# Hermes: Implementing Goal-Oriented Agent Interactions

Christopher Cheong and Michael Winikoff

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

**Abstract.** Traditional approaches to designing agent interactions focus on defining agent interaction in terms of legal sequences of messages. These message-centric approaches are not a good match with autonomous proactive agents since they unnecessarily limit the agents' autonomy and flexibility. The Hermes methodology proposes an approach for designing agent interactions in terms of *interaction goals*. In this paper we focus on how Hermes designs can be implemented by mapping the design artefacts to collections of plans.

## 1 Introduction

Existing approaches to designing agent interactions are *message-centric*. These approaches, such as using Petri nets, AUML interaction protocols [1], or finite state machines, are not a good fit with autonomous proactive agents. For instance, they do not support goals, and legal message sequences are explicitly defined in terms of messages and combining forms such as sequencing, alternatives, and loops. One consequence is that autonomous agents are forced to follow these prescribed message sequences, thus limiting the flexibility of interactions. Furthermore, due to the limited flexibility, interactions are also less robust since there are limited recovery options.

The *Hermes*[1] methodology [2, 3] aims to address this by designing interactions in terms of goals, and allowing agents to achieve these goals flexibly and robustly. Hermes uses *Interaction Goals* (IGs) as a basis for designing interactions, along with available actions and timing dependency constraints. Possible message sequences are determined by the agents in accordance with these interaction goals, actions and constraints, allowing message sequences to *emerge* from the interaction. This results in a greater degree of flexibility and robustness, and consequently, Hermes is better suited for proactive autonomous agents than current message-centric approaches.

The Hermes methodology aims to be a complete and *practical* approach to developing agents that interact flexibly and robustly. The design aspects of Hermes, including a design process, notations, techniques, and failure handling mechanisms, have been described elsewhere [3].

In this paper we focus on the *implementation* of goal-oriented agent interactions that have been designed using the Hermes methodology. We present a set of guidelines

[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).

which can be used to implement a Hermean design, and apply them to implement a sample design in Jadex [4]. To illustrate our work we use an e-commerce protocol based on the NetBill [5] protocol in which a Customer purchases goods online from a Merchant. The NetBill protocol was chosen since a number of other non-message-centric approaches have used it [6–8], and by using the same example it becomes easier to compare our approach to existing approaches.

In section 2 we briefly describe the Hermes methodology, including the design artefacts that are used in the implementation phase. Section 3, which is the central contribution of this paper, describes how a Hermean design can be systematically implemented by mapping design artefacts to a collection of plans. We then conclude in section 4.

## 2 Background: Hermes

In this section, we briefly explain the Hermes methodology. Since Hermes has been described elsewhere [2, 3] we only describe here the notations and outcomes of the design process, as these are needed in order to understand the implementation described in section 3. The aspects of Hermes which are not necessary in order to present the implementation, including the design process, are not presented here and we refer the reader to [3] for a more detailed description of the Hermes methodology.

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. Steps marked as *Final Design Artefact*s are used directly in implementing the design, and these artefacts are described in the following sections. Other artefacts, such as *Action Sequence* and *Action Message* diagrams, are used to validate that the implementation matches the design. Although these intermediate design artefacts can be useful in generating test cases for the implementation, they are not discussed further in this paper.

### 2.1 Interaction Goal Hierarchy Diagram

An *Interaction Goal* (IG) is a goal of the interaction, for example to agree on a price, or to make a trade. Note that it is *not* a goal of an individual agent, but of the interaction as a whole. The interaction goals and their relationships are captured using an *Interaction Goal Hierarchy Diagram* which is effectively a goal-tree, similar to those used in agent-oriented methodologies such as MaSE [9] or Prometheus [10]. This represents IGs as circles containing the name of the IG, the *initiator* role which initiates trying to achieve the given IG[2], and the roles which are involved in achieving the IG. Interaction Goals can be decomposed into sub-goals, where achieving an IG's sub-goals will achieve that interaction goal. For example (refer to Figure 2), the *Trade* IG can be decomposed into the IGs *Agree* and *Exchange*. Sub-goal decomposition is indicated on the diagram by

[2] An upward arrow, $\uparrow$, is used to indicate that the initiator for the IG is the same role that initiated the parent IG.

