Flexibility and Robustness in Agent Interaction Protocols*

Joshua Hutchison and Michael Winikoff

RMIT University, GPO Box 2476V, Melbourne, AUSTRALIA
Phone: +61 3 9925 2348
{johutchi,winikoff}@cs.rmit.edu.au
http://www.cs.rmit.edu.au

1 Introduction

There is increasing interest in using agents in open systems, such as the Agentcities[1] project. It is obviously important for agents to interact with each other. These interactions are usually governed by *interaction protocols* that define the legal sequences of messages in a conversation. These interaction protocols need to be common to agents and are therefore usually defined by standards bodies such as FIPA[2]. However, such protocols are rigid in that agents must adhere strictly to the order of a protocol. This runs counter to the idea of agents as being flexible and autonomous. In particular, flexibility provides error recovery – if one attempt to achieve a goal fails, another method of achieving the goal can be attempted.

Two of the key properties of autonomous and proactive agents are *goals* and *plans* [5, 6]. An agent has goals that it pursues by running plans. Each plan has a goal for which it is deemed to be relevant, and a *context condition* that is evaluated to determine whether the plan is applicable in the current situation. In order to (attempt to) achieve a goal, the agent finds all the relevant plans and filters out the ones that are not applicable. It then selects an applicable plan and runs the plan's body. The structure of a plan's body is not further specified in this paper – in some notations (such as Rao's AgentSpeak(L) [1]) it is a sequence of actions or sub-goals; whereas in others it is a full-blown programming language[3]. A crucial construct in plan bodies is a sub-goal which triggers further plan selection. This execution mechanism is used by agent systems including various implementations based on the BDI (Belief-Desire-Intention) model [2, 3].

In this work we use a case study to investigate whether protocols implemented using plans and goals can allow more flexibility and robustness than a strict message-order based protocol. A Merchant-Customer protocol, based on the Net-Bill protocol [7] was designed. The *goals* of the protocol's interactions were identified and a set of *plans*

* We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (under grant CO0106934). We would also like to thank James Harland for his assistance in preparing and reviewing this paper.

[1] See http://www.agentcities.org

[2] The **F**oundation for **I**ntelligent **P**hysical **A**gents, see http://www.fipa.org

[3] In JACK, plan bodies are a written in a superset of Java.

for use by a pair of JACK[4] BDI (Belief Desire Intention) [2, 3] agents were designed and implemented. These agents were tested in a range of scenarios in order to assess whether the use of goals and plans supported increased flexibility and robustness.

2 The Merchant-Customer Protocol

Our Merchant-Customer protocol extends the Net-Bill protocol as described by Yolum & Singh[5] [7] in a number of ways. Firstly, instead of trying to replicate e-commerce type applications, this protocol simulates purchase transactions in general which may be face-to-face or otherwise. Also, it is oriented towards being partially reliant on an agent's beliefs rather than just which messages are being sent. We make some assumptions regarding the nature of the agents using the protocol, such as that they will be truthful, and have goals relating to price of the goods representing maximum or minimum prices they will buy/sell at. Also, it is assumed that agents will continue to negotiate until a deal is done or one agent thinks a deal will be impossible.

The parts of the protocol are:

1. Availability Request – a customer will ask if some sort of goods are available.
2. Availability Response – the merchant will respond with a "yes" or "no", and if the answer is "yes" an initial asking price as well.
3. Negotiating on price of goods – requires an affirmative availability response before commencing.
4. Negotiating on the details of the goods – requires an affirmative availability response before commencing. This represents discussions on particulars of the goods – for instance, is the car to be blue or red?
5. Reservation/Holding – achieved once agreements on both the price and details have been made.
6. Payment – requires that the holding stage be met first.
7. Transfer of Goods from Merchant to Customer – requires that the holding stage be met first.
8. Transaction ends – requires both transfer of goods and payment.

The protocol allows for some stages to be carried out in parallel. For example, paying for the goods and transferring the goods could be performed in any order. Similarly, agreeing on the price or the details can be done in any order or in parallel. The reservation/holding stage serves as a waypoint in forcing both to be resolved before moving on.

We will illustrate the use of the protocol with an example of a customer looking to purchase a human skeleton. Initially, the customer will ask some merchant if it has skeletons for sale. The merchant may say "no" (and the protocol will cease), or it will say "yes" and give a price it is asking. The customer will evaluate the price and see if falls within the price range it is willing to pay. It then faces two options – either

[4] JACK is a Java based agent language for BDI agents. See http://www.agent-software.com for more information.

