Modelling Agent Protocol Requirements

Jason Heard and Rob Kremer
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada

Keywords: Multi Agent Systems, Communications Protocols, Knowledge Representation.

Abstract: Currently, there is no method for describing the requirements of a multi agent protocol. Much work has been done on describing protocols themselves, and this work has continually shifted from low-level protocols, such as situation-action pairs, to higher-level protocols, such as conversation policies. Despite the wealth of work in methods for describing policies, there is no work on describing what a policy *should do*. A method to describe what (and not how) would be useful in the area of automated protocol evaluation and even protocol generation. To address this gap, this paper outlines a *situation model* that contains all of the information needed to define the requirements of a new protocol in a declarative manner. The situation model describes the needed information to develop new protocols, try those protocols in an appropriate environment and then evaluate the performance of the protocols once they have been executed. In addition, this model has a fairly simple textual representation that is designed to be easily parsed. This paper also outlines how this model could be used to generate possible protocols and to evaluate potential protocols.

1 INTRODUCTION

Historically, the creation of multi agent communications protocols has been a long, arduous process that must be done by hand and with great care (Singh, 1998). The protocols must be described not just for the cases where everything proceeds in a straightforward and correct manner, but also for cases where things go wrong and the participants make errors, either intentionally or unintentionally.

Many great strides have been taken to make the process of creating multi agent protocols easier. Standards have been made to describe basic interactions (Foundation for Intelligent Physical Agents (FIPA), 2002). These standards aid multi agent system designers by providing known protocols that can be used in some situations instead of creating new protocols. Another improvement has been social commitments (Castelfranchi, 1995). The use of social commitments allows communications protocols to be described in terms of the actions and communications of agents (Fornara and Colombetti, 2003). With these and other advances, the process of creating agent protocols has become both simpler and more robust. Unfortunately, the process is still akin to algorithm design. The multi agent designer must create the rules, states and/or transitions of the protocol.

It would be beneficial for a multi agent designer to be able to declaratively describe the situation, including the requirements of that situation, and generate the protocol automatically. MAPC is a system that is designed to do this (Heard and Kremer, 2009). In order to do this, previous versions of MAPC required that the user describe the situation in which the protocol would execute and provide a set of fitness functions in Java. This paper outlines a new method for modelling a *communication situation*. This communication situation encompasses not just the environment of the agents in a system (including the roles and actions of those agents), but also the requirements of the desired protocol. Section 2 describes the situation model developed for this purpose. This model is presented both as a formal model and as a textual representation of situations. Section 3 explains how individual situations are used to generate environments in which to test possible protocols and to evaluate executions of possible protocols. Finally, Section 4 provides some concluding remarks.

2 MODELLING A SITUATION

A situation in a multi agent system can be defined by four separate components: (1) a description of the agents that will be in the situation, (2) a set of roles that various agents in the system will take on, (3) an outline of the digital environment or data (which is composed of information that may be different for

Heard J. and Kremer R..
Modelling Agent Protocol Requirements.
DOI: 10.5220/0004125103350338
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2012), pages 335-338
ISBN: 978-989-8565-30-3

each execution of the situation), and (4) a description of the requirements of the situation. Each of these four components is required to form a complete picture of the situation.

2.1 The Comparison Shopper Situation

To aid in the articulation of the four components, consider the *comparison shopper* situation. In the comparison shopper situation, a single purchasing agent that is looking to purchase a particular item and has multiple potential suppliers to choose from. The purchasing agent is looking for the lowest price and also has a maximum price that cannot be exceeded. The sellers each may or may not have the item in question. If they do have the item, they also have a minimum selling price.

The agents in the situation will be described as a set of agent groups. Each group has a set of one or more roles that the agents in that group are fulfilling. The roles define the abilities of a general class of agents and since they may be shared between agent groups they are described separately. In addition, each group has a defined quantity. The quantity describes the possible number of agents in that group. In the comparison shopper situation, the situation has two groups: one agent group containing the role of buyer (to be described below) and a quantity of one and a second agent group containing the seller role and a quantity of one to ten.[1]

To organize the description of the agents in the system, each situation has one or more roles which define the capabilities of various types of agents in the system. They are kept separate from the quantity of each agent type to allow the description of situations in which multiple agents share one role, but have some other differentiation. To enhance the capability to describe the roles, the roles can be arranged in a type hierarchy to indicate inheritance. The roles each describe the actions agents of that role can perform, the desires that agents of that role would like to fulfill, and negotiable values that can be used when fulfilling their desires. In the comparison shopper example, there are three roles. There is a base type called *transactor* that represents any agent with money and/or items. In addition, there are two sub-types that are named *buyer* and *seller*. The buyer is capable of giving money to another agent. The seller is capable of giving items to other agents. In Figure 1, the roles are represented as clouds. Within the roles, the *give item* and *give money* actions are represented as lightning bolts. In addition to their capabilities, the buyer and seller each have desires to receive what the other

[1]The seller quantity can vary between 1 and 10 agents.