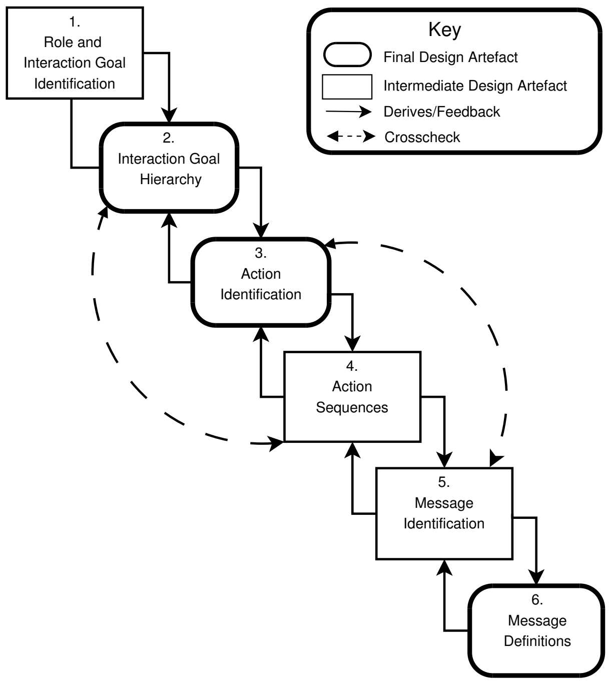

**Fig. 1.** Hermes Methodology Overview Diagram

(undirected) lines. In addition to capturing sub-goal relationships, the Interaction Goal Hierarchy Diagram also captures *temporal dependencies*, depicted as directed arrows between IGs. For example, *Exchange* is dependent on *Agree* and thus, *Agree* must be completed before *Exchange* can be started.

## 2.2 Action Maps

An *action* is a step, taken by a single agent, that moves the interaction closer to achieving its goal. The actions that can be used to achieve a given interaction goal and their relationships are captured using an *action map* for that (leaf[3]) interaction goal. Action maps are divided into “swim lanes”; one per role involved in the interaction goal. Each swim lane contains different types of actions (see below) that can be performed by that role.

The key in Figure 3 illustrates four different action types, each of which has a different meaning and use[4]. 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

[3] There is no need to identify actions for non-leaf-level goals, since they are completed when their sub-goals are completed.

[4] The fourth type, *Final Independent Action* is not used in the example, and so is not explained here.

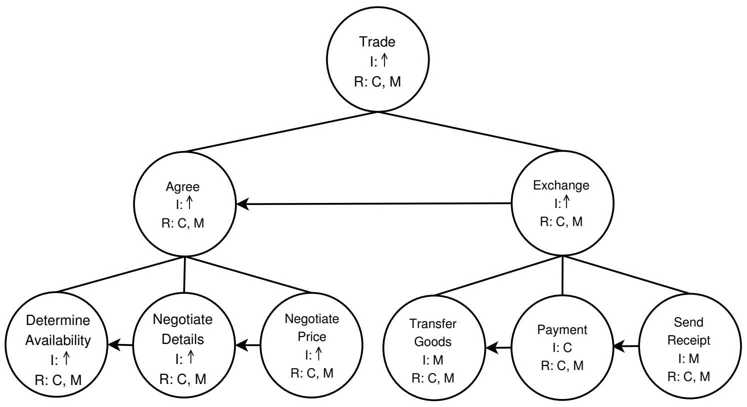

**Fig. 2.** Interaction Goal (IG) Hierarchy Diagram

another action. *Independent Action*s are entry points into interaction goals. 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. Note that performing a final 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).

Causality constraints (depicted in Figure 3 as directed arrows) specify that certain actions cannot take place until other actions have occurred. For example, the *ProposePrice* action causes the *ConsiderPrice* action. Where an action is causally linked to more than one action the causality arrows are intended to depict alternative possibilities. For example, the *ConsiderPrice* action either triggers *AcceptPrice* or *RejectPrice*, but not both. Which action is triggered will depend on certain conditions or states and it can be useful to label the causality arrows with the condition or state.

## 2.3 Messages

When using Hermes, messages are identified by considering action sequences and action message diagrams (not described in this paper). What is important to know in order to be able to understand the implementation is that the outcome of this process is a collection of messages. A message is defined whenever an action in the action map triggers another action that is performed by a different role.

