Designing Commitment-Based Agent Interactions

Michael Winikoff[*]
RMIT University,
Melbourne, Australia

Abstract

A key property of agents is that they are social, and hence the design of agent interactions is a crucial part of an agent-oriented software engineering methodology. Agent interactions are usually designed by focusing on the messages, and using interaction protocols to define permissible message sequences. This approach goes against the idea of agents being autonomous, flexible and robust by limiting agents' flexibility in interactions. In this paper we focus on the commitment machine framework of Yolum and Singh, and we provide a simple and usable process, including techniques and tips, for designing commitment-based agent interactions.

1. Introduction

A key property of agents is that they are *social*, i.e. that they interact with other entities. As a result, the design of agent interactions is a crucial part of any agent-oriented software engineering methodology.

However, although concepts and architectures for designing *individual* agents that are proactive, reactive, flexible and robust have been well developed, the design and implementation of agent interactions is usually done by focusing on possible sequences of messages. Agent-oriented software engineering methodologies such as Prometheus [11], Gaia [9], MaSE [7], and Tropos [2], all design agent interactions in terms of permissible sequences of messages, using some form of interaction protocol, such as Agent UML (http://www.auml.org). Similarly, if we look to agent-oriented programming languages such as JACK, Jadex, 3APL or Jason [1], we find agent interactions being programmed at the lowest common denominator: messages.

Such a message-focused approach is not ideal because it goes against the idea of agents being autonomous entities that are able to intelligently pursue their goals. Interaction protocols specify precisely the sequences of messages, and tend to leave the agents limited scope for flexibility in how they achieve their interaction goals.

A number of alternative approaches have emerged recently including the use of commitments as a design/implementation concept, as well as the use of "interaction goals".

In Yolum and Singh's *commitment machines* [14] interaction is based on social commitments between agents, which represent an agent's responsibility to bring about a certain condition for another agent. Flores and Kremer's *social commitment* approach [8] has a different notion of commitments: they are defined in terms of performing actions, rather than bringing about conditions. However, in both approaches the interaction is driven by the acquisition, manipulation and discharge of commitments.

Kumar *et al.* [10] use the concept of a *landmark* which represents a particular state of affairs. In this work, the agents navigate through the landmarks to reach a desired final landmark, communicating as needed along the way. This work is theoretical in nature and requires expertise in temporal and modal logics, making it impractical for software engineers who do not usually have expertise in such logics.

Cheong and Winikoff [4, 6] propose a pragmatic methodology (named *Hermes*) for the design of agent interactions. Agent interactions are designed in terms of *interaction goals*. A goal hierarchy is developed, and then for each leaf goal an action map (roughly speaking an activity diagram) is developed. A key feature of the approach is the use of "rollbacks" to recover from failure: if a given interaction goal fails then the designer may indicate that achieving a previous interaction goal may allow the interaction to continue. For example, if booking a hotel for given dates cannot be done, then going back and finding alternative travel dates may resolve the issue.

In all of these approaches message sequences are not defined explicitly, but instead *emerge* as the interaction is driven by the agents' need to fulfil their commitments, reach their landmarks, or achieve their interaction goals.

In this paper we focus on the commitment machine ap-

[*] This work is being supported by the Australian Research Council under grant LP0453486, in collaboration with Agent Oriented Software.

proach proposed by Yolum and Singh [14]. Although this approach is elegant, there is no guidance as to how to design an interaction (apart from some consistency rules provided by [13]). Given a particular interaction that is desired, it is not clear what commitments are required to realise the interaction.

This paper provides a methodology for designing commitment-based interactions.

We briefly review commitment machines in section 2, then describe our methodology (section 3) and conclude in section 4.

2. Background

A *Commitment Machine* (CM) [14] defines interaction in terms of an interaction space: a (global) state which is changed by actions. Informally, we can consider the interaction state to consist of fluents as well as (social) commitments.

A (base-level) social commitment is an undertaking by one entity, the debtor A, to another entity, the creditor B, to bring about a condition p. This undertaking by A to B to bring about p is written $\mathsf{C}(A, B, p)$ and, where the identity of the entities is obvious or unimportant, is abbreviated to $\mathsf{C}(p)$. For example, an agent might commit to paying, $\mathsf{C}(\text{pay})$.

