Modularity in Interaction Protocols

Benjamin Vitteau
Département SMA, Centre SIMMO
Ecole Nationale Supérieure des Mines de Saint-Etienne
158, Cours Fauriel
42023 Saint-Etienne
France
vitteau@csc.liv.ac.uk

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

Protocols or part of protocols are frequently reused through projects if they are sufficiently generic. For instance, the Contract Net protocol can be used verbatim or in other protocols such as the supply chain management. This reusability might be difficult to do due to the lack of reusability in current interaction protocol formalisms. In this paper, we present a new approach based on a modular architecture. A protocol is no longer monolithic but a composition of modules called micro-protocols. This approach improves modularity and reusability in interaction protocol engineering. We apply this idea to the supply chain management.

1. INTRODUCTION

Communication represents one of the main components in multiagent systems as stated in the Vowels approach [6] [7] under the letter I, which stands for Interaction. Traditionally, communication is described through protocols. Even if these protocols are as old as multiagent systems, designers are still facing challenges when designing communication. One of the main challenges is the lack of flexibility requiring designers to design protocols from scratch each time [23]. Designers cannot make profit of previous protocols to define new ones. This problem is due to the lack of flexibility in formal description techniques as stated in [17]. Formal description techniques do not consider that a protocol can be made of components (or modules) that represent a part of the interaction. For instance, one would like to represent that the interaction in the supply chain problem exhibits a first phase where customers express what they want to purchase, a second phase where customers and companies negotiate price and delay and finally, a third phase where the different tasks for the production are processed: from task allocation to shipping. Above all, one would like to express that it is possible to replace a specific part of the protocol by another one because the current one no longer fits needs. A component-based approach for protocols would allow this flexibility and reusability.

A first attempt to make interaction protocols more flexible was done in [17]. Protocols are composed of micro-protocols that represent part of the interaction. This proposal exists and is, in the best of our knowledge, the only one which is in use in applications [24]. However, this approach remains simple and will not be of any help when designers will consider negotiation or protocols that belong to a same class: Contract Net protocol and Iterated Contract Net, that is to say, when expressiveness is at stake.

The aim of this paper is to present a revision of the current approach to tackle a broader range of interaction and negotiation protocols. This revision passes by the addition of several new fields to increase the expressiveness of micro-protocols. Such fields are, for instance, the participant cardinality, the agent communication language, the ontology and the norms to name a few. A second refinement is the possibility that a micro-protocol contains micro-protocols; thus offering a hierarchy of decomposition. Finally, the last enhancement is the micro-protocol description as a XML file; although it is not presented here due to lack of space. It is then easier to read and retrieve information for both agents and designers. Moreover, documents can be generated from XML file.

The paper is organized as follows. Section 2 justifies the aim of using micro-protocols in interaction protocols and the advantages. This section summarizes what was said in [17]. As stated above, the current version of micro-protocols is insufficient to represent negotiation protocols or a broad range of protocols; Section 3 explains why the current proposal missed its objectives. Section 4 describes the new approach for micro-protocols. Section 5 considers the protocol level that binds micro-protocols together. Section 6 presents the example of supply chain management. Due to lack of space, we will focus here on only one micro-protocol that defines the negotiation between customers and companies. Section 7 concludes the paper and give future directions of work.

2. MODULARITY IN INTERACTION PROTOCOLS

Traditionally, designers either use formal description techniques coming from distributed systems [12] to represent interaction protocols or define new ones to encompass agent features such as COOL [1], temporal logic [10] or Agent UML [21]. These approaches lack of flexibility as stated in [17]. It is then difficult to easily modify a protocol as soon as a part of it no longer fits designer needs. To cope with this problem, we proposed to consider a modular architecture to describe interaction protocols [17]. The aim of such approach is to allow designers to combine these modules into a protocol to generate a specific need. Moreover, designers can easily replace a module by another if it no longer satisfies the needs. These modules are called *micro-protocols* to remind that they convey a part of the interaction usually carried by protocols. The language CPDL (*Communication Protocol Description Language*) combines these micro-protocols to form a protocol.

Several contributions are provided by a modular approach in interaction protocols:

1. It should allow for the definition of *reusable* protocols. With the current formalisms, one cannot consider replacing a piece of protocol by another. This necessitates to look for the beginning and the end of such pieces. Furthermore, it appears it is difficult to identify the exact semantics of these protocol pieces.

   By representing a protocol as a combination of micro-protocols and providing a semantics for each of them, they can be reused in other protocols. In a modular approach, a protocol becomes more flexible since one just has to remove the micro-protocol that does not suit and replace it by another one. The only constraint is to correctly connect the new micro-protocol to the others already in place.

2. It should allow for *abstraction* capabilities. Following a modular-based approach introduces a meta-level. A basic level corresponds to elements pertaining to a micro-protocol. A higher level conveys the global protocol in a more abstract fashion, i.e. a plan. With the current approaches, a protocol's global view cannot easily be perceived. The use of micro-protocols hides implementation details which are useless for a protocol's understanding.