## 2.4 Handling Failure

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

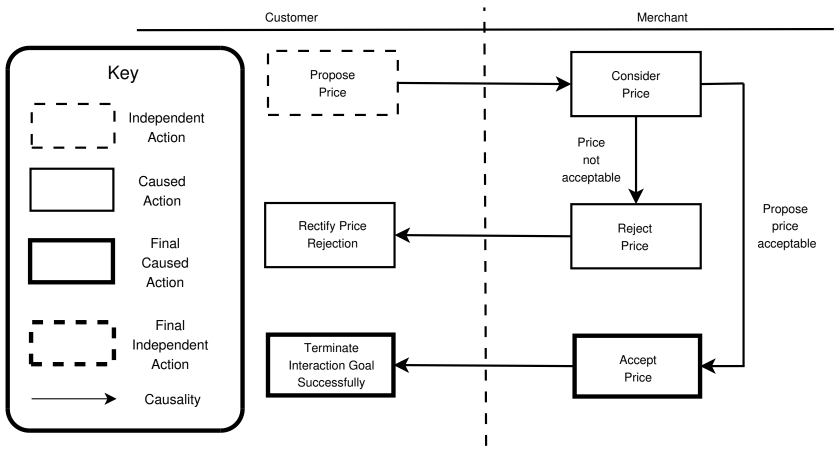

**Fig. 3.** Action Map for the *NegotiatePrice* IG

price is rejected. An action failure can be recovered from by trying further actions ("*action retry*"), or the interaction goal can be failed. 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.

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. Interaction goal failure can be handled either by failing the entire interaction, or by rolling back to an earlier interaction goal ("*rollback*").

Rollback is a failure recovery mechanism based on the idea that if a previous interaction goal is re-achieved in a different manner, the failed interaction goal may be successfully achieved. For example, if the Merchant and Customer have agreed on a product and its details (*NegotiateDetails*) but cannot agree on a price (*NegotiatePrice*) then going back and agreeing on different product details may enable agreement on a price to be reached.

Terminating the interaction and rolling back may not be appropriate for all IGs. For example, if goods have already been transferred then neither rollback nor termination should be permitted. Therefore, for each IG the designer indicates whether termination is permitted, whether rollback is permitted, and if rollback is permitted, to which (earlier) IGs should rollback be allowed. For example, the *NegotiatePrice* IG allows termination, and allows rollback to the *DetermineAvailability* and *NegotiateDetails* IGs.

## 3 Implementing the Design

In this section we discuss how the Hermean design can be systematically mapped to an implementation. Goal-oriented interactions are implemented by mapping design artefacts, such as the *Interaction Goal Hierarchy* and the *Action Map*s, to collections of plans which can be used by agent platforms. Since the design is in terms of interaction goals, we have chosen to develop an implementation scheme which targets agent platforms where the behaviour of agents is defined using plans and goals. Such platforms include those based on the Belief Desire Intention (BDI) model, such as JACK[5], Jadex[6], JAM[7], and Jason[8]. For the purposes of our work, we have implemented our design using the Jadex agent platform [4], however a Hermean design can be implemented using any of the aforementioned agent platforms.

Figure 4 shows an overview of the implementation, including different plan types and their inter-connections. Coordination plans are derived from interaction goals and are used to coordinate the participating agents through the interaction. Achievement plans are directly derived from actions and are steps which agents take towards completing an interaction goal. Interface plans are not derived from any design artefacts, but are used to transform inter-agent messages into goals and events for intra-agent processing. For example, when a Merchant receives a *NegotiateDetails* message from a Customer, the message is handled by the Merchant's *HandleProposals* Interface plan, which converts the message to a *proposeDetails* goal event and dispatches it for internal agent processing.

In the following sub-sections, we further describe the different components of the implementation. As is typical for descriptions of implementation techniques and algorithms, the description is necessarily somewhat detailed.

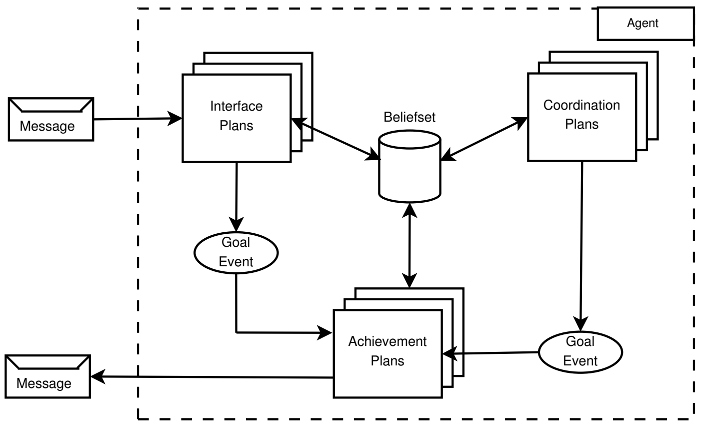

**Fig. 4.** Implementation Overview

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

## 3.1 Agent Beliefs and Interaction Goal State Representation

Agents are able to coordinate through the interaction goals via the use of a beliefset that is shared between the different types of plans within a given agent (refer to Figure 4). One important use of the beliefs is to represent the state of the interaction goals. This is done with a combination of three Boolean beliefs for each interaction goal: *in*, *finished*, and *success*. The *in* belief indicates that the IG is currently active. The *finished* belief is used to indicate whether the IG has been completed, whilst *success* indicates whether the IG has been successful.

