Executing Specifications of Social Reasoning Agents

Iain Wallace and Michael Rovatsos

School of Informatics, The University of Edinburgh,
Edinburgh EH8 9LE, UK
{Iain.Wallace,Michael.Rovatsos}@ed.ac.uk

Abstract. Social reasoning theories, whilst studied extensively in the area of multiagent systems, are hard to implement directly in agents. They often specify properties of beliefs or behaviours but not the way these should affect the computational reasoning mechanisms of a concrete agent design. The Expectation-Strategy-Behaviour (ESB) framework addresses this problem by separating and abstracting social reasoning from other practical reasoning, providing the computational machinery that is necessary to perform social reasoning in practice. We present an extension to previous work on ESB to a reasoning system which enables the execution of concise and modular declarative social reasoning rules. These rules represent the belief revision inherent to social reasoning, by representing hidden properties of other agents and social constraints. We review the foundations of the abstract ESB framework and present the implementation of a reasoner based on CTL model checking. Our system allows for conditioning agent behaviours on complex preconditions and verification of properties to aid the agent designer. It also allows for easy integration with a BDI reasoning system. We exemplify the suitability of ESB for social reasoning constructs with a specification of Joint Intention theory in ESB and illustrate the modularity of the representation by extending this to a specific example in a robotic soccer domain.

1 Introduction

In many multiagent environments, achieving coordinated behaviour requires social reasoning on the part of individual agents. This includes reasoning about others and one's relationships with them, or about social objects such as norms and social laws, commitments and conventions, deontic notions such as obligations, permissions, and prohibitions, trust and reputation. While there exists a very rich literature on social reasoning theories, there is hardly any support for implementing the suggested frameworks in actual systems at the level of general-purpose tools to aid agent design and implementation. On the one hand, there exist formal languages for describing action and cooperation theories but these do not include a computational framework for processing such specifications, e.g. [1, 2]. On the other hand, there exist implemented frameworks, but they are either specific to a *particular* social reasoning theory [3, 4], or have only been implemented in a specific software application [5]. General approaches to multiagent reasoning have a lot to offer the community, as shown by the success of BDI. Maintaining a high level of generality allows a method to be applied to many different existing ideas, for their comparison, implementation or to realise the benefits of their combination. A general method capturing key social reasoning abstractions would therefore be of benefit to research on MAS interactions.

In [6], we proposed the Expectation-Strategy-Behaviour (ESB) architecture. ESB is an abstract framework designed to unify many social reasoning approaches, whilst providing the computational machinery to process declarative specifications of a given social reasoning framework. ESB is based on specifications of a reasoning agent's beliefs regarding hidden properties of a system (such as the internal attributes of others or the status of social objects such as those mentioned above) together with belief revision mechanisms that specify how these beliefs are updated. Such a general formalism is useful for capturing social reasoning theories from the agent's point of view, rather than at the system or design level. At its most general social reasoning requires some mechanism for modifying beliefs about the state of social mechanisms, and ESB provides a method to specify these mechanisms.

Social reasoning as a discrete component has several appealing benefits. Foremost, reasoning about interaction makes programming MAS complex, and general abstractions specific to this task would ease it. Additionally, a unifying declarative specification specifically for social reasoning would allow for the design of agents interacting in varied ways by allowing combinations of social reasoning techniques from differing levels of abstraction.

We also presented in [6] an abstract interpreter for ESB specifications that constructs graph-based models of expectation sets and determines how these change over time, and allows for querying the expectation status at runtime. While this is a first step toward providing generic implementation support for social reasoning methods, the system presented previously does not specify a concrete reasoning engine that enables the whole cycle of ESB based reasoning and execution. What is required are algorithms for specification, model generation, querying mechanisms, and integration with a practical reasoning system to combine social reasoning with other rational reasoning capabilities. In this paper, we present an ESB reasoning engine that provides this missing functionality and can be readily used to implement social reasoning frameworks in BDI agents. Our system allows for:

- defining declarative ESB specifications of social reasoning rules. These specify the properties of a social reasoner without worrying about procedural processing, and are easily extended in a modular way;
- automated model generation based on these specifications, model restriction to achieve boundedly rational reasoning, and querying of logical constraints via model checking;
- an interface to generic AgentSpeak(L) [7] specifications to integrate the ESB reasoning component with specifications of BDI agent designs in a loosely coupled way so as to maximise reusability.

With this functionality, our system ESB-RS (ESB Reasoning System) provides comprehensive design and implementation support for existing and new social reasoning frameworks, while maintaining a clear separation of social reasoning from other types of practical reasoning. As far as the authors are aware, no other related work provides an implementation of a general social reasoning engine separated from the practical reasoner.

2 The ESB Framework

The ESB architecture is an abstract model for practical social reasoning systems based on the concept of *expectations*, which are defined as follows:

> An *expectation* contains a *conditional belief* regarding a statement whose truth status will be eventually *verified* by a *test* and updated according to specified *responses* to the test.

