values (parameters) **must** exactly match the number of parameters in the protocol template.
- If some confusion might arise because designers do not know, at design time, which values (parameters) in the instantiation refer to which parameters in the protocol template, then it is better to prefix each instantiation value (parameter) by the corresponding template parameter.
- If some confusion might arise between several parameters about their use, designers will suffixed with the context of use.
- Ontology, content language and agent communication language parameters must be prefixed respectively by the keyword ontology, CL and ACL.

2.1.4 Semantics

TBD

2.1.5 Examples

Figure 27 on page 30 gives the example of the FIPA Query Interaction Protocol [FIPA00027]. Several examples are available in Section 2.14.

2.2 Lifelines

2.2.1 Description

The use of lifelines in Agent UML is in contradiction with UML 2.0 specifications where it is written that a lifeline represents an individual participant in the interaction [UML2, p. 382]. Actually, Agent UML considers it is possible to represent several agents on the same lifeline [Odell2000]. A lifeline no necessarily represents an agent but also a role played by one or more agents. UML defines *role* as a “named set of behaviours possessed by a class or part participating in a particular context” [UML2, p. 557]. One can also add that a role is a specific set of behaviours, properties, interfaces and service descriptions, which allow distinguishing a particular role from another one. Readers are urged to read [Odell2003] for an in-depth description of roles in multiagent systems. The lifeline in sequence diagrams defines the time period during which a role exists for this interaction, represented by vertical dashed lines. When a lifeline is created for a role, this role becomes active for the protocol. This lifeline is present as long as the role remains active in the protocol.

Moreover, an agent is able to assume multiple roles during the interaction, as well as change roles (i.e., becomes an instance of a role and is removed as an instance of another). This is referred to as *multiple classification* and *dynamic classification*, respectively. This ability is depicted on sequence diagrams through a connection between the current role of the agent and its next role. Multiple roles imply that agents have several participations in the same interaction, for instance Seller and Participant. If a message is sent to a role *m* and a role *n* and an agent has both these roles, it will receive two messages, one for the role *m* and one for the role *n*. Dynamic classification refers to the ability for agents to change roles during the interaction. If at a specific moment *t*, agent *a* has the role *m* and it changes to role *n* at *t+1*, it no longer has the role *m* and has from now the role *n*. It means that at the moment *t+1*, it can receive messages for role *n* and not for the role *m*. If an agent plays multiple roles for the current interaction, it only loses the role *m*, other roles continue to receive and send messages. Multiple and dynamic classification are atomic operations. It means that it is not possible to consider that a message will be sent or received during the role modification. The role dynamics is not addressed in this diagram and is covered in a specific diagram called *role diagram*, especially the problem of constraints when moving from one role to others.

Finally, a specific role in a group can behave differently for the same role in other groups. As a consequence, designers can suffix the role by its group to make the distinction. A group is a set of agents that are related via their roles, where these relationships must form a connected graph within this group [Odell 2003].

Role cardinality is added to lifelines to depict the number of agents playing a specific role. These agents are involved in the interaction. Several options are offered to designers to represent this cardinality:

1. Designers know the exact number of agents involved in this role
2. Designers know the range. They then write the bounds. These bounds are inclusive.
3. The number of agents is not exactly known or is based on other facts. For instance, the role can represent only the agents that accept to bid on the proposal. In this case, the cardinality can be a logic formula or a condition

2.2.2 Notation

A lifeline is composed of two elements: a label depicted in a box on the top of the lifeline and a vertical dashed line anchoring to the box as shown in Figure 7. Several formats are possible for the label:

1. agent instance: role specifying an unique agent notated as *agent instance* playing a specific role. For instance, Mario:Employee
2. agent instance depicting a specific agent regardless of its role in the interaction.
3. role depicting a role regardless of the agents playing this role.

Roles are always underlined in lifeline labels.

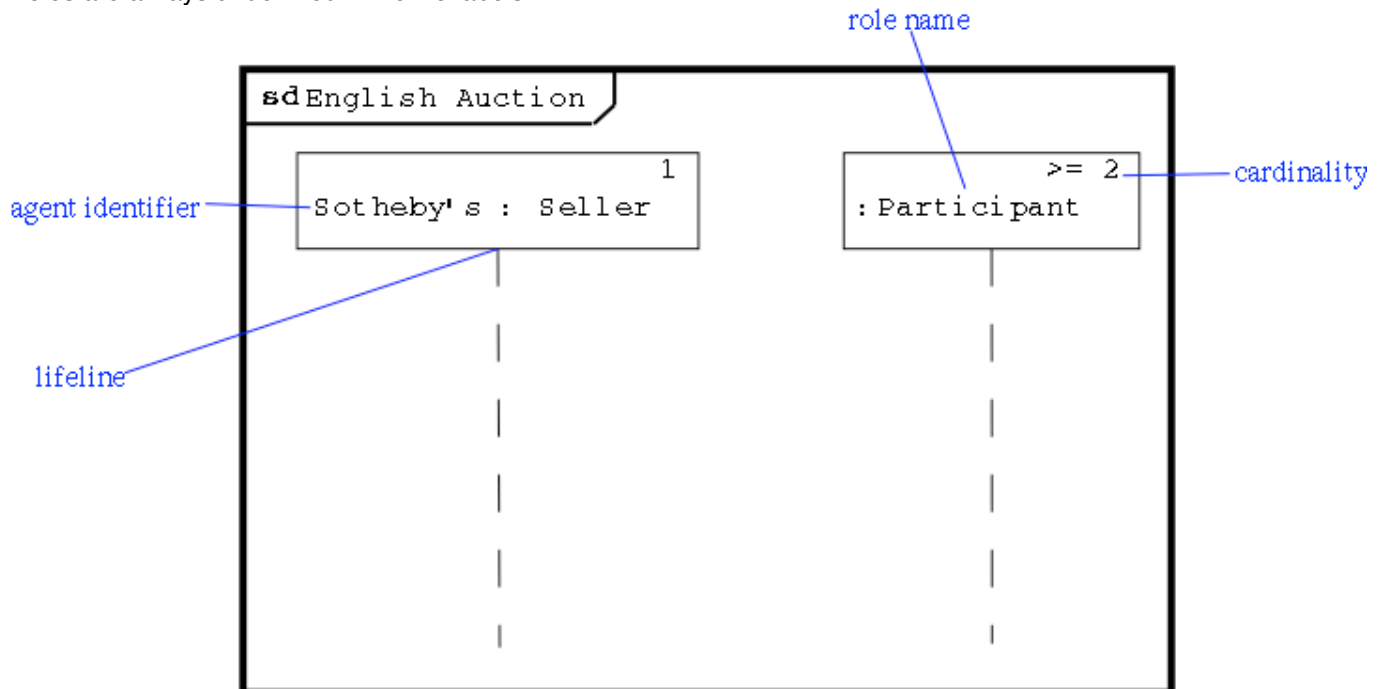


Figure 7: Agent UML Lifeline Notation

In each case given above where a role is involved, it is possible to add the group to which the role belongs. Groups are written at the far end of the label and prefixed by a slash '/'. For instance: *Smith:Sales Clerk/ACME* refers to a specific agent *Smith* playing the role of *Sales Clerk* in the group *ACME*.

The label contains as well the role cardinality. There are three formats for the role cardinality:

1. n that depicts the exact number of agents that will be in this role for this interaction
2. $m \text{ op } i \text{ op } n$ that depicts a range of values for the role cardinality. op refers to the operators $>$, $>=$, $<$, $<=$. m and n refer respectively to the lower and the upper bound for the range of values. Bounds are not written when they are implicit as shown in Figure 7 for the Participant role. It is not necessary to give the upper bound since this is infinite.
3. a condition or a logic formula depicting the number of agents

It seems better to order lifelines according to the appearance of the roles in the interaction.

As stated above, rendering a lifeline corresponds to the appearance of the role (or the agents playing this role) in the interaction, it is then important to shift the lifeline to the bottom if this role appears late in the interaction as shown in Figure 8.

Dynamic classification is rendered on sequence diagrams as a directed line from the current role to the new role. The directed line is adorned with the stereotype `<<change role>>`. For multiple classifications, agents can add new roles by a directed line to the added new role as shown on Figure 9. The directed line is adorned with the stereotype `<<add role>>`.

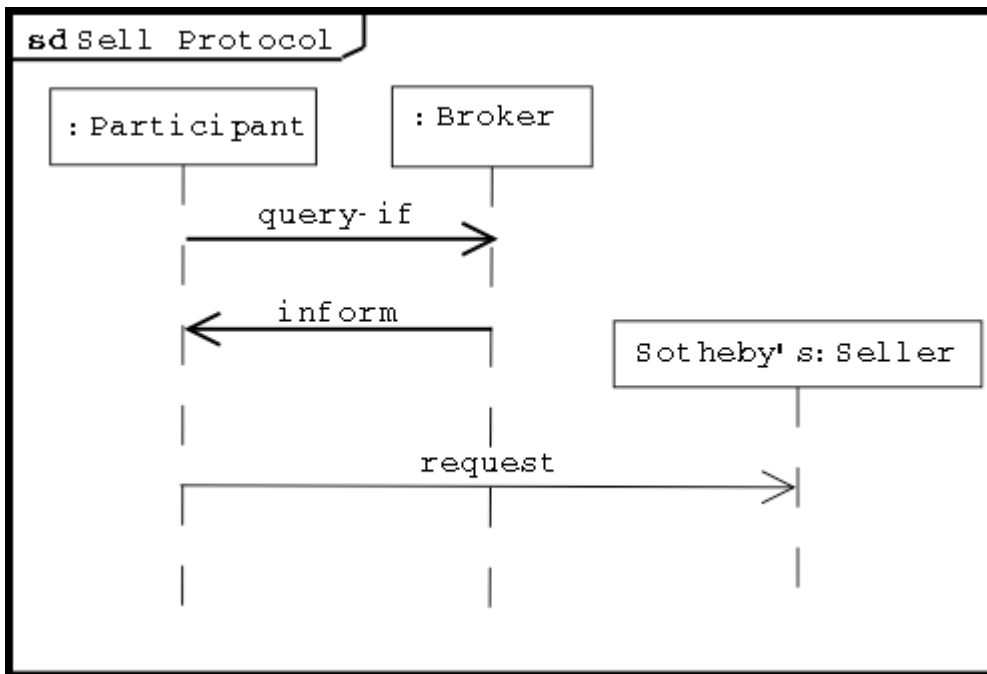


Figure 8: Sell Protocol

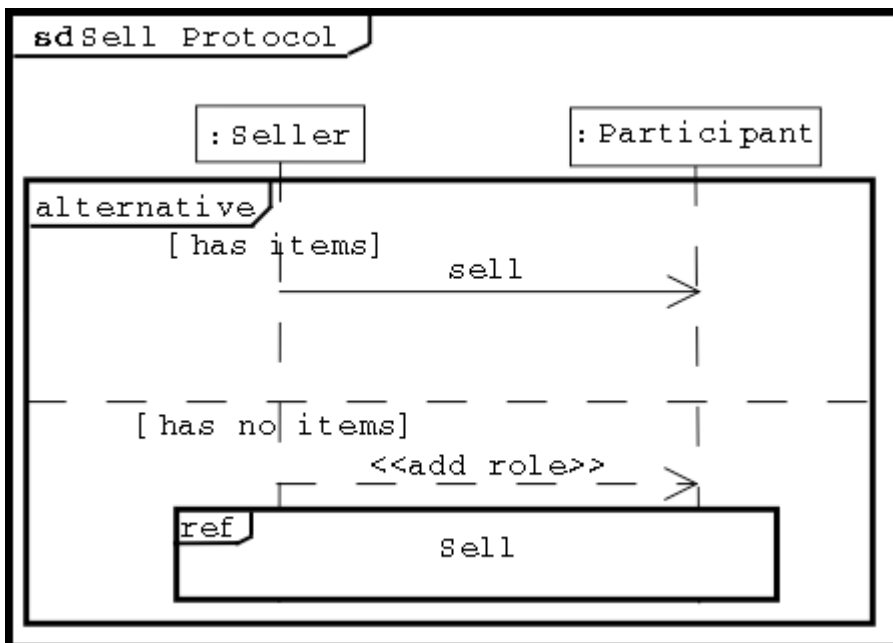


Figure 9: Multiple Role in Agent UML Sequence Diagram

2.2.3 Constraints

- A pair (role, group) without agent identifier **must** appear one and only one time on the sequence diagram
- When an agent identifier is given, it is not possible to add a participant cardinality
- A role must be underlined in the box of the lifeline
- A role cannot add or change to a lifeline where there is an agent identifier except if this is the same agent identifier for the original and the final lifeline

2.2.4 Semantics

TBD

2.2.5 Examples

Figure 33 on page 37 gives the example of the FIPA Propose Interaction Protocol [FIPA00036].

2.3 Messages

2.3.1 Description

Agents in multiagent systems communicate via messages, which express the information they want other agents consider. Here is an example of FIPA ACL message:

(inform

```
:sender (agent-identifier :name agent1)
:receiver (set (agent-identifier :name agent2))
:content "((price good 100))"
:language fipa-sl
:ontology auction
```

)

All this information is present on sequence diagrams at different locations: the sender is at the origin of the directed line between the sender and the receiver; the receiver is at the destination of the directed line between the sender and the receiver. The communicative act is adorned on the directed line. The language and the ontology are written in a comment linked to the sequence diagram. For sake of brevity and generality, the content is not included in previous figures. The content can be added right after the communicative act on the directed line corresponding to the message.

In the context of UML 2.0 and Agent UML, a message defines a particular communication between two lifelines in the sequence diagrams. Senders and receivers of a message can be on the same lifeline or not. Two situations have to be considered when the sender lifeline is the same than the receiver lifeline:

1. The sender wants to receive the message as well
2. The sender wants to be omitted from the set of recipients.

These two different situations will be distinguished on the message notation.

Constraints on the message can be added to affect how the message is used.

Agents can use either a synchronous communication as frequently encountered in distributed systems or an asynchronous communication that is generally the one used in agent communication. An asynchronous message means that agents send the message without yielding control. A synchronous message means that agents send the message with yielding of the thread of control (wait semantics), i.e. the agent role responsible of this synchronous message sending waits until an answer message is received and nothing else can be processed. The agent other roles continue to send and receive messages for this protocol or other protocols.

The content of the message is given above the arrow. There is no formalization to depict message content and depends greatly on the agent communication language used

Since lifelines may label roles and as a consequence, represent multiples agents, it is important to depict the message cardinality on sequence diagrams. The message cardinality follows the role cardinality and offers three formats:

1. n that depicts the exact number of agents that will be in this role for this interaction
2. $m \text{ op } i \text{ op } n$ that depicts a range of values for the role cardinality. op refers to the operators $>$, $>=$, $<$, $<=$. m and n refer respectively to the lower and the upper bound for the range of values. Bounds are not written when they are implicit as shown in Figure 6 for the Participant role. It is not necessary to give the upper bound since this is the infinite.
3. a condition or a logic formula

Message cardinality is given for both senders and recipients. Cardinalities are omitted when they are obvious. For instance, the cardinality is 1 or the infinite.

2.3.2 Notation

A message is shown as a directed line from the sender role to the receiver role. The form of the arrowhead reflects properties of the message (see Figure 10): Asynchronous messages have an open arrow head, Synchronous messages have a filled arrow head. If messages are addressed to a specific agent on lifelines (in case the lifeline depicts a role with several agents in this role), the agent name is given on the line near the receiver lifeline as shown on Figure 11.