There are also *conditional* commitments which specify that the debtor A undertakes to the creditor B to bring about p, conditional on q becoming true. The commitment is conditional: until q becomes true the debtor A is not required to do anything. However, when q becomes true the debtor becomes committed to bringing about p. A conditional commitment is written $\mathsf{CC}(A, B, q, p)$ and, if the identity of A and B is unimportant or obvious, is abbreviated to $\mathsf{CC}(q \rightsquigarrow p)$ where the arrow emphasises the causal nature of the conditional commitment. For example, an agent might commit to booking a room, once payment is received, $\mathsf{CC}(\text{pay} \rightsquigarrow \text{bookroom})$.

The commitment machine framework gives semantics to commitments by defining their dynamics, that is, how commitments are affected by changes to the state. The dynamics defined by Yolum and Singh specify that a base level commitment $\mathsf{C}(p)$ is discharged when p becomes true, and that a conditional commitment $\mathsf{CC}(q \rightsquigarrow p)$ is discharged when p becomes true, or when q becomes true, in which case the commitment $\mathsf{C}(p)$ is created[1]. In a subsequent paper Winikoff *et al.* [12] argued that the semantics defined had a number of anomalies, and proposed a refined semantics that was symmetrical, and that considered implied commitments[2]. In the original rules of Yolum and Singh creating $\mathsf{C}(p)$ and then making p true resulted in the state $\{p\}$ whereas making p true and then creating $\mathsf{C}(p)$ resulted in the state $\{p, \mathsf{C}(p)\}$. This asymmetry was fixed by slightly more complex rules that consider what already holds when a commitment is created. For example, when an action is meant to create the commitment $\mathsf{C}(p)$ but p already holds, then the commitment is not created. Similarly, if an action is meant to create the commitment $\mathsf{CC}(q \rightsquigarrow p)$ then depending on whether p and/or q hold either $\mathsf{CC}(q \rightsquigarrow p)$ is created (if neither p nor q hold), $\mathsf{C}(p)$ is created (if q holds but p doesn't), or no commitment is created (if p holds). In this paper we will use these refined semantics [12].

An interaction is defined by specifying:

1. The agents involved (actually the roles)
2. The possible formulae that can be contained in the state (both fluents and commitments)
3. The actions that agents can perform
4. The effects of each action on the state, and the preconditions to performing each action

An interaction implicitly defines an interaction space: a finite state machine where actions define transitions between states. A state is deemed to be final if it contains no base-level commitments: conditional commitments are, in a sense, "latent" and so do not prevent the interaction from terminating. However, an outstanding (base level) commitment must be fulfilled before the interaction can complete.

3. Designing Interactions

In this section we introduce a process for developing commitment-based interactions. We do not claim that this process is ideal or perfect. However, we believe that it is useful and that it is very clearly easier to design an interaction using this process, than without any process, as has been the case up to now.

The starting point of the process is a scenario in the style of Prometheus [11], namely a sequence of steps, each step consisting of a goal (or action or percept or "other") along with the role that performs the step and data used or produced by the step (in the example below we elide the data).

We then proceed to derive an interaction by:

1. Adding to each step its trigger ("when"), and the conditions that need to hold before the step can be performed ("why").
2. Adding commitments that "fill in the gaps" between required conditions and actual conditions.
3. Considering the sequencing of steps, and adding conditions and effects to ensure reasonable execution sequences.

1 They also defined additional rules that allowed commitments to be released by the creditor, or transferred between roles; but these three rules are the key ones which capture the dynamics of commitments.

2 We do not discuss these due to a lack of space, see [12] for details.

	Who	What
1	Organiser	Propose time
2	Participant 1	Confirm availability
3	Participant 2	Confirm availability
4	Organiser	Check venues
5	Venue Manager	Provide venues
6	Organiser	Book venue
7	Venue Manager	Book room
8	Organiser	Pay for room
9	Organiser	Confirm meeting
10	Organiser & Participants	Attend meeting

Figure 1. Scenario

4. Considering variations of the scenario, and generalising the interaction to handle variations.
5. Collecting the interaction and checking for "good" and "bad" states.