3. It should *facilitate* the *validation* process, We agree with the following idea [12]: "*A well, structured protocol can be built from a small number of well-designed and well-understood pieces. Each piece performs one function and performs it well. To understand the working of the pieces from which it is constructed and the way in which they interact. Protocols that are designed in this way are easier to understand and easier to implement efficiently, and they are more likely to be verifiable and maintainable.*" With components, it becomes easier to validate a protocol because (1) each component is less complex that a whole protocol and can be separately verified, (2) a global protocol is then verified by checking that its micro-protocols are correctly connected together.

The notion of modularity is not really addressed in interaction protocols, we can only quote work done around COSY [3] and AgenTalk [19]. Work on communication and cooperation in COSY have led to view a protocol as a combination of primitive protocols. Each primitive protocol can be represented by a tree where each node corresponds to a particular situation and transitions correspond to a possible message an agent can either receive or send. The main weakness of COSY is the semantic expression. Semantics is based on the first communicative act in the primitive protocol. As a consequence, it is difficult to distinguish two primitive protocols with the same prefix.

Work in AgenTalk is different since there is the notion of inheritance. Such approach is interesting to derive protocols to specific tasks. Each protocol is represented as a finite state machine. Unfortunately, designers in AgenTalk need to link transitions from the composite protocol to the transitions within the component protocol. This approach reduces the reuse since designers need to study the component protocols before linking states.

In the best of our knowledge, our approach seems to be the only one proposed in the literature and the only one in use [24].

3. DRAWBACKS OF CURRENT MICRO-PROTOCOLS

In this section, we first describe the current version of micro-protocols before addressing the drawbacks of the current version.

3.1 Current version of micro-protocols

Micro-protocols in our approach represent the basic component in a protocol [18]. It contains an ordered set of communicative acts. Some programmatic notions are added to this set to improve the power of expression such as loops, decisions, time management, conditions and exceptions. Micro-protocols must be seen as functions in programming. Micro-protocols have parameters which are used within the micro-protocol. It is then possible to reuse several times the same micro-protocol in the same protocol but with different parameters. Usually, the parameters correspond to the sender, the receiver and the content of the message. The content corresponds frequently to the content of the first communicative act, next contents are computed according the interaction.

Current micro-protocols are defined through four fields:

- a name
- a semantics
- a definition
- the semantics of the parameters

The *name* is the name of the micro-protocol. Micro-protocols are distinguished by their name; as a consequence, names must be unique. The *semantics* gives the semantics of the micro-protocol. This field is a free-format text. Designers can use either a natural language description or a set of logical formulae. The *definition* is the main part of micro-protocols. It contains the set of communicative acts. We add conditions, decisions, loops, time management and exceptions to increase micro-protocols' expressiveness. Each communicative act has usually three parameters: the sender, the receiver and the content of the message. The *semantics of the parameters* gives the semantics associated to these parameters. It helps agents to fill the messages correctly.

Thanks to its expressibility, it is possible to represent the Contract Net protocol [5] as a micro-protocol. The micro-protocol is given on Figure 1.

We do not explain in detail this micro-protocol, readers are urge to read [15]. The important elements in the definition field are the synchronization mechanism, the time management, alternatives and the exception. The keyword *token* is used for the synchronization between agents. The number inserted into parentheses corresponds to the number of messages the receiver expects before resuming the interaction. If a star is used instead of a number, it corresponds to a number of agents which is known at run time. Not giving the number of messages expected offers more flexibility to designers, they are not obliged to hardwire this piece of information into the micro-protocol. The synchronization is realized on the next message. The time management is performed with the keyword *time*. The number within the parentheses means that the receiver is waiting answers from senders during 10 time units. Putting *token* and *time* together has a different semantics, it means that the receiver expects several answers but it waits at most 10 time units. When the deadline is passed, it follows the interaction even if it does not receive all the messages. This approach prevents from deadlocks in interaction. The main novelty of this micro-protocol is the exception management. This notion is barely proposed in agent communication. Moore seems to be the only one to consider it [20]. Exceptions are interesting to unlock situations, for instance if a receiver is waiting for n answers and only gets m (m < n); or if an agent receives an unexpected message. The exception is given within curly brackets as well as the action to do for this exception. For instance, on the Contract Net example on Figure 1, as soon as the task manager receives a *cancel* message, it triggers an exception and do the action *exit* corresponding to stop the interaction with this agent. Finally, alternatives represent different paths in the interaction. They are written within parentheses and separated by a vertical bar. An example is given on the first line of the definition where agents can answer either with a *not-understood* message, a *refuse* message or a *propose* message. We let readers consult [13] to find other examples.

3.2 Critics against the current version

Even if micro-protocols were successfully applied to the Baghera project [24], they remain simple and several critics can be formulated. This section summarizes them.

