Models of Interaction as a Grounding for Peer to Peer Knowledge Sharing

David Robertson[1], Adam Barker[1], Paolo Besana[1], Alan Bundy[1], Yun Heh Chen-Burger[1], David Dupplaw[2], Fausto Giunchiglia[3], Frank van Harmelen[4], Fadzil Hassan[1], Spyros Kotoulas[4], David Lambert[1], Guo Li[1], Jarred McGinnis[1], Fiona McNeill[1], Nardine Osman[1], Adrian Perreau de Pinninck[5], Ronny Siebes[4], Carles Sierra[5], and Chris Walton[1]

[1] Informatics, University of Edinburgh, UK
[2] Electronics and Computer Science, University of Southampton, UK
[3] Information and Communication Technology, University of Trento, Italy
[4] Mathematics and Computer Science, Free University, Amsterdam, Netherlands
[5] Artificial Intelligence Research Institute, Barcelona, Spain

Abstract. Most current attempts to achieve reliable knowledge sharing on a large scale have relied on pre-engineering of content and supply services. This, like traditional knowledge engineering, does not by itself scale to large, open, peer to peer systems because the cost of being precise about the absolute semantics of services and their knowledge rises rapidly as more services participate. We describe how to break out of this deadlock by focusing on semantics related to interaction and using this to avoid dependency on *a priori* semantic agreement; instead making semantic commitments incrementally at run time. Our method is based on interaction models that are mobile in the sense that they may be transferred to other components, this being a mechanism for service composition and for coalition formation. By shifting the emphasis to interaction (the details of which may be hidden from users) we can obtain knowledge sharing of sufficient quality for sustainable communities of practice without the barrier of complex meta-data provision prior to community formation.

1 Introduction

At the core of this paper is an unusual view of the semantics of Web service coordination. When discussing semantics it is necessary to ground our definitions in some domain in order to decide whether our formal machinery performs appropriate inference. Normally this grounding is assumed to be in the Web services themselves, so formal specification focuses on individual services. It seems natural then to assume that having defined the semantics of services precisely we can combine them freely as long as our means of combination preserves the local semantics of those services. This assumption is ill founded for large scale systems because, when we combine services, we normally share information (by connecting inputs to outputs) and this raises the issue of whether the semantics of information provided by a service is preserved by another service obtaining that information. Universal standardisation of semantics across services appears impractical on a large scale; partly because broad ontological consensus

T.S. Dillon et al. (Eds.): Advances in Web Semantics I, LNCS 4891, pp. 81–129, 2008.

is difficult to achieve but also because the semantics of service interfaces derives from the complex semantics of the programs providing those services.

We explore an alternative approach, where services share explicit knowledge of the interactions in which they are engaged and these models of interaction are used operationally as the anchor for describing the semantics of the interaction. By shifting our view in this way we change the boundaries of the semantic problem. Instead of requiring a universal semantics across services we require only that semantics is consistent (separately) for each instance of an interaction. This is analogous to the use in human affairs of contracts, which are devices for standardising and sharing just those aspects of semantics necessary for the integrity of specific interactions.

In what follows we focus on interaction and we use models of the salient features of required interactions in order to provide a context for knowledge sharing. We are able to exchange interaction models (and associated contextual knowledge) between peers[1] that may not have collaborated before, with the context being extended and adapted as interaction proceeds. This changes the way in which key elements of distributed knowledge sharing, such as ontology alignment, are approached because semantic commitments made for the purposes of interaction are not necessarily commitments to which individual peers must adhere beyond the confines of a specific interaction. Different types of interaction require different commitments, and different levels of integrity in maintaining these.

1.1 A Scenario

To ground our discussion, we give a scenario to which we shall return throughout this paper:

> Sarah works for a high-precision camera manufacturing company and is responsible for periodic procurement for market research. Her job includes identifying competitors' newest products as they come onto the market and purchasing them for internal analysis. She knows which types of camera she needs, but, to save money, may vary the ways she purchases them. She isn't always sure what is the best way to purchase these cameras, but via recommendation service she learns of an Internet shop; an auction service (where she sets initial and maximum prices and the auction service finds a supplier by competitive bidding) and a purchasing service direct from the manufacturer (which allows some configuration of the order to take place on-line). Each of these three services has a different way of describing what they do but the system she uses to access the services can supply the translation necessary to see each of the three interactions through. She tries all three automatically; compares the prices offered; checks that she is comfortable with the way in which the interaction was performed for her; then buys from the one she prefers.

Sarah will have encountered, in the scenario above, several issues that will be explored later in this paper. Her ontology for describing a camera purchase had to be

[1] We use the term "peer" above to emphasise the independence of the services involved, rather than to suggest any specific peer-to-peer architecture.

matched to those of available services (Section 3). Her recommendation service had to know which services might best be able to interact and enable them to do so (Sections 2 and 4). When they interact the contextual knowledge needed to interact reliably should propagate to the appropriate services as part of the interaction (Section 5). When the services are being coordinated then it might be necessary to reconcile their various constraints in order to avoid breaking the collaboration (Section 6). Before she started, Sarah might have wanted to be reassured that the interaction conforms to the requirements of her business process (Section 7) and that her interaction was reliable (Section 8).

1.2 Structure of This Paper and Its Link to Computation

Central to this paper is the idea that models of interaction can be specified independently of services but used operationally to coordinate specific services. In Section 2 we describe a compact language for this purpose. The mechanisms needed to make the language operational in a peer to peer setting are sufficiently compact that they can specified in totality in this paper; and they are sufficiently close to an encoding in a declarative language that a Prolog interpreter can be obtained by a simple syntactic translation of the specification in Section 2. Similarly, the interaction model examples of Figures 2, 3, 4, 9 and 13 translate directly to their operational versions. Together, this makes it practical for the reader to understand and test at coding level the core mechanisms specified in this paper. This style of compact, complete, executable specification (made easy for programmers to pick up) is in the tradition of lightweight formal methods [26, 48].

Our lightweight language allows us to demonstrate how an interaction oriented view of semantics allows us to tackle traditional semantic web problems in unusual ways. Space prohibits us from presenting these in complete, executable detail but for each method there exists a detailed paper (see below), so it is sufficient to provide a compact formal reconstruction of these in a uniform style. Then, in Section 9, we summarise an implemented system for peer to peer knowledge sharing that embodies many of these ideas. The issues addressed are:

Dynamic ontology mapping (Section 3)**:** necessary because we cannot rely solely on *a priori* ontology mapping. Details in [8].

Coalition formation (Section 4)**:** necessary because the semantics of an interaction is sensitive to the choice of which peers participate in the interaction. Details in [46].

Maintaining interaction context (Section 5)**:** necessary because interaction models represent contracts between peers and the semantics of the interaction depends on contextual information accompanying these contracts. Details in [47].

Making interactions less brittle (Section 6)**:** necessary because peers are not as predictable or stable as subroutines in a programming language so it is useful to have mechanisms to avoid this or reduce its impact. Details in [23, 36].

Satisfying requirements on the interaction process (Section 7)**:** necessary because on the Internet languages for relating requirements (particularly business requirements) to services are becoming established, so interaction models should connect to these rather than compete with them. Details in [32].

