MATES 2003 Tutorial

Interaction Protocol Engineering in Multiagent Systems

Marc-Philippe Huget
(Agent ART Group, Liverpool, UK)
Email: M.P.Huget@csc.liv.ac.uk

Jean-Luc Koning
(Complex Systems Team, Valence, FR)
Email: Jean-Luc.Koning@esisar.inpg.fr

Outline

Context

Multiagent systems composed of multiple interacting agents

Agents have to interact together to achieve their tasks

Achieving tasks can be done through cooperation or coordination

The aim of interaction is to allow the synchronization of agents and the exchange of information

One only deals here with the interaction between agents and not between agents and environment

Context

Multiagent Interaction differs from Object-Oriented Interaction where communication corresponds to method invocation

Inappropriate approach for multiagent systems since agents are autonomous. They can accept or refuse to do a task. This is not the case in object-oriented programming

Context

Interaction uses Shannon's ideas on communication [Shannon, 1948]: *communication is the way to transmit a message from one sender to many receivers via a communication media*

Interaction uses speech act theory [Searle, 1969] for building messages: a verb and a content. For instance, *inform(agent2 (b == 10))*

Several agent communication languages available: KQML [Finin et al., 1994] and FIPA's ACL [FIPA, 2000] among others

Some Definitions

What does interaction mean? *Interaction has to do with the way agents exchange information. Such exchange amounts to message passing from one to many agents. Interaction enables agents to cooperate and to coordinate in achieving their tasks.*

What is a protocol? *This is a set of rules which guides the interaction that take place between several agents. These rules define what messages are possible for any particular interaction state. The set of possible messages is finite.*

Communication in MAS

Agents need an agent communication language to interact. Two main ACLs:

- KQML developed by the DARPA consortium through the Knowledge Sharing Effort initiative
- FIPA ACL

ACLs share notions coming from Speech Act Theory

Speech Act Theory

A speech act is an act that a speaker performs when making an utterance, including the following:

- A general act (illocutionary act) that a speaker performs, analysable as including
  - the uttering of words (utterance acts)
  - making reference and predicating (propositional acts), and
  - a particular intention in making the utterance (illocutionary force)
- An act involved in the illocutionary act, including utterance acts and propositional acts
- The production of a particular effect in the addressee (perlocutionary act)

[Linguistic dictionary]

Different Types of Speech Acts

- Representatives such as informing: *it is raining*
- Directives requesting the hearer to do something: *shut the door!*
- Commisives committing the speaker to do something: *I will be there*
- Expressives expressing the mental state of the speaker: *I am happy*
- Declarations declaring something: *I hate you*

FIPA ACL

Two parts in a FIPA ACL message:

1. Administrative part (sender, receiver, protocol, ontology, etc.)
2. Message part (communicative act, content)

An example:

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

Performative	Passing info	Requesting info	Negotiation	Performing actions	Error handling
Accept-proposal			x		
Agree				x	
Cancel		x		x	
Cfp			x		
Confirm	x				
Disconfirm	x				
Failure					x
Inform	x				
Inform-if	x				
Inform-ref	x				
Not-understood					x
Propose			x		
Query -if		x			
Query -ref		x			
Refuse				x	
Reject-proposal			x		
Request				x	
Request-when				x	
Request-whenever				x	
Subscribe		x			

Interaction Protocol Engineering

The aim of interaction protocol engineering is to develop protocols

Several phases are considered:

- Analysis
- Formal Description
- Validation
- Implementation
- Conformance Testing

Derived from communication protocol engineering [Holzmann, 1991] for tackling differences between agents and processes [Jennings and Wooldridge, 2000]

Analysis

Defines the requirements for the protocol

This informal phase makes use of natural language descriptions and sometimes chronograms, use cases or algorithms for the description of the requirements

The aim of this phase is to record:

- the definition of the protocol
- the good properties to be present
- the bad properties to avoid
- the type of the messages and the data
- the message sequences

A proposal for the analysis

A proposal for a requirements document in [Huget and Koning, 2003]

Several fields are described:

- **Name**: the name of the protocol.
- **Keywords**: a list of keywords for characterizing this protocol: auctions, secured, etc.
- **Agents**: the list of agents or roles which are involved in this protocol
- **Initiator**: the agent or the role which initiates the protocol
- **Prerequisite**: required conditions which must be true for initiating the protocol

A proposal for analysis

- **Function**: a short description of the meaning of this protocol
- **Behavior**: a detailed description of the protocol with all the messages used, the meaning of the message contents and what messages are possible for any interaction state
- **Conditions**: conditions which must hold during the execution of the protocol: deadlock freeness, liveliness, termination, etc.
- **Termination**: normal exit interaction states. For instance, the customer has the product and the merchant has the money

Use of the different fields during design cycle

	Formal description	Validation	Protocol synthesis	Conformance testing	Reuse
Name					Ö
Keywords					Ö
Agents	Ö		Ö		
Initiator	Ö		Ö		
Prerequisite		Ö	Ö	Ö	
Function					Ö
Behavior	Ö	Ö	Ö	Ö	
Conditions	Ö	Ö	Ö	Ö	
Termination	Ö	Ö	Ö	Ö	

Analysis

Not really tackled in the literature (neither for communication protocols nor for interaction protocols)

No methodology for helping designers

Need a lot of practice to handle this stage

Formal Description

Analysis generates informal documents (natural language description)

➔ ambiguities and errors can occur

➔ tools cannot use informal document

[Holzmann, 1991]

Need an accurate and rigorous tool:
formal description techniques

Formal description techniques

Several FDT available for protocol description (taken from communication protocol engineering or not):

- finite state machine, automaton, statechart, FIPA PDN
- Petri net
- LOTOS, Estelle, SDL, Z, PDL, APRIL, Grammar, FLBC
- Temporal logic, Event calculus, POS, 3APL
- UML based modeling languages: UML, UAML, Agent UML

Finite State Machine

Certainly, the simplest and the most known FDT.

A FSM can be considered as a graph.

