Improving Flexibility and Robustness in Agent Interactions: Extending Prometheus with Hermes

Christopher Cheong and Michael Winikoff

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

Abstract. A crucial part of multi-agent system design is the design of agent interactions. Traditional approaches to designing agent interaction use interaction protocols, which focus on defining legal sequences of messages. Such approaches do not naturally exhibit flexibility and robustness, and are not a good match for intelligent software agents which are autonomous, proactive, flexible and robust. The *Hermes* approach to designing agent interaction uses interaction goals, actions, and a number of failure recovery mechanisms to give a design methodology which is a good fit with intelligent software agents. However, the Hermes approach only covers part of the design process. In this paper we integrate Hermes with the Prometheus methodology, thus providing a complete methodology for designing multi-agent systems where interaction design is goal-oriented, yielding flexible and robust interactions.

1 Introduction

Since intelligent agents are social entities, a crucial part of multi-agent system design is the design of interactions between agents. Typical approaches to designing agent interactions, such as AUML [1], which is used in agent-oriented design methodologies such as Prometheus [2], Gaia [3], and Tropos [4], define interactions in terms of legal message sequences. These message-centric approaches to interaction design restrict the autonomy of intelligent agents as the agents are forced to follow them mechanistically.

The agents which partake in these interactions are goal-oriented entities, in which autonomy and proactivity are key concepts. Agents are able to deliberate about their goals, that is, determine which goal to achieve and how to achieve it. As such, there is a mismatch between the agents and their interactions.

Intelligent agents are flexible and robust. Similarly, it is desirable for the interactions between agents to also be flexible and robust, and to exploit beneficial characteristics of intelligent agents. Alternative approaches to message-centric protocols, such as the goal-oriented interactions of Hermes [5,6], are required to achieve this.

The main idea behind goal-oriented interactions is that agents partake in interactions because they have certain goals in common to achieve and thus the interaction is mutually beneficial. The main inspiration of such interactions is the way intelligent agents are structured and defined (i.e. in terms of goals and the plans which achieve them). Therefore, the interaction is modelled in terms of interaction goals (IGs), temporal constraints and actions. The IGs are common goals that the interacting agents

A. Garcia et al. (Eds.): SELMAS 2005, LNCS 3914, pp. 189–206, 2006.

want to achieve, whilst the temporal constraints allow for temporal ordering and dependencies between IGs. The agents are guided through the interaction by the IGs, which are organised in a hierarchy (with temporal constraints), and actions are used to achieve the IGs.

Unlike message-centric protocols, the message sequences are neither prescribed nor are they the focus of the interaction. Rather, the sequences emerge as interacting agents decide which IGs to achieve and which actions to use to achieve these IGs. As such, the focus is on the interaction goals and the exchanged messages are by-products of the interaction. The combination of goal-oriented interactions and intelligent agents is a more natural fit than the combination of message-centric protocols and intelligent agents.

However, the Hermes approach only addresses the design of interactions. In order to obtain a complete methodology for designing agent systems we integrate Hermes with the Prometheus methodology, thus providing a complete methodology for designing multi-agent systems where interaction design is goal-oriented, yielding flexible and robust interactions. We also refine the Hermes methodology and present a more detailed process for deriving action maps than previously described [5].

Section 2 provides the required background, whilst in section 3 we explain the integrated process and provide a partial example design on a case study. Section 4 concludes our paper.

2 Background

2.1 Hermes

The Hermes methodology is a goal-oriented approach to agent interactions which covers design and implementation. In this section the design aspects are briefly covered. For a more detailed explanation of the design process refer to [5]. The implementation process is not explained in this paper (see [6] for details).

Figure 1 provides an overview of the Hermes methodology. The first two steps involve determining which agents are to participate in the interaction (i.e. which roles they will undertake) and what they have to achieve in the interaction (i.e.interaction goals). The interaction goals (IGs) are goals of the interaction that are common to the agents involved in the interaction. Once IGs have been identified, they are structured into an Interaction Goal Hierarchy (IGH), and temporal constraints are added.

Compound IGs, that is IGs that have sub-IGs, such as *Order Book* and *Retrieve Details* (refer to Figure 6), are achieved when their own sub-IGs are achieved. Atomic IGs, such as *Retrieve Credit Card Details* from Figure 6 are achieved by agents executing appropriate actions: discrete steps that single agents take towards achieving an IG. These action flows are organised into *action maps* [1]. Therefore, each atomic IG should have a corresponding action map. Once defined, the action maps are then improved iteratively.

The last two steps of the Hermes design process require identification of messages and formally defining their structures. Messages are needed between actions that occur

[1] Action maps dictate the flow of actions between agents involved in an interaction. More details on action maps can be found in Section 3.4.

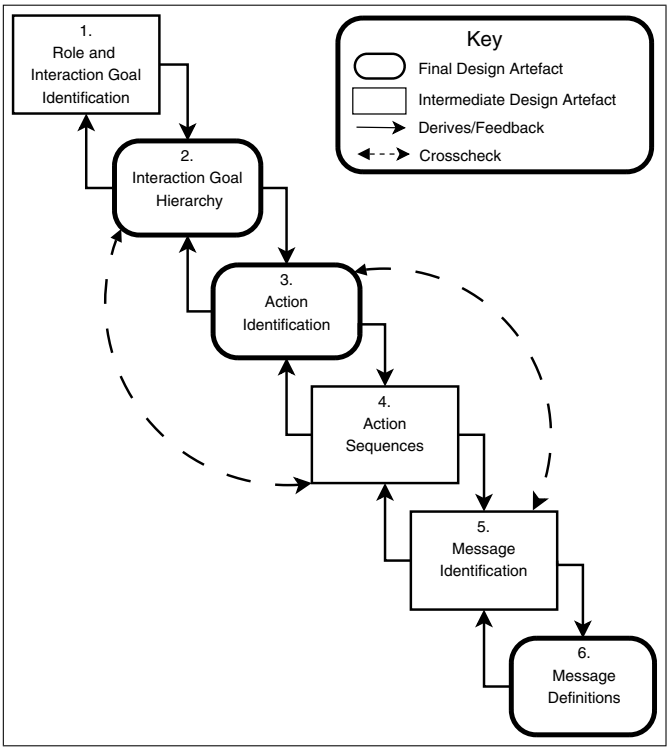

Fig. 1. Hermes Overview Diagram

in different roles that have a causality arrow in between them. For example, in Figure 7 the *Delivery Manager* will need to send a message to the *Stock Manager* after the *Log Outgoing Delivery* action to trigger the *Log Books Outgoing* action. Although Hermes provides guidelines for identifying messages, it does not provide any guidelines for developing message format. The interaction designer is free to use KQML, FIPA, SOAP or message types provided by a particular agent platform.