This is represented as a belief Φ held by an agent A under condition C. Depending on the outcome of a test T, the agent will take action ρ^+ if the expected belief was confirmed (positive response), and ρ^- if not (negative response).

Strategies constrain the style and scope of the agent's reasoning process, whilst behaviours form the link between the representation of social concepts in the expectations and the practical reasoning process.

In the general case, an expectation is represented as follows:

$$\mathbf{Exp}(N\ A\ C\ \Phi\ T^+\ T^-\ \rho^+\ \rho^-)$$

N is the *name* of the expectation, taken from a set of all expectation labels $\mathbf{EXP} = \{N, N', \ldots\}$.
A is the *agent* holding this expectation, from a set of agents $\{A, A', \ldots\}$.
C is the *condition* under which the expectation holds. Evaluated over the agent's beliefs like the guards on BDI plans.
Φ is the expected belief, i.e. an event or an expectation that another agent is assumed to hold
T^+, T^- are the *tests* which confirm (T^+), or reject (T^-), the expectation of Φ. Tests are split for the purposes of the implementation described here. E.g. a test may succeed when another agent discards an expected card, or fail when a certain number of turns elapses.
ρ^+/ρ^- are the positive/negative *responses* to the tests. Each is specified in terms of two sets of expectations: those to remove, and those to add.

To illustrate ESB we shall re-use from [6] an example based on the Rummy [8] card game. It is only required to understand that players are trying to collect either runs of cards in a suit, or sets of the same value card. On their turn a player must pick up a card. They then create any sets they can and discard, play passes to the next player. In our example domain, an expectation could be that if agent B picks up a $2\diamondsuit$ (C), then we expect (Φ) them to be collecting 2s. The test (T) is if they pickup or discard another 2. The corresponding positive response (ρ^+) is to remove expectations that they are collecting a run. The negative response (ρ^-) is to remove this expectation (leaving others suggesting they're collecting a run).

Strategies specify ways to process sets of expectations. Consider all possible sets of expectations that could arise from future observations of test outcomes - a strategy is a particular way of traversing the graph that results from mapping out all possible future expectation combinations. A strategy therefore controls how the agent reasons based on expected future outcomes. To continue our Rummy example, a strategy could consider only those future states where the opponent plays according to mini-max principles (with an evaluation function based on the utility of cards). Or more simply, when the game nears the end one could reduce the depth of the graph considered to the likely number of hands.

Behaviours determine how the overall expectation base affects the agent's practical reasoning and actions. They are conditioned on statements about expectations, and the truth value will be established by applying the agent's strategy to the expectation base. This condition is tied with an action, which is not necessarily a direct physical action, but rather a modification of the agent's beliefs. An example might be to only allow discards of 2♢ (action) if the opponent is not expected to be collecting 2s, or to be collecting them in the future (condition).

Expectations, together with *strategies* and *behaviours* provide us with a natural way for describing social reasoning methods and devising modular social reasoning designs from belief revision mechanisms regarding hidden properties in the system (such as the mental states of other agents). The notion of *expectation* captures all the elements that are necessary to link beliefs held about non-observable parts of the system. It also allows us to express belief dynamics at a practical, procedural level while benefiting from declarative representations.

2.1 The Operation of an ESB Agent

The ESB theory proposes that there are advantages to separating social reasoning from practical reasoning about actions in the world. In this work, we assume a BDI practical reasoner and the interface between the two reasoners to be the belief revision function of the BDI interpreter.

To demonstrate ESB, we adapt a simple set of expectations and their graph from [6] (Figure 1). For brevity, expectations are labelled with a number, and only the responses are shown in the table. States in the graph show the active set of expectations in that state. The responses define how the state changes according to the tests. The convention used is that an arc labelled "$\neg 2$" is the transition caused by the test for expectation 2 failing, or simply "2" for the success case. Where an arc bears more than one label, it represents parallel arcs grouped for clarity. To refer to components of these expectations we will use notation like "$\Phi(1)$" to specify the expected belief Φ of expectation 1. Correspondingly $T^+(1)$ would be the positive test condition for expectation 1 and so on.