Definition of a FSM [Salomaa, 1969]:

<Q, q0, A, T> where

Q is the set of states

q0 is the initial state

A is the alphabet of messages

T : Q x A → Q is the transition function

associated FSM:

Input/Output FSM [Lynch and Tuttle, 1988]

FSM Example

adapted from the buyer admission scene in Fishmarket [Noriega, 1998]

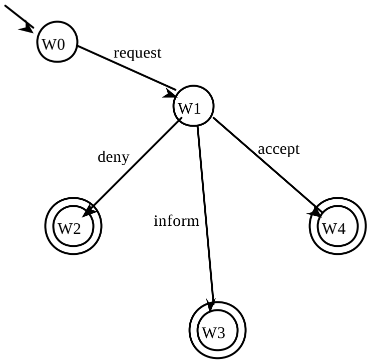
In this example:

Q = {W0, W1, W2, W3, W4}

q0 = {W0}

A = {request, accept, deny, inform}

T = { (W0, request) = W1,
(W1, deny) = W2,
(W1, inform) = W3,
(W1, accept) = W4 }



FSM in Interaction Protocols

COOL [Barbuceanu and Fox, 1995]: a FSM plus rules for governing the interaction (conversation rules for the semantics, error recovery rules for the unexpected messages, continuation rules for beginning new interaction within an interaction) and conversation classes which define what kind of rules and what states are available for a specific class of interaction

AgenTalk [Kuwabara et al., 1995]: Extended Finite State Machines, in order to encompass conditions on firing transitions

FSM in Interaction Protocols

COSY [Haddadi, 1996]. The main advantage of this approach is to consider a protocol as a composition of generic protocols such as requesting or informing

Advantages/Drawbacks of FSMs

- 👍 Easy to understand, easy method for describing protocols
- 👍 Semantics formally defined
- 👍 Several tools available for the design
- 👍 Several algorithms available for the validation (reachability analysis, model checking)
- 👎 Concurrency cannot be represented by FSMs
- 👎 Incoming and outgoing messages are not differentiated (except with Input/Output Automata)

Petri net

Petri nets are an extension of FSMs

Addition of synchronization and meeting points

Definition of Petri net [Jensen, 1991]

$<P, T, A, m_0>$ where

- P is a finite set of places
- T is a finite set of transitions
- $P \cup T = \emptyset$ $P \cap T = \emptyset$
- $A \subseteq (PxT) \cup (TxP)$ is the transition relation
- m_0 is the initial marking

Another kind of Petri nets

Colored Petri Net

- CPN = $<\Sigma, P,T,Color, A, m_0>$ where
- Σ is a finite set of types called color sets
- P is a finite set of places
- T is a finite set of transitions
- Color is the function that maps colors to places and transitions
- A is a finite set of arcs
- $A \subseteq (PxT) \cup (TxP)$ is the transition relation
- m_0 is the initial marking

Petri net Example

Produce/Consumer as a Petri net

P = {p1, p2, c1, c2, buffer}

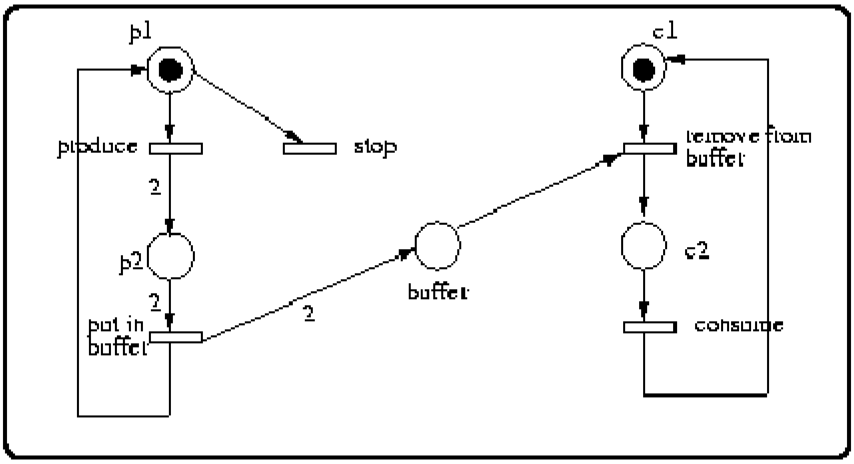
T = {produce, stop, put in buffer, remove from buffer, consume}

A =

	produce	stop	put in buffer	remove from buffer	consume
p1	1	1	0	0	0
p2	0	0	2	0	0
c1	0	0	0	1	0
c2	0	0	0	0	1
buffer	0	0	1	0	0

	p1	p2	c1	c2	buffer
produce	0	2	0	0	0
stop	0	0	0	0	0
put in buffer	1	0	0	0	2
remove from buffer	0	0	0	1	0
consume	0	0	1	0	0

Petri net Example



Petri nets in Interaction Protocols

[Cost et al., 1999]: Petri Nets are used since they allow designers to consider concurrency and verification. Colored Petri Nets are used in the context of negotiation

[Koning et al., 1998]: Petri Nets for the formalization and the validation of negotiation protocols

[El Fallah-Seghrouchni et al., 1998]: Colored Petri Nets to represent concurrency and to allow verification. These Colored Petri Nets are also employed to recognize patterns of interaction

[Ramos et al.] Colored Petri Nets used for dynamic interaction specifications and in a context of complex simultaneous interactions in Cooperative Information Systems

Advantages/Drawbacks of Petri nets

- 👍 Concurrency represented by synchronization and meeting points
- 👍 Semantics formally defined
- 👍 Several tools for the design and the validation
- 👍👎 Petri nets represented by matrices

LOTOS

- Standardized programming language (ISO/TC97/SC21/WG1) [Turner, 1989] based on
  - ACT ONE [Ehrig and Mahr, 1985] for the data
  - CCS/CSP based processus algebra [Milner, 1989] for the control
- Essentially used in distributed systems for designing synchronous systems
- A system in LOTOS is a set of processes. Interaction between processes is handled through gates
- Barely used in multiagent systems [Koning, 1999]