2.2 Prometheus

Prometheus is an agent-oriented software engineering methodology that aims to be practical and usable by software developers and undergraduate students, not only by agent researchers and postgraduate students. Its distinguishing features include that it is complete (from system specification to implementation with some work on debugging [7]), is described in considerable detail (see [2]), supports the development of agents that use goals and plans to deliver flexible behaviour, and has tool support [8].

For the purposes of this paper we do not attempt to describe the entire methodology, instead we focus on the inputs to the interaction design part of the methodology.

Prometheus' system specification phase (see Figure 2) defines system goals using a goal overview diagram that shows all of the goals and the goal-subgoal relationships. The system specification phase also uses scenarios. Each scenario is an example

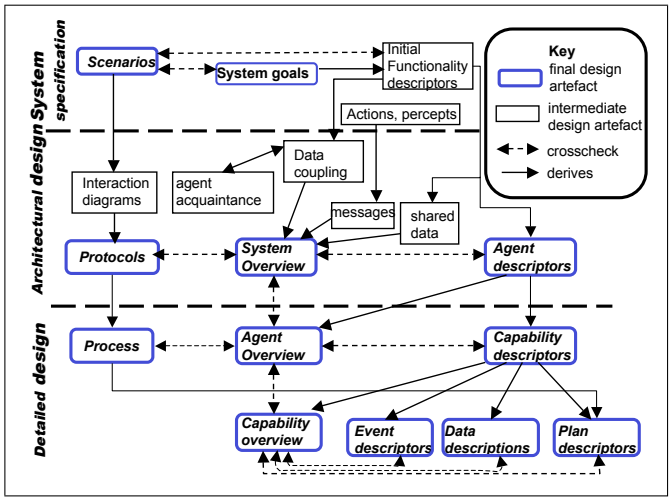

Fig. 2. Prometheus Overview Diagram

sequence of events illustrating desired system functionality. Scenarios are captured using a structured form (see Figure 5) which includes a sequence of steps. Each step is either a Goal, a Percept (incoming information from the environment), or an Action[2].

Goals are used as the basis for identifying functionalities: specific chunks of behaviour formed by grouping related goals. Based on considerations including data coupling, these functionalities are grouped to form agent types.

The interaction design part of Prometheus is fairly conventional. It begins by taking individual scenarios and creating corresponding interaction diagrams where instead of showing steps associated with functionalities, messages between agents are shown. These interaction diagrams are then generalised into interaction protocols (using AUML) which capture all possible sequences of messages.

3 Integrating Hermes and Prometheus

3.1 Overview

Since Hermes is specifically for interaction design, the obvious approach to integrating the two methodologies is to replace the typical Prometheus interaction design process with Hermes. This replacement leads to alterations to inputs needed for the interaction design process. These alterations are shown in Figure 3, which is an overview of the integrated methodology (the changed artefacts are shaded, and additional lines and arrows are shown in bold).

The major difference between Prometheus and Hermes interaction design is that the latter is *goal-oriented*. Therefore, a crucial alteration is to ensure that the design of the interaction goal hierarchy (the first Hermes artefact) is derived from both scenarios and system goals.

[2] Other step types are sub-scenario and “other”.

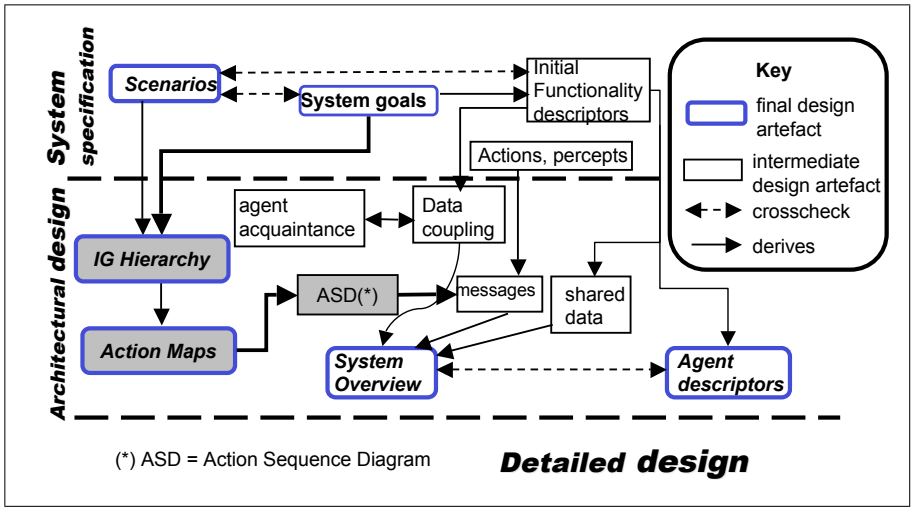

Fig. 3. Integrated Overview Diagram

Additionally, Hermes requires information about roles and data stores involved in the interactions[3]. Obviously, roles and data stores are integral parts of interactions as the former are necessary to determine inter-agent communication paths and the latter to determine the transmission of required data from agent to agent. From the Prometheus process, this information can be obtained from agent types, which summarises each type of agent in the system, including the functionalities and data stores they each possess. These details are used in the design of the action maps.

