Hermes: Designing Goal-Oriented Agent Interactions

Christopher Cheong and Michael Winikoff

RMIT University, Melbourne, AUSTRALIA
{chris,winikoff}@cs.rmit.edu.au

Abstract. Interactions between agents are traditionally specified as interaction protocols using notations such as Petri nets, AUML, or finite state machines. These protocols are a poor fit with autonomous proactive agents since protocols are message-centric and do not support goals. Additionally, interaction protocols prescribe how interactions are carried out by agents, thus limiting the flexibility of the interactions. This also limits robustness, by reducing the available options for recovering from failure. In this paper we propose a goal-oriented approach to interaction. Since we aim at a useful and practical approach that can be used by practising software engineers, a design methodology is an important part of our solution. We present the Hermes approach which includes a methodology for designing goal-based interactions, failure handling mechanisms, and a process for mapping design artefacts to an executable implementation.

1 Introduction

It has been remarked that there is no such thing as a single agent system. The ability of agents to interact with other agents is essential, and it is desirable for agent interactions to be flexible and robust. Agent interactions are traditionally specified in terms of interaction protocols, expressed in notations such as Agent-UML [1], Petri nets [2], or finite state machines. However, these approaches are not well-suited to agents that are autonomous and proactive. Interaction protocols are at a low level of abstraction and are message-centric in nature since they are defined in terms of legal message sequences.

This results in a number of drawbacks for the agent paradigm. The primary disadvantages are that the protocols are mechanistic and restrict the autonomy of intelligent agents. Since agents are autonomous and able to independently pursue goals and recover from failures, their interactions should exploit, rather than limit, these characteristics. Further disadvantages are that the flexibility and robustness of the interactions are limited (as the degree of flexibility and robustness depend on the number of legal message sequences); where flexibility refers to multiple ways to successfully achieve an interaction and the ability to take *shortcuts* (i.e. by-passing already completed parts of the interaction), and robustness is the ability to recover from and persevere through failures in the interaction.

We propose the concept of *goal-oriented interaction* which is better suited to the agent paradigm's goal-oriented nature. Goal-oriented interactions are defined in terms of the goals of the interaction (*interaction goals*) and temporal constraints. The interacting agents determine how interaction goals are achieved and are restricted by the temporal constraints placed on the interaction goals (IGs). Interactions between agents

occur because the agents involved have certain goals to achieve, and the interactions are a means of achieving the agents' goals.

In traditional protocol designs, the interaction designer explicitly defines a number of legal messages sequences in terms of messages and combining forms such as sequencing, alternatives, and loops. The Hermes design is different in that the interaction designer does not explicitly define legal message sequences. Instead, the interaction is described in terms of interaction goals, available actions, and constraints. The agents then determine what legal message sequences (according to the constraints defined by the interaction designer) are used for the interactions. As such, the message sequences *emerge* from the interaction. This results in a greater degree of flexibility and robustness since there are more legal sequences available than what an interaction designer could have explicit defined.

We aim to devise a *practical* approach that can be used to develop flexible and robust interactions in agent systems which specifically includes a design methodology and execution mechanisms. Our approach is not particularly targeted towards open systems, however, we have developed the work such that it is useable in open systems. We thus introduce *Hermes*[1], which is a domain independent methodology providing a systematic approach for creating goal-oriented interactions and thus moves away from message-centric protocols. Hermes covers the design (section 2), failure handling (section 3) and implementation (section 4) aspects of the agent interaction development process.

2 Goal-Oriented Interaction Design

In this section, we explain the Hermes design process. To illustrate our work, we use an e-commerce protocol based on the NetBill [3] protocol in which a Customer purchases goods online from a Merchant. The NetBill protocol was chosen since a number of other approaches to flexible interactions have used it [4–6], and by using the same example it becomes easier to compare our approach to existing approaches.

Figure 1 provides an overview of the Hermes design process. The process is shown as an incremental mini-waterfall model in which each step is derived from the previous step. However, as is typical of design, the process is applied in an iterative fashion where developing the design may suggest changes to previously developed aspects. For example, identifying the actions (step 3) may suggest additional interaction goals (step 2).

The Hermes design process encompasses not only designing the interaction, but also designing the internals of the agents that participate in the interaction. In order to end up with a complete design that can be implemented we must consider both the inter-agent aspects, as well as internal (intra-agent) aspects.

