Flexibility and Robustness in Agent Interaction Protocols

Joshua Hutchison
RMIT University
Melbourne, Australia
johutchi@cs.rmit.edu.au

Michael Winikoff
RMIT University
Melbourne, Australia
winikoff@cs.rmit.edu.au

1. INTRODUCTION

With an increase in agent applications has come an increasing need for agents to be able to interact with each other over open agent systems, such as the types of systems the Agentcities[1] project is developing. These interactions are usually governed by protocols which dictate the legal messages between agents for the course of that interaction. However, protocols can be rigid in the order in which messages can be sent, as well having a lack of robustness where even a small deviation will bring the protocol session to a premature end.

In this paper we investigate how protocols implemented from the point of view of plans and goals would allow more flexibility and robustness than a strict message-order based mechanical protocol. We examine an attempt to translate a protocol to a set of plans to see whether doing so adds any flexibility to using the protocol. A Merchant-Customer protocol, based on the concepts (quoting and accepting price, sending goods, etc.) as used in the Net-Bill protocol [8], was designed. The goals of the protocol interactions were identified and a set of plans for use by a pair of JACK[2] BDI (Belief Desire Intention) [3, 4] agents were designed and implemented.

More flexible and robust protocols would allow autonomous agents to not be forced to rigidly follow a sequence of messages dictated by the protocol. Rather, they could take advantage of their intelligence. By identifying problems when they occur whilst using a protocol they can intelligently resolve them, or even act proactively to avoid them. This would make using protocols easier and more able to deal with problems caused by having agents coded by two different people/groups, with different goals and beliefs, etc. In turn this should lead to better working open systems, allowing their wider application.

Section 2 of this paper outlines some of the background issues and concepts that formed the motivation and basis of our work, including the Merchant-Customer protocol that served as the test-bed for our work. Section 3 reviews an initial process for translating mechanistic protocols to plans that can handle the protocol and section 4 assesses the goal-plan approach proposed with respect to flexibility and robustness.

[1] See http://www.agentcities.org for more information

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

2. BACKGROUND

2.1 Interaction protocols in Open Systems

When agents need to communicate or interact in any meaningful or sustained way they require protocols to provide a structure to those interactions. These protocols are usually governed by standards bodies, such as FIPA[3] or written up and documented in widely distributed, immutable documents like Request For Comment documents. However, such protocols have two shortcomings:

1. Rigidity – agents must adhere to the order of a protocol and cannot recover from deviations. There is usually no scope within a protocol to allow for the communication and handling of agents with differing goals and a protocol session is a mechanical process rather than a belief and goal driven process. This is counter to the BDI agent concept of achieving goals via multiple means.
2. They are not extensible. Agents must be hardcoded with a protocol if they are to use it to communicate with other agents. In open-systems this reduces the flexibility of the agents as standard protocols must be designed, approved (by a standards body), and coded into the agents to be used.

These issues are problems in open agent systems where multiple developers are creating new agents to be part of open agent systems on an ongoing and evolving basis. Enforcing set and rigid protocols may prevent an agent from fully using its intelligence or other properties it might posses.

Our work has focused on overcoming the first issue. Enabling protocols to be more flexible for use by goal driven agents serves as the project's motivation.

2.2 Net-Bill Protocol

As stated before, the protocol that was designed for the project was based on similar concepts that form the Net-Bill protocol in [8] by Yolum & Singh[4]. The Net-Bill protocol aims to represent an

[3] FIPA – Foundation for Intelligent Physical Agents. See http://www.fipa.org for more information.

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

e-commerce transaction over a system like the internet. The idea behind the Net-Bill protocol is to complete a transaction whose end point was that the customer received some goods and the merchant received payment from the goods. Aspects specific to e-commerce (such as third-party verification of payment) have to some degree been included within their protocol.

The protocol in [8] makes some allowances for the different nature of purchases. Three starting points of a transaction using this protocol were defined, aimed at reflecting different motivations. The first of these is where a customer asks for a quote on the price of the goods s/he wants. The second scenario involves the merchant "advertising" the price of the goods by proactively sending out quotes to the customer. The third scenario is based on the assumption that the needs or means of a customer are such that they do not care about the price and accept before they are aware of the price.