3.2.1 Micro-Protocol Distinction

One of our critics on COSY's approach is that it is not possible to describe a broad range of primitives due to the way to distinguish them. The distinction is made on the first communicative act. Even if micro-protocols allow more possibilities, the problem of distinction is still present. Micro-protocols are distinguished by their name. Such an approach allows, *a priori*, an infinite number of micro-protocols but this approach falls down due to a lack of readability. Let us suppose that we want to express the Contract Net protocol as stated in Section 3, we straightly name the micro-protocol Contract Net. Now, let us suppose that we want two Contract Net protocols: the current one and an iterated version [9]. We naturally extend the name of the micro-protocol to express the different Contract Nets. This approach is quickly intractable as soon as micro-protocols only differ on some features such as the number of agents, the way to interact (broadcast, multicast), the negotiation phase, etc.

To increase the power of distinction between micro-protocols, it seems reasonable to consider as many fields as necessary to well characterize micro-protocols. For instance, a field on the cardinality is interesting to distinguish micro-protocols dealing with one-to-one communication or one-to-many communication. Several new fields are added to the current version of micro-protocols to increase the expressiveness as Section 4 demonstrates it.

3.2.2 Semantic Weaknesses

The *semantics* field in the current version of micro-protocols is a free-format text. It means that designers are free to use either a natural language description, a set of logical formulae or whatsoever they want. The absence of a well-structured field for the semantics decreases the ability to describe accurately the micro-protocol. Designers can delay writing a formal description and as a consequence providing a micro-protocol without clear semantics. The example shown on Figure 1 clearly demonstrates this point since the semantics is reduced to *ContractNet*.

In the new proposal, we advocate to increase the semantics definition by adding several fields. The semantics field is helped with fields such as prerequisite, termination or type (see Section 4).

3.2.3 Conditions on Execution

Conditions on execution are missing in the current version of micro-protocols. It is thus not possible to describe the conditions allowing or preventing such execution. Designers have no other choice that to describe this information outside micro-protocols. It is important that micro-protocols contain the conditions which are required. These conditions refer to the mental states of the agent which wants to use this micro-protocol, the conditions on the environment, etc.

The current version has some conditions via CPDL [18]. CPDL is the language responsible to link the different micro-protocols. However, these conditions are not sufficiently fine grained to be worthwhile. Conditions are applied to a set of micro-protocols and not for each micro-protocol.

3.2.4 Agent Communication Languages and Ontologies

Even if micro-protocols are using communicative acts, the agent communication language and the ontology used are not written. The piece of information *agent communication language* is important in the context of interoperability; particularly if agents are heterogeneous. Writing the agent

Name: ContractNet
Parameters' semantics:
A: sender
B: receiver
T: task
P: proposal
R: refuse
Definition:
cfp(A,B,T).token(*).time(10).(not-understood(B,A,T).exit |
refuse(B,A,R).exit | propose(B,A,P)).
(reject-proposal(A,B,R) |
accept-proposal(A,B,P).
exception{cancel=exit}.(failure(B,A,R) | inform(B,A,T)))
Semantics: ContractNet

Figure 1: The Contract Net Protocol as a Micro-Protocol

communication language used helps agents to know if they have to translate messages to other agents. The notion of ontologies allows agents to share a same meaning on terms used.

3.2.5 Micro-protocols Defined for Agents

The first version of micro-protocols is directed to agents and not to designers. Designers give a name to the micro-protocol, describe the parameters and the semantics and finally insert the definition. Agents use them during their interactions. As soon as the domain of applications of such micro-protocols augments, it is important to increase the quality and the documentation of micro-protocols. At this moment, designers have to write a documentation separately since there is no way to store this information in the micro-protocol. The principle of reuse is transgressed since it is not possible to have both documentation and definition in the same element.

The new version of micro-protocols attempts to fix this problem by inserting several new fields in the micro-protocol. Moreover, we propose to store the micro-protocol as a XML document [26]. Thus, the micro-protocol is readable by both designers and agents and it is possible for designers to document micro-protocols. Finally, another advantage of XML is XSLT which allows designers to generate documentation automatically. This point is in favour of reuse.

3.2.6 Absence of hierarchical decomposition

In the current version of micro-protocols, there are three levels: the communicative act level, the micro-protocol level and the protocol level. Communicative acts are contained in micro-protocols. Micro-protocols are contained in protocols. Even if this decomposition offers large possibilities, it is better if the decomposition can be more important. In the new proposal, we still have the communicative level and the protocol level but it is now possible to have as many levels as we want between these two levels. Hence, a micro-protocol can be composed of micro-protocols. And consequently, the notion of micro-protocol itself must be thought differently and more broadly than the smallest piece of a protocol.

4. PROPOSAL FOR NEW MICRO-PROTOCOLS

As stated in Section 3, the current version of micro-protocols is insufficient to represent negotiation protocols and protocols that differ a little bit. In this section, we present a new proposal based on the drawbacks found in Section 3. Concerns about the hierarchical decomposition are treated in Section 5. We only focus here on the micro-protocol content.