LOTOS Example

This example defines two classes AGENT1 and AGENT2, one instance of the class AGENT1 and three instances of the class AGENT2. The aim of this protocol is to broadcast the message ACK from the instance of the class AGENT1 to the three instances of the class AGENT2 [Koning, 1999]

```
specification EXAMPLE [AGENT1_0_AGENT_2_OUT,
        AGENT2_0_AGENT_1_IN,
        AGENT2_1_AGENT_1_IN,
        AGENT2_2_AGENT_1_IN ] : exit
behavior
        (
                AGENT1 [AGENT1_0_AGENT2_OUT]
                |||
                AGENT2 [AGENT2_0_AGENT1_IN]
                |||
                AGENT2 [AGENT2_1_AGENT1_IN]
                |||
                AGENT2 [AGENT2_2_AGENT1_IN]
        )
        ||
        (
                MEDIUM_AGENT2_AGENT1[
                    AGENT1_0_AGENT2_OUT,
                    AGENT2_0_AGENT1_IN,
                    AGENT2_1_AGENT1_IN,
                    AGENT2_2_AGENT1_IN
        )
where
        process MEDIUM_AGENT2_AGENT1 [
                AGENT1_0_AGENT2_OUT,
                AGENT2_0_AGENT1_IN,
                AGENT2_1_AGENT1_IN,
                AGENT2_2_AGENT1_IN ] : exit :=
                AGENT1_0_AGENT2_OUT ?M : ACK;
                AGENT2_0_AGENT1_IN !M ;
                AGENT2_1_AGENT1_IN !M ;
                AGENT2_2_AGENT1_IN !M;
                MEDIUM_AGENT2_AGENT1 [
                    AGENT2_0_AGENT2_OUT,
                    AGENT2_0_AGENT1_IN,
                    AGENT2_1_AGENT1_IN,
                    AGENT2_2_AGENT1_IN ]
[]
        exit
endproc
process AGENT1 [AGENT2_OUT]
        AGENT2_OUT ! ACK ;
        exit
endproc
process AGENT2 [AGENT1_IN]
        AGENT1_IN ? M : ACK ;
        exit
endproc
type ACK is
        sorts ACK
        opns
                ACK :→ ACK
endtype
endspec
```

Advantages/Drawbacks of LOTOS

- 👍 Standardized language
- 👍 Semantics defined
- 👍 Modularity and reusability of LOTOS code
- 👍 Tools available (CADP)
- 👍 Algorithms for validation available
- 👎 Difficulty to read the LOTOS code
- 👎 Difficulty to learn the LOTOS language

Temporal logic

Approach only used in interaction protocols

Best example : Concurrent METATEM framework
[Finger et al., 1993]

- 👍 Representation of past, present and future moments in protocols
- 👍 Allow validation of properties by theorem proving
- 👍 Semantics formally defined
- 👎 Not reusable due to variables

Temporal logic Example

Contract Net protocol in temporal logic [Fisher and Wooldridge, 1994]

start $\Rightarrow$ announce(task(Name, Desc, Req, self))

$(\neg$award(T,A)$)$ $\mathcal{S}$ announce(T) $\wedge$ competent(A,T) $\Leftrightarrow$ possible(A,T)

possible(self, T) $\Rightarrow$ bid(T, self)

bidded(T,X) $\wedge$ most-preferable(T,Y) $\Rightarrow$ award(T,Y)

$(\neg$completed(T,self,X) $\wedge$ $\neg$split(T,T1,T2)$)$ $\mathcal{S}$ award(T, self) $\wedge$ $\odot$result(T,R) $\Rightarrow$ completed(T,self, R)

UML based modeling languages

Several UML based modeling languages:

- UAML [FIPA, 1999]
- Agent UML [Odell et al., 2000]
- EAUML [Wei et al., 2001]
- UAMLe [Koning et al., 2001a]

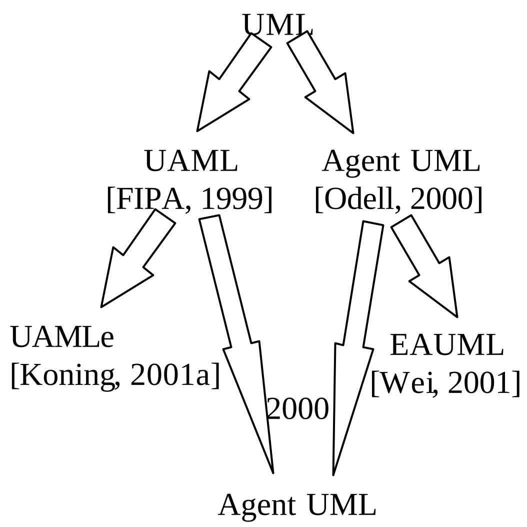
Why based on UML? [Odell et al., 2000]

– UML is well known by software engineers and multiagent system designers are frequently software engineers: capitalizing on knowledge

Why not UML? [Odell et al., 2000]

– agents and objects are too different, some notions unavailable in UML

UML based language taxonomy



Comparison between UML based languages

	Odell's AUML	Wei's EAUML	FIPA's UAML	Koning and Huget's UAMLe
Roles	All			
Synchronous/ Asynchronous	Both	Asynchronous	Both	
Concurrency	Specific connector		Messages in boxes	
Loop	At the level of messages and groups of messages			
Time	Through deadlines			
Exception		Connector		Connector
Design	UML tools	No graphical tools		DIP
Validation	No tools	Translation into Promela, use of Spin	No tool	Translation into FSM, Petri nets and Promela, use of Spin
Protocol Synthesis	No code generated	Code generation	No code generated	Code generation

[Koning et al., 2001a]

Agent UML

Several notions:

- agents described by their instances or by their roles in the protocol *instance / role : class* (UML-Airlines / Auctioneer : Seller)
- messages from sender role to receiver roles
- constraints on messages and on nested protocols, multiplicity for the number of copies of the messages, nested and interleaved protocols, messages sent in parallel, choice between several messages

http://www.auml.org

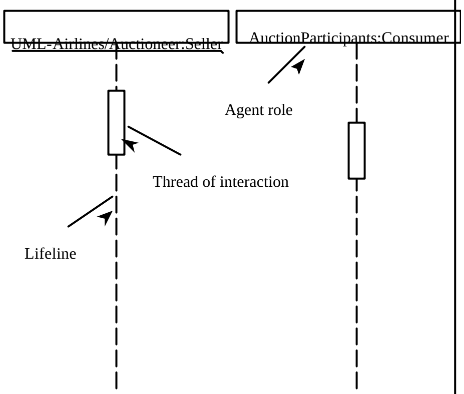
Agent UML is on the way to be standardized at the FIPA, see FIPA Modeling TC (Agent UML 2 based on UML 2)

