Interaction Protocol Design: Application to an Agent-Based Teleteaching Project

Jean-Luc Koning
INPG-CoSy
50, rue Laffemas, BP 54
26902 Valence Cedex 9, France
Jean-Luc.Koning@esisar.inpg.fr

Marc-Philippe Huget
Agent ART Group, Department of Computer Science
University of Liverpool
Liverpool L69 7ZF, United Kingdom
M.P.Huget@csc.liv.ac.uk

Abstract

In this paper we focus on the analysis and design stages of the protocol engineering development cycle. We start by sketching an application framework dedicated to a web-based learning environment called Baghera whose aim is to teach geometry problems. We then apply our protocol engineering process to protocols for checking mathematical proofs a student happens to build. The following section discusses the analysis stage of such a protocol. We then briefly introduce our component-based formal specification language in order to then describe the protocol's design stage. Finally we present a tool built upon the FIPA norm (making use of the PDN or UAML notation) which supports the analysis and design of interaction protocols.

1. Introduction

1.1. Interaction Protocol Development Cycle

The development cycle of interaction protocols for multiagent systems does not account for as large a literature as the one dedicated to communication protocols in distributed systems. Let us quote Singh's precursory work on interaction oriented programming [11] and also [2] where protocol engineering comprises three main stages.

1. **Design and validation.** A dedicated way to tackle stage 1 is through the use of colored Petri nets since such a formalism supports concurrent processing. Besides a whole set of validation tools is available.

2. **Observation of the protocols' execution.** Stage 2 deals with a post-mortem analysis of the message scheduling.

3. **Recognition and explanation of conversations.** Stage 3 checks whether the interactions unfolded according to the protocol and that the overall behavior corresponds to what the designer expected. It also highlights the agents' behavior during those interactions and helps pinpoint possible causes of failure.

Because of the quite distinctive nature of the two sets of protocols found in distributed and multiagent systems, it is not possible to fully apply results from works in communication protocols to interaction protocols. Therefore, it is necessary to define a suited development cycle that, when possible, makes use of existing techniques from distributed systems, or otherwise derives new ones.

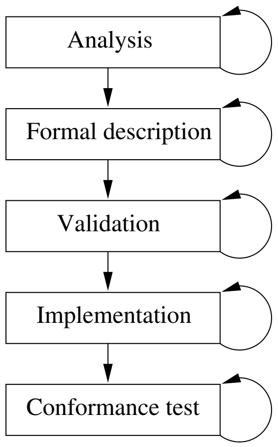


Figure 1. Interaction protocol development cycle.

The development cycle we advocate for interaction protocols consists in five main stages as shown on Figure 1 [5]. The *analysis* stage consists in a natural language description of the protocol to be designed. It identifies the various agents' role, the message exchanges, and their contents as well [6]. The next stage consists in deriving the protocol's *formal specification*. This aims at getting rid of ambiguities that might have been introduced earlier due to the natural language description. Then comes a *validation* stage which checks the protocol's structural and behavioral properties. The *implementation* stage, consists in building a protocol's executable version that will be integrated in the agents [4] and finally, the *conformance test* checks whether the implemented version of the protocol is conform to the properties defined earlier in the analysis stage. As for communication protocols, it turns out that the two crucial stages are the specification and validation.

1.2. Paper's Contents

In this paper we only focus on the analysis and design stages of the protocol engineering development cycle. We start by sketching an application framework dedicated to a web-based learning environment called Baghera whose aim is to teach geometry problems (see Section 2.1). We then apply our protocol engineering process to protocols for checking mathematical proofs a student happens to build. Section 2.2 discusses the analysis stage of such a protocol. We then briefly introduce our component-based formal specification language (see Section 3) in order to then describe the protocol's design stage (see Section 4.1). Finally we present a tool built upon the FIPA norm (making use of the PDN or UAML notation) which supports the analysis and design of interaction protocols (see Section 4.2).

2. Baghera: A Web-Based Learning Setting

2.1. Overview of the Baghera Application

The aim of the BAGHERA project [14] is to build a decentralized system which supports distance learning [1]. Such an application is originally dedicated to the teaching of eighth-grade geometry problems when a student has an enforced long stay at the hospital. In order to avoid wasting a whole school year those students are given the opportunity to log in on a local workstation. From there they are able to reach a set of new or partially known geometry problems as well as pieces of knowledge related to currently or already studied topics.