Micro-protocols are still organized as a document with fields. However, several new fields were added to the previous version of micro-protocols. The micro-protocol fields are the following:

- Name/Type
- Synonyms
- Participant cardinality
- Other attributes
- Agent communication language
- Ontology
- Semantics
- Function
- Behavior
- Roles
- Initiator
- Norms
- Constraints

- Prerequisite
- Termination
- Input/Output
- Code sample

The remaining of this section describes the different fields in the micro-protocols and which fields are used by designers and those used by agents.

Name/Type: *Name* identifies the micro-protocol. The field *Name* should be unique to ease the selection of micro-protocols. However, in case of several micro-protocols sharing the same name, designers have to study the micro-protocols and particularly, the fields *Function*, *Semantics* and *Other attributes* to distinguish the different micro-protocols. *Type* refers to the ability to categorize micro-protocols to help designers to select the correct micro-protocol. For instance, an example of *Type* could be *Auction*. Thus, this micro-protocol belongs to auction protocols. Designers can be as accurate as they want to express the type of the micro-protocol. *Name* is defined as a natural language term. The usual way to represent *Type* is to use a natural language term but it is also possible to describe the *Type* as a path description referring to an object-oriented approach: $Name_1 : Name_2 : \ldots : Name_n$. It means that the type of this micro-protocol is composed of several subtypes separated by colons. Types are ordered from left to right. As a consequence, $Name_2$ is considered as a subtype of $Name_1$ and so on. *Type* is separated from the *Name* by a slash. Several examples are given below:

```
Contract Net
English/Auction
Dutch/Auction
Vickrey/Auction
Iterated/Contract Net
Iterated/Contract Net:1 to 5
Iterated/Contract Net:1 to n
```

Synonyms: English auction is one of the name used to represent this protocol, we can also use open outcry auction or ascending price auction. *Synonyms* gives the list of synonyms to help designers to browse list of micro-protocols. *Synonyms* are represented as a list of terms.

Participant cardinality: In case protocols are described through Petri nets, considering a protocol for m participants is not the same than considering a protocol for n participants where $m \neq n$. Two formats are possible for this field: $m - n$ and m. In the first case, it means that m refers to the sender cardinality and that n refers to the recipient cardinality. Usual values are $1 - 1$, $1 - n$, $n - 1$, $n - n$ refering to one-to-one communication, one-to-many communication, many-to-one communication and many-to-many communication respectively. In the second case, m refers to the participant cardinality without addressing the sender cardinality and the recipient cardinality. In both cases, m and n must correspond to non-zero positive integers.

Other attributes: The field *Other attributes* depicts keywords that characterize the micro-protocols. It might be, for instance, that the interaction is only between authenticated agents, messages are encrypted, secured, fault tolerant or anonymous. Obviously, this list is not exhaustive and greatly depends on the context of the micro-protocol. *Other attributes* are denoted by a list of terms.

Agent Communication Language: This field refers to the language used by the agents in this micro-protocol. Usual values are KQML or FIPA ACL. It is thus possible to describe the same protocol with two micro-protocols, each one for a specific agent communication language.

Ontology: This field indicates the ontology used in this micro-protocol.

Semantics: Representing the semantics of a micro-protocol is important to help designers to understand the meaning of such a micro-protocol. The *behavior* field describes informally what the intention of the micro-protocol is. This field is insufficient to describe the micro-protocol due to ambiguities and misunderstandings of the natural language. It is important to describe formally the micro-protocol. The *semantics* field is defined to this purpose.

Function: The *Function* field gives a summary of the intention of the micro-protocol. Actually, the *Behavior* field may be too long to read in the context of reuse. It is better if designers can read an abstract. That is the aim of this field.

Behavior: The *Behavior* field is the main field in the micro-protocol since it provides a detailed description of the micro-protocol (the communicative acts, the message sequence and the message content). BDI modalities [25] and agent actions are also stored in this field. It is then possible for designers to know what is the impact of the interaction on agents. This is the field that designers will use to generate code for this micro-protocol. The *Behavior* field can be completed by a diagram that represents graphically the micro-protocol. Due to space restriction, we only give one example in Section 6.

Roles: Agents involved in interactions and interaction protocols are involved for a particular role. For instance in auctions, there are usually two roles *auctioneer* and *participant*. This piece of information is used at several levels: first knowing agents' roles allow designers to know how agents will behave in the interaction. Knowing agents' roles, designers know the responsibilities of each agent in the interaction and what they are expected to do. Second, this criterion can make a distinction between different micro-protocols. Two micro-protocols could have the same set of communicative acts but two different set of agents' roles. For instance, in the Baghera project [24], the same micro-protocol *request-information* will not have the same meaning if it is applied between a pupil and his/her companion and between a pupil's companion and the pupil's tutor. In the first case, pupil asks whether his/her proof is correct, the companion answers yes or no. In the second case, companion and tutor can exchange several messages if they need more information about the proof. Finally, agents' roles are used when defining norms for this micro-protocol. Norms are linked to agents' roles and as a consequence to their roles in this interaction. Agents will be "punished" differently according to their roles and their responsibilities.