The end of the Hermes process results in a number of artefacts which are to be re-integrated with Prometheus. Note that the action maps are closer to process diagrams, i.e. it is easier to go from action maps to process diagrams than it is to go from AUML interaction protocols to process diagrams. This is due to the fact that process diagrams contain only internal agent processes and action maps contain both inter-agent communications and internal agent processes, whereas AUML interaction protocols only contain inter-agent communications.

3.2 The Amalgamated Design Process

Figure 4 summarises the steps involved in interaction design in the integrated methodology. The first five steps are taken directly from the usual Prometheus process. Before step 6 an assessment is made of whether it is worthwhile to use Hermes, and if not, then steps 6-11 are replaced with the existing Prometheus methodology. This decision is made based on the complexity of the interaction and likelihood of failures. For example, if it is a simple query and response scenario between two agents, it would not be worthwhile to use Hermes to design the interaction.

[3] Figure 3 omits links from the shared data and agent descriptors to the action maps and IG hierarchy respectively.

1. System Description
2. Develop System Goals
3. Identify Functionalities by grouping goals
4. Develop Scenarios
5. Determine agent types by grouping functionalities
6. Identify Hermes Roles and Interaction Goals
7. Develop Interaction Goal Hierarchy (section 3.3)
8. Develop Action Maps (section 3.4)
9. Develop Action Sequence Diagrams
10. Identify Messages
11. Develop System Overview Diagram (Prometheus)
12. Proceed with rest of Prometheus process

Before step 6 an assessment is made of whether it is worthwhile to use Hermes. If not, then replace steps 6-11 with the existing Prometheus methodology.

Fig. 4. Steps in Interaction Design

The identification of roles is usually straightforward as they are usually Prometheus agent types or roles that a particular Prometheus agent type assumes in an interaction. For example, in an e-commerce system, there may be two agent types, *Merchant* and *Customer*. In this case, it is quite obvious that Merchant and Customer are the roles involved in the interactions as they are aptly named after their roles. There are some designs in which agent types are not named after roles and will need to assume roles in their dealings with other agent types. For example, in an academic conference system, there may exist an agent type, *Academic Agent*, which may undertake a number of different roles, such as *Reviewer*, *Chairperson* and *Author*. In such circumstances, it is best to use the role as opposed to the agent type in the interactions.

The interaction goals are usually determined by analysing (i.e. grouping, abstracting and decomposing) the goals in the given scenario. Designing interaction goal hierarchies and action maps is detailed in sections 3.3 and 3.4 respectively.

The remainder of the goal-oriented interaction design (steps 9 and 10) follows the typical Hermes procedure. The action sequence diagrams are derived from the action maps, the action messages are derived from the action sequence diagrams and messages are described in (Prometheus) message descriptors. As these are unchanged from the original Hermes methodology, they are not discussed in this paper.

The messages obtained from the Hermes process are re-integrated into Prometheus. Hermes interactions are shown on the system overview diagram in the same way that protocols were previously represented. This simply represents that there is a goal-oriented interaction between those agents and the Hermes design artefacts can be referred to for more details. The messages produced as part of the Hermes design process are formally defined in message descriptors. Also, the IGs from the interaction goal hierarchy and the actions from the action maps are added to the agent overview diagrams[4].

[4] Both IGs and Hermes actions are mapped to plans: The former to *coordination plans* and the latter to *achievement plans* [6].

Name: Order Book
Description: An order is received from the WWW page interface (goal Place Order). Information is obtained in order to place the order and the order is placed.
Trigger: Goal: Place Order
Steps:

Step	Type	Name	Role	Data
1	Goal	Obtain delivery options	Delivery handling	...
2	Goal	Calculate delivery time estimates	Delivery handling	...
3	Goal	Present information	Online interaction	...
4	Percept	User input	Online interaction	...
5	Goal	Obtain credit card details	Purchasing	...
6	Percept	User input	Online interaction	...
7	Action	Bank Transaction	Purchasing	...
8	Percept	Bank transaction response	Purchasing	...
9	Goal	Arrange Delivery	Delivery handling	...
10	Action	Place delivery request	Delivery handling	uses: customer order record
11	Goal	Log outgoing delivery	Delivery handling	produces: Customer Orders
12	Goal	Log books outgoing	Stock management	uses: Customer order record produces: Stock DB
13	Goal	Update customer record	Profile monitor	produces: Customer DB
14	Action	Send email	Customer contact	uses: Customer DB

Variation: Book is not currently available. Include information with delivery options. Replace steps 7–12 with steps to add the order to an orders pending file.

Fig. 5. Order Book Scenario (From [2, p. 164])

The subsequent sections explain how the interaction goal hierarchy and its action maps are developed. In these sections we will use an example from the book store described in [2]. In particular, we will develop an interaction around ordering a book, based on the *Order book* scenario [2, p. 146] which is reproduced (in abridged form) in Figure 5. In addition to the scenario, we also need to know which agent types have been defined, and for each agent type what functionalities and data it contains. The book store example defines the following agent types:

- *Sales Assistant*: comprising the functionalities of *Book finding*, *Welcoming*, *Purchasing*, and *Online interaction*.
- *Customer Relations*: comprising the functionalities *Profile monitor* and *Customer contact* and the *Customer DB* database.
- *Delivery Manager*: comprising the functionalities *Delivery handling* and *Lost goods management* and the databases *Customer orders* and *Delivery problems*. This agent type also has access to external databases of couriers and postal areas.
- *Stock Manager*: comprising the functionalities of *Stock management*, *Competition management*, *Price setting*, and *Catalogue management*; and the databases *Pending orders*, *Books DB*, *Stock orders* and *Stock DB*.