The multiagent system Baghera gathers a diversity of actors (professors of various competencies and students of various levels) and types of actors (artificial and human) spread on a network. Compared to current tele-teaching platforms Baghera's multiagent approach introduces new features. First, it affords a spatial distribution and mobility of the students and professors. They do not have to be in a given location. Second, the frequency, duration and time of login are left open for the students as well as the professors. There is no requirement to be connected at a given time. Third, it ensures a personalized learning scheme follow-up which allows each student to receive help and suggestions that take into account his/her own learning trajectory, and to support a professor's help to a student by providing him/her with a piece of information that is relevant for that student as far as the current reasoning situation s/he is in, or, more globally, his/her learning progress.

[1]http://www-baghera.imag.fr

As opposed to an interaction via a video channel for instance where a professor and a student need to rebuild a common setting in order to be able to efficiently interact (student level, special difficulties related to a knowledge domain and the current problem, etc.). Such a multiagent system contributes to this construction by providing professors pieces of a student's history or by helping to match in a relevant way a student to a professor, or even a student to another student when this pairing is enough to make the student's learning progress.

By means of such a computer system (see Figure 2), students are in direct relation with several artificial agents.

- A *companion* agent assists a student by providing him/her a graphical user interface that will help interact with the other agents and mostly with the tutor agent. The companion agent is also in charge of managing the student's school bag which encompasses both the already and yet to be solved geometry problems as well as some didactic information such as his/her knowledge level and his/her knowledge background (the theorems, axioms, proofs s/he knows).
- A *tutor* agent has clearly a didactic purpose. It coordinates the rest of the student's agents and guides his/her learning scheme via the companion agent by providing assignments that comply with his/her level. It is endowed with dialog capabilities in order to manage the interaction with professors or possibly some other students. One of its main goals is to rebuild the learning setting of a student (what s/he is knowing or lacking) whenever a professor needs it. It also possesses negotiation capabilities with the verificator agent in order to shape the derived proofs given the student's knowledge profile. It steps into the students' reasoning process if they take a wrong path in their problem solving or if they have a hard time understanding a proof. Therefore it keeps a close look at what a student is doing and what is going on in the system. It takes into account the verificator's viewpoint in order to guide the students and lead them in the right procedure towards a solution.
- A *verificator* agent analyzes the validity and coherence of a proof given by a student. It allows to build analogies between what a student is currently doing and the