2.3 Merchant-Customer protocol

Our Merchant-Customer protocol extends the Net-Bill protocol 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.
5. Reservation/Holding – achieved once an agreement on both the price and details has been made.
6. Payment – requires the holding stage be met first.
7. Transfer of Goods from Merchant to Customer – requires the holding stage be met first.
8. Transaction ends – requires both transfer of goods and payment.

Negotiations on details represents discussions on particulars of the goods – for instance, is the car to be blue or red.

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 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.

2.4 Goal-Plan Agents

We are interested in agents in the BDI (Belief-Desire-Intention) tradition. From an implementation perspective two of the concepts that distinguish such agents are *goals* and *plans*[5] 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. Finally, each plan has a *body* that is run if the plan is selected. This body is not further specified in this paper – in some notations (such as Rao's AgentSpeak(L) [2]) it is a sequence of actions or subgoals; whereas in others it is a full-blown programming language[6]. A crucial construct in plan bodies is a sub-goal which triggers further plan selection.

The execution cycle of a goal-plan agent consists of the following steps:

1. Select a goal.
2. Determine the set of relevant plans (those that handle the goal).
3. Determine the subset of these that are applicable (those that have a true context condition).
4. Select an applicable plan and begin executing it. Where the execution generates sub-goals, these are handled using this process.

Note that systems in the BDI tradition tend to reduce goals to events, and consequently talk about the execution cycle in terms of events. However, as discussed in [7, 6], this reduction is undesirable since a number of desirable properties of goals (such as persistency) do not hold for events.

3. IMPLEMENTING PROTOCOLS FOR BDI AGENTS

3.1 Implementation

The above Merchant-Customer protocol was first translated into plans (see next section) 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.

[5]Other concepts are discussed in [6].

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

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.

JACK [1] is a Java based, commercially available, agent language implementing the Belief, Desire, Intention agent paradigm [3, 4]. It supports the use of plans, and JACK agents can be configured with a set of plans to use. Each JACK plan must be designated to handle a class of event. An event can either be something sent between agents (a message), something posted by the agent to be handled within itself (signifying a change in desires/goals or beliefs) or something externally generated from the environment. When deciding which plan to execute for an agent, JACK will select a set of plans designed to handle the incoming event. It will then disregard any plan that has conditions which mean that it is inapplicable to this particular event. JACK then selects a plan from the remaining set of applicable plans.

3.2 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).
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/goals do I need to have passed through (what goals must be achieved to reach this stage?). This will allow the determination of the prerequisites for each stage.
4. Determine some action[7] 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.

For example, in the Merchant-Customer protocol the stages are

1. Determine availability (availability request and response)
2. Agree on price
3. Agree on details
4. Reserve goods (agree on price & details)
5. Pay
6. Transfer goods
7. Transaction end (goods paid for and delivered)

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.

The goals of the stages are:

1. Availability Request and Availability Response: the goal is to determine whether the desired goods are available from the merchant
2. Negotiating on price of goods: the goal is for the merchant and the customer to agree on a price
3. Negotiating 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
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.

Note that 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 [8] – 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.

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

4. RESULTS: SITUATIONS FOR PROVING ROBUSTNESS & FLEXIBILITY

Having now coded two agents that have a basic implementation of the protocol, it is possible to assess the robustness and flexibility of the protocol as it is implemented using plans and goals. We defined some simple scenarios that could occur in using the protocol given the potentially inexact nature of the agents that might use it. 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.

4.1 Multiple protocol starting points

Plan based implementations allow the commencement of the protocol to start at some other point at the very start, assuming that it has the knowledge or has achieved a goal required. 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. If, somehow, there is an existing agreement on price then that phase of the transaction can be bypassed.

Using the example of a customer that knows the price of some goods already, if the customer was able to communicate this belief to the merchant then both will believe the goal of price agreement already obtained. In our implementation the plans for the customer were closely linked in order to find the point the protocol session can be started at given the goals reached and the beliefs/knowledge possessed. They need not handle messages sent by the merchant reactively, but rather as part of a sequence it commenced. Ultimately, this would depend on the agents using the protocol and their plan sets as much as the protocol or some base plan set.