[5] In turn, this is based on earlier work by Marvin Sirbu [4]

accept the price (and send an accept message), or suggest an alternative price. Upon the reception of a counter-offer from the customer, the merchant faces the same options – accept or counter-offer again. For example, the merchant might ask for $1000. The customer might say $600 is what it wishes to pay. The merchant might then ask for $850 and so on until an agreement is reached. At the same time, the merchant and customer may negotiate over the details of the goods, and this in turn could affect the price. The customer might say it is after a real human skeleton. The merchant could say that a real skeleton will cost more.

3 From Protocols to Goals and Plans

Whilst it is impossible for this paper to be able define how well other protocols and other types of protocols (as opposed to a commerce-based one like this) are able to become more flexible and robust, we have attempted to determine some process by which other protocols could be converted into a set of goals and plans similar to what has been done in this paper with the Merchant-Customer protocol. The main requirement is to identify the *aims* of a protocol session – what is achieved (or at least desired) by using it. These aims form the goals.

The process we follow to derive a set of goals and plans for a protocol is:

1. Divide the protocol into stages. Parts of the protocol can be grouped into stages based on the desired result (e.g. establishing a connection, or confirming availability). Note that some parts of the protocol should be considered as a unit since they are working towards the same goal. For example, the availability request and the availability response are both working towards the goal of knowing whether the desired goods are available from the merchant.
2. Identify what the underlying goal of that stage is (e.g. is it part of achieving a price agreement?).
3. For each goal identified, determine what states or goals need to have been reached or achieved to reach this stage. This will allow the determination of the prerequisites for each stage.
4. Determine some action[6] for each stage and codify this action as the body of a plan.

This process de-emphasizes the order in which messages are sent and instead focuses on achieving goals in a logical sequence. This reduces the rigidity of protocols and allows protocols to be more flexible and robust.

The stages and their goals for the Merchant-Customer protocol are:

1. Determine availability: the goal is to determine whether the desired goods are available from the merchant
2. Agree on price: the goal is for the merchant and the customer to agree on a price
3. Agree on the details of the goods: the goal is for the merchant and the customer to agree on the details of the goods
4. Reservation/holding: this is a milestone, rather than an interaction. The milestone is reached when the two agents agree on both the details of the goods and the price

[6] More generally a sequence of actions and/or sub-goals.

5. Payment: the goal is for the merchant to have received payment
6. Transfer of goods: the goal is for the customer to have received the goods
7. Transaction end: like the reservation/holding stage, this stage is a milestone rather than an interaction. It is achieved if the goods and payment have both been received.

Some of the stages correspond to interactions (and hence to plans that run and do things – such as communicate, issue payments, etc.). However, other stages correspond to milestones (or synchronization points) that do not have any associated actions.

Having determined the top-level goals used in the protocol, we now identify for each top-level goal sub-goals or actions that can be used to achieve the top-level goal. For example, the goal of agreeing on a price can be met by either accepting an offer that is acceptable, or by issuing a counter-offer and waiting for a response. A crucial point is that additional flexibility and robustness can be obtained by adding additional plans. We return to this in the next section.

The process is similar to that used by Yolum & Singh to translate a Commitment machine to a Finite State Machine and vice versa [7] – at least in the early stages, given its deterministic nature. Whilst two different people undertaking this procedure may get different results for even the same protocol because of its subjective nature, it potentially allows some sort of guidelines for performing similar transitions in the future.

4 Results: Situations for Testing Robustness & Flexibility

The Merchant-Customer protocol was translated into plans from both the perspective of the customer and the merchant. Agents in JACK were then built to use these plans and set up to attempt simple transactions using the protocol. The beliefs and goals of the agent were set at compile-time but key beliefs such as the upper and lower price from the point of view of either agent, and properties of the goods sought could override these at run-time. This allowed a number of scenarios to be set up and tried to test the strengths and weaknesses of our methodology.

We defined some simple scenarios that could occur. In each scenario we consider if it tests for robustness or flexibility and how well the scenario is handled – thus showing any improvement in flexibility or robustness. In general, the tests fall into four main types, and further tests on this protocol, or any protocol in general would need to cover these areas: multiple starting and finishing points; handling and recovery from errors (e.g. timeouts); competing or undefined goals in any of the participants; and being able (for whatever reason) to terminate the protocol explicitly – and what to do afterwards.

Due to space limitations we only describe some of the scenarios that we investigated. For the missing scenarios see the version of this paper that appeared in the workshop proceedings, also available from the second author's web page.