The agents' cardinality that defines the number of agents per roles is added to agents' roles. For instance, for an auction there is one and only one auctioneer but at least two participants. This piece of information prevents from jeopardizing the system if this is an open system thus too many agents enter the system and consume the resources. Agents' cardinality format is $m-n$ where both m and n represent respectively the lower and the upper bound. There is at least m agents of this role and at most n agents. Lower and upper bounds are positive numbers.

Initiator: This field gives which role is initiator for this interaction. The role must be a role defined in the *Roles* field.

Norms: Norms were first considered in electronic institutions [8] to enforce agents to do some actions or to conform to some behaviors. Norms define what is permitted, what is obliged and what is forbidden. Norms have also a meaning in open systems to prevent actions that will jeopardize the system. Norms can be defined formally (see [8] to this purpose) or informally. Such an example of norms is: "The participant of an auction that gets the higher bid at the end of the auction must pay the amount corresponding to the bid except if the highest bid is less than the reserved price."

Constraints: *Constraints* correspond to constraints that must hold during the execution of the micro-protocol. These constraints are important when designers will check properties on micro-protocols. *Constraints* correspond to safety properties (nothing bad happens) and liveness properties (something good eventually happens). *Constraints* are defined formally via temporal logic or informally. Such examples of constraints are for instance mutual exclusion, deadlock but constraints can be also oriented to the goal of the micro-protocol, for instance, bids less than the current price are forbidden.

Prerequisite: The *Prerequisite* field describes the conditions that must be satisfied before executing the micro-protocol. Prerequisite can be defined formally through temporal logic or informally. Such example of prerequisites is agents must be authenticated before requesting information.

Termination: The *Termination* field gives the valid exits for this micro-protocol. The exit of a micro-protocol is considered valid if the exit matches one defined in the *Termination* field. For the NetBill protocol [4], a valid exit is "the customer has the good and no longer the money, the seller has the money and no longer the good."

Input/Output: Micro-protocols can be linked to express that a protocol is a composition of micro-protocols. To this aim and to add reuse, it is worthwhile if micro-protocols can exhibit entry points and exit points. These points are data (called input and output respectively) that will be used in the current micro-protocol or in the next micro-protocol. Discussion on inputs and outputs is given in Section 5.

Sample Code: Previous fields are more oriented to the definition and the documentation of micro-protocols. This one is directed to the use of the micro-protocol in agents. Two kinds of information can be found in this field, either a sample code that gives the behavior of the micro-protocol in a programming language or an ordered set of communicative acts and micro-protocols that could be directly used by agents as defined in [15]. The former proposal corresponds to the usual approach for protocol synthesis as described in [12]. The latter is related to our interaction module where definition is directly executed [14].

Table 1 describes the use of these fields by designers and agents.

Field	Designers	Agents
Name/Type	√	√
Synonyms	√	
Participant cardinality	√	√
Other attributes	√	
Agent communication language	√	√
Semantics	√	
Ontology	√	√
Function	√	
Behavior	√	
Roles	√	√
Initiator	√	√
Norms	√	√
Constraints	√	√
Prerequisite	√	√
Termination	√	√
Input/Output	√	√
Sample code	√	√

Table 1: Use of Micro-Protocol Fields

We briefly explain this table. All fields are useful for designers. Some of the fields are written in natural language. As a consequence, agents are unable to process them. Such fields are *Other attributes, Function* and *Behavior*. The field *Semantics* is written as a formal description but it will be really difficult for agents to figure it out. The field *Synonyms* cannot be applied to agents since we define it as a field for designers. The other fields can be used by agents providing that the description is structured to be understandable by agents.

5. PROTOCOL LEVEL

In Section 4, we described the micro-protocols. These micro-protocols can be decomposed into micro-protocols giving

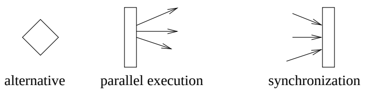

Figure 4: Protocol Level Connector

several levels of micro-protocols. At the lower end of this decomposition, there is the communicative act level containing the communicative acts exchanged in the messages. At the upper end of the decomposition, there is the protocol level that seizes all the micro-protocols used in the interaction protocol as shown on Figure 2.

For instance on Figure 2, there are four levels: one protocol level, two micro-protocol levels and one communicative act level. The protocol level is different from the micro-protocol level as it can be seen as a meta-level addressing the aim of the interaction protocol and the structuration of the interaction. The distinction between protocols and micro-protocols is similar to the one existing between goals and plans in planning.

Protocols are rendered graphically as diagrams as shown on Figure 3. A diagram contains the protocol name in the left upper corner in a "snippet" pentagon as UML 2 proposes it. Constraints on protocols are written in the right upper corner either formally (via OCL [22] for instance) or informally. The remaining of the diagram contains the different micro-protocols and the flow between them.