The aim of the first two stages is to identify the commitments that are needed. The aim of the next two stages is to generalise the interaction beyond the fixed sequence of steps described in the scenario. The last stage simply collects the interaction and checks that it doesn't allow for undesirable states to be reached.

Figure 1 shows an example scenario from a meeting scheduler application. There are four agents[3]: a meeting organiser (O) who has been requested by their user to organise a meeting, two participants (P_1 and P_2) who will be asked to attend the meeting, and a venue manager (VM) who manages room bookings. The interaction begins with interactions to determine the meeting date and time (steps 1-3), then the organiser and the venue manager interact to book a suitable room (steps 4-7), then the meeting is confirmed (step 9). Paying for the room (step 8) is shown as occurring before meeting confirmation, but could occur later (e.g. when the meeting is held). Finally, the meeting takes place (step 10). There are two documented variations of this scenario. Firstly, if the agents cannot agree on a meeting time (step 2 or 3 fails) then the remaining steps are skipped and the interaction fails. Secondly, if the room is not available (step 5 fails) then the remaining steps are skipped and the interaction fails.

3.1. Adding Conditions and Triggers

To each step we add two additional pieces of information: what does the agent require to hold before performing the step ("why", or more accurately "why is this ok to do?"), and what causes the step to execute ("when").

3 In Prometheus scenarios contain roles, not agents. We use agents here for simplicity.

The "why" condition can be a "public" condition, that is something that could be evaluated by another agent, such as whether certain commitments have been made. It could also be a "private" condition that only that agent can evaluate. For example, whether that agent is free at a given time. We denote private conditions by putting them in square brackets (e.g. "[available]"). To obtain the "why" condition for each step we ask "what needs to hold before the agent would be willing to do this step?" and can check the condition by asking "if this condition did not hold, would the agent still be willing to do the step?". For example, in step 6, the organiser will only perform the step of booking the venue if the participants have committed to attending the meeting at that time (otherwise the organiser may be committing to paying for a room which is not used). Similarly, in step 7, the venue manager would only be willing to book the room if payment has been received[4].

The trigger is determined by asking "what causes this step to happen?". It can be the performance of a previous step (which is indicated with the number of the previous step), or it can be left empty (denoted with a "-") in which case the step can occur whenever the condition is true. For the first step in the interaction the trigger will normally be the trigger of the scenario (in this case a request from the user to arrange a meeting).

The difference between the condition ("why") and the trigger ("when") is that the former is a logical condition that must hold before the agent will be willing to perform the step. The latter is an indication of what causes the step to be done (assuming the "why" condition holds).

Figure 2 shows this information added to the original scenario. For steps 2 and 3, the participants need to be available at the requested time, and the steps are triggered by the first step (so step 2 and step 3 could occur in parallel, since step 3 does not depend on step 2). Step 4 can only occur when the participants have confirmed their availability, and the step can be done when the condition is true.

Usually steps that provide information and do not change the agent's commitments do not have any "why" conditions, for example, step 5 simply provides information on venues, and the Venue Manager is happy to do so at any time.

In order to decide whether to make the "when" a preceding step or not, one considers whether it would make sense for the interaction to start with this step if its condition was true. If yes, then the "when" should be empty, if no, then the "when" entry should be a previous step. For example, it doesn't make sense for the interaction to start with step 2, even if the participant is available, and so step 2 is triggered by step 1.

4 In the next stage, introducing commitments (see section 3.2), we will modify this condition, since it is currently infeasible: payment is made later in the scenario.