The first step in the methodology involves the identification of roles and interaction goals. The roles are defined in terms of the participants of the interactions and the interaction goals can be seen as high level goals that need to be achieved for the interaction to be successful. When identifying interaction goals, it is best to think broadly and cap-

[1] In Greek mythology, Hermes was an Olympian god who acted as the herald of the gods and served as their messenger (http://www.pantheon.org).

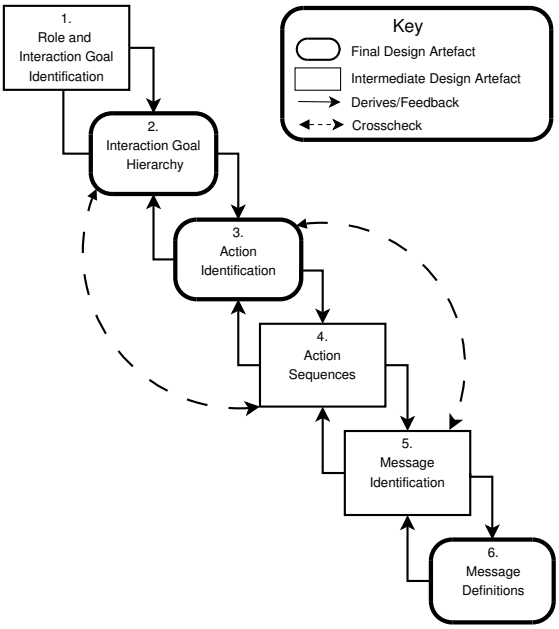

Fig. 1. Hermes Methodology Overview Diagram

ture high level interaction goals. Note that interaction goals are goals of the *interaction*, not of a particular agent.

The second step is the refinement and organisation of the interaction goals identified in the previous step. Where possible, the interaction goals identified are broken down into smaller sub-IGs and are organised in a hierarchy as in Figure 2. The hierarchy should only have a single IG at its apex, which captures the overall goal of the entire interaction.

For example, in our e-commerce protocol, the overall goal of the interaction is for the Customer and Merchant to trade cash and goods, thus the top interaction goal is *Trade*.

The *Trade* IG can be further broken down into two more concrete IGs, *Agree* and *Exchange*. Those two IGs can then be broken down even further. Figure 2 shows the interaction goal hierarchy for our protocol, in which the circles represent the interaction goals and the plain lines denote decomposition (i.e. sub-goal relationships). For example, the links between *Trade* and *Agree*, and those between *Trade* and *Exchange* denote that the *Trade* interaction goal is composed of *Agree* and *Exchange*. For the *Trade* IG to be successfully completed, its sub-IGs, *Agree* and *Exchange*, must also be successfully completed. The interaction goal hierarchy is effectively a goal-tree, similar to those used in agent-oriented methodologies such as MaSE [7] or Prometheus [8]. In developing the notations of Hermes we intentionally did not adopt the UML as a starting point. We believe that by doing this we avoided developing notations that were biased by the object-centred viewpoint of the UML.

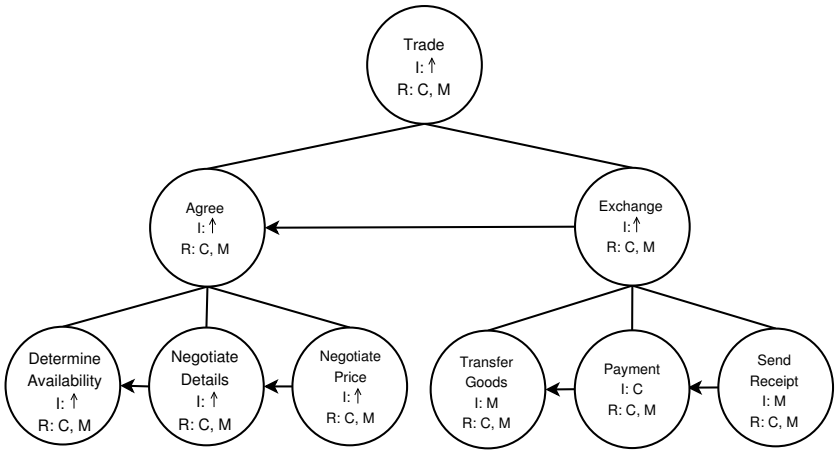

Fig. 2. Interaction Goal (IG) Hierarchy Diagram

Once the IGs have been decomposed into smaller and more concrete interaction goals, temporal dependencies (lines with arrowheads on Figure 2) are added. The temporal dependencies allow the interaction designer to place *constraints* on the sequence of the interaction. For example, the line with the arrowhead between *Agree* and *Exchange* implies that the *Agree* interaction goal must be (successfully) completed before the *Exchange* interaction goal can start. The particular design shown in Figure 2 is strongly constrained, however, alternative designs could, for instance, negotiate the details and the price simultaneously.