Building interaction models more reliably (Section 8)**:** necessary because the design of interaction models is of similar engineering complexity to the design of programs, hence we need analogous robust mechanisms to reduce error in design. Details in [42, 55].

The aim of this paper is to describe an alternative view of interaction between peers that share knowledge. What a "peer" might be is understood differently in different communities (in multi-agent systems a peer is an agent; in a semantic web a peer is a program supplying a service) and is supported on different infrastructures (in multi-agent systems via performative based message passing; in Web service architectures by connecting to WSDL interfaces). Our specification language though computational, is more compact than specification languages that have grown within those communities and infrastructures but it can be related back to them, as we discuss in Section 10. We have also applied our methods direclty in business modelling [33] and in e-Science [5, 57].

2 Interaction Modelling

In this section we describe a basic language for modelling interactions. Our use of this language in the current section will be for specification of interactions but in the sections that follow it will be used for executing interactions. We do not claim that this language, as it stands, is ideally suited to deployment in the current Web services arena - on the contrary, we would expect it to be adapted to whatever specification standards emerge (the most stable currently being RDF and OWL, see Section 10) and linked to appropriate forms of service invocation (for example, we have used WSDL). The aim of this paper is to present the essentials of our interaction oriented method in as compact a form as possible.

2.1 A Lightweight Coordination Calculus

Our aim in this section is to define a language that is as simple as possible while also being able to describe interactions like the one in our scenario of Section 1.1. It is built upon a few simple principles:

- Interactions can be defined as a collection of (separate) definitions for the roles of each peer in the interaction.
- To undertake their roles, peers follow a sequence of activities.
- The most primitive activity is to send or receive a message.
- Peers may change role (recursively) as an activity.
- Constraints may be defined on activities or role changes.

Figure 1 defines the syntax of the Lightweight Coordination Calculus (LCC). An interaction model in LCC is a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described by the type of role and an identifier

$$
\begin{aligned}
Model &:= \{Clause, \ldots\} \\
Clause &:= Role :: Def \\
Role &:= a(Type, Id) \\
Def &:= Role \mid Message \mid Def\ then\ Def \mid Def\ or\ Def \\
Message &:= M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role \\
C &:= Constant \mid P(Term, \ldots) \mid \neg C \mid C \wedge C \mid C \vee C \\
Type &:= Term \\
Id &:= Constant \mid Variable \\
M &:= Term \\
Term &:= Constant \mid Variable \mid P(Term, \ldots) \\
Constant &:= \text{lower case character sequence or number} \\
Variable &:= \text{upper case character sequence or number}
\end{aligned}
$$

Fig. 1. LCC syntax

for the individual peer undertaking that role. The definition of performance of a role is constructed using combinations of the sequence operator ('$then$') or choice operator ('or') to connect messages and changes of role. Messages are either outgoing to another peer in a given role ('$\Rightarrow$') or incoming from another peer in a given role ('$\Leftarrow$'). Message input/output or change of role can be governed by a constraint defined using the normal logical operators for conjunction, disjunction and negation. Notice that there is no commitment to the system of logic through which constraints are solved - so different peers might operate different constraint solvers (including human intervention).

2.2 Return to Scenario

To demonstrate how LCC is used, we describe the three interaction models of our scenario from Section 1.1. Figure 2 gives the first of these: a basic shopping service. This contains two clauses: the first defining the interaction from the viewpoint of the buyer; the second from the role of the shopkeeper. Only two roles are involved in this interaction so it is easy to see the symmetry between messages sent by one peer and received by the other. The interaction simply involves the buyer asking the shopkeeper if it has the item, X, then the shopkeeper sending the price, P, then the buyer offering to buy at that price and the shopkeeper confirming the sale.

The constraints in the interaction model of Figure 2 - $need(X)$, $shop(S)$, $afford(X, P)$ and $in\_stock(X, P)$ - must be satisfied by the peers in the role to which the constraint is attached (for example the buyer must satisfy the $afford$ constraint). We write $known(A, C)$ to denote that the peer with identifier A knows the axiom C. LCC is not predicated on a specific constraint language (in fact we shall encounter two constraint languages in this paper) but a common choice of constraint representation in programming is Horn clauses, so we follow this conventional path. By supplying Horn clause axioms in this way we can describe peer knowledge sufficient to complete the

$a(buyer, B)$ ::
$ask(X) \Rightarrow a(shopkeeper, S) \leftarrow need(X)\ and\ shop(S)\ then$
$price(X, P) \Leftarrow a(shopkeeper, S)\ then$
$buy(X, P) \Rightarrow a(shopkeeper, S) \leftarrow afford(X, P)\ then$
$sold(X, P) \Leftarrow a(shopkeeper, S)$

$a(shopkeeper, S)$ ::
$ask(X) \Leftarrow a(buyer, B)\ then$
$price(X, P) \Rightarrow a(buyer, B) \leftarrow in\_stock(X, P)\ then$
$buy(X, P) \Leftarrow a(buyer, B)\ then$
$sold(X, P) \Rightarrow a(buyer, B)$

Fig. 2. Shop interaction model

interaction model. For instance, if we have a buyer, b, and a shopkeeper, s, that know the following:

$$known(b, need(canonS500))$$
$$known(b, shop(s))$$
$$known(b, afford(canonS500, P) \leftarrow P \leq 250)$$
$$known(s, in\_stock(canonS500, 249))$$

then the sequence of messages in Table 1 satisfies the interaction model.

Table 1. Message sequence satisfying interaction model of Figure 2

Recipient	Message	Sender
a(shopkeeper,s)	ask(canonS500)	a(buyer,b))
a(buyer,b)	price(canonS500,249)	a(shopkeeper,s))
a(shopkeeper,s)	buy(canonS500,249)	a(buyer,b))
a(buyer,b)	sold(canonS500,249)	a(shopkeeper,s))

Figure 3 gives the second scenario in which a peer, S, seeking a vendor for an item, X, sends a message to an auctioneer, A, stating that S requires X and wants the auction for it to start at purchase value I and stop if the purchase value exceeds maximum value M with the bid value increasing in increments of I. On receiving this requirement the auctioneer assumes the role of a caller for bids from the set of vendors, Vs, that it recognises and, if the call results in a bid to sell the item at some price, P, then the auctioneer offers that price to the seeker who clinches the deal with the vendor and gets its agreement - otherwise the auctioneer signals that no offer was obtained. The role of caller (assumed by the auctioneer) involves two recursions. The first recursion is over the value set by the seeker: the caller starts with the initial value, L, and changes role to a notifier for the vendor peers in Vs that they have a potential sale of an item of type X at value L. If a vendor, V, is obtained by the notifier then the offered price, P, is set to L; if not the price is incremented by the given amount, I, and the role recurses. The