	Who	What	Why	When
1	Organiser	Propose time	-	meeting requested
2	Participant 1	Confirm availability	[available]	1
3	Participant 2	Confirm availability	[available]	1
4	Organiser	Check venues	participants confirmed	-
5	Venue Manager	Provide venues	-	4
6	Organiser	Book venue	participants confirmed	5
7	Venue Manager	Book room	paid	6
8	Organiser	Pay for room	roombooked	-
10	Organiser & Participants	Attend meeting	need to attend	-

Figure 2. Scenario with triggers and conditions

3.2. Adding Commitments

What we are aiming for is a *feasible* scenario: one where the steps can occur in sequence. Since a step requires certain conditions to hold, for the scenario to be feasible each condition that is required must be made true by previous steps. In general this doesn't hold: for example, the condition of step 7 (in Figure 2) is that payment for the room has been made, but paying is done in step 8.

More generally, a condition can fail to hold either because there is no step that makes the condition true, or because there is a step that makes it true but that step occurs too late. In the first case we change the sequence so an earlier step makes the condition true. This can be done by adding an effect to an earlier step, or by creating an additional step. In the second case we *weaken* the condition to a commitment and ensure that the commitment is created before the step in question. We know that the condition will become true later, so the commitment will be discharged. A special case of this is where the condition is already a commitment, in which case it is weakened to a *conditional* commitment.

Whenever we change (or add) a step to make a commitment true we also ask ourselves under what conditions the agent doing that step would be willing to make the commitment. This condition in turn needs to be made true, thus this stage of adding commitments is iterative: it continues until the scenario is feasible.

For example, the condition of step 7 is not true, but is made true later. We therefore change the condition (paid) to a commitment (C(paid)) and change step 6 so it creates this commitment. However, in order for the Organiser to be willing to commit to paying, it requires that the room must have been booked. Since this condition is made true later, we weaken C(paid) to the conditional commitment CC(roombooked $\rightsquigarrow$ paid), and update the condition of step 7 to be this conditional commitment. Since the Organiser is happy to make this conditional commitment we are done with the effect of step 6 and the condition of step 7.

Similarly in steps 2 and 3 we begin by modifying the steps to create the commitment C(attend) but because the participants are only willing to commit to attending if the meeting is confirmed we weaken this to CC(confirmed $\rightsquigarrow$ attend). We then define the condition "participants confirmed" as being this conditional commitment having been made by both participants, so the combined effects of steps 2 and 3 meet the condition of steps 4 and 6.

Finally, we add the effect of booking the room to step 7, and the effect of confirming the meeting to step 9.

Figure 3 shows the resulting scenario with commitments added in the "effect" column. The reader may verify that Figure 3 defines a *feasible* sequence of steps. Note that the condition of step 10 is the result of making each participant's conditional commitment to attend (if the meeting is confirmed) into a commitment to attend, since the meeting is confirmed in step 9.

3.3. Considering Sequencing

We now need to examine the possible execution sequences that are permitted by the conditions of steps, and ensure that we have sensible possible sequences. Specifically, we want to ensure that on the one hand the possible sequences are sufficiently constrained that the example sequence in the scenario can occur, with each step being triggered by the previous one; but on the other hand, we want to avoid over-constraining the possible sequences. In particular, we want to allow parallelism between steps, choices of sequences, and the ability to start in the middle of the interaction, where these make sense.

Our first step, however, is to ensure that all steps have some effect. This is done because in the commitment machine framework an action that has no effect is difficult to reason about: since it has no effect there is no reason to execute it, nor any reason to refrain from executing it repeatedly! We therefore add an effect to any step that has no effect. In some cases the effect to be added is obvious, for example the effect of the action *Attend meeting* is to make *attend* true, and the effect of the action *Pay for room* is to make *paid* true. In other cases the effect of a request or re-

	Who	What	Why	When	Effect
1	Organiser	Propose time	-	meeting requested	
2	Participant 1	Confirm availability	[available]	1	CC(confirmed $\rightsquigarrow$ attend)
3	Participant 2	Confirm availability	[available]	1	CC(confirmed $\rightsquigarrow$ attend)
4	Organiser	Check venues	participants confirmed	-	
5	Venue Manager	Provide venues	-	4	
6	Organiser	Book venue	participants confirmed	5	CC(roombooked $\rightsquigarrow$ paid)
7	Venue Manager	Book room	CC(roombooked $\rightsquigarrow$ paid)	6	roombooked
8	Organiser	Pay for room	roombooked	-	
9	Organiser	Confirm meeting	roombooked and participants confirmed	-	confirmed
10	Organiser & Participants	Attend meeting	need to attend	-	

participants confirmed = $\mathsf{CC}(P_1, O, \text{confirmed}, \text{attend}) \wedge \mathsf{CC}(P_2, O, \text{confirmed}, \text{attend})$.
need to attend = $\mathsf{C}(P_1, O, \text{attend}) \wedge \mathsf{C}(P_2, O, \text{attend})$