The states of the IG and valid transitions between states are shown in Figure 5. The dashed circles represent intermediate states that have no conceptual meaning, but are required to change state from *active* to either *succeeded* or *failed*. The Boolean string in parentheses show the values of the three beliefs, *in*, *finished*, and *success*, respectively.

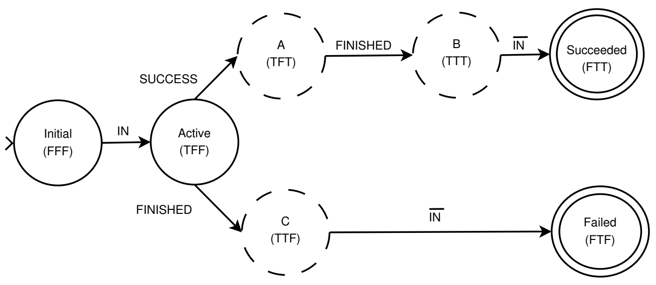

**Fig. 5.** Interaction Goals States

The general structure for an agent's beliefset, along with a brief summary of its use, is shown in Table 1.

| **Belief** | **Use** |
|---|---|
| *role* | Identifies the agent's role in the interaction. |
| *initiator* | Identifies the interaction's initiator. |
| Interaction Goal Initiators | A series of beliefs which identifies the initiator of each IG, e.g. *tradeIGInitiator* (one per IG). |
| Interaction Goal States | A series of beliefs used to represent the state of IGs, i.e. *in*, *finished*, and *success*. Used for Coordination-Achievement Plan connections. |
| Interaction Goal Retries | A series of beliefs for retrying IGs. One for each IG that is allowed to be retried (refer to Section 3.4), e.g. *retryNegDetails*. |
| Interaction Specific Beliefs | Beliefs which are specific to the given interaction, e.g. *merchantName*, *product*, etc. |

**Table 1.** Belief structure and use

### 3.2 Coordination Plans

Coordination plans are directly derived from interaction goals from the *Interaction Goal Hierarchy* diagram. There are two flavours of Coordination plans: *leaf-node* and *non-leaf-node*. The *non-leaf-node* variety are obtained from IGs which have at least one sub-IG. These types of Coordination plans deal with coordination between themselves and other Coordination plans. *Leaf-node* Coordination plans are derived from IGs which have no sub-IGs. These plans deal with coordination between themselves and actions.

All non-leaf-node Coordination plans follow the same structure as the *Trade Coordination* plan, shown in Algorithm 1. The coordination rules, obtained from the sub-goal and temporal dependencies in Figure 2, are shown in the *Coordination* section of the plan (lines 1 – 11).

**Algorithm 1** Coordination Plan for *Trade* (Non-leaf-node Coordination Plan)

```
Require: inTrade == true
 1: // Coordination
 2: inAgree = true
 3: waitFor(finishedAgree and not inAgree)
 4:
 5: if agreeSuccessful then
 6:    inExchange = true
 7:    waitFor(finishedExchange and not inExchange)
 8:    if exchangeSuccessful then
 9:       tradeSuccessful = true
10:    end if
11: end if
12:
13: // Synchronization (with other Coordination plans)
14: finishedTrade = true
15: inTrade = false
```

The *require* statement in Algorithm 1 specifies the trigger condition for the *Trade Coordination* plan. Thus when *inTrade* becomes *true*, i.e. the Trade IG is *active*, the plan begins execution. The initial step of the plan (line 2) is to set the Agree IG to the *active* state which triggers the *Agree Coordination* plan. The *waitFor()* statement blocks until the *Agree interaction goal* enters a final state (i.e. *succeeded* or *failed*), which is set by the *Agree coordination* plan.

When the Agree IG is achieved, the *Agree Coordination* plan sets the appropriate beliefs to move the Agree IG from *active* to either *succeeded* or *failed*, depending on the outcome. The condition in the *Trade Coordination* plan's *waitFor()* (line 3) is then satisfied and the plan continues executing.

If the *Agree* IG is successful, the interaction proceeds onto the *Exchange* goal, otherwise it is terminated. This is based on the temporal dependency link between the *Agree* and *Exchange* goals in Figure 2.

The *Synchronisation* section (lines 13 – 15 in Algorithm 1) is used to set the *finished* and *in* beliefs to move the IGs into a final state (either *succeeded* or *failed*).