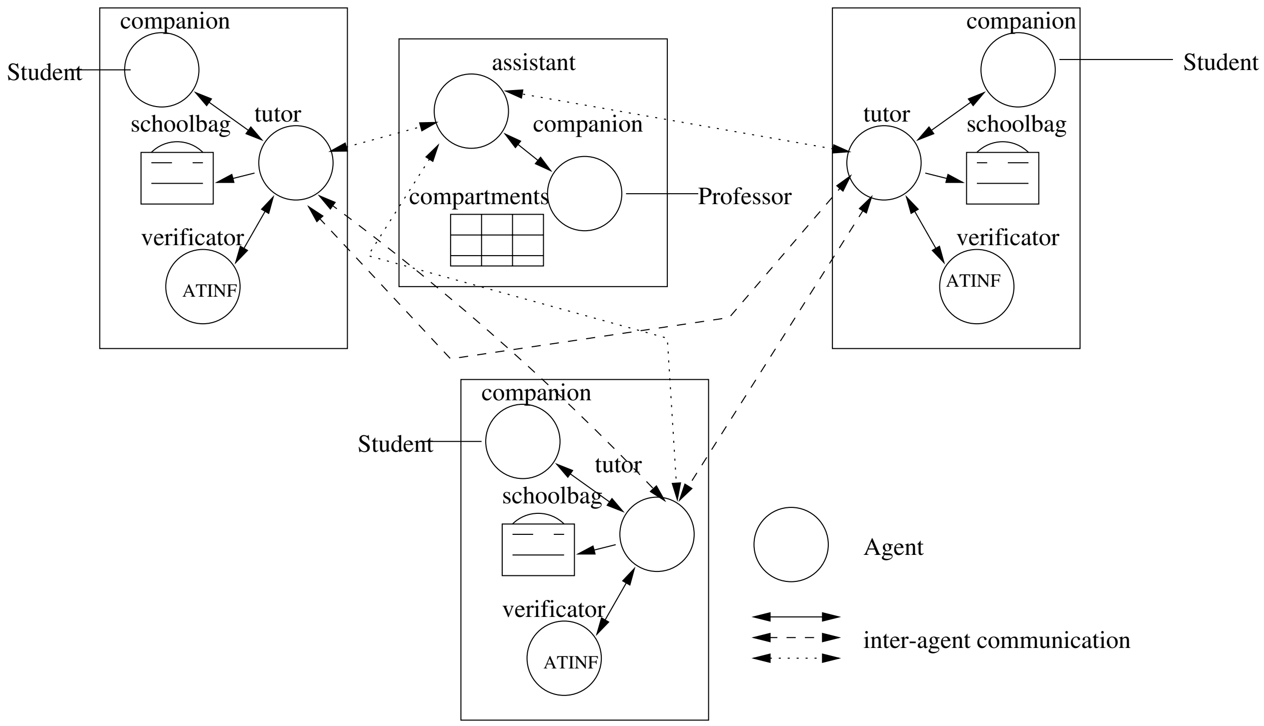


Figure 2. Baghera web-based learning system.

proofs s/he may have obtained in the past. It encapsulates a resolution module that helps in building mathematical proofs by suggesting whole or partial demonstrations. Such an agent is a formal reasoning tool capable of refutation, belief revision, and proposing counter-examples.

In a similar fashion, professors are assisted by a *companion* agent which help them interact with their *assistant* agent via a graphical user interface. This latter agent's purpose is to guide a student's learning progress through his/her tutor agent. Each professor owns a set of compartments that hold geometry problems arranged according to their type and difficulty level.

Baghera's approach consists in allowing the tutor agent to call into play the verificator's services. The scenario between a student and a computer is thus rather general compared to the one where a student receives an evaluation from a professor several days after turning an assignment back.

2.2. Deriving Baghera's Protocols From Possible Scenarios

Now that we have presented the actual agents and their role, in order to complete the protocol's analysis stage we are going to describe how the protocol unfolds (see Figure 3).

In the Baghera application, a standard scenario is when a student builds a mathematical proof for a given geometry problem and wishes to know whether is valid or not.

The student starts typing in a proof by means of its companion agent via an intelligent interface[2] dedicated to the handling of 2-D mathematical objects that is capable of performing syntactical checking (e.g., a square A, B, C, D has 2 diagonals AC and BD but not AB or CD). Once the proof is considered syntactically correct the tutor agent passes it to the related verificator agent which carries out the proof validation process[3]. Then, the verificator agent verifies the proof and passes on its analysis back to the student via her/his tutor and companion agent.

Of course such a scenario involves some inherent translation between the various formalisms. For example, the intelligent 2-D editor generates formula that need to be understood (i.e., translated) by the verificator agent in order for it to analyze them. And vice versa this analysis has to be understood (i.e., translated) by the companion to inform the student.

It may happen that for some reason the verificator agent is not capable of validating the mathematical proof. It thus informs the tutor agent which then invokes its counterparts (the other tutor agents) to see whether they have previously

[2]http://www-cabri.imag.fr/index-e.html
[3]http://www-leibniz.imag.fr/ATINF/Welcome-ENG.html

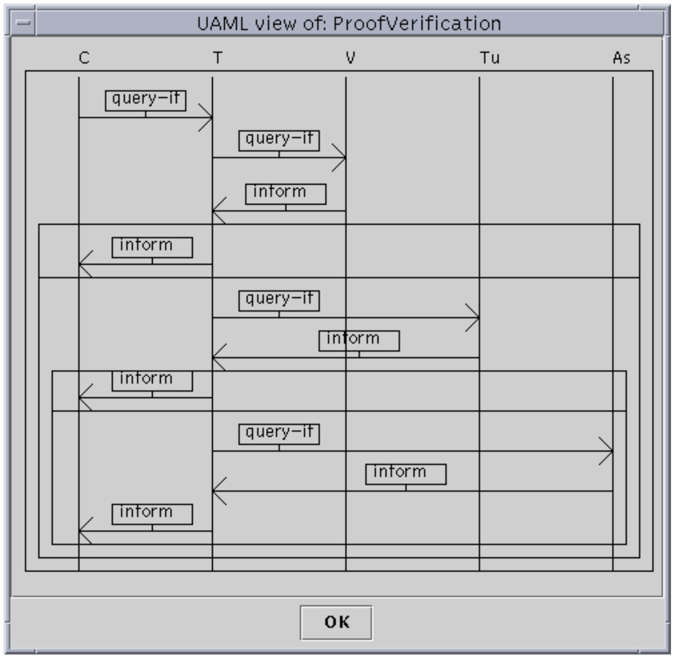


Figure 3. Complete protocol for analyzing a mathematical proof.

derived a solution to the problem at hand. If so, the tutor supplies the companion agent with it. If not, then the tutor requests the same information from the assistant agents and get back to the companion.