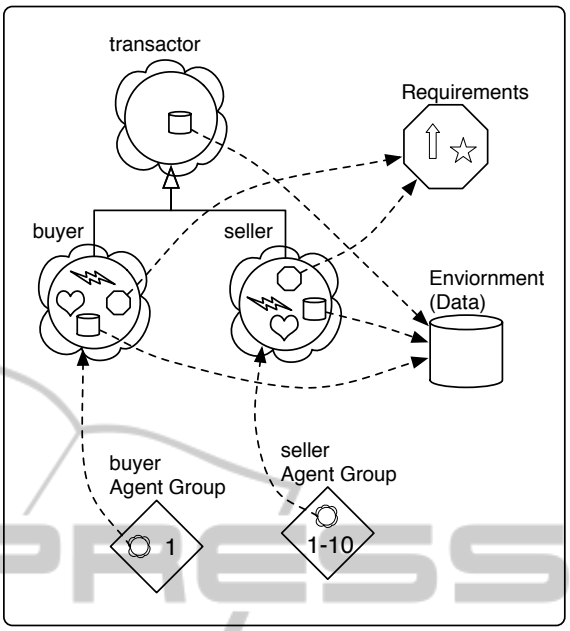

Figure 1: Comparison Shopper Situation.

can give: the buyer has a desire for a give item action with the buyer agent as the recipient and the seller has a desire for the give money action with the seller agent as the recipient. These desires are represented within the roles as hearts. In addition, both the buyer and the seller have a negotiable value *price* which the seller would prefer to be higher and the buyer would prefer to be lower.

The environment of a situation can be represented as a collection of data. This data is composed of three kinds of data. It includes information that is constant over all instances of the situation, information that may be different for different instances of the situation but is constant throughout the entire timeline of one execution of the system, and information that may change as the system executes. In addition, data may be global to the entire situation or specific to one or more roles in the situation. In the comparison shopper example, there isn't any global data, but there is data specific to each role. For example, each transactor agent has a money and item count variable. In Figure 1, data is represented by cylinders. Each of the roles has some data that is part of the environment (or data) of the entire situation.

The final component of the situation model is the requirements of the system. These include both the hard requirements that must be met and the soft requirement that the system designer would like to be met. In the comparison shopper situation the buyers and sellers must not exceed their price limits; this is a hard requirement. Another hard requirement is that money must be given for each item received. Finally,

to illustrate the ability of the situation to reflect the desires of the designers, this example will reflect a buyer's market, and so the buyer should end up with the highest possible total value at the end of the execution of the situation. As with the situation data, the requirements may be generic to the entire situation or specific to individual roles. In Figure 1, the octagons represent the requirements, parts of which refer to specific roles.

When discussing a situation and later the evolution of a protocol that will satisfy the situational requirements, it is useful to discuss *situation instances* and *executions*. Situation instances are specific examples of agent arrangements that are included in a situation. For example, one situation instance of the comparison shopper situation could include a single buyer agent and three seller agents. The situation instance could also include specific values for the number of items and amount of money held by each individual as well as the minimum and maximum prices for the sellers and buyers, respectively. An execution of a situation instance is the sequence of states from the situation instance's starting environment to an ending state where no more actions occur.

2.2 Formal Definitions

Formally, a situation (*Sit*) can be defined as the 4-tuple:

$$Sit = (AG, Rol, Dat, Req)$$

where *AG* is the set of agent groups in the situation, *Rol* is the set of roles in the situation, *Dat* is the set of possible data instances for the situation and *Req* are the requirements in the situation.

An agent group in a situation ($ag_n \in AG$) is defined as the pair:

$$ag_n = (R_n, c_n) \text{ s.t. } R_n \in Rol \wedge c_n \in \mathbb{P}\,\mathbb{Z} \wedge c_n \neq \varnothing$$

where R_n is a role in *Rol* and c_n is the set of possible numbers of agents in the agent group ag_n.

A role is a set of actions that every agent with that role can perform, desires that every agent of that role wishes to fulfill, and negotiable values that agents of that role may use when executing. Furthermore, roles are arranged in a type lattice where every sub-type contains all the actions and desires of it's ancestors. Formally:

$$Rol : poset\,\mathbb{P}(Action \cup Desire \cup Negotiable) \text{ s.t. } \forall r_i, r_j \in Rol \bullet r_i \lessdot r_j \rightarrow r_j \subseteq r_i$$

An action in a role ($a_m \in r_i$, $r_i \in Rol \wedge a_m \in Action$) is defined as the pair:

$$a_m = (f_m, In_m)$$
$$f_m : Dat \times In_m \rightarrow Dat$$

where f_m is the function that modifies the current data instance to reflect the occurrence of the action and In_m is the set of all possible parameters for the action.