The language for the behaviour conditions follows Computation Tree Logic (CTL). It is a branching time logic specifying possible futures, and only knowl-

Expectations and Responses

Name	ρ^+	ρ^-

1	add(3,4)
	remove(2,5)
		remove(1)

2	add(5)
	remove(1,3,4)
		add(3,4)
		remove(2,5)

3	-	add(2,5)
		remove(1,3)

4	-	add(2,5)
		remove(1,3)

5	add(2)
	remove(1,3,4)
		add(1,3,4)
		remove(2,5)

Behaviour Conditions: $B1.\ \mathbf{E}\Diamond(3)$
$B2.\ \mathbf{A}\Box(\mathbf{A}\Diamond 2)$

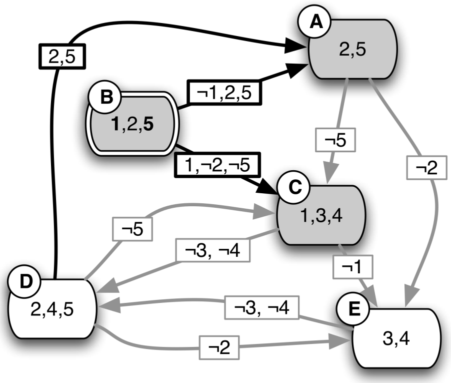

Fig. 1. A simple ESB example (adapted from [6])

edge of a few operators is required here. There are both quantifiers over paths and specific to paths. **E** and **A** mean "there exists a path" and "for all paths". For paths, □ and ◊ can be read as "always" and "eventually", i.e. in all states or in some future state on the respective paths.

An agent is provided with a specification of all expectations $\mathbf{EXP} = \{1, 2, 3, 4, 5\}$, and maintains the sets $EXP_A = \{1, 2, 5\}$ (active expectations, state B in the graph) and $EXP_C = \{1, 5\}$ (current expectations where $C = true$, shown in bold). An ESB agent's reasoning cycle proceeds as follows:

1. Update EXP_C to contain those expectations from EXP_A where $C = true$. *State B is active in the example, and the agent holds 1 and 5 to be true.*
2. For $E \in EXP_C$ add $\Phi(E)$ to the belief base. *We add* $\Phi(1), \Phi(5)$.
3. Apply the strategy to create the restricted strategy graph by considering only a subset of transitions from the current state. *Our example has an optimistic agent, that considers only* ρ^+ *responses, those where there is a positive transition in our graph (marked in bold). Applied to the active state B, this leaves the subgraph of the grey highlighted states as the strategy graph.*
4. For each behaviour, where the behaviour condition is *true*, add the specified belief (the corresponding *action*) to the belief base. *The example behaviours are written in a CTL style. B1 says "3 is possible", this is true in state C. B2 says "2 is always possible" and is false, as state A is in the strategy graph.*
5. Perform BDI plan selection and action execution as normal.
6. For $E \in EXP_C$
   (a) If test $T^+(E)$ applies, update expectations as per the response $\rho^+(E)$
   (b) else if $T^-(E)$ applies, update expectations as per the response $\rho^-(E)$
   If we assume that $T^+(5)$ *holds, then the expectations are updated to place the agent in state A.*

The responses will update the set of expectations, so there may be some overlap where one expectation is added but another response causes its removal. In this case, the current system applies them all recursively, in the order specified by the designer (if they still apply given previous effects). This results in an expectation set consistent with the agent's observations, relative to its current expectations.

3 An ESB Reasoner

ESB only specifies a technique for social reasoning, and so must be combined with a method for practical reasoning. The main components of the ESB reasoner are:

- A BDI reasoning engine, in this case Jason [9].
- A set of plans for practical reasoning, written in AgentSpeak(L) [7].
- An ESB engine, to maintain the expectations and interface with Jason.
- An ESB specification.

Of these only the last has previously been presented in [6]. The contribution here is a description of the algorithms required for the ESB engine, and coupling of this engine with the BDI interpreter. The evaluation of this takes the form of the case studies of joint intentions [10] and the robosoccer example presented later. The interactions between these components are illustrated in Figure 2. The principal interaction between the social and practical reasoning components is via the belief revision function (BRF). Expectations are updated based on the agent's perceptions and beliefs. The control of the agent's social interactions is also in terms of beliefs, as these can act as guards on plans which carry out actions.