4.2 Incomplete goals or beliefs

Incomplete goals or beliefs of participating (BDI) agents tests the flexibility of a protocol. Can two agents still use a protocol, if, for example, the customer did not know how much s/he wished to pay for a skeleton (that is to say, had no price goals)? We found that, provided both participants didn't have a complete lack of beliefs on some aspect, then the goals of the participant who did have some (be it over price or whatever) would be used as the starting point to initiate negotiations and the protocol could proceed. The other agent would simply identify the goals of the first and base its actions upon those. One way to communicate a lack of beliefs over some aspect would be to have a reserve value to flag this fact. Then, plans where the relevance condition was set to pick up this value could ensure the other agent's goals were not considered. For example, if, when specifying the price of a skeleton, it specified -1 to denote "make me an offer" then the customer could handle that message and value, send its goal price and let the merchant evaluate that offer.

Again, appropriate additions to the plan sets of both agents would allow them to communicate if they had a lack of beliefs or goals on a particular topic. Plans that handle a standard message for some interaction, but with a different relevance condition would allow the agents to reach an alignment of beliefs in an alternate manner to the norm, yet still use the protocol.

4.3 Unexpected messages

Being able to handle messages out of order, or duplicated or otherwise unexpected messages is a test of robustness that such protocols can handle to a point. The architecture of JACK means that messages that are sent that are not defined as being handled are subsequently ignored. With the establishment of proper pre-requisites for each message (established in the relevance condition of each plan) to be handled, defined messages that get sent out of context can also 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 and the use of the 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.

4.4 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 pursed concurrently potentially adds a great deal of flexibility to how agents use the protocol. With an architecture like JACK [1], 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 4.3) would ensure than 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 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.

4.5 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.

4.6 Behavior not conducive to a result

This case refers to behavior from one of the agents that will not result in the agents concluding a transaction and is a test in robustness. That is, the protocol should be robust enough to ensure an

agent can cease to continue in the protocol session rather than be caught in a cycle of useless behavior. This behavior could take the form of a pattern of messages (sent by one of the agents) during the price negotiation stage that will not lead to a price agreement. This pattern may be a continual rejection of price bids offered by one of the agents, or some type of infinite loop of messages (or what at least appears to be an infinite loop). Such behavior may be legal according to the protocol but clearly there is no benefit in continuing with a transaction in such a situation, and a robust protocol will allow an agent to safely conclude the protocol from at least its end.

Such a situation could handled in the design. In order to handle a situation as described above, an agent would need some plan that handled the offending message type (whether it be for a *reject* or an *offer*) and seeing if there is a pattern of such messages (this test would be specified in the context of the plan). The ability to detect a pattern would rest with the beliefs of the agent – if an agent kept track of the history of the negotiations it could use this knowledge in its plans. It then would be a case of stopping the interaction (by sending a termination message) once a plan had concluded that continuation was futile.

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; in particular, many improvements are possible by adding additional plans that provide alternative means of achieving a given goal.

The degree of flexibility added by applying the proposed approach could be quite great. As was discussed before, it would be possible to support multiple starting points in a protocol, dealing with agents with incomplete or unclear goals or beliefs, the ability to negotiate (at least in a more intelligent way), or to 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 desire or 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.

Whilst it may be a simplistic measure, adding more plans (usually several handling the same event but with different context and relevance conditions) allowed the agents to be used in a wider range of possible situations. Provided the agent is given some way to terminate the transaction, an agent programmer can supply an agent with plans to cover as many situations as is practical for that agent to handle, and a default plans that terminates the session can be used if none of those are applicable.

Overall, mapping protocols to plans can make at least simple protocols easier to implement and more flexible to use.

There is clearly a great deal of research to be done in the area. More sophisticated designs and a comparison between a basic mechanistic protocol and a goal/plan based one (which was an aim of this work) would prove quite useful as well. However, in the context of this work, the most useful next step would be to look at robustness with goal and plan based protocols on its own - focusing on issues such as recovery from failures and timeouts (in open agent systems), or silence.

6. ACKNOWLEDGMENTS

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.

7. REFERENCES

[1] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998.

[2] 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.

[3] 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.

[4] 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.

[5] 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).

[6] 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.

[7] 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.

[8] 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*.