3.3 Interaction Goal Hierarchy

To create the interaction goal hierarchy (IGH) we must first determine the overall intent of the scenario. Given the scenario in Figure 5, it is obvious that the overall intent is *order book* (as the scenario is aptly named). Thus, the top-most interaction goal (IG) is named *Order Book* and it is appropriately placed at the apex of the IGH (refer to Figure 6).

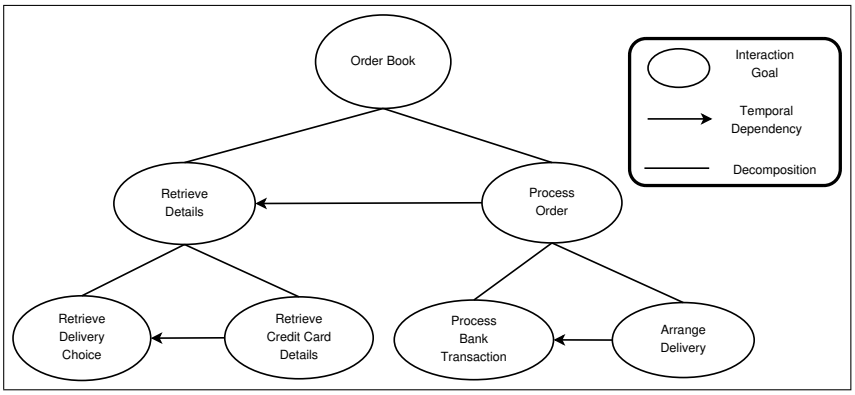

Fig. 6. Interaction Goal Hierarchy

We now further analyse the scenario to elicit other interaction goals. This can be done by abstracting or grouping related steps from the scenario, by decomposing scenario steps, or by mapping system goals.

System goals can be mapped to either interaction goals or Hermes actions; which artefact it maps to depends on the specific system goal and its properties. Typically, high-level system goals (abstract or easily decomposable into sub-goals) or those that involve more than one agent (e.g. Purchase, a goal that requires *Customer* and *Merchant* agents to achieve it) map to interaction goals. Low-level system goals (concrete or not easily decomposable into sub-goals) or system goals that involve only a single agent (e.g. *SendConfirmationEmail*) are usually mapped to Hermes actions.

In deriving IGs from scenarios it is more common to abstract or group existing steps than to decompose scenario steps. For example, from the scenario it appears that steps 1–4 are related: they gather, present and obtain information about delivery details. Thus, an IG, *Retrieve Delivery Choice* is created. Similarly, steps 5 and 6 are related; they gather, present and obtain information about credit card details. Thus, a *Retrieve Credit Card Details* IG, which represents scenario steps 5 and 6, is created.

Since the two IGs are very similar they can be further abstracted to a single IG, *Retrieve Details*. Thus, the *Retrieve Details* IG is composed of two sub-IGs, *Retrieve Delivery Choice* and *Retrieve Credit Card Details*. The scenario suggests that *Retrieve Delivery Choice* (i.e. steps 1–4) is completed before *Retrieve Credit Card Details* (i.e. steps 5 and 6). As this is a sensible suggestion (payment details are usually retrieved at the end, just before the delivery of the product), we have elected to keep this sequence. As such, we place a temporal dependency between the *Retrieve Delivery Choice* and

Retrieve Credit Card Details IGs. The *Retrieve Details* IG and its sub-IGs are added as a sub-IG to *Order Book* (refer to Figure 6) in the IGH.

Scenario steps 7 and 8 are abstracted into an IG, *Process Bank Transaction*, as they are related to performing transactions with the bank. Steps 9–14 are grouped into an IG, *Arrange Delivery*, as they deal with organising delivery of the book to the purchaser. Both *Process Bank Transaction* and *Arrange Delivery* are then grouped into an IG, *Process Order*. Again, the scenario steps suggest that *Process Bank Transaction* occurs before *Arrange Delivery* and, once more, we have chosen to retain this sequence. Thus, a temporal dependency is placed between *Process Bank Transaction* and *Arrange Delivery*. *Process Order* is then added to the IGH under the *Order Book* IG. As *Retrieve Details* should be achieved before *Process Order* is attempted (this is also implied in the scenario), we also place a temporal dependency between the two.

The resulting IGH (refer to Figure 6) is, in terms of temporal dependencies, a strongly constrained design. In order to successfully complete the *Order Book* interaction, the following atomic goals must be achieved in sequence: (1) *Retrieve Delivery Choice*, (2) *Retrieve Credit Card Details*, (3) *Process Bank Transaction*, and (4) *Arrange Delivery*.

In fact, since Prometheus scenarios are defined as a (strongly ordered) sequence of steps, any action map that takes this into account will be strongly constrained. Therefore, part of the process of moving from scenarios to interaction design, be it using the Hermes process or the existing Prometheus process, is to consider which of the temporal constraints implied by the scenario should be retained, and which temporal constraints should be weakened.

When developing the IGH it is important to consider *all* scenarios, as well as the variations of these scenarios. In some cases in order to accommodate all scenarios and scenario variations the IGH may need to be refined.

3.4 Action Maps

Action maps are used to determine how an *atomic* IG (i.e. leaf-node goal from the IGH) can be achieved. Thus, there is usually one action map per atomic IG. In a full interaction design for our case study four action maps would be created, however, due to space limitations, we only present the development of the action map for the *Arrange Delivery* IG.

The process in which action maps are designed is an iterative one. For ease of explanation, we describe the action map design as a four-step process, however, we do not intend that these four steps be followed rigidly. For example, the designer is free to skip steps if they are not deemed relevant, or to use as many iterations over these steps as are deemed necessary. The four steps, which are discussed below, are:

1. Develop initial action maps by transcribing scenario steps and assigning them to the appropriate agent
2. Add data to the action map and consider data flow issues
3. Extend action map to cover scenario variations
4. Extend action map to deal with failures

Step 1: Initial Action Maps. In this first step, the relevant steps of the scenario are transcribed onto action maps. Scenario steps that are goals correspond to goals of a single agent, and are mapped to actions in action maps. Scenario steps that are actions map (obviously) to actions. For example, step 10 of the scenario is the action *Place delivery request* which is mapped to the Hermes action of the same name. Scenario steps that are percepts also (less obviously) map to actions. In Prometheus, percepts are incoming information from the environment. When translated to Hermes, percepts are perceived as actions that somehow gain the required information in the interaction. Typically, this means that percepts are mapped to actions that either wait for incoming information (e.g. wait for an agent to send it information or for a belief to change) or retrieve the information themselves (e.g. read a file or access a database).