The sub-goal relationships between the *Trade*, *Agree*, and *Exchange* interactions goals (refer to Figure 2) are implemented by the *Trade* Coordination plan waiting for completion of the *Agree* and *Exchange* IGs (lines 3 and 7 in Algorithm 1). The dependency between *Agree* and *Exchange* is achieved by triggering the *Exchange* coordination plan after the *Agree* IG has been successfully completed (lines 5 and 6).

The leaf-node Coordination plans follow the same structure as the *NegotiatePrice* Coordination plan, shown in Algorithm 2 and are slightly different to non-leaf-node Coordination plans. The main difference lies in lines 1 – 5 of the leaf-node Coordination plan. These lines dispatch a *proposePriceGoal* if the agent's role is the initiator of this IG. Every agent has a set of common beliefs which list the *initiator* of every IG and also a *role* belief which indicates their role in the interaction (e.g. *customer* or *merchant* in this particular scenario, refer to Table 1). In our e-commerce example, the Customer is the initiator for the *NegotiatePrice* IG, thus it takes the initiative and proposes a price to the Merchant.

Although both leaf and non-leaf plans have a Synchronisation section at the end, they serve different purposes. In the non-leaf-node plans, the Synchronisation section synchronises Coordination plans with other Coordination plans, whereas in the leaf-node plans, it is used to synchronise Coordination plans with Achievement plans. The *waitFor()* statement (line 8 of Algorithm 2) allows an arbitrary number of Achievement plans to run. When the final Achievement plan (derived from a final action) has completed its execution, the IG is in either the *B* or *C* state, shown in Figure 5 (i.e. the *finished* belief is set to *true*). This in turn un-blocks the *waitFor()* method and allows the Coordination plan to change the IG state to either *Succeeded* or *Failed*.

**Algorithm 2** Coordination Plan for *NegotiatePrice* (Leaf-node Coordination Plan)

```
Require: inNegPrice == true
1: if not negPriceSuccess then
2:     if role == initiator then
3:         dispatch(new proposePriceGoal())
4:     end if
5: end if
6:
7: // Synchronisation (with Achievement plans)
8: waitFor(finishedNegPrice)
9: inNegPrice = false
```

## 3.3 Achievement Plans

Achievement plans are derived directly from actions in the *Action Map*s. There are some slight variations to the Achievement plans depending on which type of action they are implementing, however, Achievement plans all follow the same structure as the *PriceAccepted* Achievement plan, shown in Algorithm 3.

Achievement plans are triggered via goal events that are usually dispatched from Interface or Coordination plans (refer to the first line of Algorithm 3). They have two

**Algorithm 3** Achievement Plan for the *PriceAccepted* Action

```
Require: priceAcceptedGoalEvent and priceAcceptable()
 1: // Synchronisation (with Coordination plan)
 2: waitFor(inNegPrice)
 3:
 4: // Achieve IG (application specific)
 5: price = priceAcceptedGoalEvent.getPrice()
 6: if action achieves IG then
 7:     negPriceSuccess = true // Action achieves IG
 8: end if
 9:
10: // Finish IG, only done if action is final
11: // Synchronisation (with Coordination plan)
12: if action is final then
13:     finishedNegPrice = true
14: end if
```

distinct sections: *Synchronisation* and *Achieve*. The Synchronisation section is similar to that of the Coordination plans and synchronises the Achievement plan with its respective Coordination plan.

All Achievement plans begin with a Synchronisation section (lines 1 and 2), which acts as a guard condition and allows them to execute only when the interaction is achieving the correct interaction goal. Some achievement plans are only applicable in certain situations, and the additional condition, for example only agreeing to a price if it is acceptable, is included in the required condition for the plan to run (first line of Algorithm 3). Whether an action is only applicable in certain situations can be seen in the action map as labels on the causality links. The definition of the condition (e.g. *priceAcceptable()*) is provided by the agent in question. The following section, Achieve IG (lines 4 and 5), contains application-specific code for the action. Furthermore, if an Achievement plan successfully achieves an IG (determined by application-specific conditions), it sets the *success* belief to *true* (lines 6 – 8).

Achievement plans implemented from a *final* action have an additional Synchronisation section at the end (lines 10 – 14) which signals the end of the Achievement plans' execution and returns processing control back to the appropriate Coordination plan for the given IG.

Note that unlike Coordination plans the synchronisation between actions is implicitly handled by having actions trigger other actions with internal events (within a role) and messages (between roles).