A desire in a role ($d_m \in r_i$, $r_i \in Rol \wedge d_m \in Desire$) is defined as the pair:

$$d_n = (a_n, in_{part}) \text{ s.t. } a_n \in Action \wedge \exists in_n \in In_n \wedge In_n \in a_n \bullet in_{part} \subseteq in_n$$

where a_n is the desired action and in_{part} is one instance of the possible inputs for that action or a subset of one of the possible inputs (i.e. a partially defined action).

A negotiable in a role is a partially ordered set of potential values. Formally:

$$Negotiable : poset\,\mathbb{P}\,\mathbb{Z}$$

The requirements (*Req*) of a situation are given as the pair:

$$Req = (req_h, req_s)$$
$$req_h : \mathbb{P}\,(Dat \rightarrow \mathbb{B})$$
$$req_s : \mathbb{P}\,(Dat \rightarrow \mathbb{Z})$$

where req_h are the hard requirements and req_s are the soft requirements. The hard requirements are defined as functions from the final data instance to a boolean value of true for success and false for failure. The soft requirements are defined as a functions from the final data instance to an integer value indicating how well the soft requirements were met (a higher or lower value could be considered better, depending on the situation).

A situation instance is one specific instance of a situation. A situation instance ($inst_n$) can be formally defined as the 3-tuple:

$$inst_n = (Sit, A_n, dat_{n0}) \text{ s.t. } dat_n^0 \in Dat \wedge (\forall a \in A_n \bullet \exists r \in Rol \bullet a \; hasRole \; r) \wedge (\forall ag_n \in AG \bullet \exists r \in Rol, c \in (\mathbb{P}\,\mathbb{Z}) \,|\, ag_n = (r,c) \bullet |\{a \in A_n \,|\, a \; hasRole \; r\}| \in c)$$

where *Sit* is the situation, A_n is the set of agents in this situation instance and dat_n^0 is the initial data instance. Furthermore, all the agents in A_n fulfill at least one role, and every agent group in *AG* contains the proper number of agents.

3 MODEL USE

A situation model can be used to generate potential situation instances and evaluate an execution of some protocol.

3.1 Situation Instances

To generate situation instances, first the collection of agents is chosen. This is done by selecting the number of agents in each agent group from the available options. In the comparison shopper example, one possibility is that the number of seller-group agents is three and the number of buyer-group agents is one (the only option). Once the agent quantities have been determined, the initial value of all of the variables must be picked.[2]

3.2 Evaluating a Protocol

In order to evaluate a protocol using a situation, such as the comparison shopper situation described above, the protocol must give a method to go from a desire to some action. If the protocol is formed in this way, the desires of the agents in a situation instance can drive the test agents to attempt to communicate to fulfil those desires.

Once a protocol has been executed in a situation instance, it must be evaluated against the requirements of the situation. One process for doing this is to use the hard-requirements, soft-requirements and desires defined in the situation.

Each of the post-condition functions in each of the fitness objects within the roles and the situation as a whole are considered a different hard requirement. In addition, the pre-conditions of all of the actions are considered hard requirements. To aid in the evaluation process, the infraction of the pre-conditions are counted as the system executes. Since all of the aforementioned functions are Boolean functions, a relative fitness is given by comparing the number of failures of action pre-conditions, role-specific post-conditions and situational post-conditions. This gives a method for comparing the relative utility of each protocol even when none of the potential protocols met all of the hard requirements.

4 CONCLUSIONS

The situation model described in this paper has all of the information needed to allow the Multi Agent Protocol Creator (MAPC) to automatically generate protocols.

[2]In the case of random values, an appropriate function was called. In the case of the fixed starting values that value is simply used as is.

REFERENCES

Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. In Lesser, V. R., editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 41–48, San Francisco, CA, USA. AAAI-Press and MIT Press.

Fornara, N. and Colombetti, M. (2003). Defining interaction protocols using a commitment-based agent communication language. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, pages 520–527, New York, NY, USA. ACM Press.

Foundation for Intelligent Physical Agents (FIPA) (2002). FIPA ACL message structure specification. document number SC00061G, FIPA TC communication. http://www.fipa.org/specs/fipa00061/SC00061G.html.

Heard, J. and Kremer, R. (2009). Evolving social commitment based protocols using MAPC. In *Proceedings of Self-Organizing Architectures (SOAR 2009)*.

Singh, M. P. (1998). Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47.