Agent UML elements

Agents or agent roles, defined in a box

Lifelines define the time period during which an agent exists. Lifelines exist for an agent as long as the agent is present in the interaction. Lifelines rendered as a dashed vertical line

Threads of interaction show the period during which an agent is performing some tasks as a reaction to an incoming messages. Threads are rendered as a thin vertical bar linked to lifelines



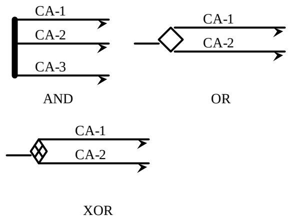
Agent UML elements

Connectors: AND, OR, XOR

AND: parallelism between several messages

OR and XOR: decision between several messages. OR: zero or several messages selected (and sent in parallel). XOR: one message selected

Conditions defined through a textual string or formally with OCL (Object-Constraint Language), allow to select the path to follow in the protocol diagram. Conditions define within curly brackets



Agent UML elements

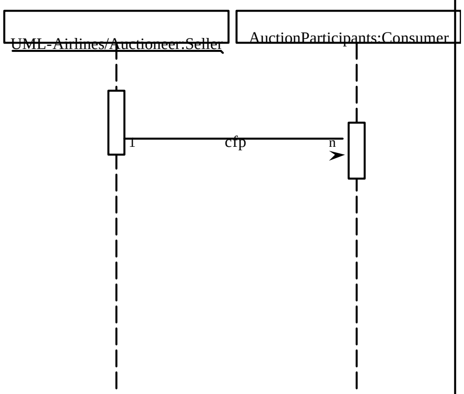
Multiplicity allows to send several copies of the same message

Two numbers at both ends of an arrow: corresponding to the number of messages sent from the sender and the number of receivers

Special value ‘*’ denotes that zero or more copies are sent

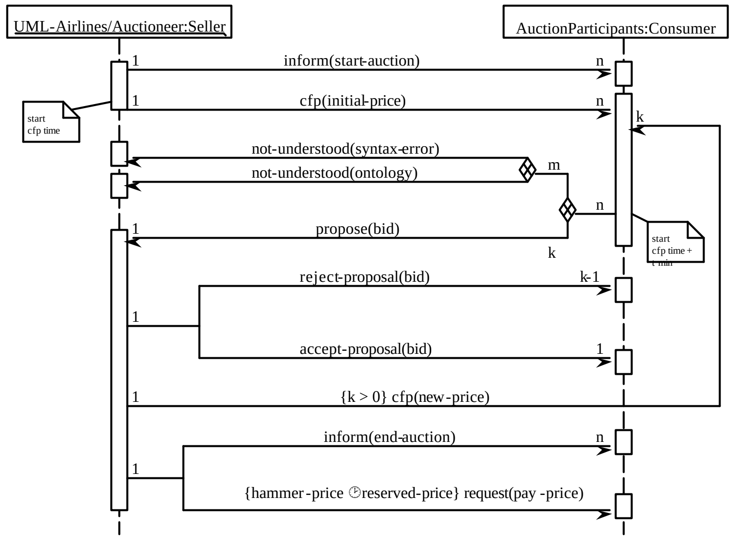
Range can be used: 1..5, from 1 to 5 messages have to be sent or received

Type of delivery, nested and interleaved protocols, templates [Bauer et al., 2000]



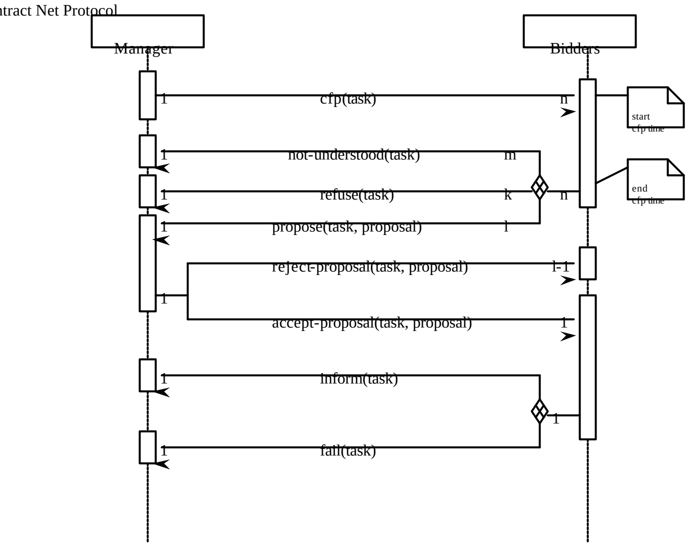
Agent UML Example

English Auction Protocol for surplus flight tickets



Agent UML Example

Contract Net Protocol



Advantages/Drawbacks Agent UML

- 👍 Easy to understand, steep learning curve for software engineers
- 👍 Several tools for the design and the generation of code
- 👍 Based on UML which is well-known
- 👎 Few work on validation [Huget, 2002a] [Koning and Romero, 2002], few work on this FDT
- 👎 No complete formal semantics yet (some work on UML however)

Tools for the formal description phase

- Automata: CoLos
  http://homes.dsi.unimi.it/~alberti/colos/automata.html
- Petri net: Design-CPN
  http://www.daimi.au.dk/designCPN
- LOTOS: CADP
  http://www.inrialpes.fr/vasy/cadp
- Temporal logic: your favorite text editor
- Agent UML: Visio
  http://www.microsoft.com/office/visio/default.asp

Formal Description

Certainly the main question: which one do we have to choose?