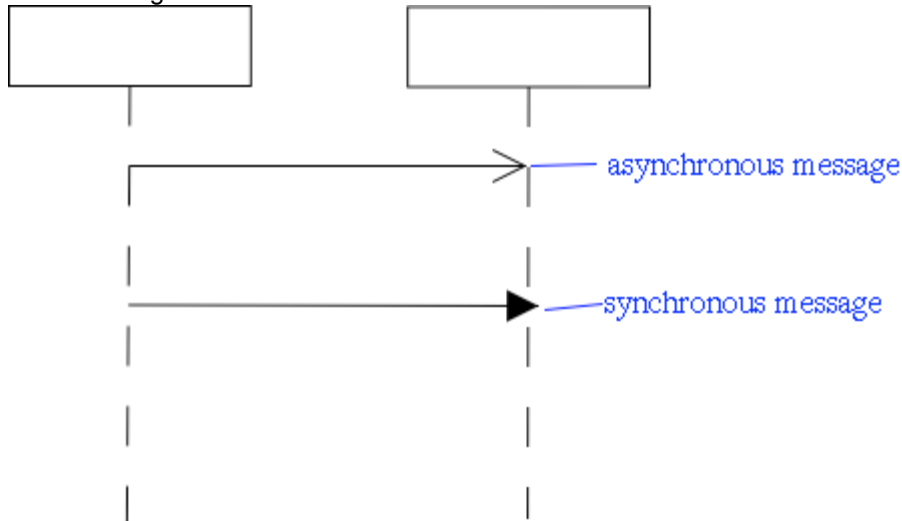


Figure 10: Agent UML Message Notation

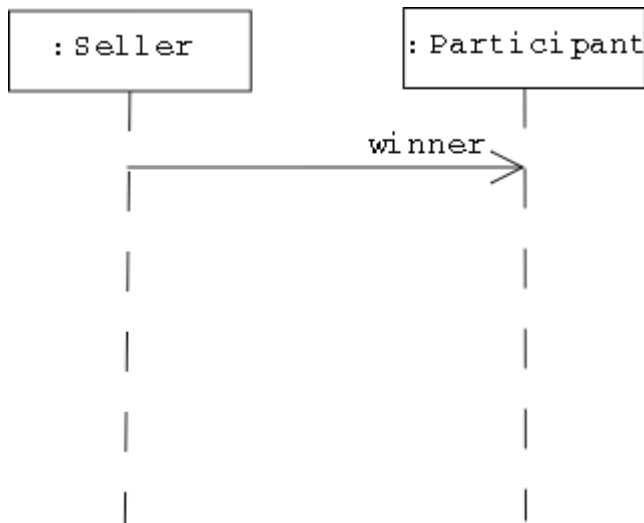


Figure 11: Addressing a Specific Agent

Constraints on messages are given within square brackets near the sending lifeline (see Section 2.4). Timing constraints on messages are given within curly brackets near the receiving lifeline (see Section 2.5).

When a message is sent and received by the same lifeline, it is necessary to consider if the sender will receive the message as well. The directed line is barred near the beginning of the line if the sender does not want to receive the message. Barring or not the line is only for asynchronous messages. For synchronous messages, the line is

always barred to prevent deadlock: the sender is blocked and waits an answer from itself, but it cannot answer since it is blocked. The notation for omitting senders on receipt is given in Figure 12.

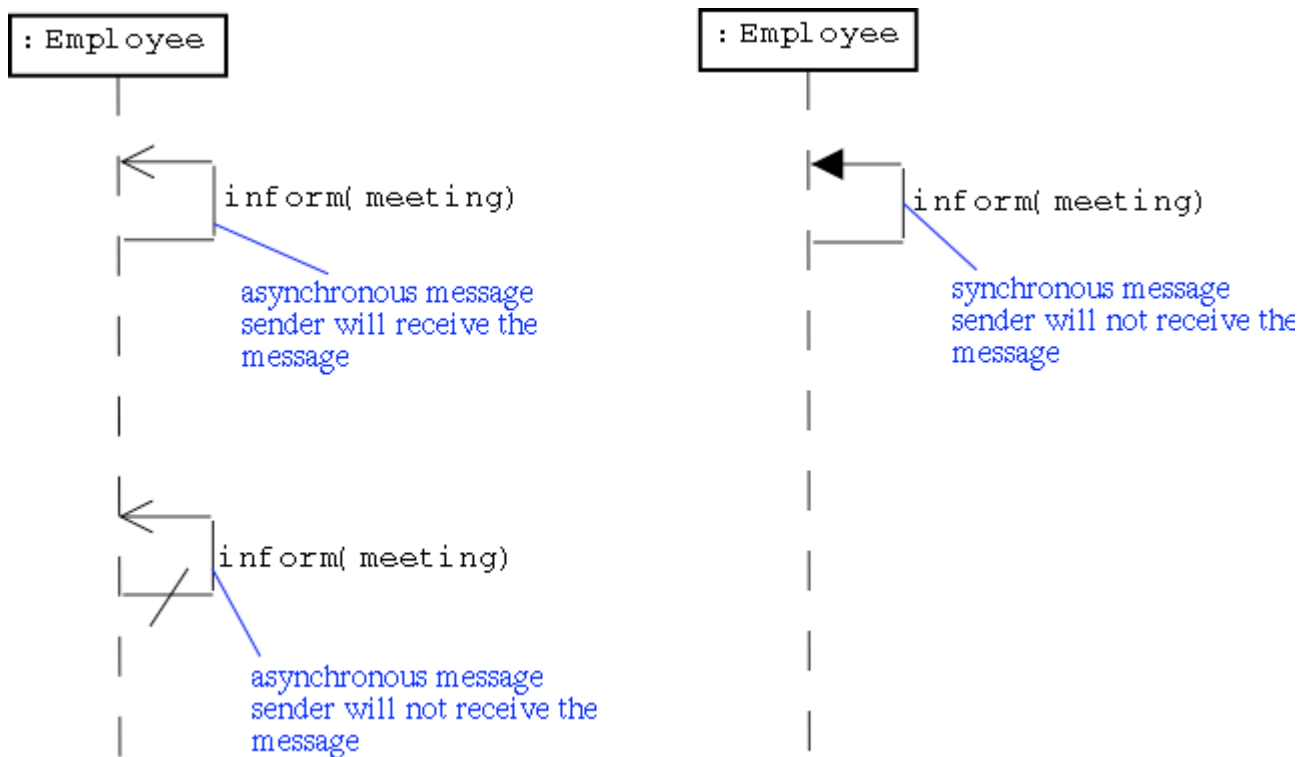


Figure 12: Omitting Sender in Sending Messages on the Same Lifeline

The last consideration in messages is how to represent the overlap of several messages. It is for instance possible that a message sent before another one arrives after. A bridge is made between the first and the second message as shown on Figure 13.

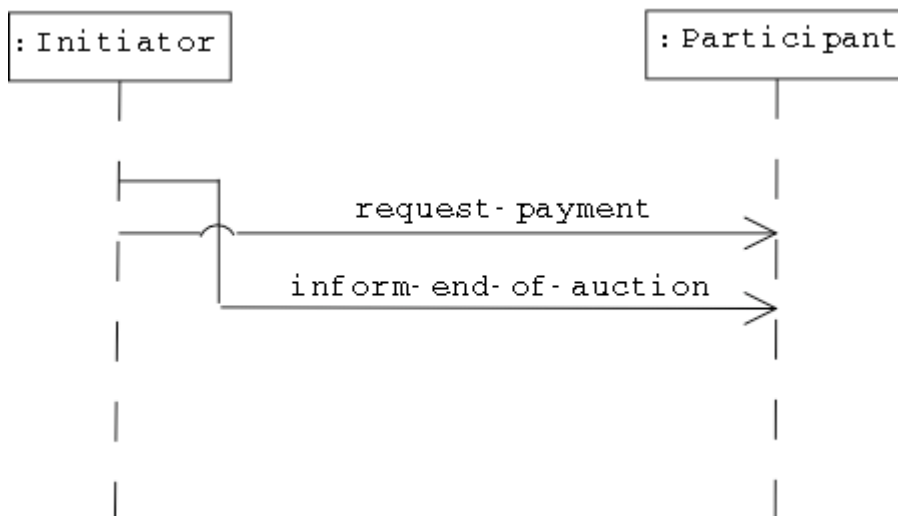


Figure 13: Message Overlapping

Section 2.7 presents messages that are sent (received) to (from) other protocols.

2.3.3 Constraints

- The directed line of the message must be adorned with a string corresponding to the message sent
- The message cardinality must respect the role cardinality if expressed
- A message cannot be adorned with an agent identifier if the receiver lifeline is defined with an agent identifier

2.3.4 Semantics

TBD

2.3.5 Examples

Figure 14 presents an example of a protocol with asynchronous messages. Figure 15 describes an example of a protocol with synchronous messages.

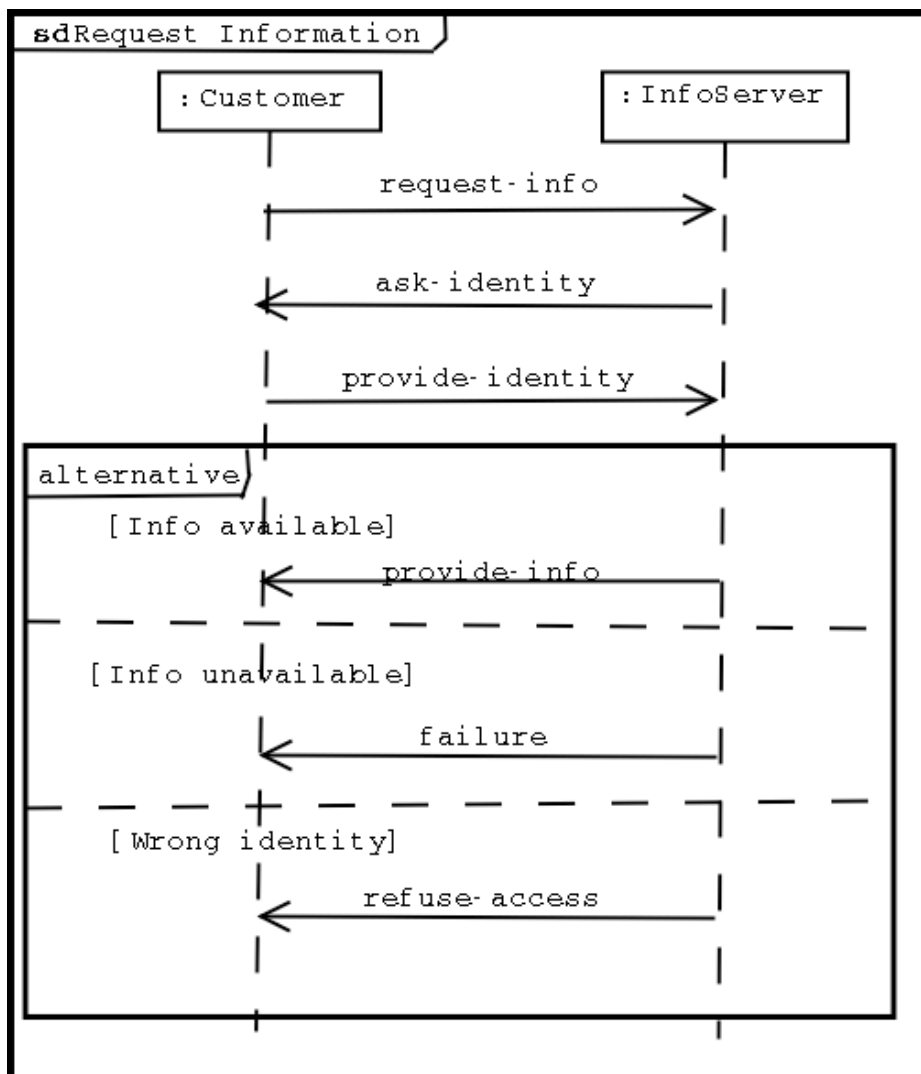


Figure 14: Request Information Protocol

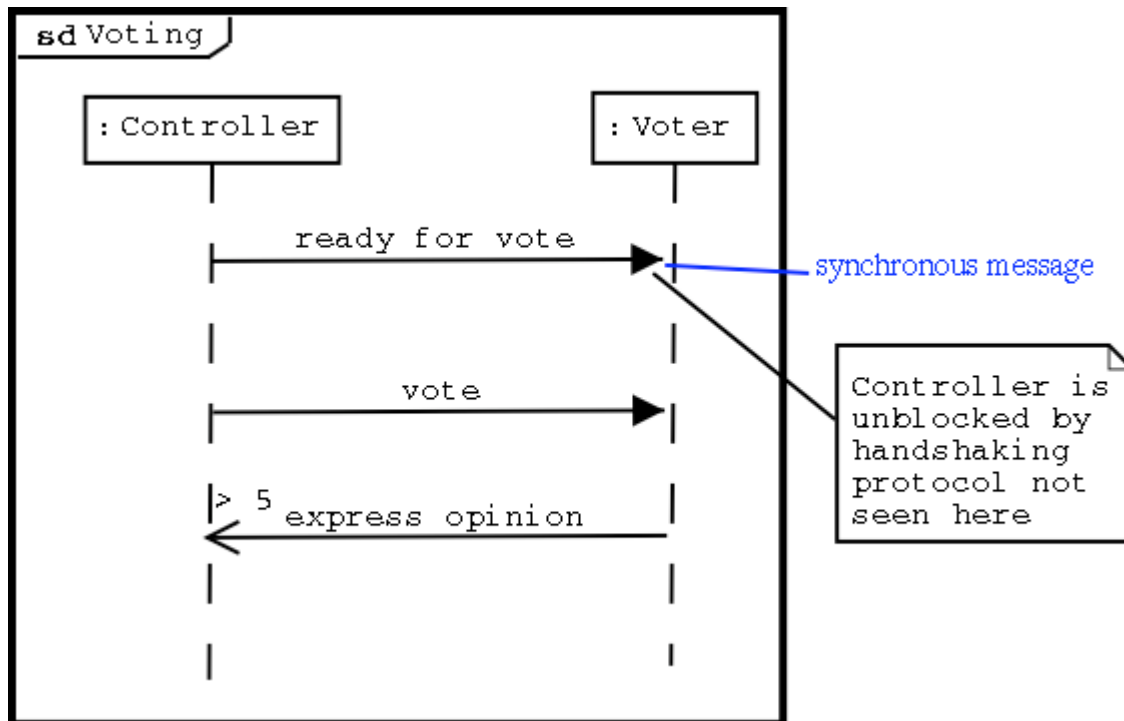


Figure 15: Vote Protocol

2.4 Constraints

2.4.1 Description

Constraints affect how messages and paths in the interaction protocol are used. Some messages can be inhibited if the constraints associated with them are not satisfied. Constraints are mainly used to choose a path between several ones in case of alternatives (see Section 2.6).