$$
\begin{array}{l}
a(seeker, S) :: \\
\quad require(X, L, M, I) \;\Rightarrow\; a(auctioneer, A) \leftarrow need(X, L, M, I)\ and\ auction\_house(A)\ then \\
\quad \left(\begin{array}{l} offer(V, X, P) \;\Leftarrow\; a(auctioneer, A)\ then \\ clinch(X, P) \;\Rightarrow\; a(vendor, V)\ then \\ agreed(X, P) \;\Leftarrow\; a(vendor, V) \end{array} \right)\ or \\
\quad no\_offer(X) \;\Leftarrow\; a(auctioneer, A)
\end{array}
$$

$$
\begin{array}{l}
a(auctioneer, A) :: \\
\quad require(X, L, M, I) \;\Leftarrow\; a(seeker, S)\ then \\
\quad a(caller(Vs, X, L, M, I, V, P), A) \leftarrow vendors(Vs)\ then \\
\quad \left(\begin{array}{l} offer(V, X, P) \;\Rightarrow\; a(seeker, S) \leftarrow not(P = failed)\ or \\ no\_offer(X) \;\Rightarrow\; a(seeker, S) \leftarrow P = failed \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
a(caller(Vs, X, L, M, I, V, P), A) :: \\
\quad a(notifier(Vs, X, L, Ps), A)\ then \\
\quad \left(\begin{array}{l} null \leftarrow s(V) \in Ps\ and\ P = L\ or \\ null \leftarrow L > M\ and\ P = failed\ or \\ a(caller(Vs, X, Ln, M, I, V, P), A) \leftarrow not(s(V) \in Ps)\ and\ Ln = L + I\ and\ Ln \leq M \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
a(notifier(Vs, X, C, Ps), A) :: \\
\quad \left(\begin{array}{l} need(X, C) \;\Rightarrow\; a(vendor, V) \leftarrow Vs = [V|Vr]\ then \\ \quad \left(\begin{array}{r} Ps = [s(V)|Pr] \leftarrow supply(X, C) \;\Leftarrow\; a(vendor, V)\ or \\ Ps = Pr \leftarrow decline(X, C) \;\Leftarrow\; a(vendor, V) \end{array} \right)\ then \\ a(notifier(Vr, X, C, Pr), A) \end{array} \right)\ or \\
\quad null \leftarrow Vs = []\ and\ Ps = []
\end{array}
$$

$$
\begin{array}{l}
a(vendor, V) :: \\
\quad \left(\begin{array}{l} need(X, C) \;\Leftarrow\; a(notifier(Vs, X, C, Ps), A)\ then \\ \quad \left(\begin{array}{l} supply(X, C) \;\Rightarrow\; a(notifier(Vs, X, C, Ps), A) \leftarrow sell(X, C)\ or \\ decline(X, C) \;\Rightarrow\; a(notifier(Vs, X, C, Ps), A) \leftarrow not(sell(X, C)) \end{array} \right)\ then \\ a(vendor, V) \end{array} \right)\ or \\
\quad (clinch(X, P) \;\Leftarrow\; a(seeker, S)\ then \\
\quad agreed(X, P) \;\Rightarrow\; a(seeker, S))
\end{array}
$$

Fig. 3. Auction interaction model

second recursion is within the notifier which tells each vendor, V, in Vs that item X is needed at current offer price, C; then receives a message from V either offering to supply or declining.

Now in order to satisfy the interaction model we define the following example knowledge possessed by seeker, b, auctioneer, a, and two vendors, $v1$ and $v2$:

$$
\begin{array}{l}
known(b, need(canonS500, 100, 200, 10)) \\
known(b, auction\_house(a)) \\
known(a, vendors([v1, v2])) \\
known(v1, sell(canonS500, 110)) \\
known(v2, sell(canonS500, 170))
\end{array}
$$

and then the sequence of messages in Table 2 satisfies the interaction model.

Table 2. Message sequence satisfying interaction model of Figure 3

Recipient	Message	Sender
$a(auctioneer, a)$	$require(canonS500, 100, 200, 10)$	$a(seeker, b)$
$a(vendor, v1)$	$need(canonS500, 100)$	$a(notifier([v1, v2], canonS500, 100, Ps1), a)$
$a(notifier([v1, v2], canonS500, 100, Ps1), a)$	$decline(canonS500, 100)$	$a(vendor, v1)$
$a(vendor, v2)$	$need(canonS500, 100)$	$a(notifier([v2], canonS500, 100, Ps1), a)$
$a(notifier([v2], canonS500, 100, Ps1), a)$	$decline(canonS500, 100)$	$a(vendor, v2))$
$a(vendor, v1)$	$need(canonS500, 110)$	$a(notifier([v1, v2], canonS500, 110, Ps2), a)$
$a(notifier([v1, v2], canonS500, 110, Ps2), a)$	$supply(canonS500, 110)$	$a(vendor, v1)$
$a(vendor, v2)$	$need(canonS500, 110)$	$a(notifier([v2], canonS500, 110, Ps3), a)$
$a(notifier([v2], canonS500, 110, Ps3), a)$	$decline(canonS500, 110)$	$a(vendor, v2)$
$a(seeker, b)$	$offer(v1, canonS500, 110)$	$a(auctioneer, a)$
$a(vendor, v1)$	$clinch(canonS500, 110)$	$a(seeker, b)$
$a(seeker, b)$	$agreed(canonS500, 110)$	$a(vendor, v1)$

Figure 4 gives the third scenario in which a peer that wants to be a customer of a manufacturer asks to buy an item of type X from the manufacturer, then enters into a negotiation with the manufacturer about the attributes required to configure the item to the customer's requirements. The negotiation is simply a recursive dialogue between manufacturer and customer with, for each attribute (A) in the set of attributes (As), the manufacturer offering the available attribute and the customer accepting it. When all the attributes have been accepted in this way, there is a final interchange committing the customer to the accepted attribute set, Aa, for X.

$$
\begin{array}{l}
a(customer, C) :: \\
\quad ask(buy(X)) \;\Rightarrow\; a(manufacturer, M) \leftarrow need(X)\ and\ sells(X, M)\ then \\
\quad a(n\_cus(X, M, []), C) \\
\\
a(n\_cus(X, M, Aa), C) :: \\
\quad \left(\begin{array}{l} offer(A) \;\Leftarrow\; a(n\_man(X, C, \_), M)\ then \\ accept(A) \;\Rightarrow\; a(n\_man(X, C, \_), M) \leftarrow acceptable(A)\ then \\ a(n\_cus(X, M, [att(A)|Aa]), C) \end{array} \right)\ or \\
\quad \left(\begin{array}{l} ask(commit) \;\Leftarrow\; a(n\_man(X, C, \_), M)\ then \\ tell(commit(Aa)) \;\Rightarrow\; a(n\_man(X, C, \_), M)\ then \\ tell(sold(Aa)) \;\Leftarrow\; a(n\_man(X, C, \_), M) \end{array} \right) \\
\\
a(manufacturer, M) :: \\
\quad ask(buy(X)) \;\Leftarrow\; a(customer, C)\ then \\
\quad a(n\_man(X, C, As), M) \leftarrow attributes(X, As) \\
\\
a(n\_man(X, C, As), M) :: \\
\quad \left(\begin{array}{l} offer(A) \;\Rightarrow\; a(n\_cus(X, M, \_), C) \leftarrow As = [A|T]\ and\ available(A)\ then \\ accept(A) \;\Leftarrow\; a(n\_cus(X, M, \_), C)\ then \\ a(n\_man(X, C, T), M) \end{array} \right)\ or \\
\quad \left(\begin{array}{l} ask(commit) \;\Rightarrow\; a(n\_cus(X, M, \_), C) \leftarrow As = []\ then \\ tell(commit(As)) \;\Leftarrow\; a(n\_cus(X, M, \_), C)\ then \\ tell(sold(As)) \;\Rightarrow\; a(n\_cus(X, M, \_), C) \end{array} \right)
\end{array}
$$