Let us note that such a protocol brings into play four agent roless, namely the companion agent, the tutor agents, the verificator agent and possible assistant agents. We shall build five protocols, since there are two kinds of tutor agents depending on whether it is the one related to the student submitting a proof or the tutor agent of some other student.

- **Protocol CP** (Companion agent's Protocol) puts the companion and the tutor agents into interaction. It deals with the validation of a proof. The companion agent sends a proof to the tutor agent which sends back the proof's analysis telling whether it is a valid proof or not. Here the companion agent is the protocol's initiator and the tutor agent is the receiver.

- **Protocol TP** (Tutor agent's Protocol) enables the tutor agent to interact with the companion and the verificator agent as explained above in the ideal case.

- **Protocol TuP** (Tutor agents' Protocol) is called when the proof's analysis cannot be obtained from the verificator agent. The tutor agent then requests it from the other tutor agents.

- **Protocol VP** (Verificator agent's Protocol) deals with the interaction between the tutor and the verificator agent. The verificator agent receives from the tutor agent a request to validate a proof and sends back the proof's analysis if it is valid or a *failure* message if not. The tutor is the one to trigger such a protocol

- **Protocol AP** (Assistant agents' Protocol) is invoked when the tutor agent does not manage to obtain the proof's analysis from the other tutor agents. It thus initiate an interaction towards the professor's assistant agents to call for such an proof's analysis (or a *failure*).

3. A Component-Based Formal Specification

3.1. Protocols and Micro-Protocols

It is important for an interaction protocol to be reusable, i.e., a piece of a protocol could be replaced by another without having to start a whole new development cycle and to globally think out the protocol but to be able to reuse parts of a protocol [8].

Here, interaction protocols are defined as sets of components, called micro-protocols (i.e., they represent interaction units that themselves contain a set of performatives and whose contents is the piece of information to be passed on), that can be assembled in a protocol via a dedicated composition language called CPDL. See [8] for an extended article on issues related to the modeling of component-based interaction protocols.

Like components in software engineering, a micro-protocol is defined by an *executable* part which is a set of performatives and an *interface* part for connecting micro-protocols together. A micro-protocol is composed of four attributes:

- Its *name* identifies a unique micro-protocol.

- Its *semantics* is used to help designers know its meaning without having to analyze its definition. These two fields make up the micro-protocol's signature. The other two attributes refer to its implementation.

- Its *parameters' semantics*. When making use of a micro-protocol it is necessary to know all the parameters' semantics since they are used for building messages.

- Its *definition* corresponds to the ordered set of performatives constituting the micro-protocol. Each performative is described along with its parameters like the sender, the receiver and the message's content.

3.2. The CPDL Aggregation Language

Combining micro-protocols into a general interaction protocol can be done with some logic formulae encompassing a sequence of micro-protocols. The relation between the micro-protocols' parameters should also be specified by telling which are the ones matching. Suppose two parameters α and β are used in a same protocol, if they handle an identical parameter, this parameter should have a unique name. This facilitates the agents' work in allowing them to reuse preceding values instead of having to look for their real meaning. This approach is very much oriented towards data reuse.

CPDL is a description language for interaction protocols based on a finite state automaton paradigm which we have endowed with a set of features coming from other formalisms such as:

- *tokens* in order to synchronize several processes as this can be done with high-level Petri nets.
- *timeouts* for the handling of time in the other agents' answers. This notion stems from temporal Petri nets.
- *beliefs* that must be taken into account prior to firing a transition. This notion is present in predicate/transition Petri nets as well as in temporal logic. Beliefs within the protocol's components as it is the case in AgenTalk [10].

Compared with a finite state automaton a CPDL formula includes the following extra characteristics:

1. a conjunction of predicates in first order logic that sets the conditions for the formula to be executed.
2. the synchronization of processes through the handling of tokens. Such behavior is given through the **token** predicate.
3. the management of time and time stamps in the reception of messages with the **time** predicate.
4. the management of loops that enable a logic formula to stay true as long as the premise is true, with a **loop** predicate.