Micro-protocols are rendered as rectangles with the name of the micro-protocol inside. Micro-protocols are linked together via transitions. Transitions are directed arrows with open arrowhead. Two kinds of information are written on transitions: (1) values used by a micro-protocol and (2) constraints on the execution of the micro-protocol. Values can be supplied by previous micro-protocols or by agent knowledge base. Values can be represented textually on diagrams if there is enough space else designers can use variables and give the content of the variables below the diagram as shown on Figure 3. Constraints are nested into square brackets. Paths in protocols can be splitted or merged. Designers have connectors to this purpose as shown on Figure 4. The diamond connector represents the OR connector meaning that several alternatives are possible. One and only one alternative will be chosen. Parallel executions of micro-protocols are depicted through vertical bars with outgoing arrows. Each arrow corresponds to a parallel execution. Synchronization of these parallel executions are done via incoming arrows on vertical bars. The beginning of the protocol is depicted as a solid circle and an arrow pointing the initial micro-protocol or connector. The end of the protocol is denoted as a solid circle with a hollow circle around. There must be one and only one initial beginning but several endings are possible.

6. THE EXAMPLE OF SUPPLY CHAIN MANAGEMENT

To illustrate our matter, we have chosen to present here the example of Supply Chain Management. At first, we describe succinctly and in a simplistic view what Supply Chain Management is. Then we show the different phases of the process and how they can be matched with micro-protocols. And finally we give the complete depiction of one of the micro-protocols, as the limitations of this paper do not allow us to depict them all.

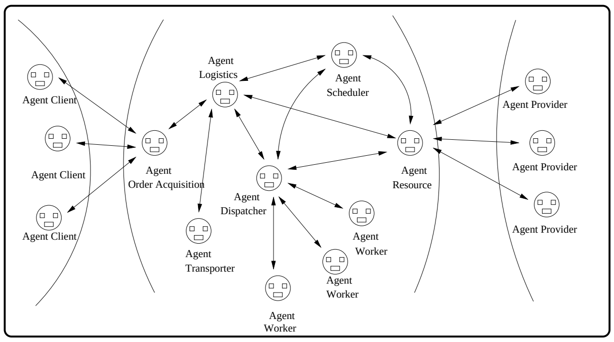

Figure 5: Agents in Supply Chain Management

In its global view, supply chain management renders the material, the information and the finance flows between customers, suppliers and manufacturers. It describes how materials are passing from suppliers to customers and how suppliers and manufacturers attempt to reduce stock. The information flow defines the ordering and the update of orders. Finally, the finance flow presents how and when payments will be made and under what conditions. Pragmatically speaking, the management of supply chain describes how customers pass orders and how suppliers process these orders.

In this paper, we limit the view of supply chain management to the order of customers, the negotiation between customers and companies to fill the order and finally, the task allocation and distribution to process the order.

We follow the idea of Barbuceanu and Fox [2] to agentify the management of supply chain as shown on Figure 5. The list of agents is given below:

- Client
- Order acquisition
- Logistics
- Scheduler
- Dispatcher
- Resource
- Transporter
- Provider
- Worker

Several agents are involved in this application as shown on Figure 5:

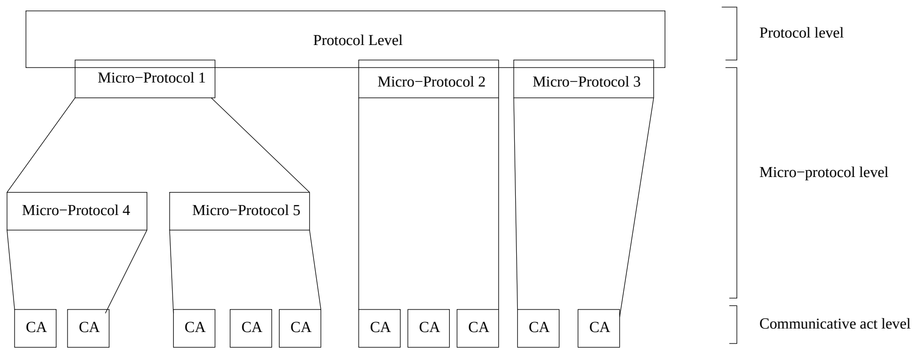

Figure 2: Decomposition of Interaction Protocols in our Model

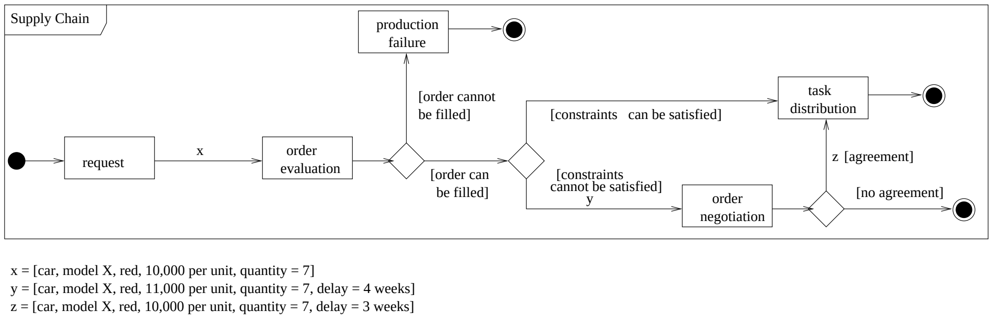

Figure 3: Supply Chain Protocol Representation

- a client agent who places, modifies and deletes orders.
- an order acquisition agent who receives the orders and negotiates the price and the delay with the logistics agent. It is responsible for the connection between clients and the company. It informs clients as soon as deliveries are postponed or delivered.
- a logistics agent who negotiates the delay and the price with the order acquisition agent. As soon as orders are accepted, it asks the scheduler agent to generate a plan for this order. This plan is sent to the agents transporter, resource and dispatcher. It asks for a new plan as soon as some modifications appear either if the client wants to modify the order or if some delays arise on the production.
- a scheduler agent who generates a plan according to the order, the delay and the allocation of workers and resources.
- a transporter agent who is responsible for the delivery of raw materials for the production and for the delivery of the final product to the client.
- a resource agent who manages the allocation of materials in order to realize the order. If the materials are not available, it places an order to provider agents.
- a dispatcher agent who manages worker agents. It allocates work to worker agents. The dispatcher agent informs the logistics agent if some problems arise occurring a delay for the delivery.
- several worker agents who realize the work. If they are unable to complete their tasks at time, they inform the dispatcher agent.
- several provider agents who provide raw materials if the company has not enough materials for the order. In fact, a provider agent is an order acquisition agent from another company.

As stated above, there are several phases in supply chain management: request, evaluation, negotiation and finally task allocation. The request phase involves customers and order acquisition agents to define orders. Then, the companies evaluate if this order can be processed in terms of products to be manufactured and constraints on company production and scheduling. If this evaluation fails, companies enter in a negotiation loop with customers to relax constraints such as prices, delays or features. Finally, companies allocate tasks to workers to fill the order. The production is finished when the order is delivered to the customer.

All these different phases defined above are good candidates to be shaped in micro-protocols. Moreover, such phases are relatively reusable through projects. For instance, the request and negotiation micro-protocols can be used as soon as designers have to deal multiagent systems for commerce. It would be the same for task allocation used either in cooperation systems or in meeting scheduling.

Several micro-protocols are proposed to represent the management of supply chain:

1. **Request micro-protocol**, used by the client agents to contact the order acquisition agent of the company and send an order.
2. **Order evaluation micro-protocol**, composed of several micro-protocols not detailed here which involve every agent of the company and whose function is to evaluate if the order is achievable or not.
3. **Production failure micro-protocol**, to inform the client that the company is unable to process the order.
4. **Order negotiation micro-protocol** used by clients and order acquision agents to negotiate price, delay and whichever elements of importance.
5. **Tasks distribution micro-protocol**, which implies the scheduler and the dispatcher agents to distribute the different tasks among the worker and the transporter agents in order to achieve the order. A Contract Net is performed prior to distribution to define which workers and transporters have to be involved.

The composition of these micro-protocols is depicted on Figure 3. This figure shows the data flow between micro-protocols.

Due to lack of space, we only give here the order negotiation micro-protocol.

Name/type: Order Negotiation / Negotiation

Synonyms: none

Participant Cardinality: 1-1

Other Attributes: none

Agent Communication Language: FIPA ACL

Ontology: Negotiation, Sale

Semantics:

Function: this protocol is employed when the order acquisition agent cannot satisfy the order, and needs to negotiate some points with the clients, most of the time, points refer to the delay or the price. The negotiation stage is based on proposals and counter-proposals and should reach an agreement else the negotiation fails.

Behavior: The order acquisition agent first sends the new proposal to the client by issuing a proposal via the *propose* message. The content of the message is the new conditions proposed by the company. The client agent receives the proposal and has three choices: (1) accepting it (via the *accept* message), (2) rejecting it (via the *reject* message) and finally, (3) countering it (via the *counter* message). In the first case, the negotiation ends since an agreement is reached. In the second case, the negotiation fails since the client gave up. In the third case, the client modifies the proposal and sends its counter-proposal to the order acquisition agent. Proposals and counter-proposals can continue as long as the client does not answer either by the *accept* message or the *reject* message. The automaton corresponding to this micro-protocol is in Figure 6.

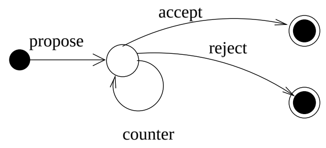

Figure 6: Order Negotiation Example

Roles: Client, cardinality = 1

Order Acquisition, cardinality = 1

Initiator: Order Acquisition

Norms:

Constraints: agents are benevolent and do not try to stuck into a loop where no solution will appear.