Two types of constraints are considered here: blocking and non-blocking constraints. The former implies that the interaction is blocked as long as the constraints are not satisfied; the latter implies that the element ruled by these constraints is executed if and only if the constraints are satisfied. Blocking constraints are useful for instance when some data or situations have to be available to continue the interaction: at least n agents to continue the interaction, for instance. Blocking constraints can be applied to a role, a specific agent in a role or finally to a set of roles. If blocking constraints are applied to a role, it means that all the agents implied in this interaction and playing this role are blocked until the constraints are satisfied. Agents are only blocked for this role and can continue their interactions for their other roles involved in the interaction. Non-blocking constraints are used in alternatives and selection of paths. In case of non-blocking constraints, it is possible to use a default case named *else*. The *else* case is used if other constraints are not satisfied. Timing Constraints in Section 2.5 are specific constraints dedicated to time constraints.

2.4.2 Notation

Constraints can be defined as free-format text or formally with Object Constraint Language (OCL) for instance. Constraints are rendered near the element on which they are applied. When constraints are applied to alternatives, they are given on top of each alternative on the sending lifeline.

Blocking constraints are prefixed by the stereotype <<blocking>>. Non-blocking constraints are written within square brackets.

Examples of blocking and non-blocking constraints are given on Figure 16.

2.4.3 Constraints

- The evaluation of the conditions must be a Boolean value

2.4.4 Semantics

TBD

2.4.5 Examples

Examples of blocking and non-blocking constraints are given on Figure 16.

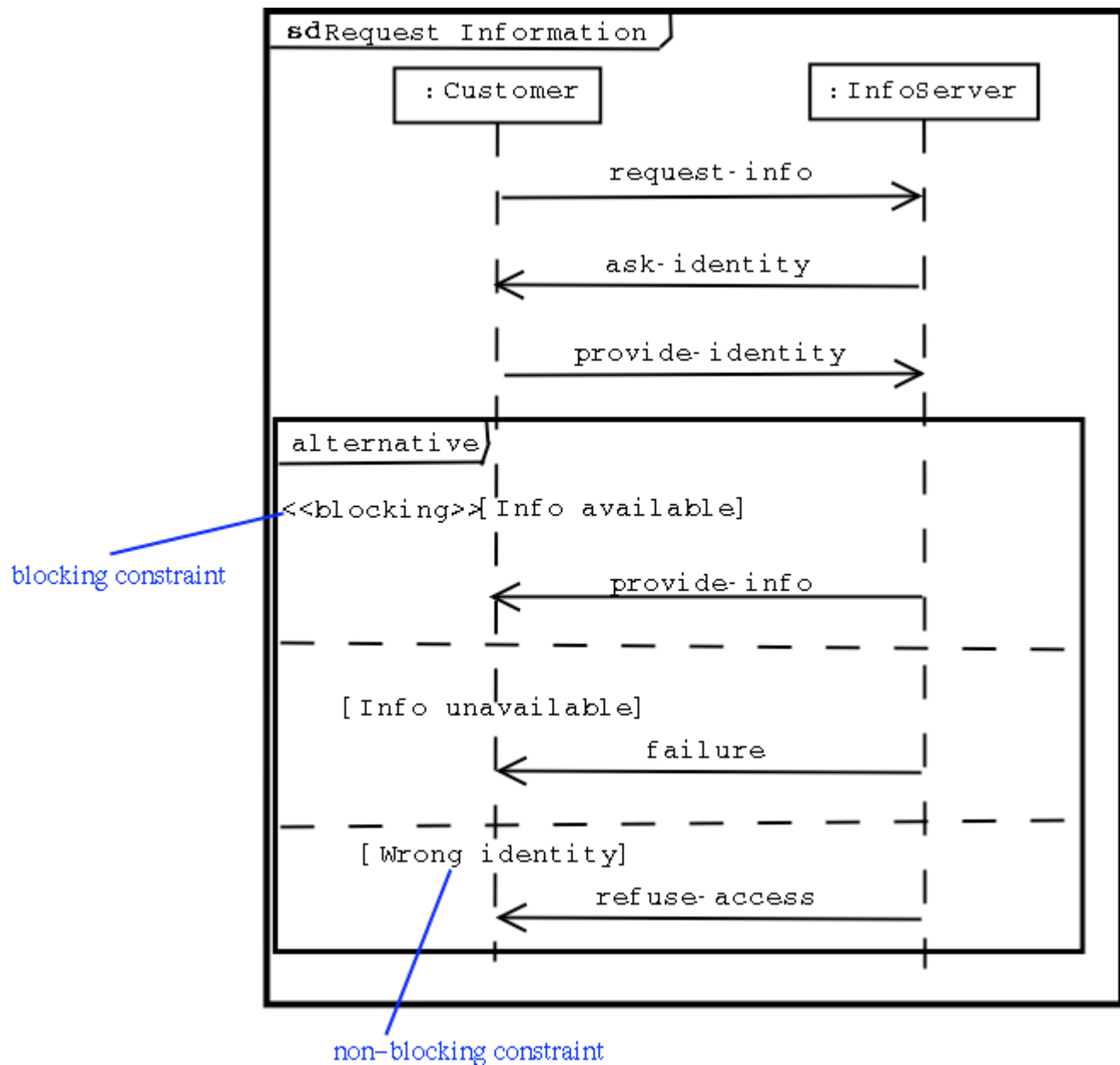


Figure 16: Agent UML Constraints Notation

2.5 Timing Constraints

2.5.1 Description

Time on sequence diagrams allows designers to represent that some messages have to be received before a certain delay. Such delays appear on Contract Net protocols or auction protocols, the answers from participants will arrive before the deadline to be taken into account. Timing constraints can be absolute or relative in the sense that they will refer to the current time or since a specific event in the interaction, the last message for instance. Relative time constraints are written as duration. The message to which the timing constraint is applied has to be received between the lower bound and the upper bound. Lower and upper bounds can be either natural numbers such as $\{0..3\}$ (the message has to be received between now and 3 units of time) or based on observation time such as $\{0..t+3\}$ (the message has to be received between now and the observation time plus 3 units of time). The observation time has to be defined before in the diagram.

Absolute time constraints are written as free-format text. It can be a date such as 2003-04-23 or a time 14:00 or any combination of these two constraints.

2.5.2 Notation

Relative time constraints are written as an interval as follows: $\{initial\ time..final\ time\}$. The interval is rendered as a horizontal bar on the first message in the interval, a horizontal bar on the last message in the interval, a vertical line directed in both ways between the two bars and the timing constraints near the vertical directed line as shown in Figure 17. Absolute time constraints respect this notation except that the interval can be a date or a time or any combination of these two elements.

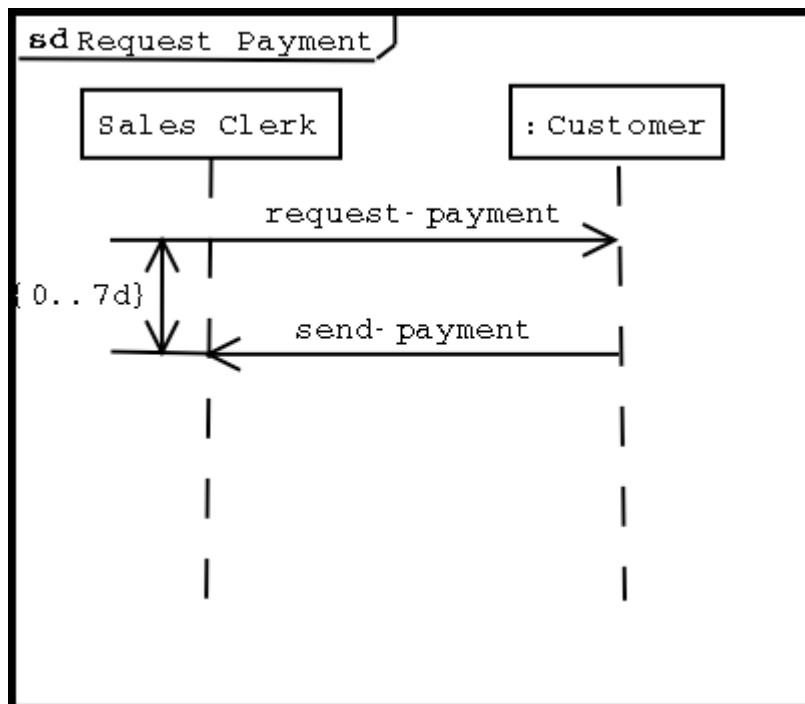


Figure 17: Agent UML Timing Constraint Notation

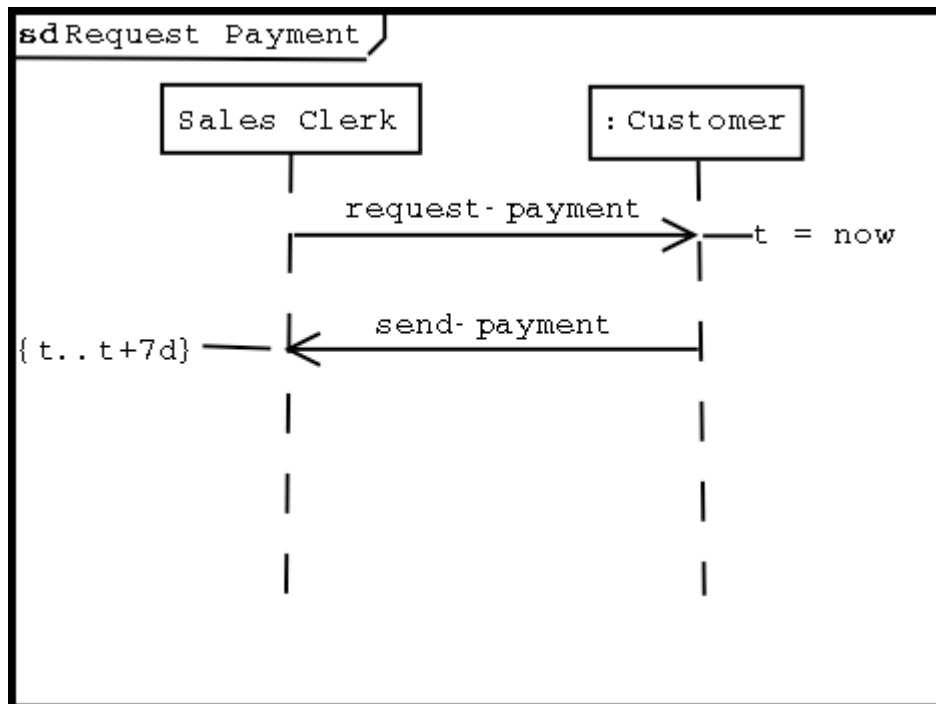


Figure 18: Alternate Timing Constraint Notation

2.5.3 Constraints

- When timing constraints refer to absolute time, lower and upper bounds **must** be natural numbers.
- The lower bound cannot be greater than the upper bound.
- The lower bound and the upper bound may have the same value.
- When timing constraints refer to relative time, the variable referring to the observation time **must** be defined before in the diagram.
- Variables for the relative time **must** be defined as follows: <variable> = now where <variable> is the variable name used after in the time intervals.

2.5.4 Semantics

TBD

2.5.5 Examples

An example of relative time constraints is given in Figure 17. It represents that customers have to pay within seven days depicted by $\{0..7d\}$. The timing constraint is computed between the message that requests payment and the payment. An alternate solution is in Figure 18 using the observation time.

2.6 Splitting/Merging Paths

2.6.1 Description

A protocol that has a single path from the initial state to the final state is worthless and reduces all the autonomy for agents. It is better to let agents the ability to consider several alternatives during the interaction based on their mental states, their intentions and the current state of the interaction. This section describes how to represent several paths in the interaction and how to merge them if needed.

In the first version of Agent UML [Odell2000], splitting or merging paths in the interaction is realized via specific connectors: the AND, OR and XOR connectors. The AND connector represents the concurrency in the interaction: the messages linked by an AND connector are sent concurrently. The OR and XOR connectors refer to decision in the interaction. The OR connector means that zero or several messages from the list of messages linked by the OR connector are sent. If several messages are chosen, they are sent concurrently. The XOR connector means that one and only one message has to be chosen. The XOR connector was a restriction of the OR connector.

These notions are considered differently in the new version of Agent UML. Actually, Agent UML adheres to the UML 2.0 proposal depicted as CombinedFragment [UML2, p. 364], which means that "boxes" contain the different paths in the interaction depending on the *interaction operator*. As UML 2 quoted it: "*Through the use of CombinedFragments, the user will be able to describe a number of traces in a compact and concise manner*" [UML2, p. 368].

The list of the different interaction operators is the following:

- Alternative,
- Option
- Break,
- Parallel,
- Weak Sequencing
- Strict Sequencing
- Negative,
- Critical Region
- Ignore/Consider,
- Assertion
- Loop.

The operator given above defines the semantics of the CombinedFragment.

The first operator is *Alternative*. *Alternative* means that several paths are possible to follow the interaction and agents have to choose at most one to continue. Guards are associated with alternatives. As a consequence, the alternative that is chosen is the alternative that has its guards evaluated to true. None alternative can be chosen in case any guards are evaluated to true. In this case, the alternative CombinedFragment is not executed. If designers at least need that an alternative is executed, they can implement an *else* alternative in the alternative CombinedFragment. It means that this alternative is always executed if all guards associated with alternatives are evaluated to false. Alternatives were handled through the XOR operator in previous Agent UML. This version extends it by adding the *else* clause. Moreover, it is possible no alternative is executed if no guard is evaluated to true.

The *Option* operator only considers one path in the CombinedFragment. If the conditions associated with this path is evaluated to true, then the path is executed else nothing happens and the interaction follows after this CombinedFragment. The *Option* operator can be represented as a CombinedFragment with an *Alternative* operator and two paths: the first one contains the set of messages that are performed if the conditions are satisfied; the second one is empty corresponding to the conditions not satisfied. The *Option* operator corresponds to the OR operator of previous Agent UML with just one alternative.

The *Break* operator is defined in UML 2.0 as a breaking scenario that stops the current execution of the sequence diagram and executes the scenario in the break CombinedFragment. The broken current execution will not be resumed. Such behavior can be used in Agent UML to represent exceptions that stop the current execution to execute a specific sequence of messages. Such an example is when agents receive an unexpected message; they can stop the current execution and enter in a break CombinedFragment to send a *not-understood* message.

The *Parallel* operator depicts the parallel execution of different paths in any order. It allows designers to represent the sending of several messages concurrently. *Parallel* is similar to the AND connector in the previous version of Agent UML.

The *Weak Sequencing* operator as defined in UML 2.0 means a weak sequencing in a sequence of messages, that is to say, the messages within this CombinedFragment on the same lifeline are ordered but it is not possible to make any assumption for message ordering coming from different lifelines in the same CombinedFragment. This *Weak Sequencing* operator seems to be useful in case of message trace observation when agents are distributed and do not share a global clock. Distributed systems speak about partial observation.

The *Strict* operator refines the *Weak Sequencing* operator and ensures that all the messages in the CombinedFragment are ordered from top to bottom.

The *Negative* operator describes the set of messages that are considered as invalid in the interaction.

The *CriticalRegion* operator represents the critical section in distributed systems. It allows the description of the sequence of messages in the critical region has to be done atomically and this sequence of messages cannot be interleaved with other sequences of messages. Such an example of critical region is in the context of financial transactions between the creditor and the debtor. The order of payment and the payment can be combined in a critical region to ensure that both operations are processed atomically.

The *Ignore* operator defines the set of messages that have to be ignored during the interaction. An illustration of this ignore operand is when delayed messages arrive and are no longer interesting for the state of the interaction, for instance, when the task manager in the Contract Net receives bids and the task is already awarded.