Figure 3. Scenario with commitments

sponse is to modify the state with a fluent that notes that the request has been made, or (respectively) that a response has been received. For example, the effect of *Check venues* is to make the fluent *venuesrequested* true, and the effect of *Provide venues* is to make the fluent *venuesprovided* true. Similarly the effect of the first step is to make *timeproposed* true.

In order to check for sensible execution sequences we use a graph to visualise the possible sequences. In this graph each step number is a node, a step being triggered by another step is indicated with an edge, and a step that is triggered by a condition has a dashed edge from the step which makes that condition true. If the graph indicates an under-constrained set of steps, then additional constraints can be added to prevent steps from being executable at inappropriate points in the interaction. If the graph indicates an over-constrained set of steps, then constraints and dependencies need to be relaxed.

Figure 4 shows the execution sequences for the example. Looking at this figure we can see that the possible sequences of execution appear to be sensible: the sequences are sufficiently constrained that a sensible order (including the order in the scenario) can occur. The significant differences between the strict sequential ordering given in the scenario and the possible orderings allowed by the triggers and conditions are:

- That steps 2 and 3 can be done in parallel, which makes sense, because the two participants can check their availability in parallel.
- Step 8 can be done in parallel with steps 9 and 10. This also makes sense because payment doesn't depend on confirming the meeting, or the meeting occurring.
- The interaction can begin at the start (step 1), or in the middle (see below).

We now consider whether the sequence is too constrained, and ensure that various desirable flexibilities are supported. Due to space limitations we focus on one particular, key, type of flexibility that is a distinguishing feature of commitment machines: the ability to begin an interaction in the middle. In order to design a commitment machine that supports this sensibly we consider "sequence blocks" — a sequence of steps that, conceptually, achieve a single goal (cf. Interaction Goals [4]) — and then ensure (i) that a sequence block cannot be executed if the goal it achieves is already true, and (ii) that a sequence block can begin execution whenever conditions are right, i.e. that it isn't triggered directly by a previous step. We achieve (i) by adding the negation of the goal of the block to the condition of its first step. A useful heuristic is that the goal of a sequence block can often be identified by looking at the precondition of the subsequent block. We achieve (ii) by ensuring that the first step of a sequence block does not have an entry in its "When" column. In our example it would make sense for the interaction to begin at step 4 or at step 8: the aim of the first three steps is to agree on a meeting date and time (*participants confirmed*), if this has already been done, then one can skip to organising a room. Similarly, the aim of steps 4-7 is to book a suitable venue, and if this has been done, then one can skip to steps 8-10. We therefore identify steps 1-3 as being a sequence block, and achieve condition (i) by adding $\neg$ *participants confirmed* to the condition of step 1. Similarly, we add $\neg$ *roombooked* to the condition of step 4. In this case condition (ii) already holds: neither step 4 nor step 8 are directly triggered by their preceding step.

Finally, we need to ensure that the dependencies indicated by numbers in the "when" column, i.e. steps being triggered by previous steps, are captured solely by the con-

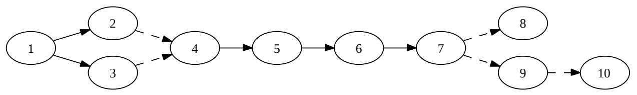

Figure 4. Execution Sequences in Example

ditions and effects of steps. This is because the commitment machine framework does not provide any way for a step to trigger another step, except through its changes to the state making the condition of the other step true. This is where it is important that all steps have some effect. Considering the steps in the scenario we add a number of additional conditions: to steps 2 and 3 (*Confirm availability*) we add the precondition *timeproposed*; to step 5 (*Provide venues*) we add the precondition *venuesrequested*; and to step 6 (*Book venue*) we add the precondition *venuesprovided*. Step 7 does not need any additional conditions, since its precondition is already made true by step 6.