Answer: no definitive answer…

Several reasons:

- Depends greatly on the use and the kind of protocols (description for documentation, for validation, for code generation) (simple, concurrency, reliability, open systems)
- Depends on the background of the designer (formal skills, software engineering skills)

Validation

Designers have a formal description **but** it is not sure that the properties are present or absent of the formal description

Aim of this stage: checks if good properties are present and bad properties are absent

Good properties: properties based on the goal of the protocol, mutual exclusion, termination

Bad properties: deadlock, standard non-conformance

Validation (Pseudo-)Algorithm

P is the formal description of the protocol

p is the properties to check on this formal description

M is the model of the formal description

1. Generating the model M from P
2. Representing the property p as a graph
3. Verifying that the property p holds in M

Difficulties: how to generate M? How to generate the graph of p? How to validate p on M?

Validation

Properties can be sort of in two classes:

- Structural properties, usually general properties (termination, deadlock): validation by generating a graph property from this structural property
- Semantic properties, depending on the goal of the protocol: validation by generating a graph from this property via a temporal logic formula

Structural properties verified via reachability analysis [Holzmann, 1991]

Semantics properties verified via model checking [Clarke et al., 2000]

Reachability analysis

This technique defines the reachability graph

The graph is defined from the finite state automaton

Each state in the graph corresponds to a state in the interaction

Each transition in the graph corresponds to a sending or a receiving message

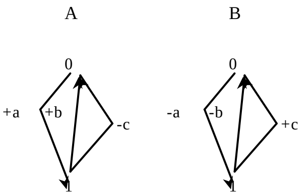
Generating the graph corresponds to an exhaustive depth-first search in the finite state automaton

Algorithm for the reachability analysis

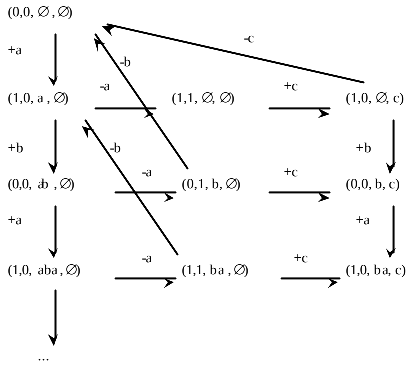
Holzmann's algorithm [Holzmann, 1991]:

```
start() {
    W = initial state; /* work set: to be analyzed */
    A = ∅; /* previously analyzed states */
    analyze();
}
analyze() {
    if (W is empty) return;
    q = last element of W;
    add q to A;
    if (q == error state)
        report-error();
    else
        for (each successor state s of q)
                if (s is not in A or W)
                        add s to W;
                        analyze();
    delete q from W;
}
```

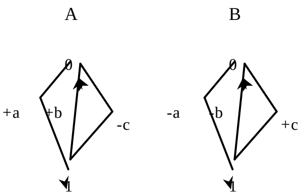
Reachability Analysis Example



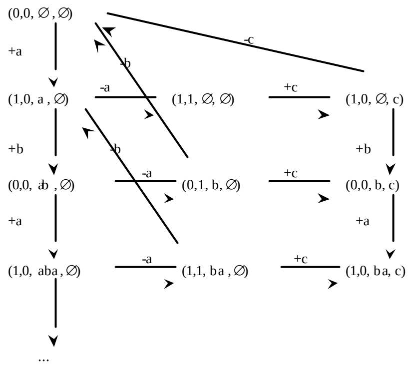
+ : send
- : receive
a : open request
b,c : close request



Reachability Analysis Validation



+ : send
- : receive
a : open request
b,c : close request



There is at least one termination: the current state is 0 and the last input is b or c

Model checking

Uses the graph of the reachability analysis: the model

Define the property as a temporal logic formula

Model checking = checks that a given finite state model satisfies a given formula.

Several operators for the formula:

[] *always* <> *eventually* X *next time* U *until*

A *for all* E *exists*

depends greatly on the temporal logic used: CTL, CTL*: usually CTL

Temporal logic formulae

if crit_A and crit_B represent the critical section of A and B

[] (crit_A && crit_B) means

it is impossible that both A and B are in their critical section at the same time

test if the final state is reachable:

[] ((states == Wi) → [] ¬exit)

we test the opposite, it is more convenient in model checking (and in Spin in particular)

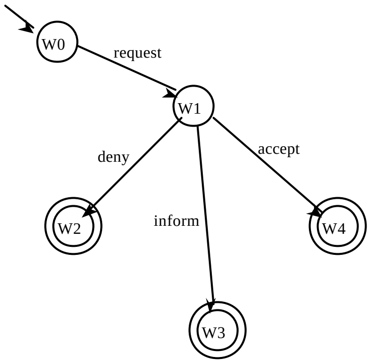
Model checking Example

<> (state == W2)

<> (state == W3)

<> (state == W4)

(<>state == W2 ||
<>state == W3 ||
<>state == W4)



Model checking Tools

Several model checkers available (tool for model checking systems):

SMV [McMillan, 1992]

SPIN/PROMELA [Holzmann, 1991] [Holzmann, 1997]

CADP [Fernandez et al., 1991] with LOTOS and FSM

Validation

One of the main domain of work with formal description in interaction protocol design

Several different approaches usually considering different tools: NuSMV, SMV, SPIN

Usually, validation does not take into account agents and multiagent systems (one exception among others [Wooldridge et al., 2002]

Protocol Synthesis

The output of previous stage: a formal description verified

Time to implement this protocol within agents

Aim of this stage: generates a program that behaves like the formal description

Generates code for usual programming languages: C, C++, Java

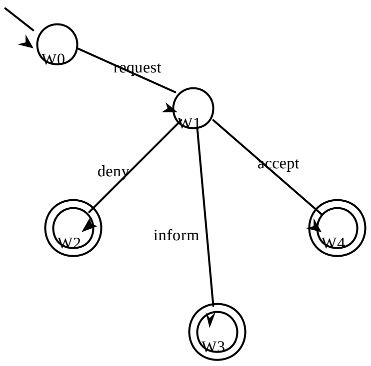
Protocol Synthesis

Two phases in this stage:

- Automatic generation of the protocol skeleton
- Manual generation of the protocol semantics. The semantics corresponds to the actions when moving from one state to another one