The *Consider* operator represents the converse of the *Ignore* operator. It describes all the messages that have to be considered in the interaction.

The *Assertion* operator describes that the sequence of messages in the CombinedFragment is the only accepted sequence of messages for the current state of interaction.

The *Loop* operator allows designers to represent that an ordered set of messages has to be applied several times. Designers can use either lower and upper bounds or a Boolean expression. As long as conditions are satisfied, the loop is executed and the messages are sent and received.

Previous explanations are on splitting paths, merging paths are done through the Continuation from UML 2.0 [UML2, p. 369]. The Continuation is used within compartments of splitting. A Continuation is identified by a name. Figure 8-136 in UML 2.0 specifications [UML2, p. 370] describes an example involving two sequence diagrams but it is also possible to use Continuations on just one sequence diagram. Such cases help designers to represent paths that merge after having been split. Continuations are matched by their names. The Continuations are distinguished according to whether they are the merging point or a path that follows to the merging point.

2.6.2 Notation

The splitting notation is a solid-outline rectangle with a "snipped corner" pentagon where the operator is given within the pentagon as shown on Figure 19. The splitting separator depicted as a dashed line separates each path from others.

The notation for the name of the operator corresponds to the name of the operator as shown in Figure 19: alternative, option, break, parallel, weak, strict, negative, critical, ignore, consider, assertion and loop. This is contrary to UML 2, which abbreviates the operator name for some of them and not for all. In case of the *Ignore* and *Consider* operators, the operator is followed by the list of messages that have to be ignored or considered. When using a *Loop* operator, designers have to write after the bounds or the condition in parentheses.

Each path is separated from others via a splitting separator. A splitting separator is rendered as a dashed line.

Continuation is rendered as a rounded rectangle with a name within as shown on . A Continuation that follows in a merging point has a filled triangle after its name (*outgoing Continuation*), a Continuation that is the merging point has a filled triangle before its name (*incoming Continuation*). All the outgoing Continuations must be matched with one and only one incoming Continuations.

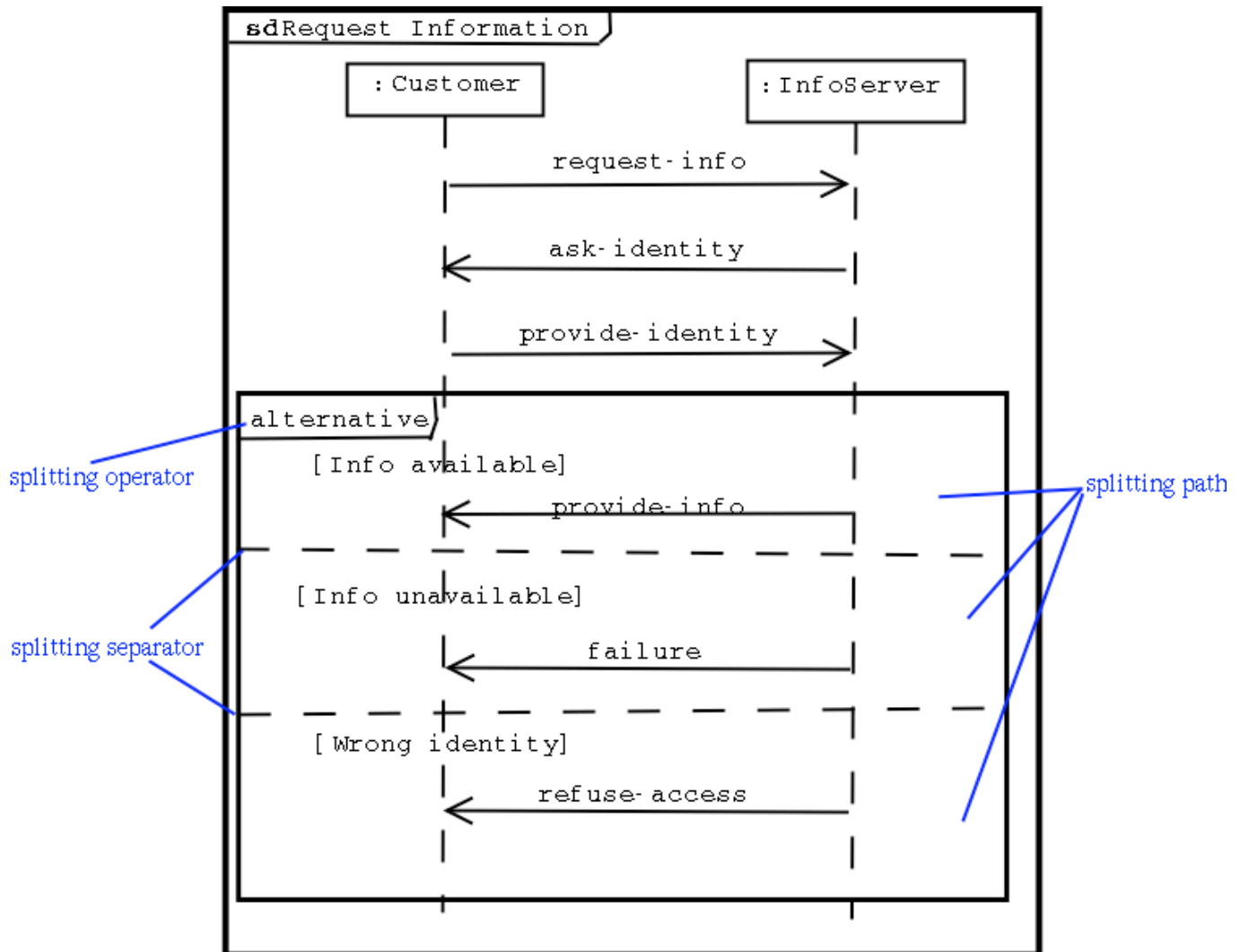


Figure 19: Agent UML Splitting Notation

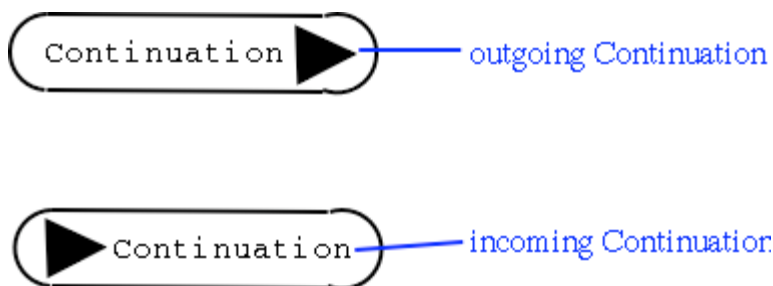


Figure 20: Agent UML Continuation Notation

2.6.3 Constraints

- An outgoing Continuation must be matched with an incoming Continuation.
- All the outgoing Continuations for a specific name are matched with one and only one incoming Continuations.

2.6.4 Semantics

TBD

2.6.5 Examples

Examples of CombinedFragments and Continuations can be found in Section 2.14.

2.7 Protocol Interactions

2.7.1 Description

Section 2.9 describes the combination of several protocols where a protocol is executed within another one. In this case, there is no interaction between the two protocols since the inner protocol has no knowledge of the outer protocol. In some situations, it is important that there is an interaction between the two protocols: the outer protocol sends a message to the inner protocol and passes information. At the end of the execution of the inner protocol, this one ends and may want to pass information to outer protocol. It then sends a message to the outer protocol. These interactions are processed through Gates in UML 2.0 [UML2, p. 377]. We use these Gates in Agent UML to represent the interactions between protocols.

Two kinds of protocol interactions have to be considered: blocking interaction and non-blocking interaction. In the former, the outer protocol waits the inner protocol termination to resume its interaction. In the latter, the outer protocol resumes its interaction without waiting the inner protocol termination.

2.7.2 Notation

There is no specific notation for Gates in Agent UML. The notation corresponds to the message notation given in Section 2.3. A non-blocking interaction is distinguished from a blocking interaction is the outer protocol continues its execution, that is to say some messages within the outer protocol are exchanged during the inner protocol execution.

2.7.3 Constraints

- If designers use blocking interactions, they have to provide a mechanism (message, action, etc.) that the inner protocol can trigger to inform the outer protocol of its termination
- Designers have to be cautious of some deadlock situations that can occur if two protocols mutually interact together and wait each other of an answer from the other

2.7.4 Semantics

TBD

2.7.5 Examples

Figure 21 depicts the examples of two protocols in interaction through gates. These gates are blocking since the request payment protocol is pending till the do payment protocol ends.

2.8 Interaction Termination

2.8.1 Description

The stop operator allows designers to represent that a specific agent playing a specific role or a role no longer interacts in this interaction. If an agent has multiple roles, stopping a specific lifeline does not have consequence on other roles that still continue to interact. The stop operator is also interesting to denote that a path (in the context of paths running in parallel) comes to end.

2.8.2 Notation

The stop operator is depicted by a cross on the form of an X at the bottom of a lifeline as shown on .

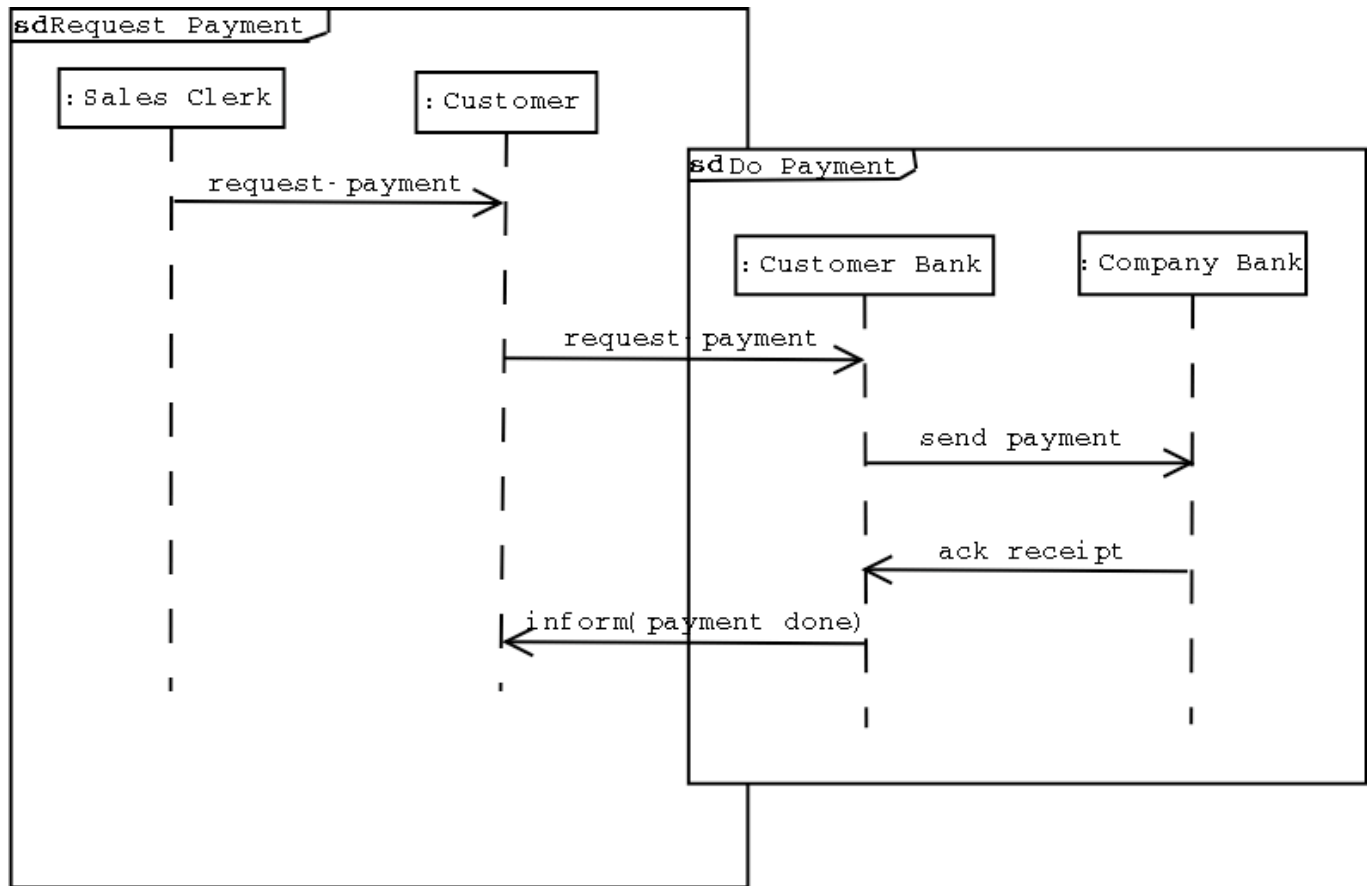


Figure 21: Request Payment Protocol with Gates



Figure 22: Agent UML Termination Notation

2.8.3 Constraints

- No messages can appear below this stop operator except if this operator intervenes in a CombinedFragment.

2.8.4 Semantics

TBD

2.8.5 Examples

The example of FIPA Contract Net Protocol on Figure 29 depicts the use of the stop operator.

2.9 Protocol Combination

2.9.1 Description

In the first version of Agent UML, it was possible to represent nested and interleaved protocols, i.e. protocols that are called within another protocol. A nested protocol is a protocol that is included in another one whereas an interleaved protocol is a protocol that is called during another one. In the new version of Agent UML, we propose to keep the notion of interleaved protocols and we follow the InteractionOccurrence from UML 2.0 [UML2, p. 378]. An interleaved protocol is depicted by its name and the parameters in case the sequence diagram associated with this protocol is a protocol template. As soon as the called protocol ends, the execution returns to the caller protocol.

Designers have to be cautious how they use protocol combination, especially when different agent roles are at stake. For instance, in a protocol, the client is depicted through the role *Customer* and in the second one, as *Client*. Designers need to match the different roles if they are differing from protocol to protocol either with a comment or with a role diagram.

2.9.2 Notation

The call to another interaction protocol rendered as a sequence diagram is done through a solid-outlined rectangle with a "snipped" corner pentagon. The content of the rectangle is the name of the interaction protocol. The keyword **ref** is present in the pentagon as shown on Figure 23.

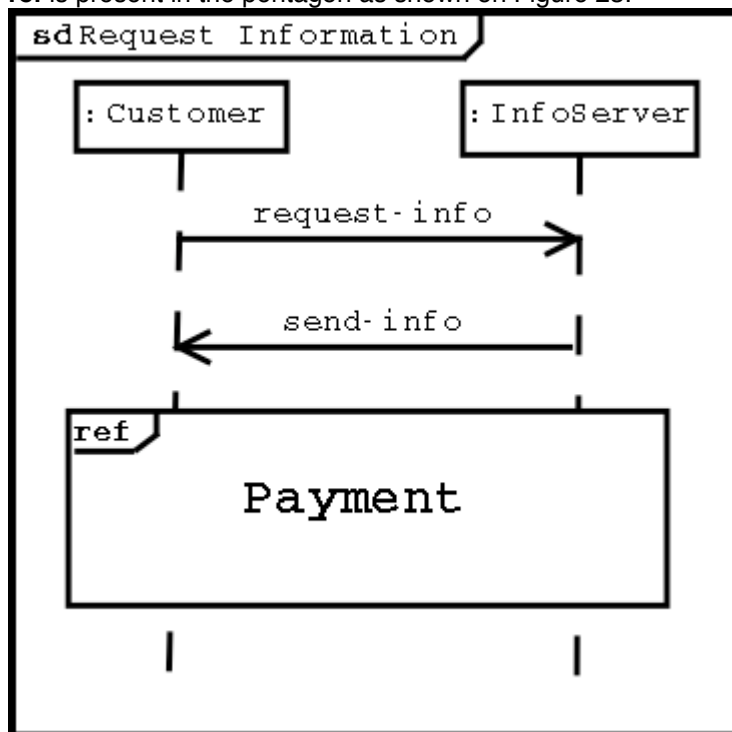


Figure 23: Agent UML Interleaved Protocol Notation

2.9.3 Constraints

- A sequence diagram may call itself through the interleaving notion but it is mandatory in this case to provide a guard to prevent infinite loop except it is the requested behaviour.