3.4. Considering Variations

This stage continues the generalisation process by considering variations of the scenario, and ensuring that the interaction defined is able to handle these variations. If the interaction is not able to handle a given variation then it is revised so it can do so. A range of techniques are used to do this, depending on the nature of the variation. These techniques include weakening conditions to allow additional sequences, and adding alternative steps that provide additional possible sequences of interaction.

For example, one variation of the original scenario is where the participants and organiser cannot agree on a meeting date/time, in which case the meeting is abandoned. This variation is already supported by the interaction defined: step 4 will not proceed if agreement has not been reached, and since there are no base-level commitments, the interaction can end there.

Another variation of the original scenario is that a room may not be obtainable for the desired date/time. One possible response is to abandon the interaction, which is already supported by the interaction designed: the interaction cannot continue without a room booking, and can be stopped since there are no outstanding base-level commitments. Another possible response to the lack of a room is to find an alternative date. This response is not handled by the interaction as designed, but could be handled by introducing an action that unconfirms the date/time when there is no room, and re-triggers the first step.

3.5. Collecting and Checking the Interaction

We now collect the interaction: make note of the actions that can be performed, what each action's effects and precondition are, and what fluents and commitments can occur in the state.

Collecting the interaction designed we have the roles of Organiser, Participant 1, Participant 2 and Venue Manager.

The fluents are those conditions that occur in the "why" or "when" columns (except for private conditions such as [available]) or that were added in the third stage. In our example the fluents are: *timeproposed*, *confirmed*, *venuesrequested*, *venuesprovided*, *attend*, *paid*, and *roombooked*.

The commitments that are used are those which are created by steps, namely: $\mathsf{CC}(P_i, O, \text{confirmed}, \text{attend})$, and $\mathsf{CC}(O, VM, \text{roombooked}, \text{paid})$.

The actions are the steps, where the precondition of each action is its "why" column and its effect is the commitment or fluent listed in its "effect" column. When collecting the conditions and effects of the actions we need to recall the additional conditions and effects added in section 3.3.

- Propose Time, performed by the Organiser, has the effect of making *timeproposed* true, and has the precondition that participants have not been confirmed.
- Confirm Availability, performed by either Participant, requires that the participant is free at the desired time (a private condition) and that *timeproposed* is true. It creates the commitment $\mathsf{CC}(P_i, O, \text{confirmed}, \text{attend})$.
- Check Venues, performed by the Organiser, has the effect of making *venuesrequested* true, and has the precondition that the room has not been booked, and that the participants have been confirmed (participants confirmed $\wedge$ $\neg$roombooked).
- Provide Venues, performed by the Venue Manager, has the effect of making *venuesprovided* true, and has the precondition that venues have been requested (*venuesrequested*).
- Book Venue, performed by the Organiser, creates the commitment $\mathsf{CC}(\text{roombooked} \leadsto \text{paid})$, and has the precondition that the participants have been confirmed and that the venues have been provided.
- Book Room, performed by the Venue Manager, has the effect of making the fluent *roombooked* true, and has the precondition $\mathsf{CC}(\text{roombooked} \leadsto \text{paid})$.
- Pay for room, performed by the Organiser has the effect of making the fluent *paid* true, and has the precondition that *roombooked* must be true.
- Confirm Meeting, done by the Organiser, has the precondition that the room is booked and participants confirmed, and makes the fluent *confirmed* true.

- Attend meeting, done by the Organiser and Participants[5], has the effect of making *attend* true, and (because no one likes to attend meetings that they don't have to) has the precondition that there is a commitment to attend ($\mathsf{C}(P_1, O, \text{attend}) \wedge \mathsf{C}(P_2, O, \text{attend})$).