Protocol Synthesis Example

```
protocol() {
  while(state != W2 || state != W3 ||
        state != W4) {
    switch(state) {
        case W0:
          printf(‘‘request’’);
          state = W1;
          break;
        case W1:
          if (scene_closed) {
            printf(‘‘inform’’);
            state = W3;
          }
          else {
            if (admission) {
              printf(‘‘accept’’);
              state = W4;
            }
            else {
              printf(‘‘deny’’);
              state = W2;
            }
          }
          break;
    }
  }
}
```



Protocol Synthesis

Difficult stage since for multiagent systems, it has an impact on agents, it is not *really* possible to generate code without considering agents

The protocol semantics need tight relationships with knowledge and means-end reasoning within agents

Few work exist in interaction protocol design: FIPA, [Huget, 2001], generally done without considering agents

Second approach in [Huget, 2001]: directly executing the formal description

Main reference: Chu's thesis [Chu, 1989]

Conformance Testing

Previous stage gives you a program that behaves **normally** like the formal description

but:

- Adding code can insert errors or misconceptions
- Programs have to be conformed to standards
- Are programs deadlock-free?
- Are all the requirements document properties present?

We need to verify the programs as well via the conformance testing stage

Conformance Testing

Conformance testing stage is **equivalent** to the validation stage **except** it concerns the program and not the formal description.

This stage is totally forgotten during interaction protocol design

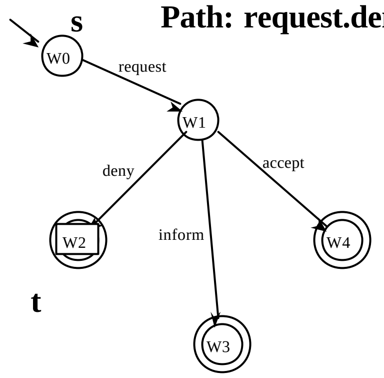
Conformance Testing Algorithm

S is the formal specification of the protocol
P is the program corresponding to the protocol
s and t are two states of S
s is the initial state of S, t is the state to be checked
find a path from s to t
find all the outgoing transitions from t, store the corresponding states
apply the path to P
apply each outgoing transition
test if states of P correspond to the state of S

Conformance Testing Example

```
protocol() {
  while(state != W2 || state != W3 ||
          state != W4) {
    switch(state) {
          case W0:
             printf(‘‘request’’);
             state = W1;
             break;
          case W1:
             if (scene_closed) {
              printf(‘‘inform’’);
              state = W3;
            }
             else {
               if (admission) {
                printf(‘‘accept’’);
                state = W4;
              }
              else {
                printf(‘‘deny’’);
                state = W2;
              }
            }
            break;
    }
  }
}
```

Path: request.deny



P **S**

Conformance Testing

This stage has some similarities with software testing

Main reference on the subject:

Jan Tretmans (http://www.cs.kun.nl/~tretmans)

[Tretmans, 1992]

Designers have to be cautious: conformance testing says that requirements document properties are present or are absent **but** do not say if some other properties not checked are absent

Main difficulty: protocols are embedded within agents, problems can come from agent implementation.

Conformance Testing

Conformance testing has some similarities with ACL conformance testing [Wooldridge et al., 1998] [Wooldridge, 1999] [Huget and Wooldridge, 2003]

Applications

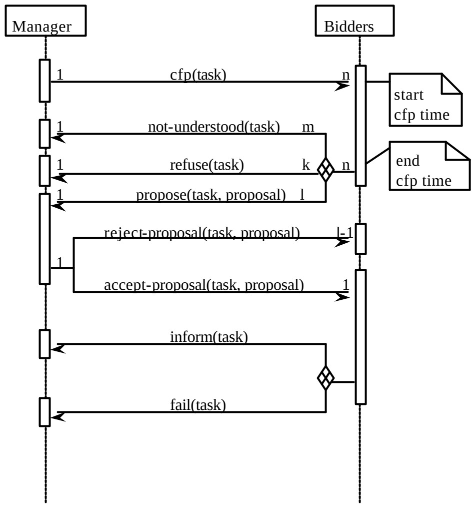
Examples of interaction protocols

Negotiation: Contract Net protocol

[Davis and Smith, 1983]

Aim: contracting agents doing tasks that agents are unable to do alone

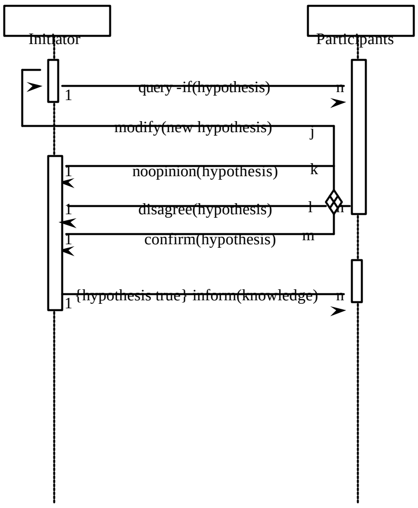
Protocol used in electronic commerce and robotics



Examples of interaction protocols

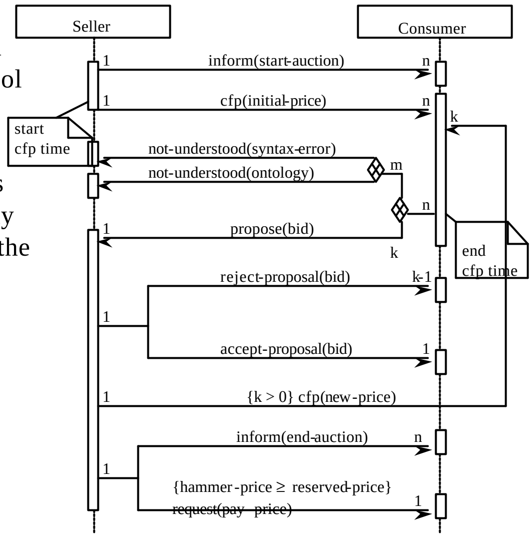
Learning: Sian's protocol [Sian, 1991]

Aim: one agent makes an assertion, agents give their opinion about the assertion, a final decision is made according to opinions