Designing the initial action map for the *Arrange Delivery* IG (scenario steps 9–14) involves placing five steps (10–14, 9 is omitted as it is achieved by steps 10–14) into the action map. It is best to follow the scenario steps as closely as possible. However, it is likely that some slight deviations from the scenarios will be necessary. Deviations include changing the ordering of some of the actions or creating new actions to clarify certain parts of the action maps.

The action maps have the roles placed as headings of the swim lanes, so the designer needs to ensure that actions are placed in the correct swim lane. With our case study, the roles involved in the interaction are simply the agent type identified in the earlier (Prometheus) steps of design. Determining the correct swim lane is a matter of assigning steps to the agent type that has been formed out of the corresponding functionality. For example, step 10 in the scenario, *Place delivery request*, is associated with the *Delivery handling* functionality which is part of the *Delivery Manager* agent type. Thus, this action is placed in the *Delivery Manager* swim lane.

It is also important to select the correct *action type* for each action. In some cases, the action types will need to be revised during other iterations of the design. An explanation of the different action types (*independent*, *caused* and *final caused*) follows.

An *independent action* is an action which can start without being triggered by another action, that is, it is not necessarily triggered by another action, but *may* be triggered by another action. Typically, independent actions are used as entry points into action maps. An independent action is denoted as a rectangle with dashed borders. For example, since *Place Delivery Request* (refer to Figure 7) is the first action to be executed, it is defined as an *independent action*.

A *caused action* is denoted by a rectangle and can only be triggered by another action. For example, *Log Outgoing Delivery* in Figure 7 only occurs after *Place Delivery Requested* is executed.

A *final caused action*, denoted by a rectangle with bold borders, is a caused action which terminates the IG for a particular role. For example, *Send Email* is the final action for the Customer Relations in Figure 7.

Once the actions have been placed, causality arrows are added between the actions to identify the flow of actions (based on, but not restricted to, the scenario sequence). After adding the causality arrows, the designer is free to make any alterations as sometimes the flow process will need to be different to the scenario.

The result of this first step is the action map in Figure 7.

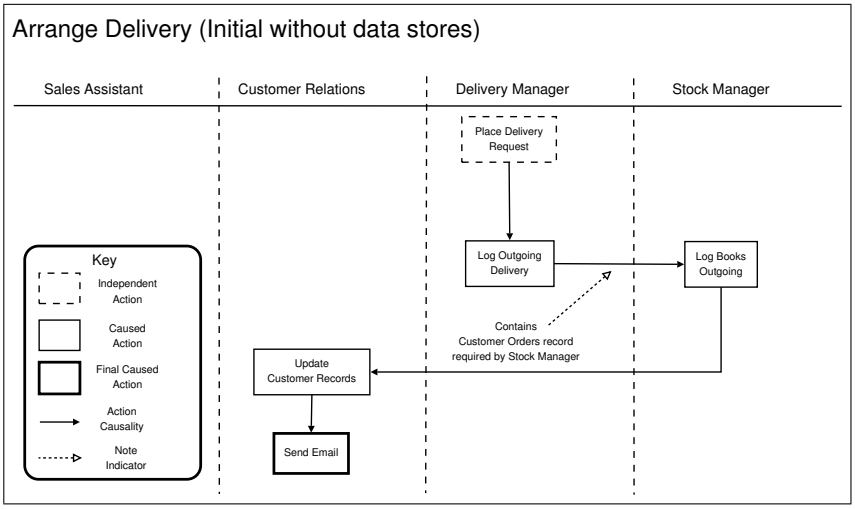

Fig. 7. Initial Action Map without Data Stores: Arrange Delivery

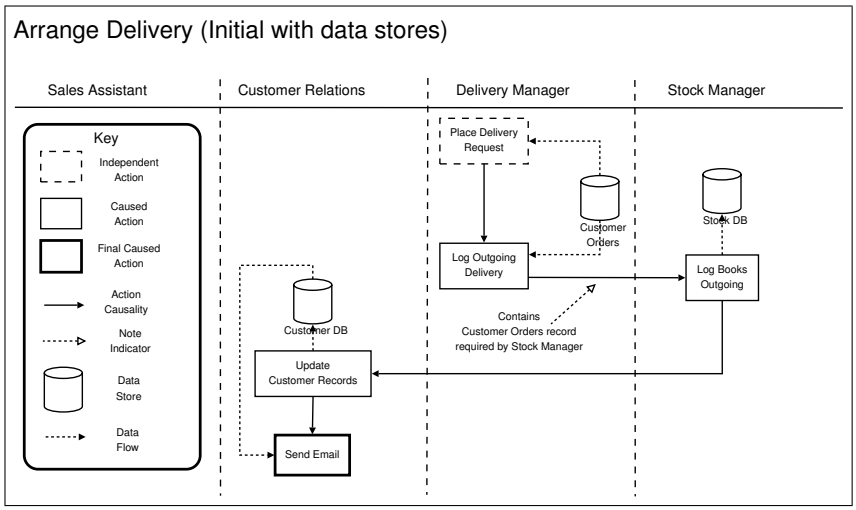

Fig. 8. Initial Action Map with Data Stores: Arrange Delivery

Step 2: Data Flow. This step focuses on incorporating data stores into the action maps and ensuring that all the data each agent requires are accessible. Firstly, we add the data stores onto the action maps. Data stores usually belong to specific agents (as defined in the agent groupings summarised earlier) and should be placed in the correct swim lanes in the action maps. It is not necessary to display every data store an agent contains, only the data stores that are used in the action map should be displayed. For example, in scenario step 10, *Place Delivery Request*, the *Delivery Handling* role requires data from *Customer Orders*. As the *Place Delivery Request* step is represented by the *Place*

Delivery Request action on the *Arrange Delivery* action map (Figure 8), the *Customer Orders* data store is added to the *Delivery Manager* agent's swim lane (since the data store belongs to that role).