Finally, as discussed in [12], we should identify which states are considered to be "bad", i.e. states that should not occur, and analyse the interaction to ensure that these states cannot be reached. Similarly, we should also identify desirable, or "good", states and ensure that they can be reached.

In the example the (final) states that need to be avoided ("bad") are where a room has been booked but not paid for, where payment has been made but a room has not been booked, and where a room has been booked (and paid for) but the meeting is not attended by the Organiser and both Participants.

Figure 5 shows the Finite State Machine corresponding to the defined interaction[6]. Interestingly, there is one bad state that can be reached: state 9 in figure 5 has a room that has been booked and paid, but is a final state: the interaction can terminate here without the meeting actually taking place! The issue is that the commitments to attend the meeting are still conditional (on the meeting being confirmed), and that there is no commitment to confirm the meeting. This could be fixed either by changing the conditional commitments from $\mathsf{CC}(\text{confirmed} \rightsquigarrow \text{attend})$ to $\mathsf{CC}(\text{roombooked} \rightsquigarrow \text{attend})$ (which fixes the problem, and has the side effect of making the *confirm* action unnecessary), or by ensuring that once the room is booked the meeting must be confirmed, which can be done by having the *Pay for room* action create a commitment to confirm the meeting.

4. Discussion

We have presented an approach for designing commitment-based interactions. The approach comprises a simple process, along with techniques and tips for how to perform each step in the process. To the best of our knowledge, this is the first published process for designing interactions using the commitment machine framework of Yolum and Singh.

There are two comparisons that need to be performed. The first is to compare this approach with a traditional message-centric approach to interaction design. It has already been argued elsewhere [14] that the commitment machine framework yields greater flexibility in interactions than more traditional approaches. Additionally, an empirical comparison between the *Hermes* methodology (another non-message-centric approach to designing interactions) and the interaction design part of *Prometheus* (a traditional message-centric approach[7]) has shown that the Hermes methodology produces interactions that are significantly (in a statistical sense) more flexible and robust than interactions produced using the part of Prometheus that is concerned with interaction design [3].

However, for a more direct comparison Figure 6 shows a conventional interaction protocol (omitting the step of attending the meeting), developed using the Prometheus process. Some of the execution sequences that are permitted by the commitment-based interaction but not by the conventionally designed interaction protocol are:

- Beginning the interaction midway. For example, if agreement on a meeting date/time has been obtained.
- Skipping certain steps, for example, if the Organiser agent already knows about the available venues (*venuesprovided* is true) then obtaining venue information can be skipped.
- Payment for the room can occur before the meeting is confirmed, after it is confirmed but before the meeting occurs, or even after the meeting occurs.

The second comparison that needs to be performed is between different non-message-centric approaches, e.g. between the approach in this paper and *Hermes*. This is left for future work.

One key area for future work is to develop a mapping scheme for implementing commitment-based interaction, in the spirit of [5]. Mapping to a BDI-like agent programming language would appear to be straightforward: each action A with preconditions P and effects $\{e_1, \ldots, e_n\}$ would be mapped to a plan with context condition P and with a plan body that makes the effects true.

Another key area for future work is to develop tool support for designing commitment-based interactions. A particular focus is to support checking for various undesirable situations, such as infeasible designs, or the existence of reachable "bad" states. The existing tool (http://www.winikoff.net/CM) already provides some support: it allows the complete interaction space for a commitment machine to be generated and visualised. It is straightforward to extend the tool to check for good and bad states (and perhaps colour them appropriately).

5 It is modelled as a single action, to capture that it's really done at one point in time.

6 The finite state machine is software-generated: the nodes and connections were computed by an implementation of the axioms (available from http://www.winikoff.net/CM) and were then laid out by *graphviz* (http://www.graphviz.org/, http://www.pixelglow.com/graphviz/).

7 Briefly, this process involves introducing messages between steps that are to be performed by different agents, and then generalising to protocols by considering at each point what alternative messages might be sent.