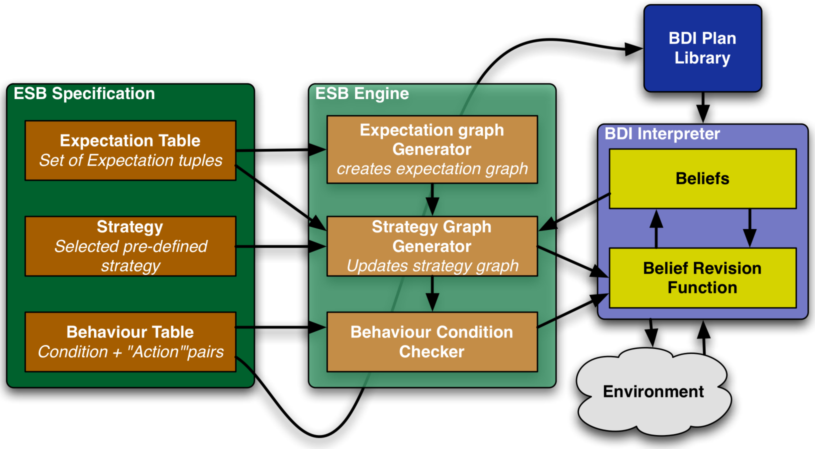

Fig. 2. Overview of an ESB-RS Agent

The execution of ESB-RS can be roughly described as follows. The expectation graph is generated from the specifications. This is only done once, as after this initial step the agent can simply track its current state in the graph. Then, in each reasoning cycle, the beliefs from the BDI part, this expectation graph and the strategy specification are used to create the reduced strategy graph. This is then used by the behaviour condition checker to select the applicable behaviour actions, which are used to update the agent's beliefs accordingly. With the belief-based interface to the BDI practical reasoner, it is necessary for the designer to develop behaviour actions that work in tandem with relevant plans.

3.1 Expectations

The language of expectations is defined formally in [6]. The agent's current state is captured by three sets of expectations: current, active and inactive, updated according to the algorithm described in section 2.1.

In ESB-RS we transform the expectations into a finite state machine (FSM) representation at a lower level of abstraction than the expectation graph of previous work. Behaviour conditions are however defined in terms of the expectation graph. Accordingly we must transform a declarative expectation specification into a form suitable for efficient behaviour checking. The graph may be thought of as a FSM, where it is necessary to check if certain properties hold (the behaviour conditions) given an initial state (defined by EXP_C). This intuition naturally suggests the approach of model checkers, which aim to solve a similar problem. This requires algorithms to create the state descriptions and transition relation from our sets of expectations.

For specification, we do not have to consider all the combinations of expectations that form states in the expectation graph explicitly. This is one of the advantages brought by the model checker, as it implicitly considers these combinations. The FSM is defined at the level of individual expectations, which are specified more easily. Testing a condition on a particular expectation graph state, or setting the current state, is done by specifying the state of all individual expectations. This finer grained approach is possible as each state in an expectation graph can be considered not as one state, but as a high level abstraction of the set of states where only a subset of expectations are relevant. This concept, and the following algorithm to create an FSM with equivalent properties to the conceptual expectation graph model represent the key new development of this work. This transformation allows the use of the NuSMV [11] model checker, and ultimately makes it possible to easily implement the previous ESB theory.

Creating the FSM, we are concerned with only the parts of each expectation that describe the dynamics:

Condition $\in \{True, False, DC\}$ - Each expectation can be considered to be True or False, if it is in the active set EXP_A, or "don't care" (DC) if it is not in current expectation-graph state (that is, $E \in \mathbf{EXP} \setminus EXP_A$).

$\Phi \in \{True, False\}$ - This is as per *condition*, or $False$ where condition is DC.

Test $\in \{Tp, Tm, NA\}$ - The test is either positive (Tp), negative (Tm) or not-applicable (NA) whenever the expectation is DC.

Responses ρ^+, ρ^- - The add- and remove-sets of expectations in each response are used to define the transition relation of the FSM.

The states in the FSM are all unique combinations of each expectation's condition (though of course many will be unreachable). The transition relation is defined as expressions specifying the next state in terms of the current state variables. This is calculated as below:

```
For each expectation E
 next(E.Condition) :=
  case:
   For each expectation O
    If (O.Test = Tp) & (E in O.Tp.addSet) then
     next(E.Condition) : {True,False}
    If (O.Test = Tm) & (E in O.Tm.addSet) then
     next(E.Condition) : {True,False}
    If (O.Test = Tp) & (E in O.Tp.removeSet) then
     next(E.Condition) : {DC}
    If (O.Test = Tm) & (E in O.Tm.removeSet) then
     next(E.Condition) : {DC}
   end for
   Condition = True : {True,False};
   Condition = False : {True,False};
   Condition = DC : {DC};
  end case
 next(E.Test) :=
  case:
   Condition = DC : {NA}
   true : {Tp, Tm, NA}
  end case
end for.
```

The choices in the case statement choose the value in the next state randomly from the set specified, and match the first valid rule. The basic operation of the FSM is easily described in a more natural manner. Each expectation may at the next step change between `False` or `True`, however if it is `DC` then it stays as such. The only time a `DC` expectation becomes active, is when it is in the add-set of another expectation where the relevant test applies. The test for an expectation is not applicable (NA) if the expectation is inactive, otherwise it may be true, false, or neither.

3.2 Strategies

Strategies are defined as restrictions on the expectation graph, to reduce the search space when checking behaviour conditions thus bounding the agent's social reasoning. This is a function from one graph to another, where either the edges or vertices are reduced in some way. So in our implementation it is the number of accessible states in the FSM we wish to reduce to simplify checking. Strategies are so called, as they influence the overall style of the agent's social reasoning rather than being concerned with the details. A simple example would be a optimistic agent, which considers only the positive response transitions - effectively assuming its assumptions about other agent's mental states are always correct.