Prerequisite: Such criteria for the order to fill are not possible to satisfy. The order acquisition must have a counter-proposal which is processable by the company.

Termination: Three situations are possible for the termination:

1. Client accepts the counter-proposal
2. Order Acquisition rearranges the production to fill this order and satisfy the constraints
3. Order cannot be processed since no agreement was found

Input/Output: Input: counter-proposal

Output: order with a set of agreed constraints or no order

Code Sample:

7. CONCLUSION

Modularity and reusability are now two terms frequently in use in multiagent system design, particularly in agent design. Unfortunately, reuse is barely considered when designing interaction protocols. As Singh [23] quoted it, designers need to start from scratch each time. A first proposal for reusability was made in [16]. This approach is based on micro-protocols and the CPDL language. Micro-protocols represent part of the interaction. They are composed through the CPDL language. This proposal allows flexibility, modularity and reusability but remains simple and in a certain way, missed its objectives. Actually, reusability is not properly addressed in this proposal since two protocols relatively similar will be difficult to represent and to distinguish. This paper attempts to fix this problem through the addition of several fields to increase the micro-protocol expressiveness. Moreover, micro-protocols can now be used by both agents and designers where previous version of micro-protocols are dedicated to agents.

Several directions of work are already considered. First, we will generate micro-protocols for the protocols defined in the FIPA library [9] in order to exemplify this approach. Then, we will provide a library that will be accessible by both agents and designers. Designers can add, delete and modify micro-protocols through a Web interface. Agents can search and retrieve micro-protocols. The final step of our work on this approach will be to update the interaction module [14] and define an application that emphasizes the use of micro-protocols and their reuse.

Acknowledgments This work was supported by the EC under project IST-1999-10948 (SLIE) and by the EPSRC under project GR/R27518.

8. REFERENCES

[1] M. Barbuceanu and M. S. Fox. COOL : A language for describing coordination in multiagent system. In *First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 17–24, San Francisco, USA, June 1995. AAAI Press.

[2] M. Barbuceanu and M. S. Fox. Coordinating multiple agents in the supply chain. In *Proceedings of the Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 96)*, Stanford University, USA, june 1996.

[3] B. Burmeister, A. Haddadi, and K. Sundermeyer. Generic, configurable, cooperation protocols for multi-agent systems. In C. Castelfranchi and J.-P. Muller, editors, *From Reaction to Cognition*, volume 957 of *Lecture notes in AI*, pages 157–171, Berlin, Germany, 1995. Springer Verlag. Appeared also in MAAMAW-93, Neuchatel.

[4] B. Cox, J. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, july 1995.

[5] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem-solving. *Artificial Intelligence*, 20:63–109, 1983.

[6] Y. Demazeau. From interactions to collective behaviour in agent-based systems. In *First European Conference on Cognitive Science*, Saint-Malo, France, April 1995.

[7] Y. Demazeau. Steps towards multi-agent oriented programming. slides Workshop, 1st International Workshop on Multi-Agent Systems, IWMAS '97, October 1997.

[8] M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos. *On the formal specifications of electronic institutions*, pages 126–147. Number 1991 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2001.

[9] FIPA. *Specification*. Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/fipa2000.html, 2000.

[10] M. Fisher and M. Wooldridge. Specifying and executing protocols for cooperative action. In *International Working Conference on Cooperating Knowledge-Based Systems (CKBS-94)*, Keele, 1994.

[11] S. Fonseca and S. Thompson. An agent negotiation pattern language. In J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell, editors, *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, Seattle, USA, November 2002.

[12] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[13] M.-P. Huget. *Une ingnierie des protocoles d'interaction pour les systmes multi-agents*. PhD thesis, Universit Paris 9 Dauphine, June 2001.

[14] 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.

[15] 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.

[16] 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.

[17] J.-L. Koning and M.-P. Huget. Validating reusable interaction protocols. In H. Arabnia, editor, *The 2000 International Conference on Artificial Intelligence (ICAI-00)*, Las Vegas, NV, June 2000. CSREA Press.

[18] 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.

[19] K. Kuwabara, T. Ishida, and N. Osato. AgenTalk: Coordination protocol description for multiagent systems. In *First International Conference on MultiAgent Systems (ICMAS-95)*, San Francisco, June 1995. AAAI Press. Poster.

[20] S. A. Moore. On conversation policies and the need for exceptions. In *Autonomous Agents'99 Special Workshop on Conversation Policies*, 1999.

[21] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of First International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, june, 10 2000. Springer-Verlag.

[22] OMG. UML 1.4. Technical report, OMG, 2001.

[23] M. P. Singh. On the semantics of protocols among distributed intelligent agents. In *IEEE Conference on Computers and Communication*, pages 1–14, Phoenix, USA, 1992.

[24] 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.

[25] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[26] XML. The eXtensible Markup Language. See `http://www.w3c.org/XML`.