Fig. 4. Manufacturer interaction model

In order to satisfy this interaction model we define the following example of knowledge possessed by customer, b, and manufacturer, m:

$$known(b, need(canonS500))$$
$$known(b, sells(canonS500, m))$$
$$known(b, acceptable(memory(M)) \leftarrow M \geq 32)$$
$$known(b, acceptable(price(M, P)) \leftarrow P \leq 250)$$
$$known(m, attributes(canonS500, [memory(M), price(M, P)]))$$
$$known(m, available(memory(32)))$$
$$known(m, available(memory(64)))$$
$$known(m, available(memory(128)))$$
$$known(m, available(price(M, P)) \leftarrow P = 180 + M)$$

and then the sequence of messages in Table 3 satisfies the interaction model.

Table 3. Message sequence satisfying interaction model of Figure 4

Recipient	Message	Sender
$a(manufacturer, m)$	$ask(buy(canonS500))$	$a(customer, b)$
$a(n\_cus(canonS500, m, Aa1), b)$	$offer(memory(32))$	$a(n\_man(canonS500, b, \begin{bmatrix} memory(32), \\ price(32, P) \end{bmatrix}), m)$
$a(n\_man(canonS500, b, As1), m)$	$accept(memory(32))$	$a(n\_cus(canonS500, m, []), b)$
$a(n\_cus(canonS500, m, Aa2), b)$	$offer(price(32, 212))$	$a(n\_man(canonS500, b, [price(32, 212)]), m)$
$a(n\_man(canonS500, b, As2), m)$	$accept(price(32, 212))$	$a(n\_cus(canonS500, m, [att(memory(32))]), b)$
$a(n\_cus(canonS500, m, Aa3), b)$	$ask(commit)$	$a(n\_man(canonS500, b, []), m)$
$a(n\_man(canonS500, b, As3), m)$	$tell(commit(\begin{bmatrix} att(price(32, 212)), \\ att(memory(32)) \end{bmatrix}))$	$a(n\_cus(canonS500, m, \begin{bmatrix} att(price(32, 212)), \\ att(memory(32)) \end{bmatrix}), b)$

Note the duality in our understanding of the interaction models we have described in this section. The interaction models of figures 2, 3 and 4 are programs because they use the data structures and recursive computation of traditional (logic) programming languages. They also are distributed process descriptions because their purpose is to constrain the sequences of messages passed between peers and the clauses of interaction models constrain processes on (possibly) different physical machines.

2.3 Executing Interaction Models

LCC is a specification language but it is also executable and, as is normal for declarative languages, it admits many different models of computation. Our choice of computation method, however, has important engineering implications. To demonstrate this, consider the following three computation methods:

Interaction model run on a single peer: With this method there is no distribution of the model to peers. Instead, the model is run on a single peer (acting as a server). This is the style of execution often used with, for example, executable business process modelling languages such as BPEL4WS. It is not peer to peer because it is rigidly centralised but we have used LCC in this way to coordinate systems composed from traditional Web services offering only WSDL interfaces ([57]).

Interaction model clauses distributed across peers: Each clause in an LCC interaction model is independent of the others so as peers assume different roles in an interaction they can choose the appropriate clause from the model and run with it. Synchronisation is through message passing only, so clauses can be chosen and used by peers independently as long as there is a mechanism for knowing from which interaction model each clause has been derived. This is a peer to peer solution because all peers have the same clause interpretation abilities. It allows the interaction model to be distributed across peers but, since it is distributed by clause, it is not always possible to reconstruct the global state of the interaction model (as we can when confined to a single peer), since this would mean synchronising the union of its distributed clauses. Reconstructing the global state is not necessary for many applications but, where it is, there is another peer to peer option.

Interaction model transported with messages: In this style of deployment we distribute the interaction model clauses as above but each peer that receives a message containing an interaction model, having selected and used an appropriate clause, replaces it with the interaction model and transmits it with the message to the next peer in the interaction. This keeps the current state of the interaction together but assumes a linear style of interaction in which exactly one message is in transit at any instant - in other words the interaction consists of a chain of messages totally ordered over time. Many common service interactions are of this form, or can be constructed from chains of this form.

In the remainder of this paper we shall adopt the linear model of computation in which global state is transmitted with messages, because this is simpler to discuss. Many of the concepts we introduce also apply to non-linear interactions without global state.

Figure 5 describes informally the main components of interaction between peers. Later, Figure 6 gives a precise definition, using a linear computation model, that is consistent with the informal description. Ellipses in Figure 5 are processes; rectangles are data; and arcs denote the main inputs and outputs of processes. The large circle in the diagram encloses the components effecting local state of the peer with respect to the interaction, which interacts with the internal state of the peer via constraints specified in the interaction model. The only means of peer communication is by message passing and we assume a mechanism (specific to the message passing infrastructure) for decoding from any message appropriate information describing the interaction model associated with that message (see Section 2.4 for an example of this sort of mechanism). To know its obligations within the interaction the peer must identify the role it is to perform and (by choosing the appropriate clause) find the current model for that role. It then must attempt to discharge any obligations set by that model, which it does by identifying those constraints the model places at the current time and, if possible, satisfying them. In the process it may accept messages sent to it by other peers and send messages out to peers. Each message sent out must be routed to an appropriate peer, and the choice of recipient peer may be determined by the sender directly (if our infrastructure requires strictly point to point communication between peers) or entrusted to a message router (if a routing infrastructure such as JXTA is used).

The overview of Figure 5 is high level, so it makes no commitment to how messages are structured or how the obligations of interactions are discharged. To make these

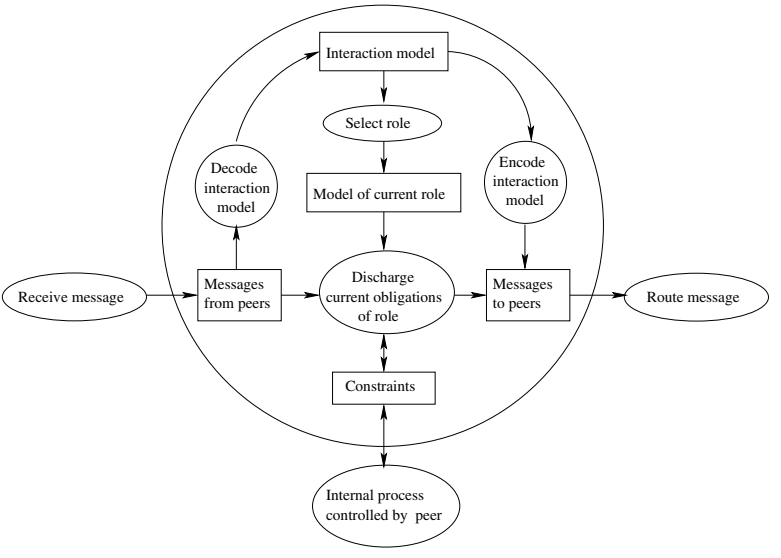


Fig. 5. Conceptual model of local state change of a peer

commitments we must be more specific about computation, which is the topic of our next section.