Strategies are implemented as a set of constraints on the transition relation. Each constraint is a boolean expression which must be satisfied in all states. This means the transitions from a given state are constrained to those where the next state meets all the strategy constraints. The atoms in the constraints are the components of each expectation. These can be combined using various operations defined by the model checker used to check behaviour constraints, NuSMV [11]. Operations can include logical propositions, case statements etc.

For example, the graph could be restricted to consider only those states reachable assuming tests, if they apply, are always the positive case Tp. The constraint to achieve this would be to limit the next state as follows:

$$next(E.Test : \{Tp, NA\})$$

This says that for any state, in the next state the test must be the positive case, if it applies at all. Specific expectations, or expectation graph states (combination of expectations) can also be excluded by name, for example:

$$next(\mathit{Exp1}.C : \{DC\}) \;\&\; next(\mathit{Exp3}.C : \{DC\})$$

This can allow for more complex strategies, which can be generated. An example would be removing states that do not meet some game theoretic criteria as suggested in [6].

3.3 Behaviours

Behaviours are defined in ESB as condition-action pairs. The actions are beliefs to be added to the agent's belief base, and work in conjunction with BDI plans which are included or excluded based on these. The conditions may capture a combination of current and future expectations. It may be desirable for example to only pick up one end of a table if I expect another to lift the other end. Or I might only discard a card in a game if I do not expect the other agent to be collecting it, and cannot expect them to do so in the future.

Constructing a graph from a set of expectations, and checking conditions based on the current state and possible (or impossible) future states naturally suggests an approach based on model checking, as the problem is very similar. This presents an attractive advantage - it is only necessary to describe the states each expectation may have, and the relatively simple relations between them, and the model checker will allow us to form relatively complex queries about expectation graph concepts.

NuSMV is chosen as it allows the checking of CTL logic formulae, allowing us to express possibilities and necessities. A simple example is the condition "If I might hold expectation 2 in the future" which is captured as $\mathbf{E}\Diamond(\Phi^2)$ (there exists a path where eventually Φ^2). Having described the implementation of the

ESB-RS engine to execute declarative specifications, we now proceed to several examples illustrating its utility.

4 Joint Intentions in ESB

In this section we evaluate ESB's suitability for capturing social reasoning concerns by demonstrating that this key concept can be easily expressed. The ability to establish joint commitment is vital to MAS. Without it agents cannot work together - the principal advantage of MAS. The most basic form of joint commitment is Joint Intentions [10], where two agents have interlocking commitments to each other to achieve a joint goal.

Kumar et al. [12] describe the Request Conversation Protocol (RCP) for establishing a joint persistent goal (JPG). The difference between a Joint Intention (JI) and a JPG is in the mutual belief throughout executing the action that it is done as a team. So given a pair of agents believing they are doing the action as a team, the request protocol brings about a state of joint intentions between the agents. ESB is ideally suited to capture Joint Intentions, as the JI centres around commitments the agent holds based on its beliefs about expected commitments of the other agent.

The RCP is a set of communication acts, with associated semantics and mental states that two agents go through to form a JPG. Figure 3 shows the communication acts, and the states the agents go through. If agent X is requesting an action of Y, the intuition is simple. X firsts requests an action from Y, X now holds a persistent weak achievement goal (PWAG) toward Y. Y either agrees or refuses, in the "agree" case Y now also holds a PWAG toward X to achieve the goal. There is now a JI. X has a goal for Y to do some action, and Y has a goal to do this action relative to X's desire to do so.

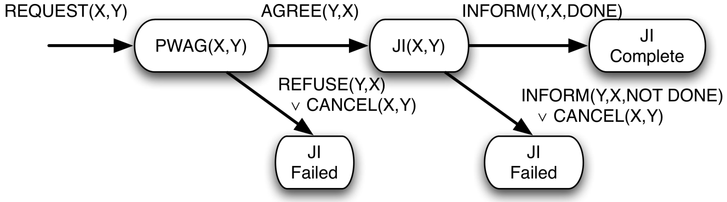

Fig. 3. The Request Conversation Protocol (adapted from [12])

To simplify the example one assumption is made. We omit the case where the requesting agent cancels the request.

There are several primitive components that the protocol is built upon: the individual commitments; persistent goal (PGOAL) and persistent weak achievement goal (PWAG); and the speech acts REQUEST, AGREE, REFUSE and INFORM. The formal definitions can be found in [12], but the description of a

PWAG is useful here. “PWAG(X,Y,A,Q)” says agent X has a persistent goal to achieve A, given relevancy condition Q, and will have a persistent goal to notify Y that A is achieved, or becomes impossible or irrelevant (¬Q holds).

These primitives are all considered as actions the agent can perform, be they communicative or holding an individual commitment. They are not social reasoning, this will be separated out into the ESB side of the reasoner, and acts on top of these basic components. These components are therefore implemented in the BDI practical reasoner component, and not described here.