2.9.4 Semantics

TBD

2.9.5 Examples

Figure 23 describes an example of interleaved protocol. The customer first requests information then the server sends the information. The payment for this information is done via the protocol payment defined outside this protocol.

2.10 Actions

2.10.1 Description

Even if sequence diagrams only deal with messages, it is important to link actions that occurred when receiving a message. In the context of FIPA Agent Communication Language, receiving an *inform* message implies that the recipient will believe the content of the message from now. The aim of this feature is not to adorn the sequence diagram with all the actions that can occur during the interaction but to pinpoint some actions that are important during the interaction. Designers are responsible to select the accurateness of the action specification from generic action to more specific one linked to message content for instance.

2.10.2 Notation

An action is depicted on sequence diagrams as a round-cornered rectangle as shown on Figure 24. The action is linked to the message that triggers it by an association.

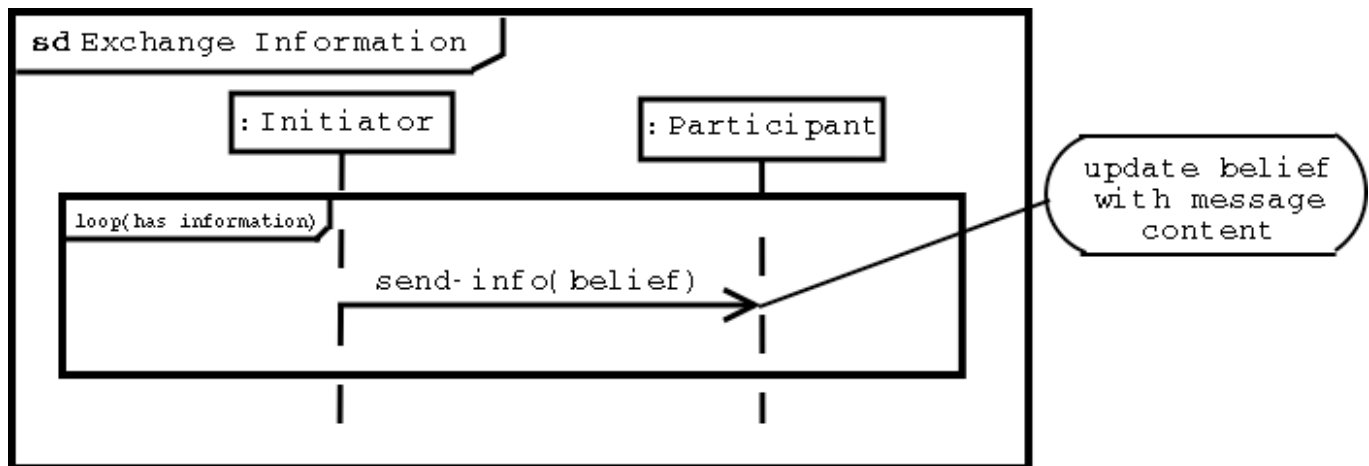


Figure 24: Agent UML Action Notation

2.10.3 Constraints

2.10.4 Semantics

TBD

2.10.5 Examples

Figure 24 describes the example where an action occurs every time the initiator sends information to the participants. The action is to update participants' beliefs with the belief given in the message.

2.11 Protocol Templates

2.11.1 Description

The purpose of protocol templates is to create reusable patterns for useful protocol instances. The protocol as a whole is treated as an entity in its own right, which can be customized for other problem domains. A parameterised 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. Protocol templates refer to abstract classes in object-oriented theory.

2.11.2 Notation

A protocol template is defined as well as a sequence diagram but there are some differences to distinguish it from instantiated protocols. The first difference is that the name of the protocol is prefixed by the stereotype `<<template>>` as shown on Figure 25. The second difference is about the information carried by the protocol. If the information (communicative act, content, variable, etc.) is prefixed by the stereotype `<<unbound>>`, it corresponds to an unbound parameter else this is a bound parameter.

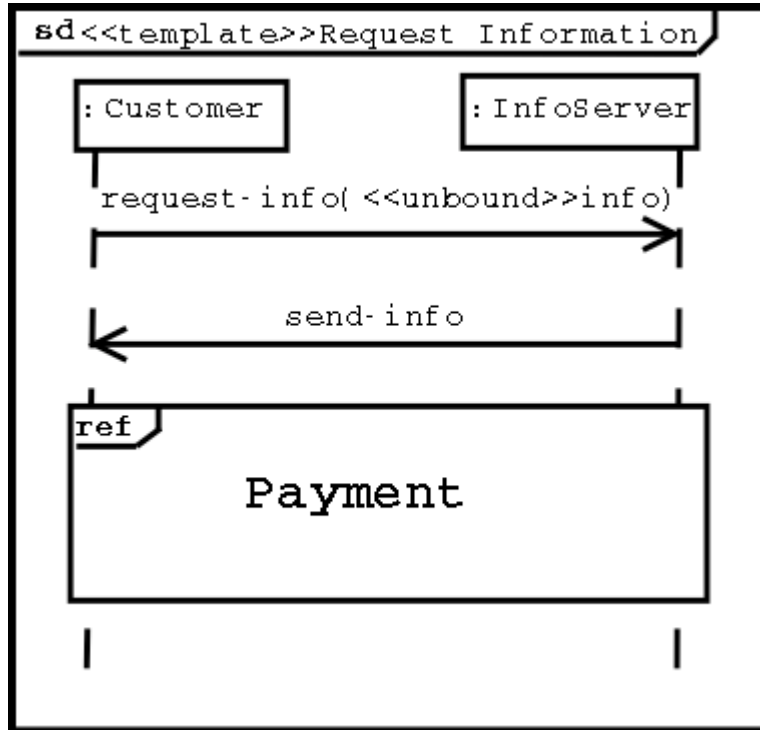


Figure 25: Agent UML Protocol Template Notation

2.11.3 Constraints

TBD

2.11.4 Semantics

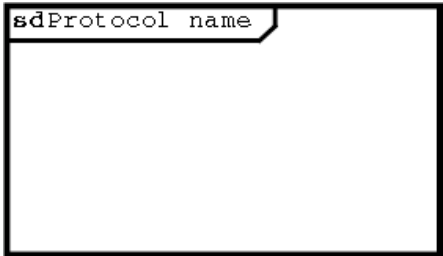
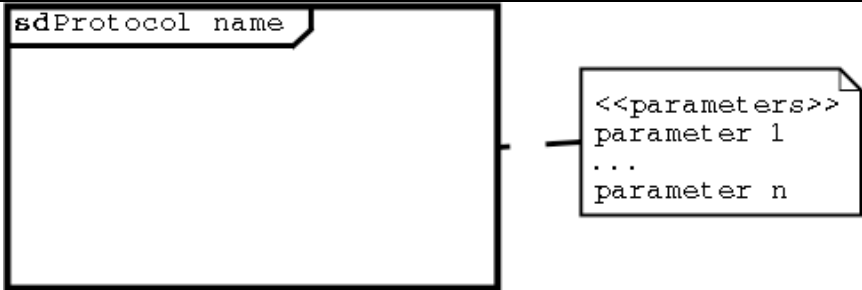
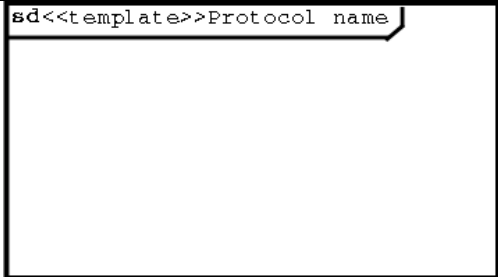
TBD

2.11.5 Examples

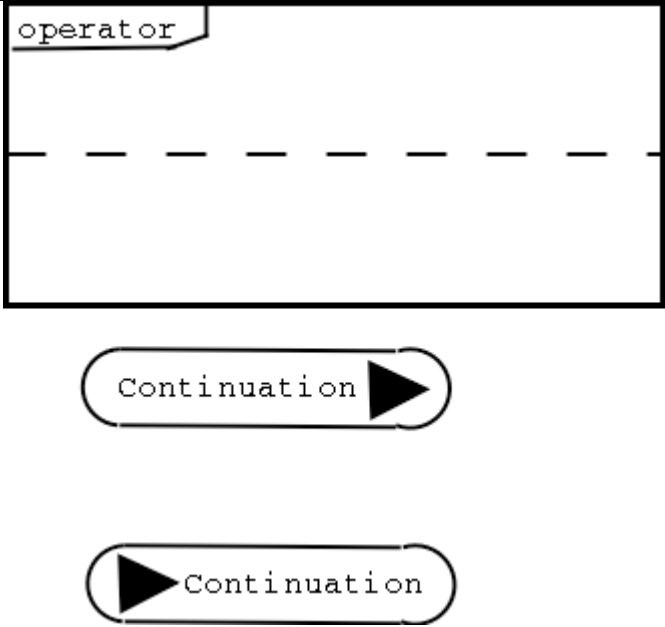

Figure 25 gives an example of protocol template. The variable *info* is unbound and the exact value will be given when this template will be instantiated.

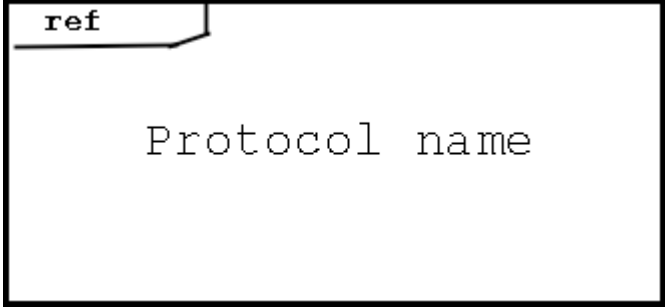
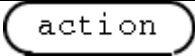
2.12 Graphical Notation Summary

The table given below sums up the different graphical notation used in Agent UML.

Protocol		Defines the frame of the sequence diagram. It contains the message sequence, the agent role, and the constraints on the interaction. See Section Error! Reference source not found. page Error! Bookmark not defined..
Protocol with parameters		Defines a protocol with parameters written in a comment linked to the protocol frame. See Section Error! Reference source not found. page Error! Bookmark not defined..
Template protocol		Defines a template protocol distinguished by the stereotype <<template>>. See Section 2.11 page 23.

<p>Lifeline</p>	<p>The diagram illustrates seven different ways to represent a lifeline in an interaction diagram:</p> <ul style="list-style-type: none"> a) <code>:agent role</code> b) <code>agent id:agent role</code> d) <code>:agent role/group</code> c) <code>agent id:agent role/group</code> e) <code>n :agent role</code> (where <i>n</i> is a cardinality) f) <code>m op i op n :agent role</code> (where <i>m</i>, <i>i</i>, and <i>n</i> are cardinalities and <i>op</i> is an operator) g) <code>condition :agent role</code> (where <i>condition</i> is a condition) 	<p>Lifelines shape the presence of agents in interaction through their identity or through their role in the interaction. a) describes an agent role, b) an agent with an identifier and a role, c) is the complete form for the label: an agent identifier, a role and the group of the agent. d) describes a role and its group regardless of a specific agent. e), f) and g) consider agent cardinality. For e), the exact number of agents is known, f) considers a range for the cardinality. Finally, agents cardinality is based on a condition in g). See Section 2.2 page 6.</p>
<p>Message</p>	<p>The diagram illustrates nine different ways to represent a message in an interaction diagram:</p> <ul style="list-style-type: none"> a) asynchronous message b) synchronous message c) message with cardinality for both sender and receiver (the exact number of agents involved), labeled <i>m</i> and <i>n</i> d) message cardinality with a range this time, labeled <i>m op i op n</i> e) message cardinality where the cardinality is expressed as a condition, labeled <i>condition</i> f) asynchronous message for the specified agent, labeled <i>winner</i> g) synchronous message with cardinality h) asynchronous message with cardinality i) asynchronous message for the same lifeline, the sender receives a 	<p>a) asynchronous message, b) synchronous message, c) message with cardinality for both sender and receiver (the exact number of agents involved), d) message cardinality with a range this time, e) message cardinality where the cardinality is expressed as a condition, f) asynchronous message for the specified agent, g) synchronous message with cardinality, h) asynchronous message with cardinality, i) asynchronous message for the same lifeline, the sender receives a</p>

		copy too, h) asynchronous message for the same lifeline, the sender does not receive a copy and l) synchronous message for the same lifeline, the sender does not receive a copy. See Section 2.3 page 9.
Constraint	<p>[condition] <<blocking>>[condition]</p> <p>a) b)</p> <p>{ t..t+i} — t = now</p> <p>c) d)</p>	a) is a non-blocking constraint, b) is a blocking constraint, c) is a timing constraint, d) presents an alternate representation. See Sections 2.4 and 2.5 on page 13 and following.
Splitting/Merging Path, Continuation	<p>operator</p> 	The splitting operator is written inside the “snipped” corner pentagon. It must be alternative, option, break, parallel, weak, strict, negative, critical, ignore, consider, assertion or loop. A dashed vertical line separates each path from others. The continuation name is given the round-cornered rectangle. The filled triangle at left (right) means an incoming (outgoing) Continuation. See Section 2.6 on page 16.
Protocol interaction	See Message notation	Protocol interaction is depicted through gates. Gate notation corresponds to message notation. See Section 2.7 on page 20.
Termination		The message path is stopped after this

		stop operator. See Section 2.8 on page 20.
Combination		The protocol of name <i>protocol name</i> is executed during the outer protocol execution. The outer protocol is resumed as soon as the inner protocol finishes. See Section 2.9 on page 22.
Action		The name of the action to be performed is written within the double vertical line rectangle. See Section 2.10 on page 23.

2.13 Textual Notation Summary

This Section will contain the XSD (XML Schema Document) to define Agent UML as an XMI file in the context of interoperability and diagram storage.

2.14 FIPA Interaction Protocols

2.14.1 Request Interaction Protocol

The FIPA Request Interaction Protocol (IP) allows one agent to request another to perform some action [FIPA00026].