As the interaction goals are identified and as the interaction goal hierarchy is being laid out, the *roles* involved in the interaction are assigned to interaction goals. In this particular example, it is quite obvious that there are two roles, *Customer* and *Merchant*, and that both roles are involved in every goal of the interaction. In Figure 2, the roles involved are shown in the circles as *R: C, M*, denoting that a particular interaction goal involves the *Customer* and *Merchant* roles.

It is also necessary to identify an *initiator* for every goal of the interaction. The initiator represents the role which initiates and is initially responsible for a particular goal of the interaction. Identifying an initiator is necessary in order to ensure that when an interaction goal is reached, at least one agent has the initiative and will begin interacting in order to achieve the IG. Valid initiators are specified as one of the roles involved in a particular IG (e.g. *C* or *M*) or as ↑ if it is an *inherited* role, i.e. the parent interaction goal's initiator. In Figure 2, the interaction initiator, whether it is the *Merchant*, or *Customer* role, is always responsible for determining the availability of the goods. By contrast, the *Merchant* role is always responsible for the *TransferGoods* and *SendReceipt* IGs, no matter who the interaction initiator is.

The interaction goal hierarchy provides an overview of what goals need to be achieved to complete the interaction. The next step is to determine what actions can be used to achieve a particular (leaf) interaction goal, and what constraints hold between these actions. It is here that we begin to consider the internal design of the agents. Note that

there is no need to identify actions for non-leaf-level goals, since they are completed when their sub-goals are completed (e.g. *Agree* is achieved when *DetermineAvailability*, *NegotiateDetails* and *NegotiatePrice* are all achieved).

An *action* is a discrete step, taken by a single agent, towards achieving an interaction goal. Actions that can be used to achieve (leaf) IGs are captured in *Action Map*s, such as Figure 3, which is an action map for the *NegotiatePrice* IG. Action maps are divided into "swim lanes"; one per role involved in the interaction goal. As there are two roles involved in the *NegotiatePrice* IG, Figure 3 is divided into two swim lanes, one for the *Customer* role and one for the *Merchant* role.

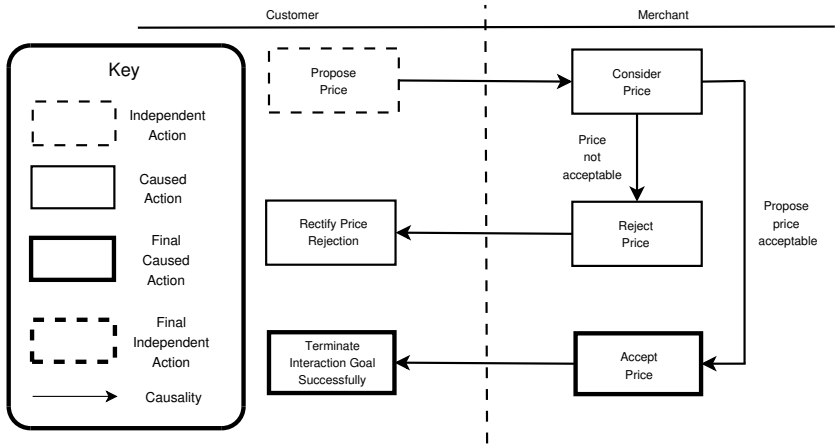

Fig. 3. Action Map

The key in Figure 3 illustrates four different action types, each of which has a different meaning and use.

An *Independent Action* is one that can start independently from other actions, i.e. it is not necessarily caused by another action, but it *may* be caused by another action. *Independent Action*s are typically used as entry points into interaction goals and as such, each action map should contain at least one *Independent Action*.

A *Caused Action* is one which cannot start independently and *must* be triggered by another action.

A *Final Caused Action* is a *Caused Action* which terminates the interaction goal for a particular role.