4.1 Implementing JI with ESB-RS

Only three expectations are needed to allow the agent to use the above primitives to form and act on a joint intention. The expectations used are shown in table 1. The agent holding each expectation is assumed to be “self”, and the other either X or Y depending on the role played. The three expectations can be explained as follows:

1. When I have a PWAG, and expect Y also has the PWAG relative to my own goal, I expect a JI to do A exists. The positive test outcome is I believe A (the action was performed), not Q (is no longer relevant) or F (failed). The negative result is harder to express neatly. Loosely it is that A, not Q or F is believed and this is not communicated by the other agent - they have not upheld their part of the joint commitment. Even if they do not directly observe the completion event, they should reply to ensure mutual belief.
2. When I receive AGREE(A,Q,F) from Y, I expect Y holds PWAG(Y,X,A,[Q ∧ PWAG(X,Y,A,Q)]). The relevancy condition for Y’s PWAG also includes that X still has the commitment toward the goal. Intuitively, it makes no sense for an agent to perform a requested action if the requester is no longer committed to it. The positive test is that Y informs me of A, not Q or F, negative test is I observe A, not Q or F and Y does not inform me as per 1.
3. When I receive REQUEST(A,Q,F) from X expect X holds PWAG(X,Y,A,Q). The tests are as 2, only obviously I expect X to inform me.

Although these expectations follow logically from the semantics of the communication acts, they are still *assumptions* about another agent’s mental state, and so are separated out as social reasoning, for the advantages described in [6].

Only a couple of responses are required. In the failure cases the belief that the other agent has agreed or requested an action is removed. The effect is to remove the expected PWAG from the belief base, and thus the expectation of a JI (if any). In addition, it may be desirable to add some response actions as a result of JI being upheld or not. An obvious case would be maintaining a list of other agents who have been proven willing to co-operate, to aid future decisions.

In this simple set of expectations, there is no real need to choose a strategy, as there are not significant numbers of transitions in the graph. Strategies are envisaged as becoming more useful to bound and direct reasoning in more complex situated examples.

Table 1. JI Expectations

Expectation	1. ExpJI	2. ExpAgree	3. ExpReq
Condition	pwag(self,Y,A,Q) $\land$ pwag(Y,self,A,[Q $\land$ pwag(self,Y,A,Q)])	agree(A,Q,F) [source(Y)]	request(A,Q,F) [source(X)]
Φ	ji(self,Y,A)	pwag(Y,self,A,[Q $\land$ pwag(self,Y,A,Q)])	pwag(X,self,A,Q)
T+	A $\lor$ $\neg$Q $\lor$ F	(A $\lor$ $\neg$Q $\lor$ F) [source(Y)]	(A $\lor$ $\neg$Q $\lor$ F) [source(X)]
T−	(A $\lor$ $\neg$Q $\lor$ F) [source($\neg$Y)]	(A $\lor$ $\neg$Q $\lor$ F) [source($\neg$Y)]	(A $\lor$ $\neg$Q $\lor$ F) [source($\neg$X)]
ρ^+	-	-	-
ρ^-	-	remove agree(A,Q,F)	remove request(A,Q,F)

The behaviour (*condition* $\rhd$ *action*) pairs represent the interface between the social and practical reasoner. Only one behaviour is required for an agent to act upon joint intentions, that is:

$$\text{ji(self,Y,A)} \rhd (\textbf{bel}\ \text{haveJI(A)})$$

This says when I expect I have a JI to do an action A, add the synthetic belief *haveJI(A)* to the belief base. This belief represents the JI and should act as a guard on all plans that require the JI to succeed. So it can be read simply as, "allow joint plans only when I have a joint intention". This seems very obvious and simple but this is the nature of Joint Intentions. It is the most basic building block on top of which more complex social reasoning and interaction can be built. Here it is specified explicitly, rather than designed into a system implicitly.

Although the above is all that is strictly necessary to express joint intentions, ESB provides an easy route to simply and generically answer the social reasoning questions of when to request a joint action, and when to agree to one. Both are necessary considerations for any implementation, and bridging the gap from theory to implementation is a key driver of our work.

The first consideration is when to request a JI. Specifically, when it is possible in the future to hold a JI toward A, and it is desired, then request:

$$\mathbf{E}\Diamond(\text{ji(self,Y,A)}) \land (\textbf{desire}\ \text{A}) \rhd (\textbf{bel}\ \text{requestJI})$$

The condition is similar for agreeing to a request, the only difference being the requirement that a request has been received:

$$\text{request(A,Q,F)} \land \mathbf{E}\Diamond(\text{ji(self,Y,A)} \land (\textbf{desire}\ \text{A}) \rhd (\textbf{bel}\ \text{agreeRequest(A)})$$

In terms of expectation graphs, the condition says that if from the current state, a possible future expectation state includes one in which I hold a JI - then it is sensible to request (or agree) to one. The converse is that it is not rational to request or agree to a JI if an agent doesn't believe that its future reasoning (or the other agent's) can bring about this state.