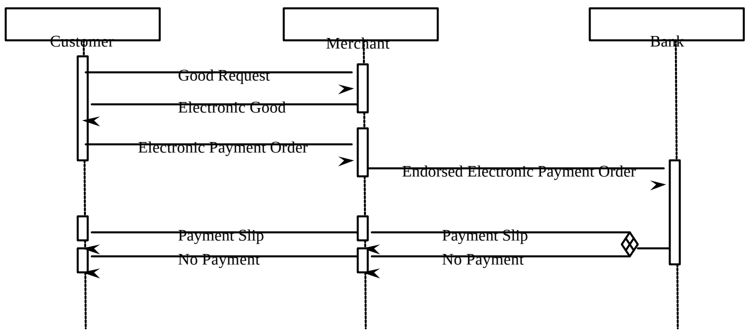


Examples of interaction protocols

- Auctions: English Auction protocol, Dutch Auction protocol, Vickrey protocol
- Aim: selling goods by auctions, one agent proposes an item, agents bid on this item until only one agent agrees to pay the current bid
- Protocols used in electronic commerce or resource sharing



Examples of interaction protocols



Electronic commerce: NetBill protocol

[Cox et al., 1995]

Aim: selling small amount value electronic goods on Internet. Involving three kind of agents: consumers, merchants and banks. This protocol is secured.

Interaction and Agent Design

There is still a *divorce* between interaction design and agent design.

Most of the methodologies are agent-oriented design:

First, agent design

Second, the remaining

There is no real impact on agents and multiagent systems when designing interactions.

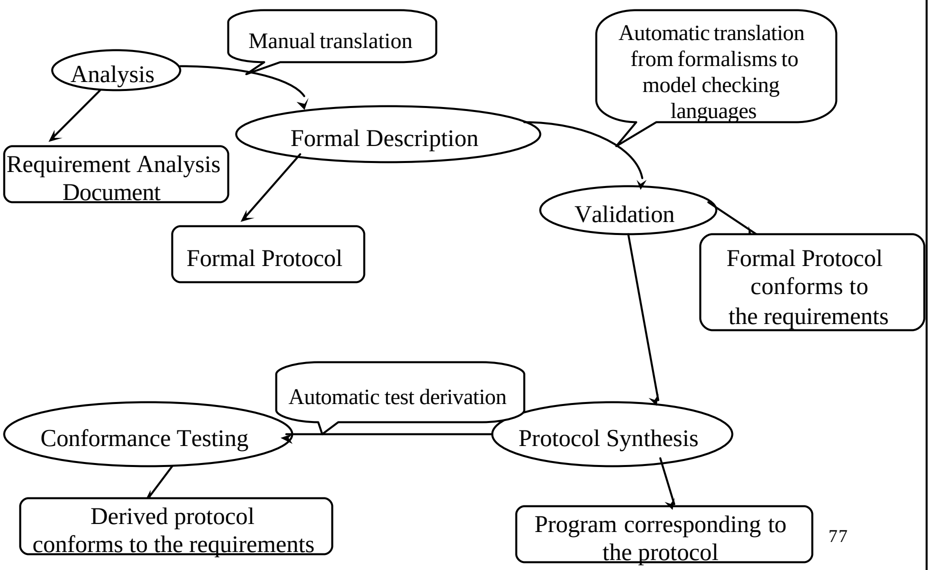
Conclusion

Description of a full design-cycle for interaction protocol engineering: from analysis to implementation and test

Use communication protocol engineering if possible (formalisms, model checking, implementation, conformance testing)

Derive an interaction protocol engineering for tackling differences between agents and processes

Conclusion



Future directions

Specific agent formal description techniques emerge: a lot of work to do. Few works on validation, on protocol synthesis for Agent UML. No dedicated tools for Agent UML (ongoing effort at the FIPA for a standardization)

Conformance testing is more or less forgotten by designers: work have to be done on this subject

What maintenance do in the context of interaction protocols?

Future directions

Few works on the design of interaction protocols within agents: protocol management, interaction management. Some scarce proposals:

[Müller, 1996], [Skarmeas and Clark, 1999], [Huget, 2002]

Verification that agents correctly implement an agent communication language or interaction protocols [Wooldridge, 1998] [Wooldridge, 1999]

Interoperability testing in communication protocol engineering: what does it mean in the context of agents and interaction protocols?

References

[Barbuceanu and Fox, 1995] M. Barbuceanu and M.S. Fox, «COOL: a language for describing coordination in multiagent systems». In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), AAAI Press, 1995.

[Bauer et al., 2000] B. Bauer and J. Müller and J. Odell, «An extension for UML by protocols for multiagent interaction». In Proceedings of Fourth International Conference on Multi-Agent Systems (ICMAS-00), 2000.

[Chu, 1989] P.-Y. M. Chu, "Towards Automating Protocol Synthesis and Analysis", Ohio State University, 1989.

[Clarke et al., 2000] E. Clarke and O. Grumberg and D. Peled, *Model checking*. MIT Press, 2000.

[Cost et al., 1999] R.S. Cost and Y. Chen and T. Finin and Y. Labrou and Y. Peng, «Modeling agent conversation with colored Petri nets». In Proceedings of Autonomous Agents Workshop on Conversation Policies, 1999.

[Cox et al., 1995] B. Cox and J.D. Tygar and M. Sirbu, «Netbill security and transaction protocol». In Proceedings of the First USENIX Workshop on Electronic Commerce. 1995.

[Davis and Smith, 1983] R. Davis and R.G. Smith, «Negotiation as a metaphor for distributed problem-solving». In Artificial Intelligence, 20, 1983.

[El Fallah-Seghrouchni et al., 1999] A. El Fallah-Seghrouchni and S. Haddad and H. Mazouzi, «A formal study of interaction in multi-agent systems». In Proceedings of Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-99), 1999.

References

[Erhig and Mahr, 1985] H. Erhig and B. Mahr, *Fundamentals of Algebraic Specification*. Springer-Verlag, 1985.

[Fernandez et al., 1991] J.-C. Fernandez and H. Garavel and L. Mounier and A. Rasse and J. Sifakis, «Une boîte à outils pour la vérification de programmes LOTOS». Technical Report, INRIA, 1991.

[Finger et al., 1993] M. Finger and M. Fisher and R. Owens, «MET ATEM: modeling reactive systems using executable temporal logic». In Proceedings of International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE), 1993.