Once the data stores have been added, it is necessary to ensure correct data flow between actions, which are depicted by dotted directed lines. In the case of the *Place Delivery Request* action, as it reads from the *Customer Orders* data store, the data flows from the data stores to it. If the action was writing to a data store, the data flow would be from the action to the data store, as with the *Log Books Outgoing* action and the *Stock DB* data store on on Figure 8.

It is not sufficient to simply add data stores and ensure that actions which read and write have *direct* access to the data stores. The designer must ensure that actions will have access to required data even if the data belongs to a data store in a different agent. This may mean that required data are read from a data store and passed through a number of different actions to reach a particular action that requires the data. For example, in scenario step 12, *Log Books Outgoing*, the *Stock Manager* (Prometheus) role requires data from the *Customer Orders* data store (which belongs to the *Delivery Manager* role). Thus, in the action map, the designer must ensure that the *Log Books Outgoing* action has access to the data store. In Figure 8, this is done and noted along the causality arrow that flows between the actions *Log Outgoing Delivery* and *Log Books Outgoing*.

The result of this second step is the action map in Figure 8.

Step 3: Incorporating Scenario Variations. Adding scenario variations provides alternative paths to successfully complete the interaction goal. As a result, this improves the flexibility and robustness of the interaction. There are no set guidelines for adding scenario variations into action maps as the variations will vary greatly depending on the domain, the agents involved and the actual interaction. In some cases, extending the interaction to cover scenario variations will require changes to the interaction goal hierarchy (IGH) if the variation affects more than a single IG. However, in the case of the *Order Book* scenario the variation only affects a single IG, and can be incorporated into the *Arrange Delivery* IG, and hence only the action map needs to be changed, and not the IGH.

In the case of the *Order Book* scenario, the variation states that if the book ordered is not available, replace steps 7–12 with steps to add a pending order. This can be incorporated into the action map by having two ways in which the *Arrange Delivery* IG can be achieved: (1) when the ordered book is available, the delivery order is placed and processed (as depicted in Figure 8), and (2) when the ordered book is unavailable, a pending order is created. Once the book is available, the pending order is filled and the delivery is processed.

Note that in our interaction, the availability of the ordered book is not explicitly queried; it is assumed to be part of the delivery options. In order to improve clarity we decide to make querying for book availability explicit. We do this by adding two new actions at the start of the action map: *Check Book Availability* and *Check Stock* (refer to Figure 9). These two actions are used to determine how to arrange the delivery. *Check Book Availability* is used to query *Stock Manager* about the availability of the ordered book. *Check Stock* is the action in *Stock Manager* that replies to the query. If

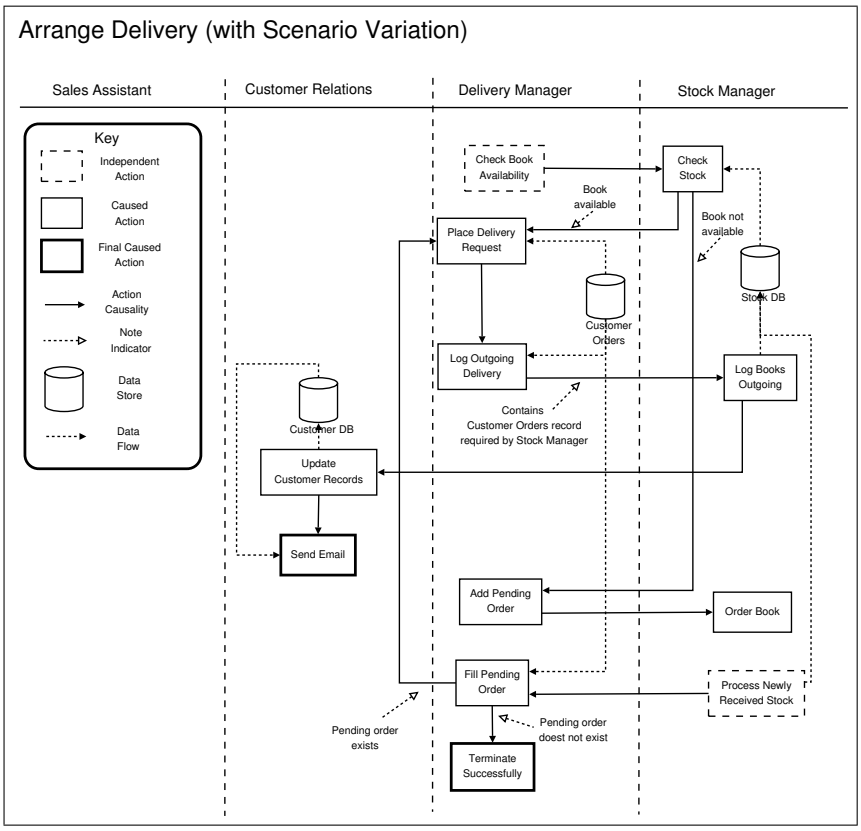

Fig. 9. Action Map: Arrange Delivery (with scenario variation)

the ordered book is available, the delivery order is placed and processed. If the ordered book is not available, the *Add Pending Order* action is used to order the book (from the publishing firm). Once the book comes in (from the publishing firm), *Process Newly Received Stock* is triggered, the pending order is filled and the delivery is processed.

The result of this third step is the action map in Figure 9.