2.4 A Basic, Linear Computation Method for Interaction Models

To be precise in our analysis of the interaction between peers we introduce, in Figure 6, a formal definition of the linear computation introduced in the previous section. Although this assumes a shared interaction model (transmitted via messages), each transition is performed by a specific peer using only the part of the interaction visible to that peer. A sequence of transitions is initiated by a single peer with some goal to achieve and the sequence terminates successfully only when that peer can infer from its local knowledge of the interaction that it has obtained that goal.

Expressions 1 and 2 in Figure 6 generate transition sequences. Expression 3 in Figure 6 describes the general form of any such sequence. Following through expression 3 for a peer $p1$ attempting to establish goal G_{p1}, it begins with the selection by $p1$ of an interaction model ($\Omega \overset{p1}{\ni} \mathcal{S}_1$); then selection of the local state pertaining to $p1$ from the shared model ($\mathcal{S}_1 \overset{s}{\supseteq} \mathcal{S}_{p1}$); then a transition of the local state for $p1$ according to the model ($\mathcal{S}_{p1} \xrightarrow{M_1,\mathcal{S}_1,M_2} \mathcal{S}'_{p1}$); then merging of the ensuing local state with the shared state to produce a new shared interaction state ($\mathcal{S}'_{p1} \overset{s}{\cup} \mathcal{S}_1 = \mathcal{S}_2$); then repeating the transitions until reaching a final state in which the required goal can be derived using $p1$'s local knowledge ($k_{p1}(\mathcal{S}_f) \vdash G_{p1}$).

$$\sigma(p, G_p) \quad \leftrightarrow \quad \Omega \overset{p}{\ni} \mathcal{S} \ \wedge \ i(\mathcal{S}, \phi, \mathcal{S}_f) \ \wedge \ k_p(\mathcal{S}_f) \vdash G_p \tag{1}$$

$$i(\mathcal{S}, M_i, \mathcal{S}_f) \quad \leftrightarrow \quad \mathcal{S} = \mathcal{S}_f \ \vee \ \begin{pmatrix} \mathcal{S} \overset{s}{\supseteq} \mathcal{S}_p \wedge \\ \mathcal{S}_p \xrightarrow{M_i, \mathcal{S}, M_n} \mathcal{S}'_p \wedge \\ \mathcal{S}'_p \overset{s}{\cup} \mathcal{S} = \mathcal{S}' \wedge \\ i(\mathcal{S}', M_n, \mathcal{S}_f) \end{pmatrix} \tag{2}$$

where:

- p is a unique identifier for a peer.
- G_p is a goal that peer P wants to achieve.
- $\mathcal{S}$, is a state of interaction which contains the interaction model used to coordinate peers; the knowledge shared by peers participating in the interaction; and a description of their current progress in pursuing the interaction. Ω is the set of all the available initial interaction states. $\Omega \overset{p}{\ni} \mathcal{S}$ selects an initial interaction state, $\mathcal{S}$, for peer P from Ω.
- M is the current set of messages sent by peers. The empty set of messages is ϕ.
- $\sigma(P, G_p)$ is true when goal G_p is attained by peer P.
- $i(\mathcal{S}, M_1, \mathcal{S}_f)$ is true when a sequence of interactions allows state $\mathcal{S}_f$ to be derived from $\mathcal{S}$ given an initial set of messages M_1.
- $k_p(\mathcal{S})$ is a function giving the knowledge visible to peer P contained in state $\mathcal{S}$.
- $\mathcal{S} \overset{s}{\supseteq} \mathcal{S}_p$ selects the state, $\mathcal{S}_p$, pertaining specifically to peer P from the interaction state $\mathcal{S}$.
- $\mathcal{S}_p \xrightarrow{M_i, \mathcal{S}, M_n} \mathcal{S}'_p$ is a transition of the state of peer P to a new state, $\mathcal{S}'_p$, given the current set of inter-peer messages, M_i, and producing the new set of messages M_n.
- $\mathcal{S}_p \overset{s}{\cup} \mathcal{S}$ is a function that merges the state, $\mathcal{S}_p$, specific to peer P with interaction state $\mathcal{S}$ (replacing any earlier interaction state for peer P).

Every successful, terminating interaction satisfying $\sigma(p1, G_{p1})$ can then be described by the following sequence of relations (obtained by expanding the 'i' relation within expression 1 using expression 2):

$$\Omega \overset{p1}{\ni} \mathcal{S}_1 \overset{s}{\supseteq} \mathcal{S}_{p1} \xrightarrow{M_1, \mathcal{S}_1, M_2} \mathcal{S}'_{p1} \overset{s}{\cup} \mathcal{S}_1 = \mathcal{S}_2 \overset{s}{\supseteq} \mathcal{S}_{p2} \xrightarrow{M_2, \mathcal{S}_2, M_3} \mathcal{S}'_{p2} \overset{s}{\cup} \mathcal{S}_2 = \mathcal{S}_3 \ldots k_{p1}(\mathcal{S}_f) \vdash G_{p1} \tag{3}$$

Fig. 6. Formal model of linearised peer interaction

Having defined, in Figure 6, a formal model of interaction, we describe in Sections 2.5 to 2.8 how this connects to the LCC language that we introduces in Section 2.1. The operations that our language must support in order to conform to Figure 6 are given as part of each section heading.

2.5 Initial Interaction State: $\Omega \overset{p}{\ni} \mathcal{S}$

For a peer, p, to initiate an interaction it must select an appropriate initial state, S, from the set of possible such initial states, Ω. In Section 2.1 we have given a way to describe S, based on interaction models. This, however, allows infinitely many possible elements

of Ω to be constructed in theory (through manual coding, synthesis, *etc*. In practise, these interaction models are (like traditional programs) built or borrowed for specific tasks of use to p, so Ω might come from a local library. In an open system, where knowing of the existence of helpful interaction models is an issue, then the contents of Ω may not fully be known to p and mechanisms of discovery are required, as we discuss in Section 4. Automated synthesis of some elements of Ω is discussed in Section 6.