A *Final Independent Action* is an *Independent Action* which terminates the interaction goal for a particular role. *Final Independent Action*s are typically used for roles that only have one action which both starts and terminates an interaction goal.

Performing a final action (either *Final Independent Action* or *Final Caused Action*), does not necessarily mean the interaction goal is successfully achieved, only that it is completed. For example, the interaction designer may wish to end the *NegotiatePrice*

IG with failure when a price offer is rejected by the Merchant (but this is not the case in Figure 3).

The causality arrows in figure 3, which can be inter-agent and intra-agent, are used to specify temporal restrictions between the actions. Later in the design process, messages are introduced to allow us to realise these constraints. For example, when the *Customer* executes the *ProposePrice* action, this will cause the *Merchant* to perform the *ConsiderPrice* action, which will trigger another action and so on until the interaction goal is completed.

Where an action has causality links to more than one action the causality arrows are intended to depict alternative possibilities. For example, in the case of the *Consider-Price* action on Figure 3, it either triggers an *AcceptPrice* action or a *RejectPrice* action, but not both. Which action is triggered will depend on certain conditions or states. For such situations, labelling the causality arrows with the conditions or states is useful in clarifying the causality path on the action map.

The next step in the design process is for the designer to develop *Action sequence* diagrams by following specific traces from the action maps. Unlike the action maps, which show all possible execution sequences, each action sequence diagram shows *one possible* sequence of actions that can be carried out to achieve the interaction. Figure 4 is an example of a partial action sequence diagram which shows how the *NegotiateDetails* and *NegotiatePrice* goals could be achieved. Actions by particular roles are indicated with the name of the action in a box on the agent's lifeline. Which actions belong to which interaction goal is shown by shading, with the name of the IG at the top-left side of the shaded region.

The purpose of the action sequence diagrams is to check that the actions identified in the action maps are sufficient to allow for a complete and successful interaction to take place, and to ensure that specific interactions that are desired can be generated by the interaction goal hierarchy and associated actions. Action sequence diagrams can also be used to show typical interactions and possible failures.

Once the actions of the interactions have been identified and checked, the inter-agent messages and their format must be determined. The messages are used to realise the constraints of the action maps. Although Hermes provides guidelines to assist with identifying the messages, details of the message format are typically specific to the application and the implementation platform, and thus Hermes does not provide any guidelines for developing the message format, nor any constraints on the message format: one could choose to use KQML, FIPA, SOAP, or the message types provided by the implementation platform (for a non-open agent system).

A good starting point for identifying messages is to expand on the action sequence diagrams by determining what messages are required between actions. Whenever an action by one role is followed by an action of another role there needs to be a message between the two roles (assuming that both actions are within the same interaction goal). This results in *action message* diagrams, for example see Figure 5, which is derived from Figure 4.

Note that when identifying messages, there is no need to have messages between the interaction goals, because moving between IGs is done by the coordination plans (see section 4) .

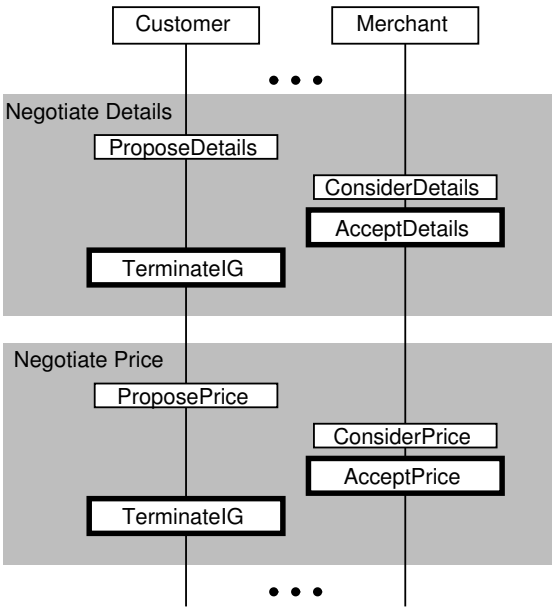

Fig. 4. (Partial) Action Sequence Diagram

3 Failure Handling

Successfully handling failure is an important part of enabling agent interactions to be flexible and robust. There are two types of failures in the Hermes methodology: *action failure* and *interaction goal failure*. An action failure is where an action does not achieve its interaction goal. For example, offering a price may fail to achieve the goal of agreeing on a price if the proposed price is rejected. An interaction goal failure is where an interaction goal cannot be achieved. For example, if the price proposed is rejected but a better offer cannot be made then the goal of agreeing on a price cannot be achieved.