Step 4: Adding Failure Tolerance. This step of the action maps focuses on failures in the interaction and how to handle them and is of crucial importance as failure handling is what gives the action maps, and thus Hermes, the majority of their flexibility

Table 1. Possible Failures and Remedial Actions for *Arrange Delivery*

Action	Possible Failures	Remedial Actions
Order Book	Book out of print	Suggest alternative title or edition
Place Delivery Request	Invalid address	Get details from user and validate
Send Email	Email bounces	Use different medium to contact user (e.g. send mail via post)

and robustness. This iteration has three sub-steps: *Failure Identification*, *Adding Action Retries* and *Adding Rollbacks*.

In the first sub-step, *Failure Identification*, action maps are analysed to determine where possible failures can occur. In general, think about each action on the action maps and determine whether it can fail or not. If it can fail then determine what different types

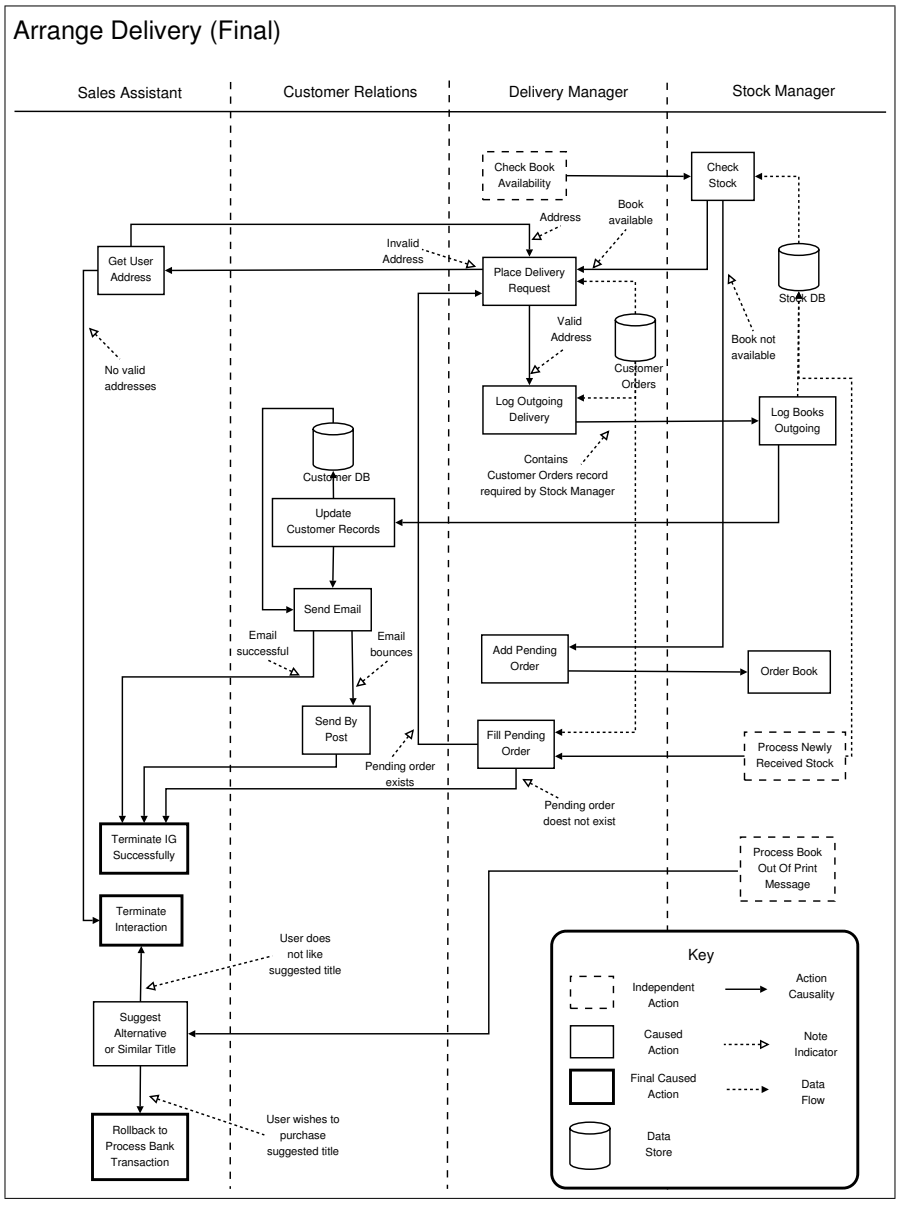

Fig. 10. Final Action Map: Arrange Delivery

of failures can result from each of the actions. Once the failures have been identified, determine ways in which they can be addressed.

For each action map we summarise in a table the possible failures and what remedial actions can be taken to recover from the failure. For example (see Table 1), the *Order Book* action can fail if the book is out of print, and in this case one, way of dealing with the failure is to suggest alternative titles or editions to the shopper.

To further enhance flexibility and robustness, the designer can also analyse each action and determine different ways in which they can succeed (i.e. alternative success paths). However, this was not carried out in this case study.

The remaining two sub-steps deal with handling the identified failures by incorporating the remedial actions into the design using respectively action retries and rollbacks.

In the *Adding Action Retries* sub-step, the action maps are updated with *action retries* to incorporate remedial actions. For example, it has been identified that the *Order Book* action could fail if a book is out of print, and the suggested remedial action for this is to suggest an alternative title. In Figure 10, this is achieved by the *Process Book Out Of Print Message* action which leads to the *Suggest Alternative or Similar Title* action.

Adding *action retries* can create loops between actions, and it is important to ensure that there are no unintended endless loops. For example, in Figure 10, there is a loop between *Place Delivery Request* and *Get User Address*. The intention is that the user is re-prompted for an address every time invalid addresses are encountered. However, an additional action, *Terminate Interaction*, has been provided in the case the user cannot or does not wish to enter an address and would like to exit the interaction at that point.