A CPDL well-formed formula looks like:

$$\boxed{\alpha, \{b \in \mathcal{B}\}^{\star}, loop(\bigwedge p_i) \mapsto \text{micro-protocol}^{*}, \beta}$$

A CPDL formula corresponds to an edge going from an initial vertex to a final one in a state transition graph. Such an arc is labeled with the micro-protocols, the beliefs and the loop conditions. α denotes the state the agent is in prior to firing the formula and β denotes the state it will arrive in once the formula has been fired.

$\{b \in \mathcal{B}\}^{\star}$ represents the guard of a formula. Such a guard is a conjunction of first-order predicates that needs to be evaluated to true in order for the formula to be used. $\mathcal{B}$ is the set of the agent's beliefs. This guard is useful when the set of formulae contains more than one formula with a same initial state. Only one formula can have a guard evaluated to true, and therefore it is fired. This requires that no formula be nondeterministic and that two formulae cannot be fired at the same time. In the current version of CPDL, predicates used for beliefs are defined within the language, and agents have to follow them.

As indicated earlier the **loop** predicate aims at handling loops within a formula. Its argument is a conjunction of predicates. It loops on the set of micro-protocols involved in the formula while it evaluates to true.

4. Designing Baghera's Interaction Protocols

4.1. The Formal Description stage

During the analysis stage a designer supplies a natural language description of a protocol. Since this possibly conveys some ambiguities that may lead to a faulty interpretation one needs to provide a formal description.

In section 2.2 the analysis stage presented five protocols. Protocol *CP* between an companion agent and its tutor agent reads:

```
init ↦ needAnalysis(C,T,P), end
```

where variables *C*, *T* and *P* respectively stand for the companion agent, its tutor agent and the proof to be analyzed. Micro-protocol *needAnalysis* embodies two performatives *query-if(A,B,Proof)* and *inform(B,A,Analysis)* whose semantics has been lifted from FIPA [3]. Variables *A*, *B* and *Proof* respectively match variables *C*, *T* and *P*. *Analysis* is a piece of information supplied by the tutor agent. Such a micro-protocol is going to be reused within other protocols.

Protocol *TP* is the main one since it brings into play a companion agent, its tutor agent, its verificator agent, the other tutor agents and the professors' assistant agents. Its CPDL description is:

```
init ↦ queryIf(C,T,P), A1
A1 ↦ needAnalysis(T,V,P), A2
A2, proof=validated ↦ inform(T,C,A), end
A2, proof=notValidated ↦ needAnalysis(T,Tu,P), A3
A3, proof=validated ↦ inform(T,C,A), end
A3, proof=notValidated ↦ needAnalysis(T,As,P), A4
A4, proof=validated ↦ inform(T,C,A), end
A4, proof=notValidated ↦ inform(T,C,F), end
```

Variables C, T, V, Tu and As respectively correspond to a companion agent, its tutor agent, its verificator agent, the other tutor agents and the professors' assistant agents. Variables P, A and F respectively denote the student's proof, the proof's analysis, and a failure situation. Note that the needAnalysis micro-protocol is being reused here twice.

Formulas starting with states $A2$, $A3$ et $A4$ show beliefs that change according to the proof's analysis. If it is a correct one the proof belief evaluates to validated it otherwise evaluates to notValidated when the analysis' contents is failure.

Protocol *TuP* between an student's tutor agent and the other student's tutor agents is described as follows:

$$\boxed{\text{init} \mapsto \text{needAnalysis(T,Tu,P), end}}$$

Micro-protocol needAnalysis is being reused again. Here T is the one asking for the proof's analysis.

Protocols *AP* between a tutor agent and the professors' assistant agents, and *VP* between a tutor and verificator agent are identical to the one just seen.

In these three protocols, the receiver agent either sends back the whole proof's analysis or the failure message when no analysis can be derived.

4.2. A Tool that Supports the Design of Interaction Protocols

We have developed a platform with a tool dedicated to helping design interaction protocols. This tool called DIP follows the component-based approach presented here above. As shown on figure 4, such a tool enables to design and bring into play a protocol in a graphical manner by relying on micro-protocols and on the compositional language CPDL.