An action failure can be recovered from by trying further actions ("*action retry*"). For example, given the actions in Figure 3, suppose that the Customer performs *ProposePrice* and the Merchant rejects the price (i.e. performs the *RejectPrice* action after considering the price). This results in the Customer using the *RectifyPriceRejection* action, which can be used to *retry*, for example by re-performing *ProposePrice* with a different price.

Alternatively, an action failure can cause the interaction goal to fail. In this case, either the interaction as a whole can be *terminated*, or the interaction can be *rolled back* to a previous IG (discussed below). If an action failure is to be handled by failing the interaction goal being pursued, then the appropriate action (e.g. *RectifyPriceRejection*) needs to request a termination of the current IG, or a rollback to a previous IG, specifying an earlier interaction goal as the rollback target.

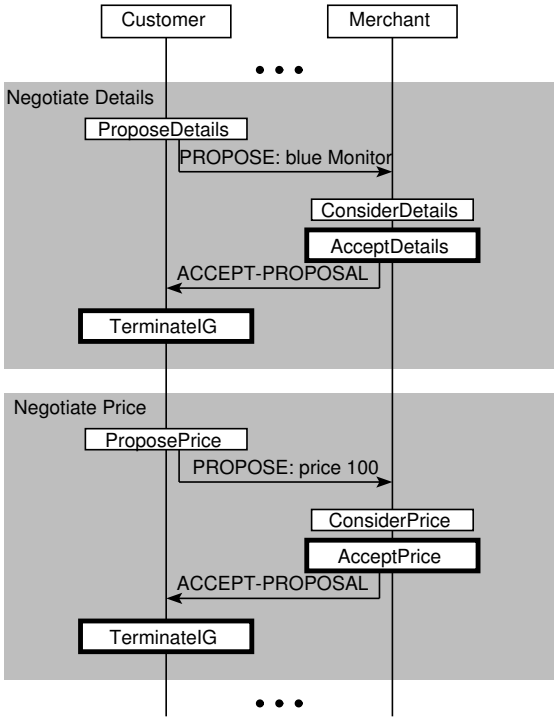

Fig. 5. (Partial) Action Message Diagram

Rollback is a failure recovery mechanism based on the idea that if previous interaction goals are re-achieved in a different manner, the failed interaction goal may be successfully achieved. Consider an example in which the Customer and Merchant have completed the *NegotiateDetails* interaction goal and have agreed on a product and its details (refer to Figure 2). They proceed to the *NegotiatePrice* IG but cannot agree on a price. One solution is to terminate the interaction, however, a better alternative is for the Customer and Merchant to move back to the *NegotiateDetails* IG, re-negotiate the details of the product and then proceed into the *NegotiatePrice* IG again with different product details.

When and where an interaction can be terminated is domain and application specific, and therefore it is up to the designer of the interaction to determine this. Similarly, when and to where rollback is permitted is domain and application specific, and is determined by the designer. The designer thus needs to indicate for each interaction goal whether termination is permissible from that IG, whether rollback is permissible, and if so, to which interaction goals it should be allowed to roll back to. For example, the *NegotiatePrice* IG allows termination, and allows rollback to the *DetermineAvailability* and *NegotiateDetails* IGs.

4 Implementing Goal-Oriented Interactions

We implement goal-oriented interactions by mapping the design artefacts to collections of plans, to be used by an agent platform which supports goal-plan agents (e.g. JACK[2], Jadex[3], JAM[4], and Jason[5]). In this section we briefly sketch how the design produced by following the Hermes process is mapped to collections of plans. For more details on this process, refer to [9].

There are three types of plans: *Interface*, *Coordination*, and *Achievement* plans. Coordination plans are used for coordinating the agents through the interaction. They contain coordination rules as defined by the temporal links and sub-goal relationships on the IG-hierarchy diagram. Achievement plans are used to take steps towards achieving an *interaction goal* (e.g. a *ProposeDetails* Achievement plan is a step towards achieving the *NegotiateDetails* IG). Interface plans (which are not part of the Hermean design process) are used to convert inter-agent messages into events and goal events for internal agent processing. For example, when a Merchant receives a *NegotiateDetails* message from a Customer. The message is handled by the Merchant's *HandleProposeMessage* Interface plan which converts the message to a *proposeDetails* goal event and dispatches it.