### 3.4 Implementing Failure Handling Mechanisms

This section details how the failure handling mechanisms described in Section 2.4 are implemented. An action failure can be addressed by either terminating the interaction or by attempting to recover from the failure (by using *action retry* or *rollback*).

Termination is implemented by adding three actions to each interaction goal that permits termination: *RequestTermination*, *TerminateOnRequest* and *Terminate*. For example, when the Customer wants to terminate the interaction, it uses the *RequestTermination* action (which is only available in particular IGs as defined by the interaction designer). The Merchant responds by using the *TerminateOnRequest* action. Once the Merchant has terminated the interaction, it replies to the Customer, which then performs the *Terminate* action. The interaction is then ended.

The *RequestTermination* plan is an achievement plan that simply requests a termination of the interaction from the current interaction goal. The *TerminateOnRequest* plan, also an achievement plan, contains IG specific details to terminate the interaction at that point. This may include matters such as re-setting beliefs or general clean up of the interaction and agent state.

*Action retry* can be implemented by incorporating an action which loops through the interaction again. For example, in Figure 3, the *RectifyPriceRejection* could be used to loop through the interaction again. This will involve *RectifyPriceRejection* triggering *ProposePrice* to send a new proposal with a higher price to the Merchant (not shown in Figure 3). Of course, an upper limit would have to be placed on the price to ensure that the Customer and Merchant do not haggle over the price endlessly.

The implementation of *rollback* is the most complicated of the three failure handling mechanisms. It is implemented by adding the following actions to every interaction goal which permits rollback: *ProposeRollback* and *Rollback* (one for each role). For example, if the Customer and Merchant cannot agree on a price, it is possible for them to rollback to re-negotiate the details and then try to negotiate on the price again. The Customer will perform *ProposeRollback* and the Merchant will use *Rollback* to roll back to the *NegotiateDetails* IG, after which the Merchant will send a message to let the Customer know that it has rolled back. The Customer will then use *Rollback* to roll back to the *NegotiateDetails* IG.

The *ProposeRollback* achievement plan simply sends a request to roll back. The *Rollback* achievement plan is the plan that does the actual rolling back. Rollback plans follow the same structure as Algorithm 4, an example of the Customer's rollback plan for rolling back from the *NegotiatePrice* IG to the *NegotiateDetails* IG.

Rollback is achieved by re-starting the interaction from a specific interaction goal. Therefore, when the *rollbackGoalEvent* is received and the interaction is in the correct IG (lines 1 and 2), the first step is to terminate the current IG, which will terminate the entire interaction. This is achieved by setting its three state representation beliefs to *false* (lines 4–6). Once the interaction is terminated (line 8), the appropriate beliefs are set to flag at which IG the interaction should re-start from (lines 9–17). This includes setting a Boolean belief (*retryNegDetails*) to notify that a rollback has been issued (line 17). The *retryNegDetails* belief is used when the interaction re-starts so that the *ProposeDetails* achievement plan will request a different solution (i.e. colour) to the previous one in order to achieve a different result so that the *NegotiateDetails* IG may succeed. The remainder of the *rollback* plan simply re-starts the interaction and notifies any relevant agents. In the case of the Customer, it does not have to notify any agents that it has rolled back.

**Algorithm 4** Customer Rollback Plan (from *NegotiatePrice* to *NegotiateDetails*)

```
Require: rollbackGoalEvent
 1: // Synchronise (with Coordination plan)
 2: waitFor(inNegPrice)
 3: // 1. Terminate current IG
 4: negPriceSuccessful = false
 5: finishedNegPrice = true
 6: inNegPrice = false
 7: // 2. Wait for apex IG to terminate
 8: waitFor(finishedTrade and not inTrade)
 9: // 3. Set appropriate beliefs to re-start interaction and to begin at desired IG (shortcut)
10: // 3.1. Reset current IG beliefs
11: finishedNegPrice = false
12: // 3.2. Set beliefs of IG to begin next interaction from (shortcut)
13: negDetailsSuccessful = false
14: finishedNegDetails = false
15: inNegDetails = true
16: // 3.3. Set beliefs for "retry" attempt
17: retryNegDetails = true
18: // 4. Re-start interaction, set "in" belief of apex stage to "true"
19: inTrade = true
20: // 5. Notify relevant agents
```

## 3.5 Sample Execution

In this section, we provide an example trace of the implementation in which the Customer is attempting to purchase a monitor at the maximum price of $100 with the following colour preferences: red, blue, yellow, and green and the Merchant is selling blue and yellow monitors at the minimum prices of $110 and $100 respectively. In this situation, for a successful sale to occur, the Merchant must sell a yellow monitor to the Customer at $100. We demonstrate how such an interaction executes on an implementation based on a goal-oriented Hermean design. Figure 6 presents the initial execution steps graphically.