[Finin et al., 1994] T. Finin and R. Fritzson and D. McKay and R. McEntire, «KQML as an agent communication language». In Third International Conference on Information and Knowledge Management (CIKM-94), ACM Press, 1994.

[FIPA, 1999] Agent Communication Language Specification. Foundation for Intelligent Physical Agents, 1999.

[FIPA, 2000] Agent Communication Language Specification. Foundation for Intelligent Physical Agents, 2000.

[Fisher and Wooldridge, 1994] M. Fisher and M.J. Wooldridge, «Specifying and executing protocols for cooperative actions». In Proceedings of the International Working Conference on Cooperative Knowledge-Based Systems (CKBS-94), 1994.

References

[Holzmann, 1991] G.J. Holzmann, *Design and validation of computer protocols*. Prentice-Hall, 1991.

[Holzmann, 1997] G.J. Holzmann, «The model checker SPIN». In IEEE Transactions on Software Engineering, 23(5), 1997.

[Huget, 2002] M.-P. Huget, «Design agent interaction as a service to agents». In Proceedings of workshop on Agent Communication Languages and Conversation Policies (ACL-02), 2002.

[Huget, 2002a] M.-P. Huget, «Model Checking Agent UML Protocol Diagrams». In Proceedings of workshop Model Checking Artifical Intelligence (Mochart-02), 2002.

[Huget and Koning, 2003] M.-P. Huget and J.-L. Koning, "Requirement Analysis for Interaction Protocols". In Proceedings of CEEMAS 2003, 2003.

[Huget and Wooldridge, 2003] M.-P. Huget and M.J. Wooldridge, "Model Checking for ACL Compliance Verification". In ACL 2003 Proceedings, 2003.

[Jennings and Wooldridge, 2000] N.R. Jennings and M.J. Wooldridge, «Agent-oriented software engineering». In *Handbook in Agent Technology*, J. Bradshaw (ed.), MIT Press, 2000.

[Jensen, 1991] K. Jensen, *High-level Petri nets: theory and application*. Springer-Verlag, 1991.

[Koning et al., 1998] J.-L. Koning and G. François and Y. Demazeau, «Formalization and pre-validation for interaction protocols in multiagent systems». In Proceedings Thirteenth European Conference on Artificial Intelligence (ECAI-98), 1998.

References

[Koning, 1999] J.-L. Koning, «Algorithms for translating interaction protocols into a formal description». In Proceedings of IEEE Conference on Systems, Man and Cybernetics (SMC-99), 1999.

[Koning et al., 2001] J.-L. Koning and M.-P. Huget and J. Wei and X. Wang, «Engineering electronic commerce interaction protocols». In Proceedings of Intelligent Agents, Web Technology and Internet Commerce (IAWTIC-01), 2001.

[Koning et al., 2001a] J.-L. Koning and M.-P. Huget and J. Wei and X. Wang, «Extended modeling languages for interaction protocol design». In Proceedings of Agent-Oriented Software Engineering (AOSE-01), 2001.

[Kuwabara et al., 1995] K. Kuwabara and T. Ishida and N. Osato, «AgenTalk : describing multiagent coordination protocols with inheritance». In Seventh International Conference on Tools for Artificial Intelligence (ICTAI-95), 1995.

[Lynch and Tuttle, 1988] N.A. Lynch and M.R. Tuttle, «An introduction to Input/Output automata». Technical Memo MIT/LCS/TM-373, MIT, 1988.

[McMillan, 1992] K. McMillan, «Symbolic model checking: an approach to the state explosion problem». PhD Thesis, Carnegie Mellon University, 1992.

[Milner, 1989] R. Milner, *Communication and concurrency*. Prentice-Hall, 1969.

[Müller, 1996] J. Müller, *The design of intelligent agents - a layered approach*. LNAI 1177, Springer-Verlag, 1996.

References

[Noriega, 1998] P. Noriega, «Agent Mediated Auctions: The Fismarket Metaphor». PhD Thesis, Universitat Autonoma de Barcelona, 1998.

[Odell et al., 2000] J. Odell and H. V.D. Parunak and B. Bauer, «Extending UML for agents». In Proceedings of Agent-Oriented Information Systems Workshop (AOIS-00), 2000.

[Ramos et al., 1998] F. Ramos and J. Fausto and F. Camargo, «A Methodology for Modeling Interactions in Cooperative Information Systems Using Coloured Petri Nets», International Conference on Software Engineering and Knowledge Engineering, 1998.

[Salomaa, 1969] A. Salomaa, *Theory of Automata*. Pergamon Press, 1969.

[Searle, 1969] J. Searle, *Speech act: an essay in the philosophy of language*. Cambridge University, 1969.

[Shannon, 1948] C.E. Shannon, «A mathematical theory of communication». In The Bell System Technical Journal, 1948.

[Sian, 1991] S.S. Sian, «Adaption based on cooperative learning in multi-agent systems», In Decentralized AI 2, 1991.

[Skarmeas and Clark, 1999] N. Skarmeas and K. Clark, «Component based agent construction». In Journal of Autonomous Agents and Multi-Agent Systems, 1999.

References

[Tretmans, 1992] J. Tretmans, «A formal approach to conformance testing». PhD Thesis, University of Twente, 1992.

[Turner, 1989] K. Turner, «The formal specification language LOTOS: a course for users». Department of Computing Science and Mathematics, University of Stirling.

[Wei et al., 2001] J. Wei and S.C. Cheung and X. Wang, «Towards a methodology for formal design and analysis of agent interaction protocols: An investigation in electronic commerce». In Proceedings of the International Software Engineering Symposium, 2001.

[Wooldridge, 1998] M. Wooldridge, «Verifiable Semantics for Agent Communication Languages». In Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98), 1998.

[Wooldridge, 1999] M. Wooldridge, «Verifying that Agents Implement a Communication Language». In Proceedings of AAAI, 1999.

[Wooldridge etal., 2002] M. Wooldridge and M. Fisher and M.-P. Huget and S. Parsons, "Model Checking Multiagent Systems with MABLE". In Proceedings of AAMAS 2002, 2002.