Figure 6 depicts an overview of the different plan types required for goal-oriented interactions and shows how they are inter-connected. The beliefset between the different plan types is used to coordinate the agents through the different interaction goals.

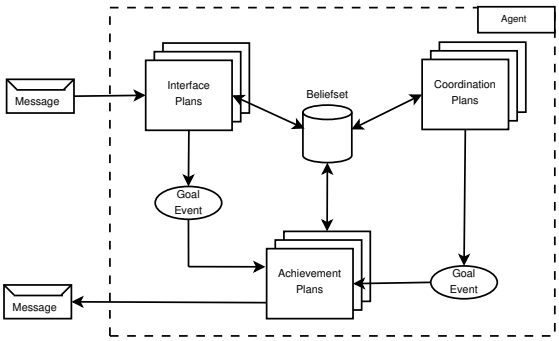

Fig. 6. Implementation Overview

5 Results

The design for the NetBill interaction that we have presented has been implemented by following the mapping of the previous section, and this implementation is able to pro-

[2] http://www.agent-software.com/
[3] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/
[4] http://www.marcush.net/IRS/irs_downloads.html
[5] http://jason.sourceforge.net/

duce a range of interactions, including the following sample execution trace in which a Merchant sells blue and yellow monitors at the minimum prices of $110 and $100 respectively, and a Customer seeks to purchase a monitor at a maximum price of $100 with the following colour preferences: red, blue, yellow, and green. It is obvious that a successful interaction will result in the Merchant selling a yellow monitor to the Customer at $100. In this example, we demonstrate how this is achieved and explain the advantages of using goal-oriented interactions and the Hermes methodology.

The first interaction goal to achieve is straight-forward (refer to Figure 2). The Customer checks the availability of monitors with the Merchant. If monitors are available, the interaction proceeds, otherwise it terminates. In this example, we assume that monitors are available.

To achieve the next interaction goal, *NegotiateDetails*, the Customer proposes a red monitor to the Merchant. However, as the Merchant only sells blue and yellow monitors, it sends a rejection message. Upon receiving the rejection message, the Customer uses the *action retry* failure recovery mechanism and proposes its second colour preference, blue.

As the Merchant sells blue monitors, it sends an accept message and the interaction proceeds to the *NegotiatePrice* IG. The Customer and Merchant then haggle for a while, however, as the Merchant's minimum price for blue monitors ($110) is higher than the Customer's maximum price ($100) an agreement cannot be reached, and the Merchant sends a reject message. At this point, the current IG fails and the interaction cannot proceed successfully unless the colour of the monitor can be altered. Thus, the Customer uses the *rollback* failure recovery mechanism and proposes a rollback to the *NegotiateDetails* IG. The Merchant rolls back to the proposed IG, sends an accept message to the Customer, and the Customer also rolls back to the *NegotiateDetails* IG.

As the Customer knows that red and blue monitors have resulted in failure, it proposes its next preference, yellow. The Merchant accepts the details and the *NegotiateDetails* goal is achieved. Haggling then re-commences, but this time the Customer is able to propose a price ($100) which the Merchant accepts. The interaction then continues with the *Exchange* IG.

6 Discussion

We have presented Hermes, a goal-oriented agent interaction methodology that includes a design process, failure recovery mechanisms and a mapping from design artefacts to an executable implementation. The goal-oriented approach to agent interactions adds a greater degree of flexibility and robustness to interactions than message-centric protocols. This is largely due to the emergent message sequences of goal-oriented interactions.

The flexibility and robustness of the interaction are further increased by adding failure recovery mechanisms, which essentially increase the number of legal message sequences by allowing agents to retry particular actions (i.e. action retry) and (in certain circumstances) to return to previous points in the interaction and re-perform parts of the interaction to attain more desirable results (i.e. rollback). Therefore, our example can be made even more flexible and robust by adding more actions and rollbacks, e.g.

more actions to achieve the *TransferGoods* and *Payment* IGs, and rollbacks from the *TransferGoods* and *Payment* IGs.

There are also other approaches which achieve similar results by moving away from message-centric protocols. These include approaches based on *social commitments* [5, 6, 10], Kumar *et al.*'s *landmark-based* approach [11], and Hutchison and Winikoff's *goal-plan* approach [4].