The interaction begins with the Customer receiving a request to start the interaction. This is handled by its Interface plan, *HandleRequests*, which flags a Boolean belief (*inTrade*) to start the interaction. The Customer then enters the *Trade* IG, then the *Agree* IG, and then *DetermineAvailability* IG (based on the Interaction Goal Hierarchy and the Coordination plans, refer to Figure 2 and Algorithm 1 respectively).

The *DetermineAvailability* Coordination plan executes and triggers its achievement plan, *RequestAvailability*, which executes and sends a message to the Merchant, enquiring about the availability of a monitor.

The message is received by the Merchant's Interface plan and is converted into a *checkAvailablity* goal which triggers the Merchant's *GoodsAvailable* plan (since it does sell monitors). Although the *GoodsAvailable* plan is triggered, it does not execute as it is waiting for the Merchant to enter the *DetermineAvailability* IG. Since the Merchant has not started the *Trade* interaction, when it receives the message from the Customer,

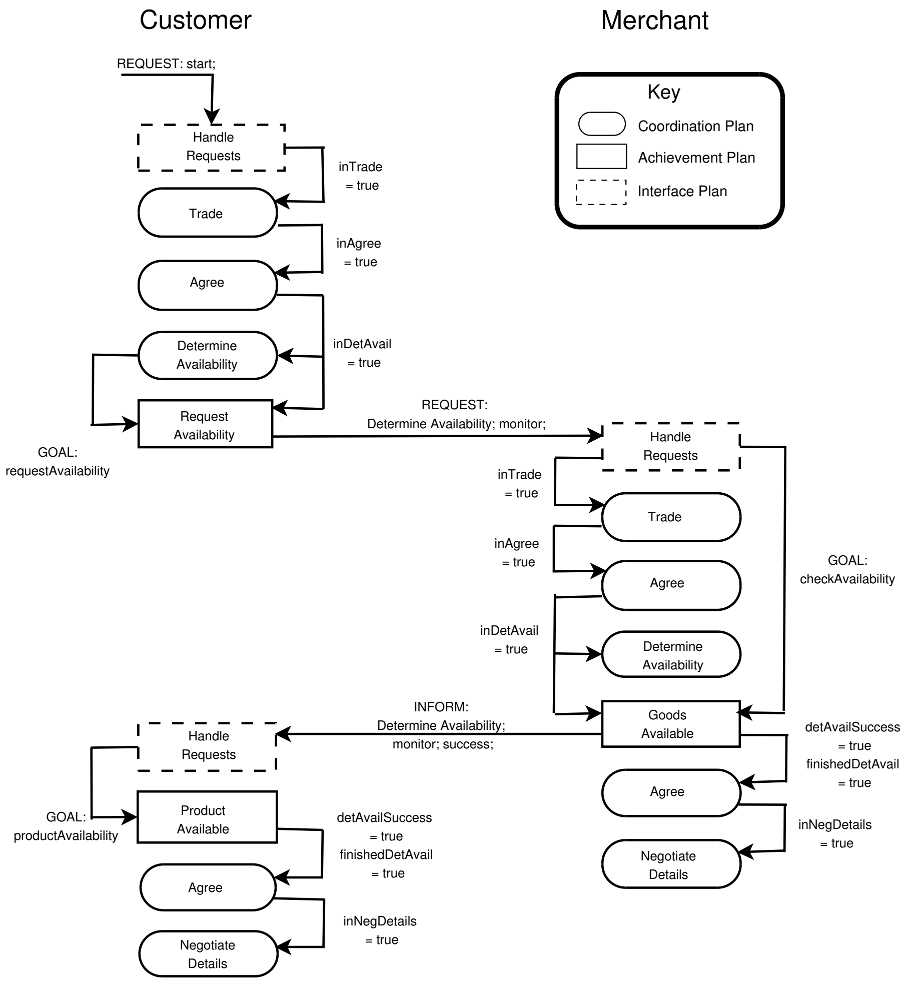

**Fig. 6.** Sample execution

it starts the interaction (at the *Trade* IG) and moves into the *DetermineAvailability* IG. The *GoodsAvailable* plan then executes and sends a message to the Customer, informing it that there are monitors available. The *DetermineAvailability* IG is then successfully achieved for the Merchant.