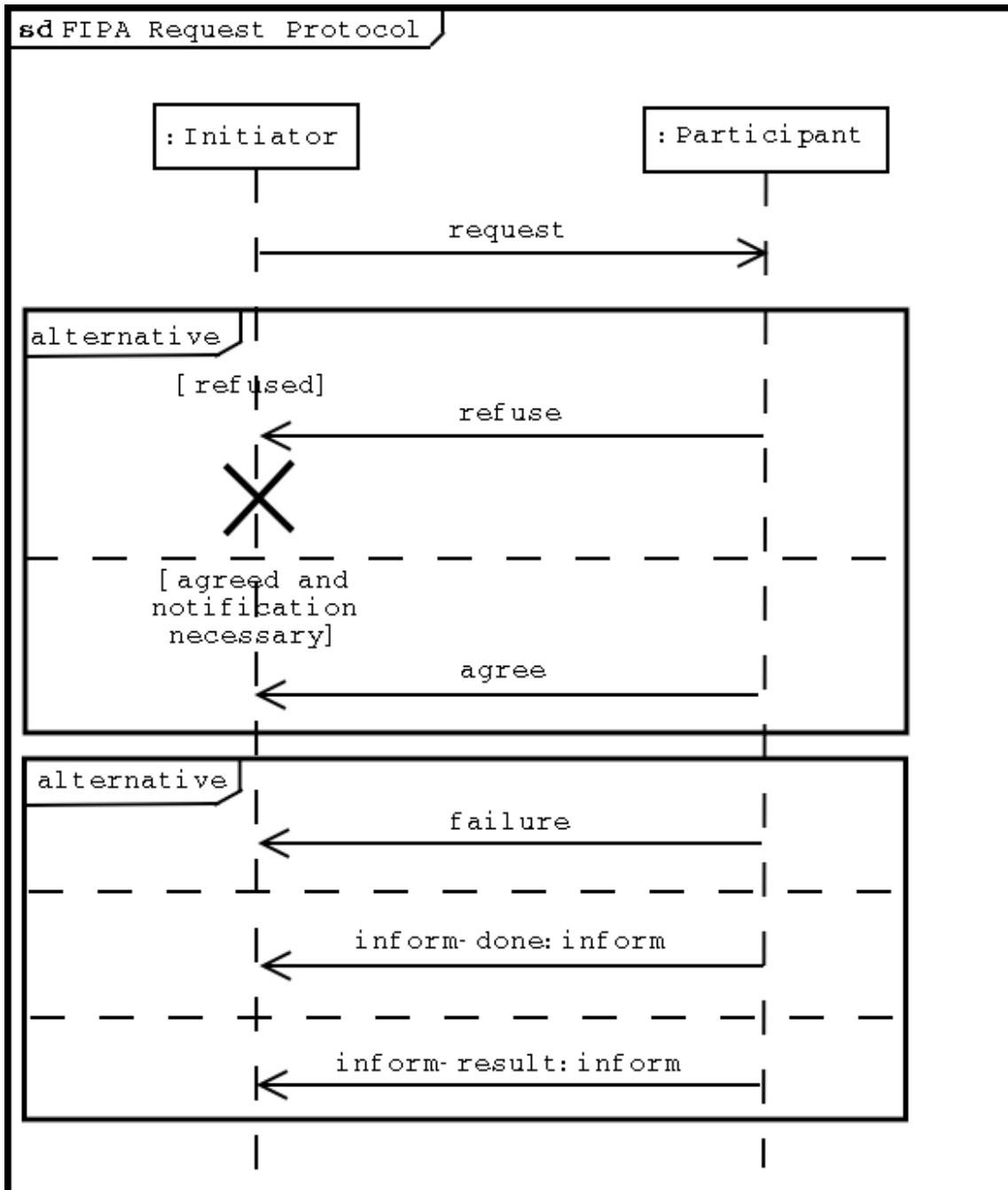


Figure 26: FIPA Request Interaction Protocol

2.14.2 Query Interaction Protocol

In the FIPA Query Interaction Protocol (IP), the receiving agent is requested to perform some kind of inform action (see [FIPA00037]). Requesting to inform is a query, and there are two query-acts: *query-if* (see [FIPA00037]) and *query-ref* (see [FIPA00037]) and either act may be used to initiate this protocol. The requesting agent has several options to answer: *not-understood* if it does not understand the message, *refuse* if it refuses to answer, *failure* if it unable to answer due to an unexpected error and finally, *inform* that gives the information requested.

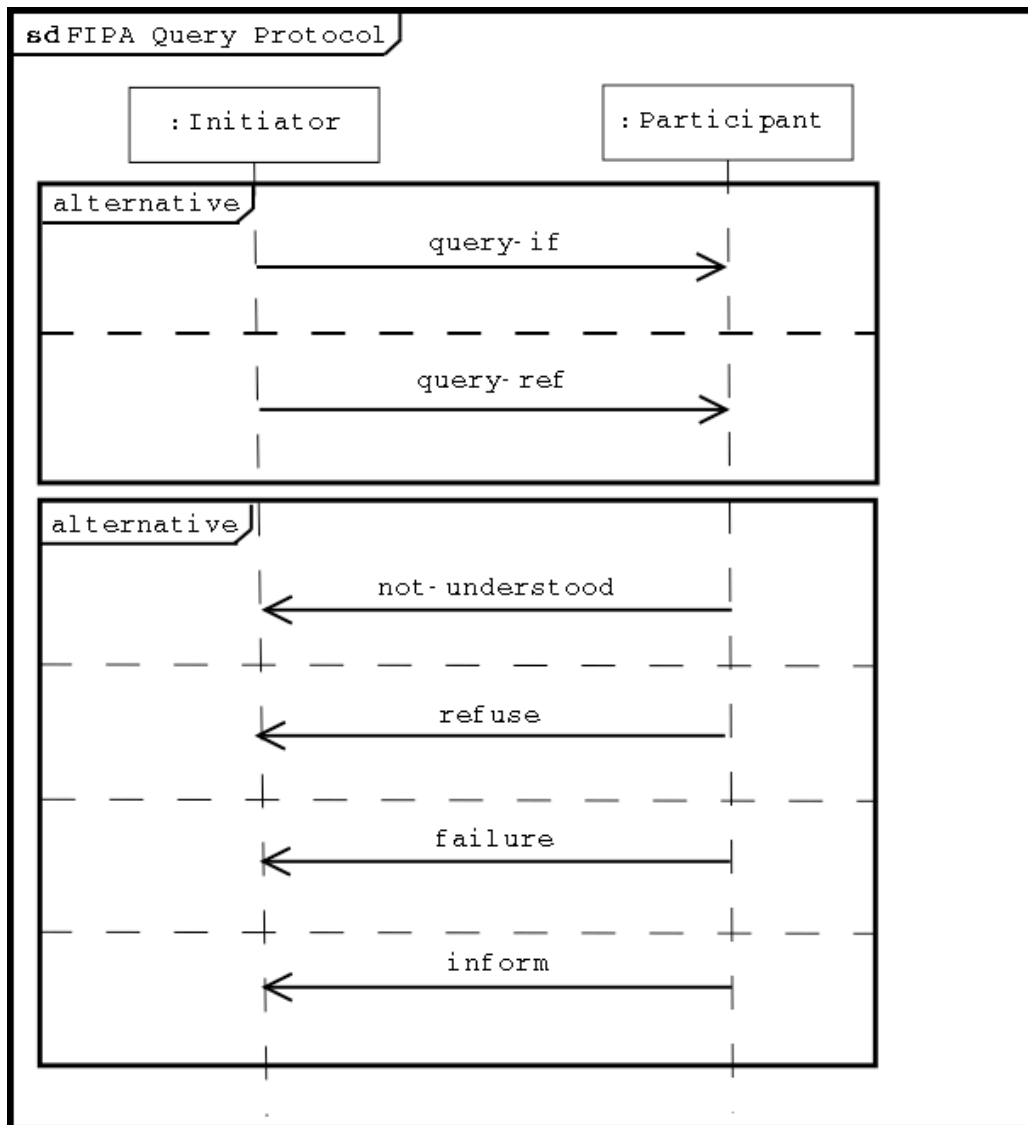


Figure 27: FIPA Query Interaction Protocol

2.14.3 Request When Interaction Protocol

The FIPA Request When Interaction Protocol (IP) allows an agent to request that the receiver perform some action at the time a given precondition becomes true. This IP provides a framework for the request-when communicative act (see [FIPA00037]).

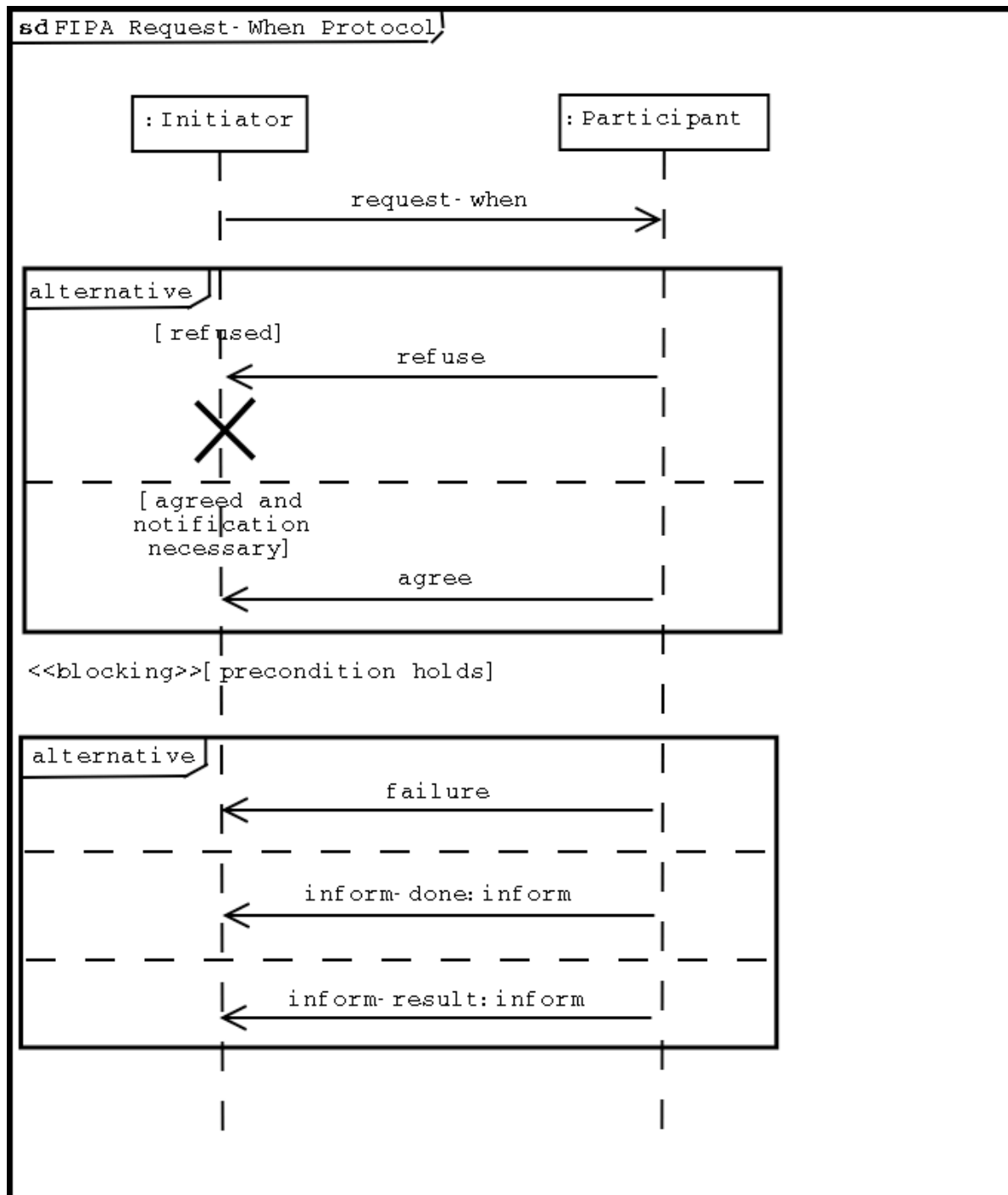


Figure 28: FIPA Request When Interaction Protocol

2.14.4 Contract Net Interaction Protocol

The FIPA Contract Net Interaction Protocol (IP) is a minor modification of the original contract net IP pattern in that it adds rejection and confirmation communicative acts. In the contract net IP, one agent (the Initiator) takes the role of manager which wishes to have some task performed by one or more other agents (the Participants) and further wishes to optimise 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. For a given task, any number of the Participants may respond with a proposal; the rest must refuse. Negotiations then continue with the Participants that proposed.

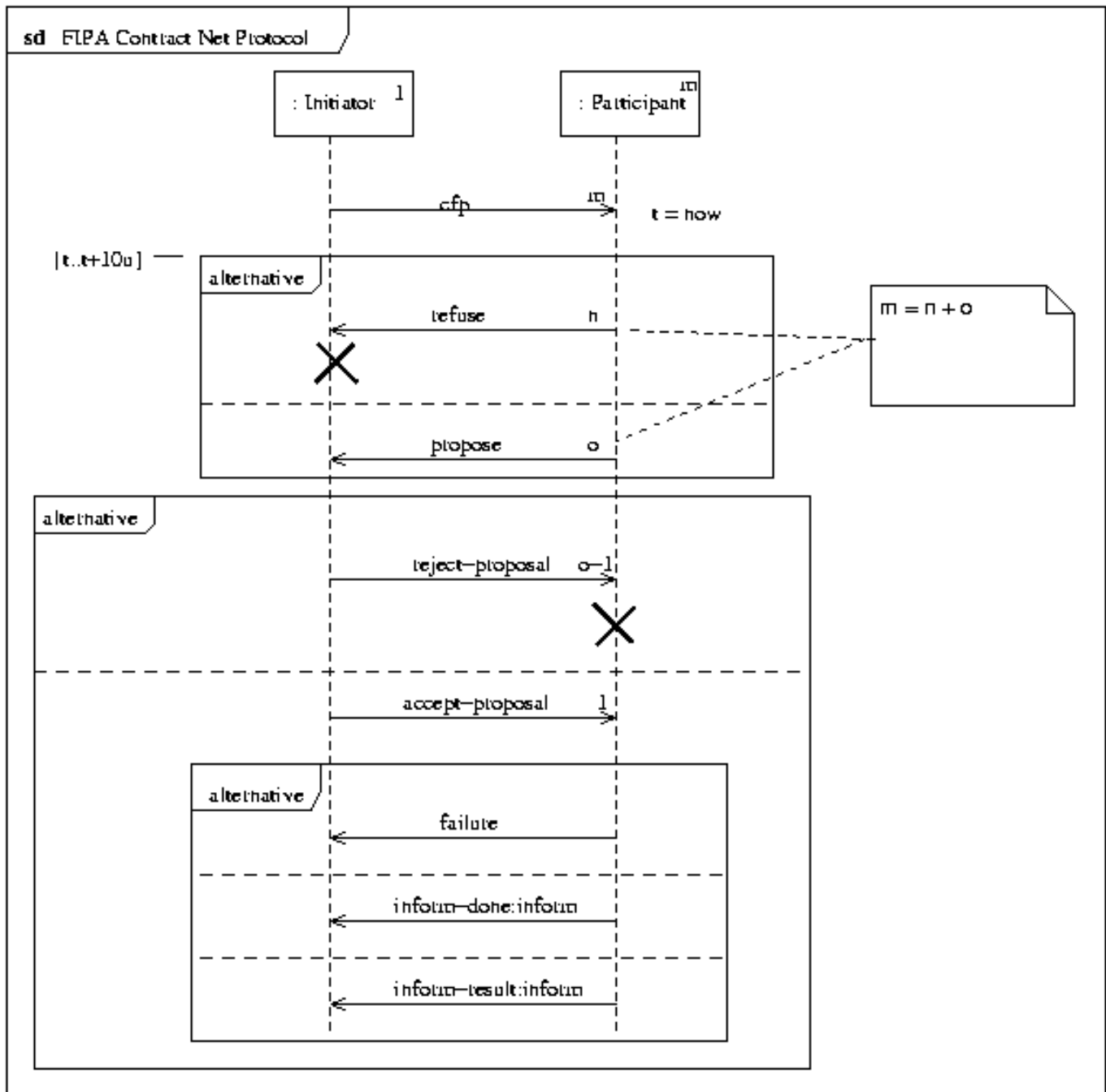


Figure 29: FIPA Contract Net Protocol

2.14.5 Iterated Contract Net Interaction Protocol

The FIPA Iterated Contract Net Interaction Protocol (IP) is an extension of the basic FIPA Contract Net IP (see [FIPA00029]), but it differs by allowing multi-round iterative bidding.