Approaches based on social commitments such as Yolum and Singh's commitment machines [5, 6] or the work of Flores and Kremer [10] capture the meanings of agents' actions in terms of their effects on social commitments. A social commitment is made from one agent to another and represents a condition which an agent will endeavour to bring about for another agent[6]. Commitments are attained and manipulated through inter-agent communicative acts. Therefore, in the course of interacting, agents create and manipulate commitments. Although both approaches allow for complex interactions which would be difficult to implement with message-centric protocols, their design aspects are not well defined. It is not obvious how to determine what commitments are required for a given interaction.

In Kumar *et al.*'s work [11], it is argued that the states of affairs brought about by a communicative act is more important than the communicative act itself. As such, the focus of the work is on the states of affairs, which are represented as landmarks. Thus, an interaction involves navigating through landmarks to reach a desired final state of affairs. Their work is theoretical in nature, and requires significant expertise in modal and temporal logics. Although an implementation ("STAPLE") has been mentioned, no details have been published beyond two posters [12, 13].

Hutchison and Winikoff's approach [4], involves modelling protocols as goals and plans. This involves determining the goals of the protocol and defining plans which are able to achieve the goals. Their work can be seen as a predecessor to our work: it gives neither a detailed design process, nor a mapping from design to implementation.

The Hermes design approach is incomplete in that it only covers interaction: other aspects of design such as determining what agent types should exist, are not addressed. Consequently one area for future work is to integrate Hermes into a complete agent-oriented methodology such as Prometheus [8]. Additionally, since some of the Hermean design diagrams appear to be similar to UML (e.g. Hermean action maps and UML activity diagrams), we will investigate to what extent Hermes can use the UML. Finally, since we aim for Hermes to be practical, tool support is an important area for future work.

Hermes' design methodology and notation will also require further refinement as we undertake research into adapting Hermes to function with a wider range of interactions, including those which involve many agents and many instances of a given role (such as auctions, where there are N bidders).

Currently, the implementation sketched in section 4 assumes that the agents are implemented using a goal-plan platform. One area for further work is to look at ways of supporting a wider range of agent platforms.

[6] Flores and Kremer define commitments as being to perform actions, rather than to bring about conditions.

Other, longer term, areas for future work include looking at the verification of goal-oriented interactions, and an experimental evaluation of the approach.

Acknowledgements

We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant LP0453486.

References

1. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent UML. In: Proceedings of the Fifth International Workshop on Agent Oriented Software Engineering (AOSE). (2004)
2. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science. Springer-Verlag (1985) ISBN 0-387-13723-8.
3. Sirbu, M., Tygar, J.D.: NetBill: An Internet Commerce System Optimized for Network-Delivered Services. IEEE Personal Communications **2** (1995) 34 – 39
4. Hutchison, J., Winikoff, M.: Flexibility and Robustness in Agent Interaction Protocols. In: Workshop on Challenges in Open Agent Systems at the First International Joint Conference on Autonomous Agents and Multi-Agents Systems. (2002)
5. Yolum, P., Singh, M.P.: Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artificial Intelligence (AMAI), Special Issue on Computational Logic in Multi-Agent Systems **42** (2004) 227–253
6. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS). (2002) 527–534
7. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent systems engineering. International Journal of Software Engineering and Knowledge Engineering **11** (2001) 231–258
8. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons (2004) ISBN 0-470-86120-7.
9. Cheong, C., Winikoff, M.: Hermes: Implementing goal-oriented agent interactions. In: Proceedings of the Third international Workshop on Programming Multi-Agent Systems (ProMAS). (2005)
10. Flores, R.A., Kremer, R.C.: A principled modular approach to construct flexible conversation protocols. In Tawfik, A., Goodwin, S., eds.: Advances in Artificial Intelligence, Springer-Verlag, LNCS 3060 (2004) 1–15
11. Kumar, S., Huber, M.J., Cohen, P.R.: Representing and executing protocols as joint actions. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, ACM Press (2002) 543 – 550
12. Kumar, S., Cohen, P.R., Huber, M.J.: Direct execution of team specifications in STAPLE. In: Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002), ACM Press (2002) 567–568
13. Kumar, S., Cohen, P.R.: STAPLE: An agent programming language based on the joint intention theory. In: Proceedings of the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2004), ACM Press (2004) 1390–1391