The Customer's Interface plan, *HandleRequests*, handles the message and converts it into a goal for internal agent processing. The *DetermineAvailability* IG is successfully achieved for the Customer, it moves into the *NegotiateDetails* IG and its *ProposeDetails* achievement plan is triggered. The *ProposeDetails* plan sends a message to the Merchant to request a red monitor. As the Merchant does not have red monitors, it sends a rejection message to the Customer. The Customer's *DetailsRejected* plan is triggered (after the message is converted to a goal by the Customer's Interface plan). The *DetailsRejected* plan then creates a new goal to trigger the *ProposeDetails* plan to send a message requesting a blue monitor.

As the Merchant sells blue monitors, it returns a positive reply to the Customer and moves into the *NegotiatePrice* IG. The Customer receives the message and also moves into the *NegotiatePrice* IG.

The negotiations over the price of the monitor proceed similarly to the negotiation of the colour of the monitor. When the Merchant rejects the Customer's highest price of \$100 (as the Merchant's minimum is \$110), the Customer's *RectifyPriceRejection* plan triggers the *ProposeRollback* plan, which sends a rollback request to the Merchant. The Merchant then uses its *Rollback* plan to return to the *NegotiateDetails* IG and notifies the Customer that it has successfully rolled back. The Customer then executes its *Rollback* to roll back to the *NegotiateDetails* IG.

The Customer and the Merchant re-negotiate the colour of the monitor and settle on yellow. The interaction then proceeds to the negotiation of the price and is able to terminate successfully.

## 4 Conclusion

We have (briefly) outlined the Hermes design process, focussing on its notations and outcomes, and then presented a mapping from Hermes designs to plans that realise the designed interaction. This mapping produces collections of plans that can be implemented using a goal-plan agent architecture.

A Hermean design for a trading scenario based on the NetBill protocol has been implemented by following this mapping. We have also implemented a Hermean design for a brokering scenario based on [11]. These implementations have shown that the mapping works and that the implementations are capable of realising flexible and robust interactions.

### 4.1 Related Work

There are other approaches which aim to provide more flexible agent interaction by moving away from a message-centric approach. These include approaches based on *social commitments* [7, 8, 12], Kumar *et al.*'s *landmark-based* approach [13], and Hutchison and Winikoff's *goal-plan* approach [6].

Approaches based on social commitments such as Yolum and Singh's commitment machines [7, 8] or the work of Flores and Kremer [12] captures 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[9]. 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 [13], it is argued that the state 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 [14, 15].

Hutchison and Winikoff's approach [6], 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.

### 4.2 Future Work

One area for future work is to develop a mapping for non-goal-plan agents. In addition there are a number of areas where the Hermes methodology can be further developed including the provision of tool support for the design process. One possibility that we are considering is to develop this by extending the Prometheus Design Tool[10]. We envisage that this tool support will encompass the generation of skeleton code in accordance with the mapping described in this paper.

The mapping described in this paper targets plan-goal agent platforms. One area for future work is to target other platforms that do not define agents in terms of goal-triggered plans. One approach is to compile down to interaction protocols using some representation such as finite state machines, Petri nets, or AUML.

Agent interactions are only one part of creating an agent system. As such, we intend to integrate Hermes with an agent methodology, such as Prometheus [10]. The design methodology and notation will also require further refinement as we undertake research into adapting Hermes to function with protocols which involve many agents. Other, longer term, areas for future work include looking at the verification of goal-oriented interactions, and an experimental evaluation of the approach.

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

[10] http://www.cs.rmit.edu.au/agents/pdt

**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. Cheong, C., Winikoff, M.: Hermes: A Methodology for Goal-Oriented Agent Interactions (Poster). In: The Fourth International Joint Conference on Autonomous Agents and Multi-Agents Systems. To appear. (2005)
3. Cheong, C., Winikoff, M.: Hermes: Designing goal-oriented agent interactions. In: Proceedings of the 6th International Workshop on Agent-Oriented Software Engineering (AOSE-2005). (2005)
4. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: Implementing a BDI-Infrastructure for JADE Agents. EXP - In Search of Innovation (Special Issue on JADE) **3** (2003) 76 – 85
5. Sirbu, M., Tygar, J.D.: NetBill: An Internet Commerce System Optimized for Network-Delivered Services. IEEE Personal Communications **2** (1995) 34 – 39
6. 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)
7. 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
8. 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
9. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent systems engineering. International Journal of Software Engineering and Knowledge Engineering **11** (2001) 231–258
10. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons (2004) ISBN 0-470-86120-7.
11. Mbala, A., Padgham, L., Winikoff, M.: Design options for subscription managers. In: Proceedings of the Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS). (2005)
12. 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
13. 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
14. 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
15. 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