To conclude, we have shown it is easy to capture Joint Intentions in ESB-RS, and so we now move on to a more complex example.

5 Case Study

To evaluate how ESB can be easily used in practice, and extended in a modular way, we present a robotic soccer example. The situation we consider here is an example of a team plan to score a goal - thus requiring joint commitment between agents, as presented above. The scenario we consider involves three team-mates, one with the ball. The agent in possession of the ball wishes one agent to go up either side of the pitch, one as a feint and the other to pass the ball to, then shoot.

If the agent with the ball is Andy, wanting to pass to Barney and have Cathy perform the feint, the intended operation is as follows. Andy must form a JI with Barney to move up the pitch receive a pass and shoot, and with Cathy to move up the pitch. But we can also assume that each agent also has practical reasoner plans to score goals a variety of ways, in different situations. To handle when to request and agree to joint actions in this example, we take the set of expectations presented previously, and add two expectations as follows:

ExpFree When another agent doesn't have the ball, expect they are "free" (to accept requests). The test for this is if they accept or refuse a request. The positive response is to add ExpJI and ExpAgree from table 1 to the current set, and remove ExpTeam.

ExpTeam When another agent has the ball, expect it to desire a joint intention for some team action to score. The test is if they request some joint action. The positive response is to add ExpJI and ExpReq from table 1 to the current set, and remove ExpFree.

This produces the expectation graph shown in Figure 4. The basic idea of these expectations is to demonstrate one of ESB's strengths - bounding reasoning. By only adding the expectations about joint intentions to the current set when they become applicable, it reduces the number of conditions and expectations that must be maintained. This also provides a very natural way to represent any scenario where an agent may act in different "scenes" performing different roles. Also simply by extending the above general JI expectations, in a modular fashion, we get this benefit. So it is easy to add and integrate reasoning rules together.

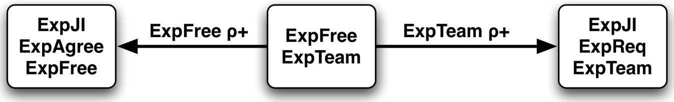

Fig. 4. Expectation Graph for the Robosoccer Agents

We now have enough of the example defined to demonstrate another advantage of ESB. It is possible to easily perform checks on sets of expectations to aid the designer. For example, in this case the intention is that an agent should only ever eventually either be in a state to expect to receive a request for a JI, or to try and form one. By exploiting the automatically generated FSM, we can query the model checker at design time, and check if the set of expectations meets our design. In this case we test to ensure it's always the case that both expectations do not hold together:

$$\mathbf{A}\Box\neg(\mathit{ExpAgree} \wedge \mathit{ExpReq})$$

If this test should fail,the model checker provides an error trace shows us exactly the states and transitions leading to the failure, and we can correct our design accordingly.

As well as implementation specific tests, it is also possible to describe general tests that the ESB formalism allows and that can be applied to any design. It is reasonable to expect that if there is an expectation which can never hold, then there is a problem with the design. We check the following condition for each expectation:

$$\mathbf{E}\Diamond(\mathit{Expectation})$$

In our example, if it were to be broken by making the response of ExpTeam empty, then this test would fail for ExpReq. We could then either fix our design through manual inspection of the expectations, or by identifying more specifically when the behaviour is not as intended by checking a more complex property and inspecting the error trace.

Behaviours are as easily extended as the expectations. As before, we have the behaviour to add a belief noting when we have a joint intention, so that BDI plans can take advantage of this. When we expect that a JI is held, plans depending on it are allowed via a guard belief representing this. The behaviours presented above to dictate when we expect it is possible to form a JI can be included as before, or for convenience slightly extended as follows:

$$\mathbf{E}\Diamond(\text{ji(self,Y,A)}) \wedge \text{free(Y)}\triangleright (\textbf{bel}\ \text{requestJI(Y)})$$

$$\text{request(A,Q,F)} \wedge \mathbf{E}\Diamond(\text{ji(self,Y,A)}) \wedge \text{free(self)} \triangleright (\textbf{bel}\ \text{agreeRequest(A)})$$

Here we add in the requirements on the expectations that relate to agents being free to accept JIs, and only agreeing to a JI if an agent is itself free. The intuition here is that if an agent has the ball, it will have better options for scoring itself and will not agree to a proposed joint action (though it may propose one itself). As the first behaviour uses the temporal operators, even although an agent initially is in a state where no expectations about JIs are current, the condition takes into account possible future reasoning, and so the behaviour holds.