In the *Adding Rollbacks* sub-step the action map is extended with rollbacks. In this case study none of the identified remedial actions involved rolling back to a previous stage, so no changes are made in this sub-step.

The result of this fourth step is the action map in Figure 10. Although the action map appears fairly complex, it provides flexible and robust interaction, including dealing with the failure cases summarised in Table 1. The action map in Figure 10 and the IGH in Figure 6 were developed by following the sequence of steps that comprise the amalgamated methodology.

4 Conclusion

There are a number of other approaches to agent interactions which, like Hermes, focus on providing more flexible and robust interactions. These approaches include those based on social commitments [9,10,11,12], Kumar *et al.*'s Landmark-based approach [13,14], and Hutchison and Winikoff's goal-plan approach [15].

The work of Yolum and Singh [9,10] defines a social commitment as an agreement between two agents in which one agent is responsible for bringing about a certain condition. Flores and Kremer [11,12] have a slightly different definition of social commitments in that an agent is responsible for performing certain actions as opposed to bringing about certain conditions. To progress through commitment-based interactions, agents acquire, manipulate and discharge commitments. Both these approaches support complex agent interactions, however, the design aspects are not well defined. Given a

particular interaction, it is difficult to determine what commitments are required for the design of the interaction.

The work of Kumar *et al.* is focused on landmarks, which can be thought of as states of affairs. Their work argue that the states of affairs are more important than the actions that bring about the states of affairs. That is, the states that are brought about by communicative acts are more important that the communicative acts themselves. In this regard, landmark-based interactions are about navigating through landmarks to reach a desired final state. This work relies heavily on expertise in modal and temporal logics.

In Hutchison and Winikoff's work [15], protocols are defined in terms of goals and plans, and can be seen as predecessor to the Hermes methodology; its design process is not detailed and it does not provide a mapping from design to implementation.

None of the aforementioned studies, which aim to provide increased flexibility and robustness in agent interactions, have been integrated with full agent system design methodologies. For such work to be practical, integration with full agent system design methodologies is important.

The authors are not aware of any such integration. The closest work to integrating more flexible and robust agent interaction design with full agent system design methodologies is in methodologies in which agent interactions are treated as first class entities. On such example is SODA [16]. However, although SODA deals with inter-agent design and regards interactions as first class entities, it has a different aim to the enhanced methodology we have presented.

There are two main differences between SODA and the integrated Hermes and Prometheus methodology. Firstly, SODA is for the analysis and design of Internet-based systems whilst the integrated Hermes and Prometheus methodology is for general purpose agent system design (i.e. not specifically for Internet-based systems). Secondly, and most importantly, the SODA interaction design appears to be message-centric as the design process is focused on resources and on information passing.

We have shown how the Prometheus methodology can be modified by replacing its current interaction design process with the Hermes methodology. We have also refined the Hermes methodology and presented a more detailed process for developing action maps than has previously been published. The enhanced methodology provides a guided iterative process for the design of agent interactions. Since the design artefacts produced are pure Hermes artefacts, they can be mapped to an implementation on goal-based agent platforms as explained in [6].

One of the main advantages of the amalgamation is the flexibility and robustness of the interactions, which is a direct result of merging Hermes with Prometheus. Furthermore, since Hermes is goal-oriented, it is more congruent with the agent paradigm. Additionally, the Hermes design process explicitly considers possible failures and how they can be recovered from, which tends to lead to more robust interactions. Preliminary results from a brief comparison of AUML and Hermes have shown that AUML, which focuses on alternative message sequences, cannot easily identify some types of failures which are easily identifiable by the Hermes process.

One disadvantage of Hermes is that it is time-consuming and, depending on the design and the particular interaction, is likely to result in a greater number of design artefacts than the AUML approach. However, the result provides for greater flexibility

and robustness, and is more compatible with intelligent software agents that are proactive and autonomous. It should be noted that we do not propose that all interactions be designed using the Hermes process: as discussed in section 3.2, the decision whether to use Hermes or Prometheus to design a given interaction should be made explicitly based on the complexity of the interaction, the need for flexibility, and the extent to which failure may arise.

A key piece of future work that we have begun is an evaluation of the ease-of-use and effectiveness of the Hermes methodology. This comparison involves having a number of designers design an interaction using either the original Prometheus process (using the AUML notation) or the process presented in this paper. We will then compare the designs produced in terms of the range of interaction sequences supported (flexibility) and the types of failure that can be recovered from (robustness).

Another area of future work is tool support for the amalgamated methodology. Tool support is important for the amalgamated methodology to be practical, and as such we intend to develop such tool support. We currently have developed a prototype tool using UMLet[5] for the aforementioned evaluation. Future tool support development may involve building upon the existing Prometheus Design Tool (PDT) [8].

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. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons (2004) ISBN 0-470-86120-7.
3. Jennings, N., Kinny, D., Wooldridge, M., Zambonelli, F.: The Gaia methodology. In Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems. Kluwer Academic Publishing (New York) (2004)
4. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An agent-oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems **8** (2004) 203–236
5. 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)
6. Cheong, C., Winikoff, M.: Hermes: Implementing goal-oriented agent interactions. In: Proceedings of the Third international Workshop on Programming Multi-Agent Systems (ProMAS). (2005)
7. Padgham, L., Winikoff, M., Poutakidis, D.: Adding debugging support to the Prometheus methodology. Engineering Applications of Artificial Intelligence **18** (2005) 173–190 Special issue on Agent-oriented Software Development.

[5] http://qse.ifs.tuwien.ac.at/ auer/umlet/

8. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the Prometheus methodology. In: Proceedings of the First International Workshop on Integration of Software Engineering and Agent Technology (ISEAT). (2005)
9. 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
10. 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
11. Flores, R.A., Kremer, R.C.: A pragmatic approach to build conversation protocols using social commitments. In: Autonomous Agents and Multi-Agent Systems (AAMAS). (2004) 1242–1243
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. 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)
16. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In: Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000). (2000) 185–193