Given some way of (partially) populating Ω, there remains the issue for p of choosing which initial state to use. This is determined by some choice made by p based on the interaction model, $\mathcal{P}_g$, and shared knowledge, K, components of the initial states (recall that in LCC the initial state is a term of the form $m(\phi, \mathcal{P}_g, K)$). We denote this choice as $c(p, \mathcal{P}_g, K)$ in the expression below but do not define the mechanism by which that choice is made, since it varies according to application - anything from a fully automated choice to a decision made by a human operator.

$$\Omega \overset{p}{\ni} \mathcal{S} \quad \leftrightarrow \quad \mathcal{S} \in \Omega \ \wedge S = m(\phi, \mathcal{P}_g, K) \ \wedge \ c(p, \mathcal{P}_g, K) \tag{4}$$

2.6 State Selection by a Peer: $\mathcal{S} \overset{s}{\supseteq} \mathcal{S}_p$

Given that in LCC the state of interaction always is expressed as a term of the form $m(\mathcal{P}_s, \mathcal{P}_g, K)$, the selection of the current state for a peer, p, simply requires the selection of the appropriate clause, $a(R, p) :: D$, defining (in D) the interaction stare for p when performing role R.

$$\mathcal{S} \overset{s}{\supseteq} \mathcal{S}_p \quad \leftrightarrow \quad \exists R, D.(\mathcal{S}_p \in \mathcal{S} \ \wedge \ \mathcal{S}_p = a(R, p) :: D) \tag{5}$$

2.7 State Transition by a Peer: $\mathcal{S}_p \xrightarrow{M_i, \mathcal{S}, M_n} \mathcal{S}'_p$

Recall that, from Section 2.6 we can know the state of a specific role in the interaction by selecting the appropriate clause. This clause gives us $\mathcal{S}_p$ and we now explain how to advance the state associated with this role to the new version of that clause, $\mathcal{S}'_p$, given an input message set, M_i, and producing a new message set, M_n, which contains those messages from M_i that have not been processed plus additional messages added by the state transition. Since we shall need a sequence of transitions to the clause for $\mathcal{S}_p$ we use C_i to denote the start of that sequence and C_j the end. The rewrite rules of Figure 7 are applied to give the transition sequence of expression 6.

$$\begin{aligned} C_i \xrightarrow{M_i, \mathcal{S}, M_n} C_j \ \leftrightarrow \ & \exists R, D.(C_i = a(R, p) :: D) \wedge \\ & \left(\begin{array}{ll} C_i & \xrightarrow{R_i, M_i, M_{i+1}, \mathcal{S}, O_i} \quad C_{i+1} \wedge \\ C_{i+1} & \xrightarrow{R_i, M_{i+1}, M_{i+2}, \mathcal{S}, O_{i+1}} \quad C_{i+2} \wedge \\ \cdots & \\ C_{j-1} & \xrightarrow{R_i, M_{j-1}, M_j, \mathcal{S}, O_j} \quad C_j \end{array} \right) \wedge \\ & M_n = M_j \cup O_j \end{aligned} \tag{6}$$

$$
\begin{array}{ll}
R :: B \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} A :: E & \text{if } B \xrightarrow{R,M_i,M_o,\mathcal{S},O} E \\
A_1 \; or \; A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E & \text{if } \neg closed(A_2) \wedge A_1 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E \\
A_1 \; or \; A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E & \text{if } \neg closed(A_1) \wedge A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E \\
A_1 \; then \; A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E \; then \; A_2 & \text{if } A_1 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E \\
A_1 \; then \; A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} A_1 \; then \; E & \text{if } closed(A_1) \wedge A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O} E \\
C \leftarrow M \Leftarrow A \xrightarrow{R_i,M_i,M_i-\{m(R_i,M \Leftarrow A)\},\mathcal{S},\emptyset} c(M \Leftarrow A) & \text{if } m(R_i, M \Leftarrow A) \in M_i \wedge satisfy(C) \\
M \Rightarrow A \leftarrow C \xrightarrow{R_i,M_i,M_o,\mathcal{S},\{m(R_i,M \Rightarrow A)\}} c(M \Rightarrow A) & \text{if } satisfied(\mathcal{S}, C) \\
null \leftarrow C \xrightarrow{R_i,M_i,M_o,\mathcal{S},\emptyset} c(null) & \text{if } satisfied(\mathcal{S}, C) \\
a(R,I) \leftarrow C \xrightarrow{R_i,M_i,M_o,\mathcal{S},\emptyset} a(R,I) :: B & \text{if } clause(\mathcal{S}, a(R,I) :: B) \wedge satisfied(\mathcal{S}, C)
\end{array}
$$

An interaction model term is decided to be closed as follows:

$$
\begin{array}{l}
closed(c(X)) \\
closed(A \; then \; B) \leftarrow closed(A) \wedge closed(B) \\
closed(X :: D) \leftarrow closed(D)
\end{array} \tag{7}
$$

$satisfied(\mathcal{S}, C)$ is true if constraint C is satisfiable given the peer's current state of knowledge. $clause(\mathcal{S}, X)$ is true if clause X appears in the interaction model $\mathcal{S}$, as defined in Figure 1.

Fig. 7. Rewrite rules for expansion of an interaction model clause

2.8 Merging Interaction State: $\mathcal{S}_p \overset{s}{\cup} \mathcal{S} = \mathcal{S}'$

The interaction state, $\mathcal{S}$, is a term of the form $m(\mathcal{P}_s, \mathcal{P}_g, K)$ and the state relevant to an individual peer, $\mathcal{S}_p$, always is a LCC clause of the form $a(R, p) :: D$. Merging $\mathcal{S}_p$ with $\mathcal{S}$ therefore is done simply by replacing in $\mathcal{S}$ the (now obsolete) clause in which p plays role R with its extended version $\mathcal{S}_p$.

$$
(a(R,p) :: D) \overset{s}{\cup} \mathcal{S} \; = \; (\mathcal{S} \overset{s}{-} \{a(R,p) :: D')\}) \cup \{a(R,p) :: D)\} \tag{8}
$$

2.9 Interaction-Specific Knowledge: $k_p(\mathcal{S}) \vdash G_p$

Shared knowledge in LCC is maintained in the set of axioms, K, in the interaction state $m(\mathcal{P}_s, \mathcal{P}_g, K)$ so a peer's goal, G_p, can be satisfied if it is satisfiable from K or through the peer's own internal satisfiability mechanisms. This corresponds to the $satisfied$ relation introduced with the rewrite rules of Figure 7.

$$
k_p(\mathcal{S}) \vdash G_p \; \leftrightarrow \; satisfied(\mathcal{S}, G_p) \tag{9}
$$

This completes our operational specification for LCC combined with the style of linear deployment given in Figure 6. In an ideal world (in which all peers were aware of each other; conformed to the same ontology and cooperated perfectly to obtain desired interactions) we would now simply deploy this system by implementing it for an appropriate message passing infrastructure. The Internet, however, presents many obstacles to so doing. In the remainder of this paper we consider how some of these obstacles may be overcome. An important point throughout is that, by anchoring our solutions in a peer to peer architecture with a strong notion of modelling of interaction, it is technically possible to tackle these obstacles using well known methods that do not necessarily require huge investments up-front from the Web community.