The final component of the soccer agents is the set of BDI plans. In AgentSpeak(L) plans are of the form:

```
+!goal : guard
 -> actions or subgoals.
```

Plans are selected on a priority ordering from the set of relevant plans - those whose guard holds, and trigger meets the current goal. So, amongst other plans to achieve the goal `score`, the social reasoning agents have the following plans:

```
 +!score : haveBall & ji(self,Barney,Pass)
  & ji(self,Cathy,Feint)
   <- passTo(Barney).
+!score: request(Andy,Action) & agreeRequest(Action)
   <- .tell(Andy,agree(Action)).
+!score : ji(self,Andy,Action)
   <- do(Action).
+!score : haveBall & requestJI(Barney,Cathy)
   <- request(Barney,Pass);
       request(Cathy,Feint).
```

The beliefs `underlined` are synthetic ones added by behaviours, and represent the interface between the ESB social reasoner and BDI practical reasoner. Using beliefs added by the ESB reasoner as guards on plans helps bound the agent's reasoning in another important way - it is only necessary to consider plans consistent with the agent's current social reasoning.

6 Conclusion

This paper has presented ESB-RS, a practical social reasoning system built upon ESB theory [6]. We have shown through the specification and implementation of Joint Intentions, that although general (indeed because of the generality) it is capable of representing social reasoning theories whilst removing the concern of managing the procedural processing. The extension of this example to a robosoccer agent allowed us to evaluate the modularity of the agent specification, showing how additional reasoning schemes can be combined, and how the agent's reasoning can be bounded. Additional benefits of ESB-RS to the agent designer have been demonstrated. The model checker based implementation allows a user to check the operation of the agent's reasoning matches their intent, and the ability to perform general tests on the sanity of an expectation set has been described.

An obvious question is "how general is ESB-RS?", and the related question of "what classes of reasoning system can be captured?" By showing that the most basic social requirement of Joint Commitment can be implemented with ESB-RS we have taken a first step toward answering this question. Further work will focus on evaluating not only this, but other ESB-RS implementations of social reasoning schemes, to explore its adequacy to represent a wide range of social methods. A problem here is the lack of easy classification of social reasoning techniques, their often wildly different levels of abstraction, and the llack of existing implementations for comparison.

Taking a broader view, we have shown that it is possible to create an implementation of the ESB framework, which itself forms the bridge necessary for us to implement varied examples of social reasoning for agents. This is the driver of our contribution, as ESB-RS now makes it possible to create agents using previously un-implemented social reasoning schemes, and eases the development of novel social agents. This is the direction we hope to take in future work. By implementing other social reasoning schemes we will refine the ESB-RS reasoner, and in the process hopefully gain a new understanding of practical agent social reasoning. It is also hoped that ESB-RS will provide a viable method to easily combine reasoning schemes in new ways.

References

1. Allouche, M., Boissier, O., Sayettat, C.: Temporal social reasoning in dynamic multi-agent systems. In: ICMAS '00: Proc. of the Fourth International Conf. on MultiAgent Systems (ICMAS-2000), IEEE (2000) 23
2. Sauro, L., Gerbrandy, J., van der Hoek, W., Wooldridge, M.: Reasoning about action and cooperation. In: In AAMAS '06: Proceedings of the fifth International Joint Conference on Autonomous Agents and Multi-agent Systems, ACM Press (2006) 185–192
3. Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., van der Torre, L.: The boid architecture - conflicts between beliefs, obligations, intentions and desires. In: In Proc. of the Fifth International Conference on Autonomous Agents, ACM Press (2001) 9–16
4. F. Dignum, D. Kinny, L. Sonenberg: From Desires, Obligations and Norms to Goals. Cognitive Science Quarterly **2**(3-4) (2002) 407–430
5. Sichman, J.S., Demazeau, Y.: On social reasoning in multi-agent systems. Revista Iberoamericana de Inteligencia Artificial **13** (2001) 68–84
6. Wallace, I., Rovatsos, M.: Bounded Social Reasoning in the ESB Framework. In Decker, S., Sichman, J., Sierra, C., Castelfranchi, C., eds.: Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), May 10-15, Budapest, Hungary (2009) 1097–1104
7. Rao, A.: AgentSpeak(L): BDI agents speak out in a logical computable language. Agents Breaking Away. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '96 Proceedings (1996) 42 – 55
8. : Rummy.com - The Rules of Rummy (2008) http://rummy.com/rummyrules.html.
9. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley (2007)
10. Cohen, P.R., Levesque, H.J.: Teamwork. Noûs **35** (1991) 487–512
11. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new Symbolic Model Verifier. In: Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99). Number 1633 in Lecture Notes in Computer Science (1999) 495–499
12. Kumar, S., Huber, M.J., Cohen, P.R., McGee, D.R.: Toward a formalism for conversation protocols using joint intention theory. Computational Intelligence **18**(2) (2002) 174 – 228