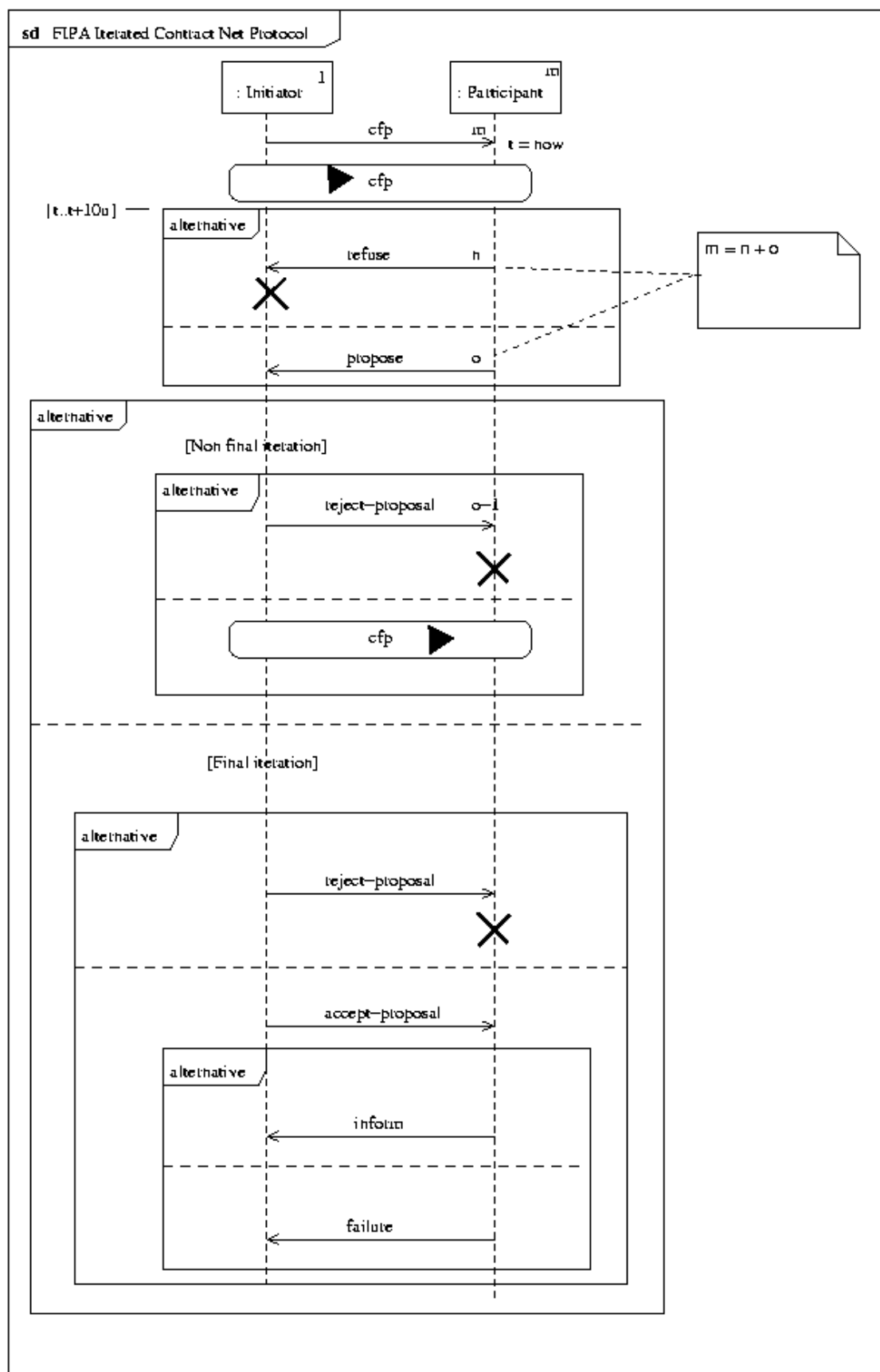


Figure 30: FIPA Iterated Contract Net Interaction Protocol

2.14.6 English Auction Interaction Protocol

In the FIPA English Auction Interaction Protocol (IP), 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 Figure 1, the auctioneer's calls, expressed as the general cfp act (see [FIPA00037]), 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 and 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 - see [FIPA00037]) has been accepted. Hence the appearance in the IP of the accept-proposal (see [FIPA00037]) and reject-proposal acts (see [FIPA00037]), despite this being implicit in the English Auction process that is being modelled.

Note that the proposals that are submitted by the bidders primarily concern the bidding process. In response to a cfp to submit bids to purchase a good X, a proposal would be something of the order: ?I propose that the bidding level be raised to purchase price Z and I assert that I am able to pay Z for X.? This allows the auctioneer to be confident that the bidder can indeed pay the price without committing to actually paying it until the auctioneer specifically requests X (at price Z) from the winning bidder.

At the end of the IP, the auctioneer will typically enter a request IP (see [FIPA00026]) with the winning bidder to complete the auction transaction.

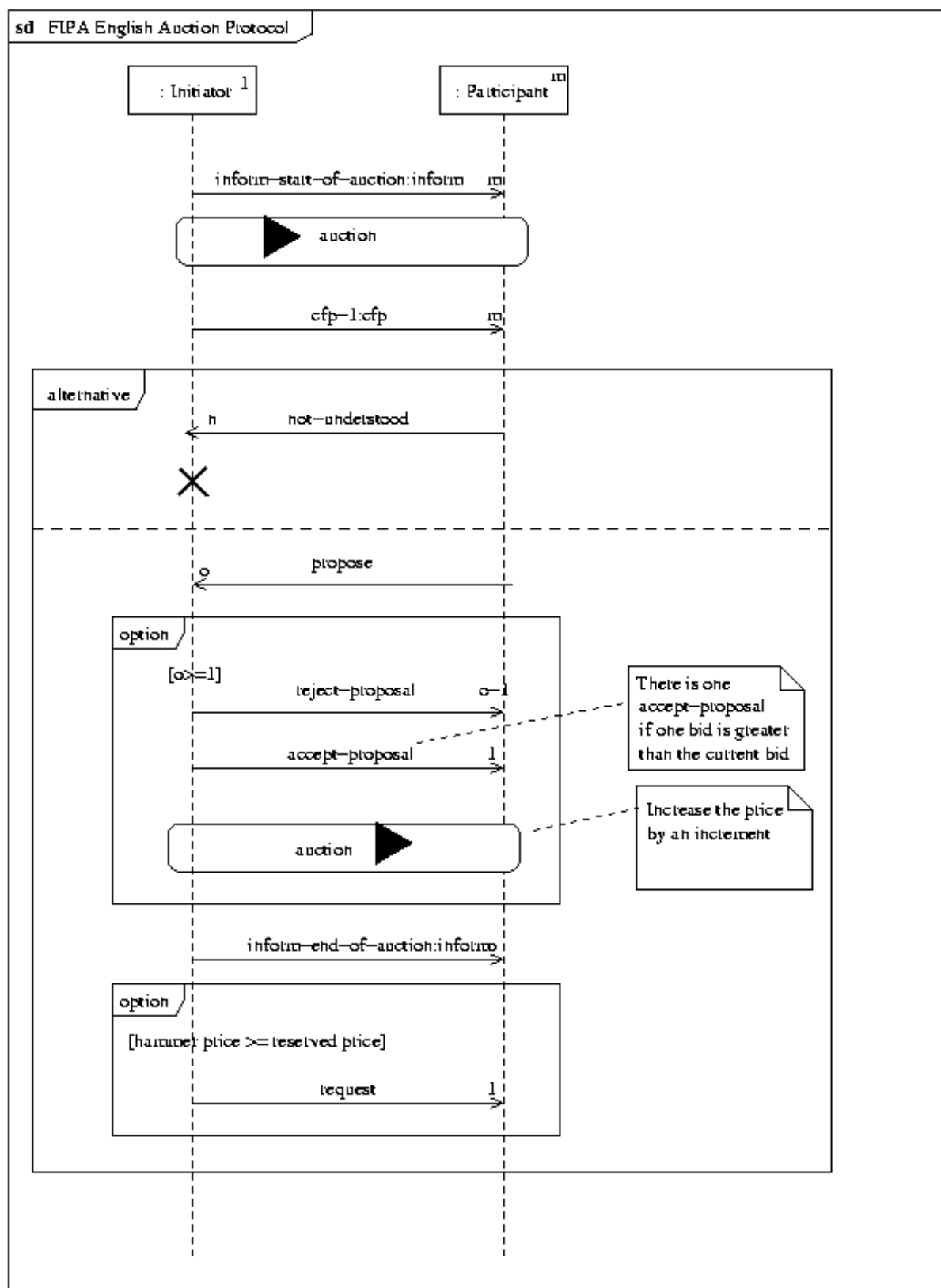


Figure 31: FIPA English Auction Interaction Protocol

2.14.7 Subscribe Interaction Protocol

The FIPA Subscribe Interaction Protocol (IP) allows an agent to request a receiving agent to perform an action on subscription and subsequently when the referenced object changes.

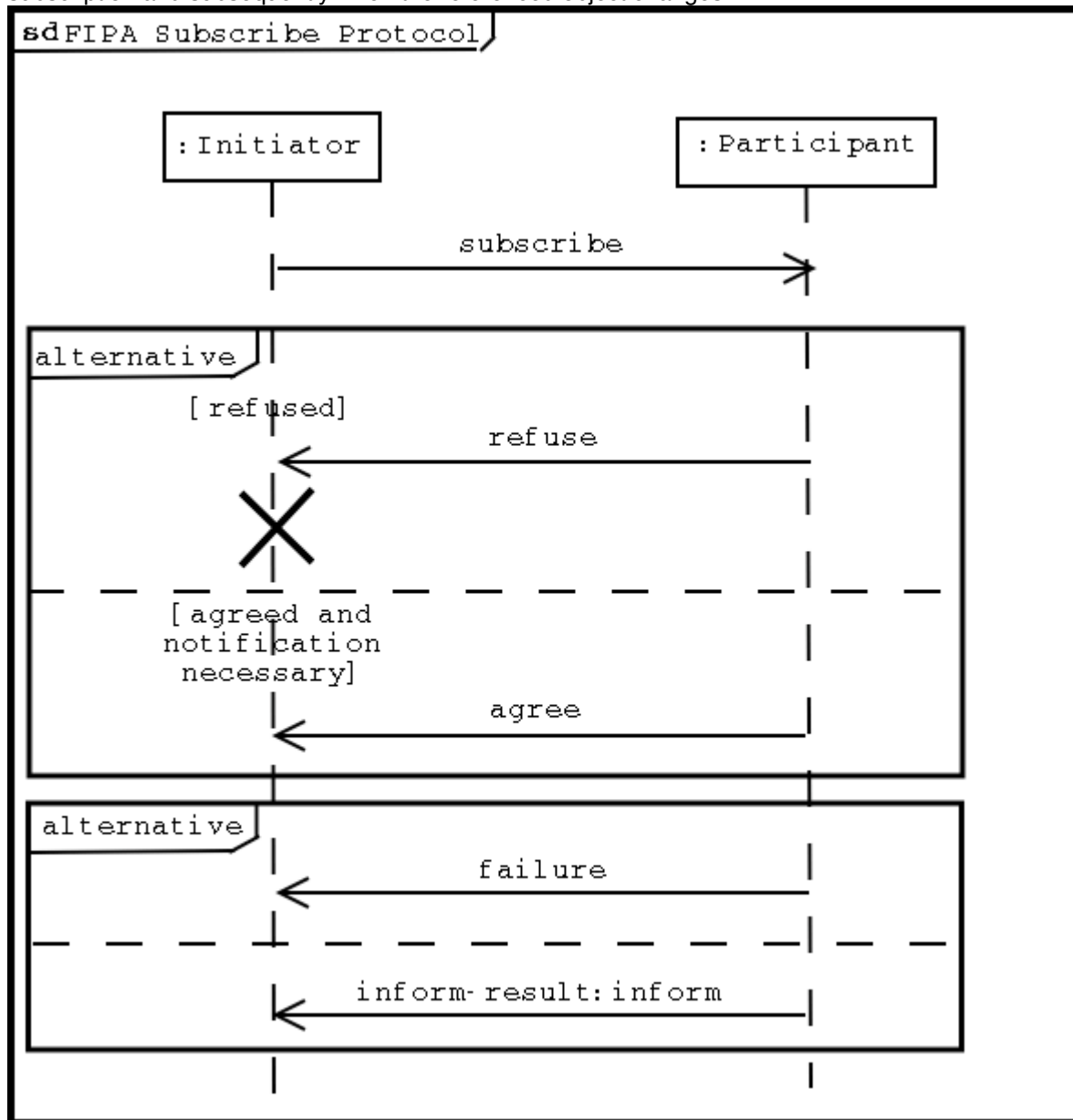


Figure 32: FIPA Subscribe Interaction Protocol

2.14.8 Propose Interaction Protocol

The FIPA Propose Interaction Protocol (IP) allows an agent to propose to receiving agents that the initiator will do the actions described in the *propose* communicative act (see [FIPA00037]) when the receiving agent accepts the proposal.

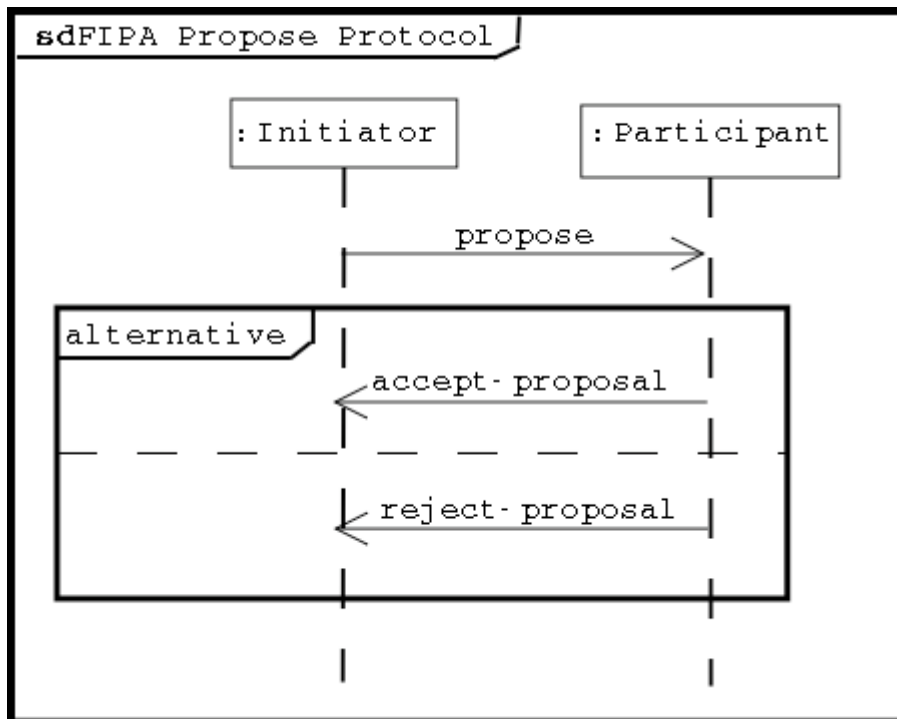


Figure 33: FIPA Propose Interaction Protocol

3 Interaction Overview Diagram

Interaction Overview Diagrams define Interactions (described in Chapter 8, “Interactions”) through a variant of Activity Diagrams (described in Chapter 6, “Activities”) in a way that promotes overview of the control flow. Interaction Overview Diagrams focus on the overview of the flow of control where the nodes are Interactions or InteractionOccurrences. The Lifelines and the Messages do not appear at this overview level.