Like the SDL communication protocol description language [13], we advocate two methods for representing protocols: a textual one by means of logical formulas (CPDL) and a graphic one (UAMLe) that is based on a super-set of the UAML language (*Unified Agent Modeling Language*) [3] devised by FIPA. UAML is capable of representing the various types of agents involved in a protocol as well as the message exchanges that happen. The graphic representation relies on a composition of boxes, each of which corresponding to a particular state of the interaction. UAMLe extends UAML in a way to take micro-protocols into account [8].

Protocol described on figure 4 shows four edges labeled with micro-protocols along with their parameters (micro-protocols are written inside boxes). Two of these CPDL formulas have beliefs attached to them (between brackets).

Our platform is endowed with a true graphic editor that enables to define interaction protocols in the graphic language UAMLe. Such a tool allows to (1) build and (2) modify protocols. For this, DIP maintains some information about a protocol: its name, its set of micro-protocols, its semantics and its set of CPDL formulas

Another feature is (3) the automatic translation into CPDL of a protocol represented by a high-level Petri net. (4) DIP allows to display a protocol in the alternate FIPA's notation called the *Protocol Description Notation* (PDN). Unlike UAML (and UAMLe), PDN is a tree-like description of a protocol where each node represents a protocol state and the transitions going out of a node correspond to the various types of message that can be received or sent at the time the interaction takes place. Since DIP is also used in analysis and implementation phases, it is possible to store a description of the protocol in natural language and the designer can generate a skeleton of the protocol in a programming language.

5. Discussion

In the field of multiagent systems, there are very few tools supporting the design of interaction protocols to date. Let us mention AgentBuilder [12] and AgentTool [1].

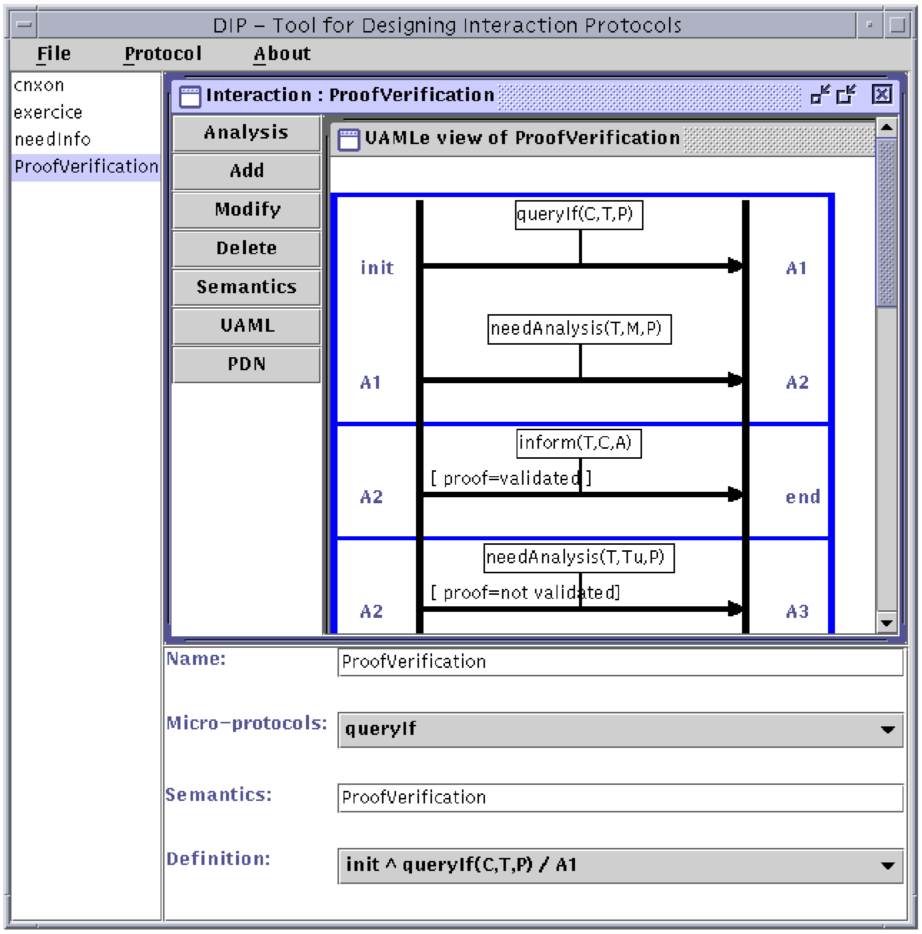
As opposed to the approach advocated in this paper, AgentBuilder does not allow for an easy reuse of existing protocols in order to build new ones as with DIP. Protocols in AgentBuilder are defined by means of finite state automata which unfortunately do not efficiently handle synchronization nor meeting points among agents. Furthermore, AgentBuilder leans on a proprietary protocol's structure which makes it very difficult to utilize any external tool to perform validation tests.