4.1 Multiple protocol starting points

Plan based implementations allow conversations to begin at some point other than the very start of the protocol, assuming that an agent has required knowledge or has

achieved a required goal. For example, the customer does not need to ask the merchant if it has some goods for sale if it knows that to be the case. A goal-based representation naturally handles this scenario since attempting to achieve a goal that does not need to be achieved (because it already has been achieved) does nothing.

4.2 Unexpected messages

Being able to handle messages out of order, or duplicated or otherwise unexpected messages is a test of robustness. The architecture of JACK means that messages that are sent that are not defined as being handled are ignored. With the establishment of proper pre-requisites for each message to be handled (established in the relevance condition of each plan), defined messages that get sent out of context can be ignored, or dealt with in an appropriate way. For example, a merchant would not react to a price offer message if it knew a price had been agreed. This would restrict the problems caused by unexpected messages, by limiting the context for which they were dealt with but would not solve all of the problems associated with such messages.

Whereas with traditional representations of protocols, any out of order message can be seen as a fatal error, agents that use goals and plans can detect and handle such messages and be robust enough to carry on if it detects a message that should not have been sent.

4.3 Different threads of negotiation

When a negotiation (like those represented by the Merchant-Customer protocol) has more than one aspect that can run concurrently, then allowing those goals of the negotiation to be pursued concurrently potentially adds a great deal of flexibility to how agents use the protocol. With an architecture like JACK, where the pursuit of multiple goals can be supported, and which handles incoming messages individually, or as a reply to some other message, each potential goal, and its associated messages, will simply activate an appropriate plan to handle it. Appropriate pre-conditions (as in 4.2) would ensure that each goal was pursued in the correct context (e.g. not too early), and in any acceptable order. Existing protocols force agents to pursue just one goal at a time.

Thus, the goal/plan protocol representation allows the two goals to be pursued at the same time, at the discretion of the agents. Like threads of a conversation, they can be paused and resumed when needed.

4.4 Competing goals of agents

The protocol, as implemented, is currently able to handle competing goals of agents. It encapsulates this flexibility by providing a mechanism for these competing goals to be resolved. By allowing negotiation (currently limited to negotiation on price and details), the competing goals of the participating agents can be allowed for, and hopefully the two agents can compromise their goals to a state where the goals are not in direct competition. The implementation contains a framework to allow negotiations on price by allowing the agents to send each other what they think the price will be which should

eventually converge, if the goals of the agents allow it (i.e. the minimum the agent is willing to sell for is less than or equal to the maximum price the customer is prepared to pay). Thus the protocol has the flexibility to allow two agents with different (price) goals to use it.

5 Conclusions & Future Work

We have taken a protocol, defined as a rigid sequence of interactions/messages, and shown how it can be reformulated in terms of goals and plans that achieve these goals. We have assessed this formulation and shown that it has improved flexibility and robustness. Additionally, it is possible to support multiple starting points in a protocol, deal with unexpected messages, allow for parallel activities more easily, and resolve conflicting goals between two agents using the protocol. The key to this is that linking a protocol based interaction between agents to their goals allows communication to take advantage of the properties of BDI agents (such as the ability to achieve goals by multiple means, persistence in achieving a goal, and the ability to choose between goals – i.e. sacrificing one goal to achieve another). This linking of the "mechanics" of a protocol to the underlying architecture of the agents using it, in our opinion, makes a great deal of sense.

References

1. Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perrame, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, pages 42–55. Springer Verlag, January 1996. LNAI, Volume 1038.
2. Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-Architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference*, pages 473–484, April 1991.
3. Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 439–449, San Mateo, CA, 1992. Morgan Kaufmann Publishers.
4. Marvin A. Sirbu. Credits and debits on the internet. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 299–305. Morgan Kaufman, 1998. (Reprinted from *IEEE Spectrum*, 1997).
5. Michael Winikoff, Lin Padgham, and James Harland. Simplifying the development of intelligent agents. In Markus Stumptner, Dan Corbett, and Mike Brooks, editors, *AI2001: Advances in Artificial Intelligence. 14th Australian Joint Conference on Artificial Intelligence*, pages 555–568. Springer, LNAI 2256, December 2001.
6. Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative & procedural goals in intelligent agent systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, April 2002.
7. P. Yolum and M.P. Singh. Synthesizing finite state machines for communication protocols. Technical Report TR-2001-06, North Carolina State University, 2001. Available from *http://www.csc.ncsu.edu/research/tech-reports/README.html*.