Interaction Overview Diagrams are specialization of Activity Graphs that represent Interactions, and may differ from Activity Diagrams in some respects.

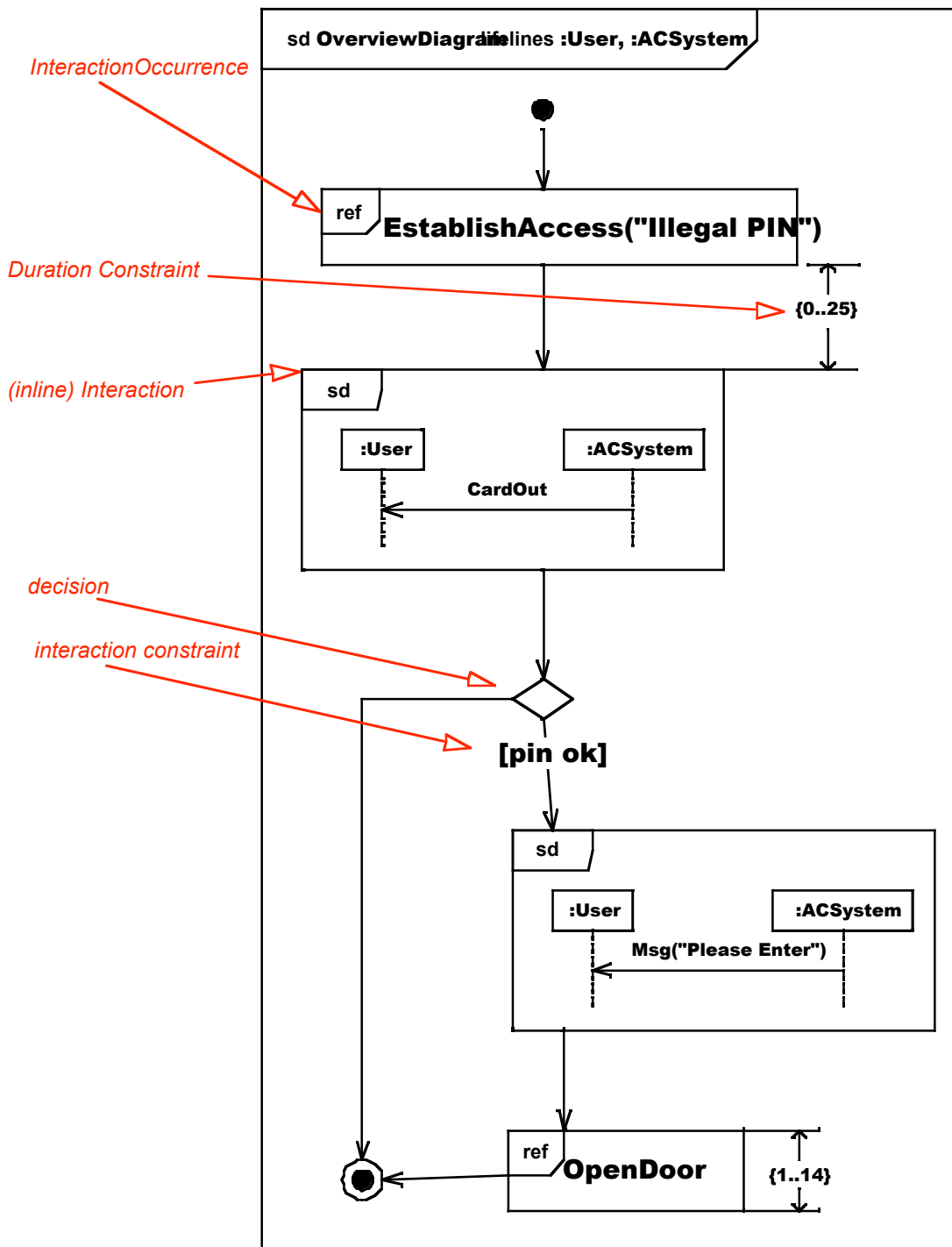


Figure 34: Example of an Interaction Overview Diagram

These kind of UML diagram is useful because it can often express the flow of interactions more clearly than by using the frame of an interaction operand.

4 Communication Diagram

Communication Diagrams focus on the interaction between Lifelines where the architecture of the internal structure and how this corresponds with the message passing is central. The sequencing of Messages is given through a sequence numbering scheme.

Communication Diagrams correspond to simple Sequence Diagrams that use none of the structuring mechanisms such as InteractionOccurrences and CombinedFragments. It is also assumed that message overtaking (i.e. the order of the receptions are different from the order of sending of a given set of messages) will not take place or is irrelevant.

4.1 Communication Diagram Expressions

The Interaction described by a Communication Diagram in Figure 35 shows messages m1 and m3 being sent concurrently from :r towards two instances of the part s. The sequence numbers shows how the other messages are sequenced. 1b.1 follows after 1b and 1b.1.1 thereafter etc. 2 follows after 1a and 1b.

The sequence-expression is a dot-separated list of sequence-terms followed by a colon (':').

sequence-term '...' ':'

Each term represents a level of procedural nesting within the overall interaction. If all the control is concurrent, then nesting does not occur. Each sequence-term has the following syntax:

[integer / name] [recurrence]

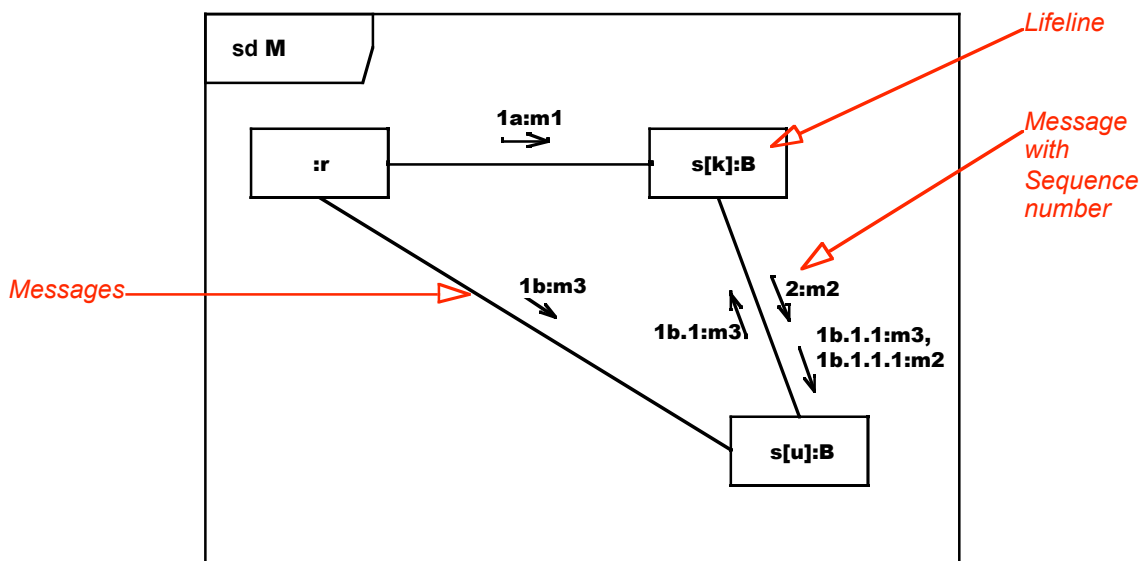


Figure 35: Example of a Communication Diagram

The integer represents the sequential order of the Message within the next higher level of procedural calling. Messages that differ in one integer term are sequentially related at that level of nesting. Example: Message 3.1.4 follows Message 3.1.3 within activation 3.1. The name represents a concurrent thread of control. Messages that differ in the final name are concurrent at that level of nesting. Example: Message 3.1a and Message 3.1b are concurrent within activation 3.1. All threads of control are equal within the nesting depth.

The recurrence represents conditional or iterative execution. This represents zero or more Messages that are executed depending on the conditions involved. The choices are:

iteration *iteration-clause* 'iteration-clause' an iteration

branch 'guard' a branch

An iteration represents a sequence of Messages at the given nesting depth. The iteration clause may be omitted (in which case the iteration conditions are unspecified). The iteration-clause is meant to be expressed in pseudocode or an actual programming language, UML does not prescribe its format. An example would be: *[i := 1..n].

A guard represents a Message whose execution is contingent on the truth of the condition clause. The guard is meant to be expressed in pseudocode or an actual programming language; UML does not prescribe its format. An example would be: [x > y].

Note that a branch is notated the same as an iteration without a star. One might think of it as an iteration restricted to a single occurrence.

The iteration notation assumes that the Messages in the iteration will be executed sequentially. There is also the possibility of executing them concurrently. The notation for this is to follow the star by a double vertical line (for parallelism): *||.

Note that in a nested control structure, the recurrence is not repeated at inner levels. Each level of structure specifies its own iteration within the enclosing context.

5 Timing Diagram

Timing Diagrams are used to show interactions when a primary purpose of the diagram is to reason about time. Timing diagrams focus on conditions changing within and among Lifelines along a linear time axis. Timing diagrams describe behavior of both individual classifiers and interactions of classifiers, focusing attention on time of occurrence of events causing changes in the modeled conditions of the Lifelines.

Timing diagrams show change in state or other condition of a structural element over time. There are a few forms in use, where one is depicted with a simple Timing Diagram in Fig. 26. The primary purpose of the timing diagram is to show the change in state or condition of a lifeline (representing a Classifier Instance or Classifier Role) over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli. The received events are annotated as shown when it is desirable to show the event causing the change in condition or state.

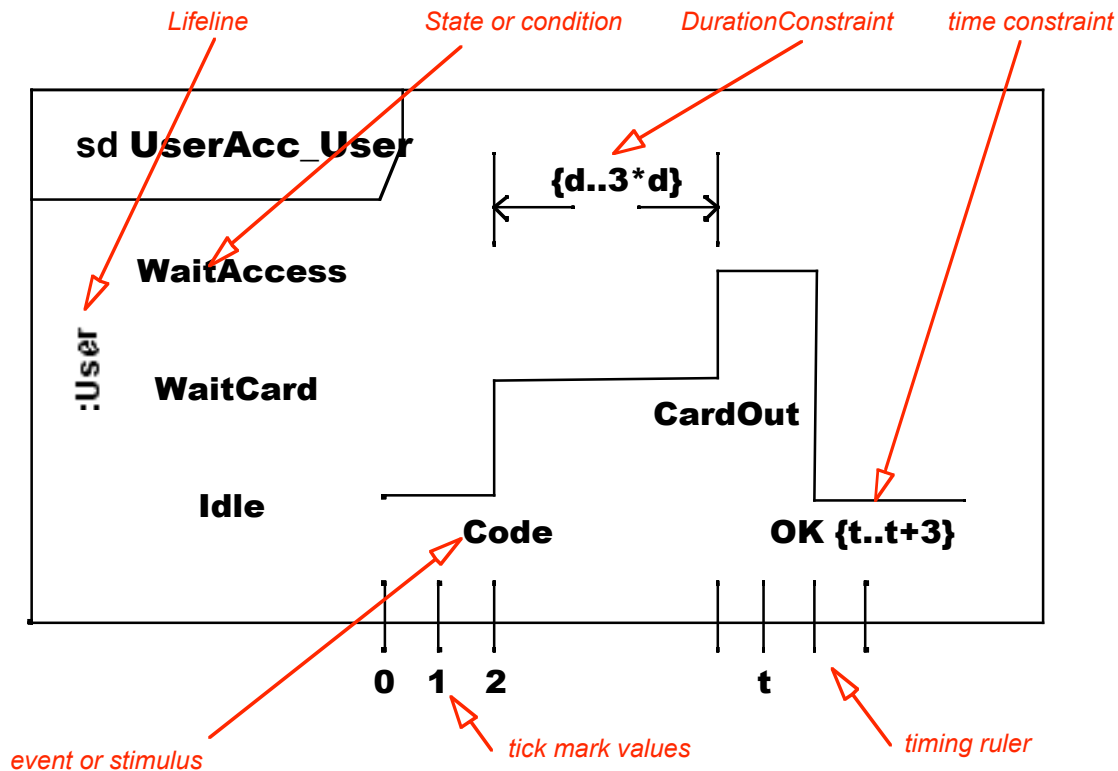


Figure 36: Example of a Timing Diagram

6 References

- [Bauer 2001] Bauer, B., Jörg P. Müller, James Odell, "Agent UML: A Formalism for Specifying Multiagent Interaction," *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001. (Held at the 22nd International Conference on Software Engineering (ISCE)) Can be obtained from <http://www.jamesodell.com/AOSE-Bauer.pdf>
- [FIPA00026] FIPA Request Interaction Protocol Specification. Foundation for Intelligent Physical Agents, 2000.
<http://www.fipa.org/specs/fipa00026/>
- [FIPA00027] FIPA Query Interaction Protocol Specification, Foundation for Intelligent Physical Agents, 2000.
<http://www.fipa.org/specs/fipa00027/>
- [FIPA00029] FIPA Contract Net Interaction Protocol Specification, Foundation for Intelligent Physical Agents, 2000.
<http://www.fipa.org/specs/fipa00029/>
- [FIPA00036] FIPA Propose Interaction Protocol Specification, Foundation for Intelligent Physical Agents, 2000.
<http://www.fipa.org/specs/fipa00036/>
- [FIPA00037] FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, 2000.
<http://www.fipa.org/specs/fipa00037/>
- [Odell 2000] Odell, J., H.V.D. Paranak, and B. Bauer, "Extending UML for Agents," in *Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, G.W. Yves Lesperance, and Eric Yu, Eds, pp. 3-17, 2000.
Can be obtained from <http://www.jamesodell.com/ExtendingUML.pdf>
- [Odell 2003] Odell, J., H. Van Dyke Parunak, and Mitch Fleischer, "The Role of Roles in Designing Effective Agent Organizations," *Software Engineering for Large-Scale Multi-Agent Systems*, Alessandro Garcia *et al* eds., LNCS, Springer, 2003. Can be obtained from http://www.jamesodell.com/Role_of_roles.pdf
- [UML2] Unified Modeling Language (UML) Superstructure: version 2.0 (ad/2003-04-01).