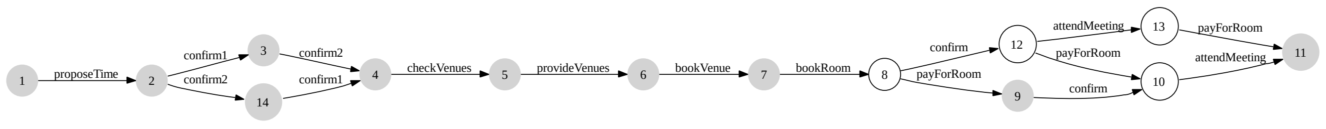

Figure 5. FSM for the Interaction (shaded circle = final state)

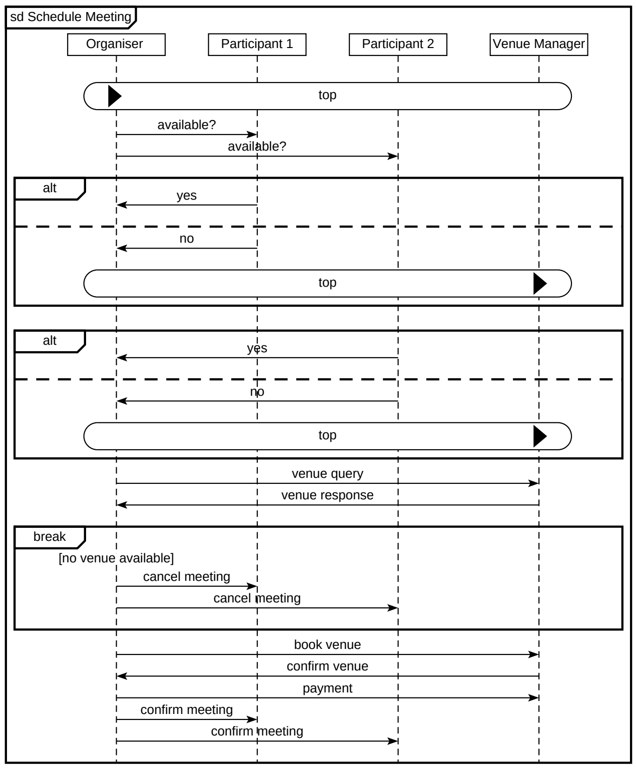

Figure 6. Conventional Interaction Protocol (in Agent UML)

References

[1] R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-agent Programming: Languages, Platforms, and Applications*. Springer, 2005.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8:203–236, May 2004.

[3] C. Cheong and M. Winikoff. Hermes versus Prometheus: A comparative evaluation of two agent interaction design approaches. Submitted for publication.

[4] C. Cheong and M. Winikoff. Hermes: Designing goal-oriented agent interactions. In *Proceedings of the 6th International Workshop on Agent-Oriented Software Engineering (AOSE-2005)*, July 2005.

[5] C. Cheong and M. Winikoff. Hermes: Implementing goal-oriented agent interactions. In *Proceedings of the Third international Workshop on Programming Multi-Agent Systems (ProMAS)*, July 2005.

[6] C. Cheong and M. Winikoff. Improving flexibility and robustness in agent interactions: Extending Prometheus with Hermes. In *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*. Springer-Verlag, 2006.

[7] S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[8] R. A. Flores and R. C. Kremer. A pragmatic approach to build conversation protocols using social commitments. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1242–1243. ACM Press, 2004.

[9] N. Jennings, D. Kinny, M. Wooldridge, and F. Zambonelli. The Gaia methodology. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, chapter 4. Kluwer Academic Publishing (New York), 2004.

[10] S. Kumar, M. J. Huber, and P. R. Cohen. Representing and executing protocols as joint actions. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 543 – 550, Bologna, Italy, 15 – 19 July 2002. ACM Press.

[11] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 2004. ISBN 0-470-86120-7.

[12] M. Winikoff, W. Liu, and J. Harland. Enhancing commitment machines. In J. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Declarative Agent Languages and Technologies II*, number 3476 in Lecture Notes in Artificial Intelligence (LNAI), pages 198–220. Springer, 2004.

[13] P. Yolum. Towards design tools for protocol development. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 99–105. ACM Press, 2005.

[14] P. Yolum and M. P. Singh. 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*, 2004.