It is also impossible to import protocols expressed in some other formalism which limits reusing. In the approach we have presented, interaction protocols are given in an open formalism which makes it possible to import foreign protocols expressed by means of Petri nets for instance.

AgentTool has not been completely tested due to numerous bugs in the software.

In the domain of distributed systems protocol engineering has been tackled for a long time. This has led to numerous effective tools. Let us mention Design/CPN [7] which allows to manage protocols by means of colored Petri nets.

Design/CPN enables to graphically design and test Petri nets. Such a software tool is capable of simulating the execution of a protocol but is not open to other tools. Its proprietary formalism for representing protocols forces designers to address the protocol implementation issue with still another formalism which is not adequate as far as validation is concerned. On the other hand DIP aims at keeping a same formalism up to the point of validation [9].



Name: ProofVerification

Micro-protocols: queryIf

Semantics: ProofVerification

Definition: init ^ queryIf(C,T,P) / A1

Figure 4. Tool for designing interaction protocols.

References

[1] Agenttool information page. http://en.afit.af.mil/ai/agentool.htm, 2000.

[2] A. El Fallah Seghrouchni, S. Haddad, and H. Mazouzi. A formal study of interaction in multi-agent systems. In *Modelling Autonomous Agents in Multi-Agent Worlds (MAAMAW)*, 1999.

[3] FIPA. *Specification: Agent Communication Language*. Foundation for Intelligent Physical Agents, http://www.fipa.org/spec/fipa99spec.htm, September 1999. Draft-2.

[4] M.-P. Huget. Design agent interaction as a service to agents. In M.-P. Huget, F. Dignum, and J.-L. Koning, editors, *AAMAS Workshop on Agent Communication Languages and Conversation Policies (ACL2002)*, Bologna, Italy, July 2002.

[5] M.-P. Huget and J.-L. Koning. Interaction protocol engineering in multiagent systems. In M.-P. Huget, editor, *Communication in Multiagent Systems: Background, Current Trends and Future*, number 2650 in LNCS/LNAI State of the Art Survey. Springer-Verlag, 2003.

[6] M.-P. Huget and J.-L. Koning. Requirement analysis for interaction protocols. In V. Marik, J. Mueller, and M. Pechoucek, editors, *Proceedings of the Third Central and Eastern European Confereence on Multi-Agents Systems (CEEMAS 2003)*, Prague, Czech Republic, June 2003.

[7] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1, Basic Concepts of *Monographs in Theoretical Computer Science*, chapter 6, Overview of Design/CPN. Springer-Verlag, 1992. ISBN: 3-540-60943-1.

[8] J.-L. Koning and M.-P. Huget. A component-based approach for modeling interaction protocols. In *10th European-Japanese Conference on Information Modelling and Knowledge Bases*, Finland, May 2000.

[9] J.-L. Koning and M.-P. Huget. A semi-formal specification language dedicated to interaction protocols. In H. Kangassalo, H. Jaakkola, and E. Kawaguchi, editors, *Information Modelling and Knowledge Bases XII*, Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, 2001.

[10] K. Kuwabara, T. Ishida, and N. Osato. AgenTalk: Describing multiagent coordination protocols with inheritance. In *Seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 460–465, Herndon, Virginia, November 1995.

[11] M. P. Singh. Toward interaction oriented programming. Technical Report TR-96-15, North Carolina State University, May 1996.

[12] R. Systems. Agentbuilder, an integrated toolkit for constructing intelligent software agents. Technical report, Reticular Systems, 1999.

[13] K. J. Turner. *Using Formal Description Techniques - An Introduction to Estelle, LOTOS and SDL.* John Wiley and Sons, Ltd, 1993.

[14] C. Webber, L. Bergia, S. Pesty, and N. Balacheff. The baghera project: a multi-agent architecture for human learning. In *Proceedings of the Workshop Multi-Agent Architectures for Distributed Learning Environments, AIED2001*, pages 12–17, San Antonio, TX, 2001.