3 Dynamic Ontology Matching

We defined, in Section 2.4, a mechanism by which a peer may make a state transition, $\mathcal{S}_p \xrightarrow{M_i, \mathcal{S}, M_n} \mathcal{S}'_p$ but we assumed when doing this that the terminology used in the message set, M_i, is consistent with the terminology actually used by the host peer, p. In an open system (where there is no restriction on the terminology used by each peer) this need not be true. When a message is created concepts in the sender's representation of the domain are mapped to the terms that compose the message, conforming to the syntax of that language. Then the receiver must map the terms in the message to the concepts in its own representation, helped by the syntax rules that structure the message. If the terms are mapped to equivalent concepts by the sender and by the receiver peers, then the understanding is correct. A misunderstanding happens if a term is mapped to different concepts by the sender and the receiver, while the interaction may fail spuriously if the receiver does not succeed in mapping a term that should correspond.

To avoid such misunderstandings we have two means of control: adapt the peer's state, $\mathcal{S}_p$ or map between terms in $\mathcal{S}_p$ and M_i. Mappings are normally viewed as a more modular form of control because they allow the ontological alignment of interaction to remain distinct from whatever inference is performed locally by a peer. Indeed, much of traditional ontology mapping is supposed to be done on ontology representations independent of interactions and prior to those ontologies being used [19, 28]. The problem with this *a priori* approach is that mapping cannot be determined separately from interaction state unless it is possible to predict all the states of all the peers in all potential interactions. This is demonstrated by examining our example interaction models in Figures 2 to 4, all of which contain constraints that an appropriate peer must satisfy in order to change its state in the interaction. For example, the peer adopting the buyer role in Figure 2 must satisfy the constraint $need(X)\ and\ shop(S)$ in order to send the first message, $ask(X)$ to shopkeeper S. The identity of X is determined by the current state of the buyer but the ontology used has to match that of the shopkeeper. We can only judge whether a mapping of terms was needed for X if we know the buyer's and seller's constraint solving choices, which normally are part of the private, internal state of each peer. Therefore, we cannot expect reliable ontological mapping in open systems (in which peer state is not known and interaction models are not fixed) without some form of dynamic mapping to keep ontology alignment on track in situations that were not predicted prior to interaction.

It is straightforward to insert an ontology mapping step into our conceptual model of interaction from Section 2.4 by adapting expressions 1 and 2 to give expressions 10 and 11 respectively. Expression 10 is obtained by defining interaction models in the set Ω with an accompanying set, O, of ontological constraints (which may be empty). These constraints are carried along with S into the interaction, producing as a consequence some final set of constraints, O_f. Expression 11 applies state transitions for each peer, $\mathcal{S}_p$, but requires that a mapping relation, $map(M_i, O_i, \mathcal{S}_p, M_i', O_n)$, applies between the current message set, M_i, and the message set, M_i', used in applying the appropriate state transition. This mapping can adapt the ontological constraints from the current set, O_i, to the new set, O_n.

$$\sigma(p, G_p) \quad \leftrightarrow \quad \Omega \overset{p}{\ni} \langle \mathcal{S}, O \rangle \ \wedge \ i(\mathcal{S}, \phi, O, \mathcal{S}_f, O_f) \ \wedge \ k_p(\mathcal{S}_f) \vdash G_p \tag{10}$$

$$i(\mathcal{S}, M_i, O_i, \mathcal{S}_f, O_f) \quad \leftrightarrow \quad \mathcal{S} = \mathcal{S}_f \ \vee \ \begin{pmatrix} \mathcal{S} \overset{s}{\supseteq} \mathcal{S}_p \wedge \\ map(M_i, O_i, \mathcal{S}_p, M_i', O_n) \wedge \\ \mathcal{S}_p \xrightarrow{M_i', \mathcal{S}, M_n} \mathcal{S}_p' \wedge \\ \mathcal{S}_p' \overset{s}{\cup} \mathcal{S} = \mathcal{S}' \wedge \\ i(\mathcal{S}', M_n, O_n, \mathcal{S}_f, O_f) \end{pmatrix} \tag{11}$$

The purpose of the map relation is to define a set of axioms that enable the ontology used in the messages, M_i, to connect to the ontology used in $\mathcal{S}_p$. For example, in Section 2.2 the shopkeeper, s, using the interaction model of Figure 2 must receive from the buyer, b, a message of the form $ask(X)$. Suppose that the message actually sent by b was in fact $require(canonS500)$ because b used a different ontology. We then want the map relation at to add information sufficient for S to conclude $require(canonS500) \rightarrow ask(canonS500)$. Although this might seem an elementary example it introduces a number of key points:

- There is no need, as far as a specific interaction is concerned, to provide a mapping any more general than for a specific object. We do not care whether $require(X) \rightarrow ask(X)$ holds for any object other than $canonS500$ or for any peer other than s because only that object matters for this part of the interaction. This form of mapping will therefore tend, on each occasion, to be much more specific than generalised mappings between ontologies.
- We would not want to insist that s accept a general $\forall X.require(X) \rightarrow ask(X)$ axiom because, in general, we don't necessarily ask for all the things we require (nor, incidentally, do we always require the things we ask about).
- The word "require" is used in many different ways in English depending on the interaction. For instance a message of the form $require(faster\_service)$ places a very different meaning (one that is not anticipated by our example interaction model) on $require$ than our earlier requirement for a specific type of camera. We could attempt to accommodate such distinctions in a generic ontology mapping but this would require intricate forms of engineering (and social consensus) to produce "the correct" disambiguation. Wordnet defines seven senses of the word "ask" and four senses of the word "require" so we we would need to consider at least all the

combinations of these senses in order to ensure disambiguation when the words interact; then follow these through into the ontological definitions. To address just one of these combinations we would have to extend the mapping definitions to have at least two different classes of $require$, each applying to different classes of object - one a form of merchandise; the other a from of functional requirement - then we would need to specify (disjoint) classes of merchandise and functional requirements. This becomes prohibitively expense when we have to, simultaneously, consider all the other word-sense combinations and we can never exclude the possibility that someone invents a valid form of interaction that breaches our "consensus".

The above are reasons why the strong notion of global sets of pre-defined ontology mappings do not appear attractive for open peer to peer systems, except in limited cases where there is a stable consensus on ontology (for example when we are federating a collection of well known databases using database schema). What, then, is the least that our interaction models require?

Returning to our example, we want the map relation when applied by seller, s, interacting with buyer, b, as follows:

$$
\begin{array}{l}
map(\ \{m(a(shopkeeper, s), require(canonS500) \ \Leftarrow \ a(buyer, b))\}, \\
\quad \{\}, \\
\quad a(shopkeeper, s) :: \\
\qquad ask(X) \ \Leftarrow \ a(buyer, B)\ then \\
\qquad price(X, P) \ \Rightarrow \ a(buyer, B) \leftarrow in\_stock(X, P)\ then \\
\qquad buy(X, P) \ \Leftarrow \ a(buyer, B)\ then \\
\qquad sold(X, P) \ \Rightarrow \ a(buyer, B) \\
\quad M_i', \\
\quad O_n)
\end{array}
$$

to give the bindings:

$$
\begin{array}{l}
M_i' = \{m(a(shopkeeper, s), ask(canonS500) \ \Leftarrow \ a(buyer, b))\} \\
O_n = \{require(canonS500)@a(buyer, b) \rightarrow ask(canonS500)@a(shopkeeper, s)\}
\end{array}
$$

where the expression $T@A$ denotes that proposition T is true for the peer A. Obtaining this result needs at least three steps of reasoning:

Detection of mapping need: It is necessary to identify in the LCC clause describing the current state of the seller, s, the transition step for which mapping may immediately be required. In the example above this is the first step in enactment of the $seller$ role, since no part of the clause has been closed. Targets for mapping are then any terms in this step which do not have matching terms in the message set. The mapping need in the example is thus for $ask(X)$.

Hypothesis of mappings: Knowing where mappings are immediately needed, the issue then is whether plausible mappings may be hypothesised. In our example (since there is only one message available to the $seller$) we need to decide whether $require(canonS500)$ should map to $ask(X)$. There is no unique, optimal algorithm for this; we present an evidence-based approach below but other methods (such as statistical methods) are possible.

Description of mappings: Mappings may, in general, be in different forms. For instance, two terms (T_1 and T_2) may map via equivalence ($T_1 \leftrightarrow T_2$ or subsumption ($T_1 \rightarrow T_2$ or $T_1 \leftarrow T_2$.) This becomes a complex issue when attempting to map two ontologies exhaustively and definitively but for the purposes of a single interaction we are content with the simplest mapping that allows the interaction to continue. The simplest hypothesis is that the term in the message received allows us to imply the term we need - in our running example $require(canonS500) \rightarrow ask(canonS500)$.

The most difficult step of the three above is hypothesising a mapping. For this we need background information upon which to judge the plausibility of our hypothesis. Several sources of such information commonly are available:

Standard word usage: We could use a reference such as Wordnet to detect similar words (such as $require$ and ask in our example). A similarity detected via Wordnet would raise our confidence in a mapping.

Past experience: If we have interactions that have been successful in the past using particular sets of mappings we may be inclined to use these again (more on this subject in Section 4). In this way earlier successes raise our confidence in a mapping while failures reduce it.

Type hierarchies for peers and interactions: Any message passing event involves three elements: the peer sending the message; the peer receiving it and the interaction model in which they are involved. Any or all of these elements may possess ontological information influencing a mapping hypothesis. A mapping might be implied by one or more of the ontologies, raising our confidence in it, or it might be inconsistent with the ontologies, lowering our confidence.

Human operators: In some circumstances it may be necessary for a human to decide whether a hypothesised mapping is valid. This choice might be informed by evidence from any or all of the sources above.

Notice that all of the sources of evidence above are unreliable: standard word usage doesn't always cover a specific circumstance; past experience may no longer apply; type hierarchies aren't necessarily complete or compatible between peers; human operators make errors. Our method does not depend on the usefulness of any of these methods, however. One can program interactions without ontological mapping but those interactions then will need perfect matches between terms (like normal programs). Ontological mapping permits more terms to match during interaction and the major programming concern is whether this "looseness" can be controlled in appropriate programming settings. Some settings require no looseness - only a perfect match will do. Other settings, in which we know in advance the ontologies used in an application but do not know which peers will be involved, allow us to define a sufficient set of mappings (O in expression 10) along with the initial interaction model. More open settings require the mapping relation (map in expression 11) to hypothesise mappings that extend the ontological "safe envelope" maintained around the interaction (relating O_i to O_n in expression 11).

4 Coalition Formation

Interaction states change via the state changes of individual peers - giving the $\mathcal{S}_{pN}$ sequence in expression 3 of Figure 6. Crucial to the success of the interaction is the choice of pN at each step. For interactions involving finite numbers of peers for which the identity is known in advance there is no coalition formation problem: the LCC interaction model simply is executed with the given peers. Notice that the examples of Figures 2, 3 and 4 are like this - we were careful to define constraints that required the peers contacting others to determine precisely which those are (for example in Figure 2 the $shop(S)$ constraint determines which shop is contacted). It is more common, however, for the choice of individual peers not to be prescribed by the interaction model - for example in the interaction model of Figure 2 what happens if the buyer doesn't know which shop might be appropriate? In open systems, a peer is often unaware of the existence and capabilities of other peers in its world. When one peer must collaborate with another to achieve some goal, a mechanism must exist to enable the discovery of other peers and their abilities.

This problem is well known, an early instance appearing in the Contract Net system [51]. It remains a crucial issue in the deployment of agent-like systems [15, 30, 59], and is resurfacing as a fundamental problem in newer endeavours like the Semantic Web and Grid computing, a recent example being the Web Services Choreography Definition Language (WSCDL) [29]. The most popular response to this problem has been to focus on specialised agents, often called "middle agents" [15, 30] or "matchmakers". The first multi-agent systems to offer explicit matchmakers were ABSI, COINS, and SHADE. These set the mould for the majority of subsequent work on matchmaking, by defining two common features: matching based on similarity measures between atomic client requests and advertised provider services; and a consideration of purely two-party interaction. OWL-S [35] and many other matchmaking architectures presume a universe where a client wishes to fulfil some goal that can be achieved by a single service provider (which may interact with other services at its own discretion). Finding collaborators for multi-party web-service interactions is discussed in [62]. Our use of performance histories is predated by a similar approach found in [63], although that only examines the case of two-party interactions.

Our aim in this section is to show how the interaction models used in LCC support matchmaking based on data from previous interactions. Suppose that in our running example we decide to automate the purchase of a list of products. We define a LCC interaction model consisting of expressions 12 and 13 below plus the original clauses from Figures 2, 3 and 4.

$$a(purchaser(L), A) :: (a(buy\_item(X), A) \leftarrow L = [H|T] \; then \; a(purchaser(T), A)) \; or \\ null \leftarrow L = [] \tag{12}$$

$$a(buy\_item(X), A) :: (a(buyer(X), A) \; or \; a(seeker(X), A) \; or \; a(customer(X), A)) \tag{13}$$

Let us further suppose that the constraints used to identify the vendors in each of the original interaction models ($shop(S)$ in Figure 2, $auction\_house(A)$ in Figure 3 and $sells(X, M)$ in Figure 4) are removed. If we now wish to buy three different

types of camera by performing the role of $a(purchaser([canonS500, olympusE300, canonEOS1]), b)$ then we have three purchases to make and it will be necessary, when performing the role of $buyer$ for each item, to choose one of the three available forms of buying model with appropriate choices of vendors. For example, we might satisfy our interaction model with the sequence of roles given below:

$a(purchaser([canonS500, olympusE300, canonEOS1]), b)$
$a(buy\_item(canonS500), b)$
$a(buyer(canonS500), b))$
Message sequence given in Table 1 when interacting with $a(shopkeeper, s)$
$a(purchaser([olympusE300, canonEOS1]), b)$
$a(buy\_item(olympusE300), b)$
$a(seeker(olympusE300), b))$
Message sequence given in Table 2 when interacting with $a(auctioneer, a)$
$a(purchaser([canonEOS1]), b)$
$a(buy\_item(canonEOS1), b)$
$a(customer(canonEOS1), b))$
Message sequence given in Table 3 when interacting with $a(manufacturer, m)$
$a(purchaser([]), b)$

in which case our interaction will have involved the set of peers $\{b, s, a, m\}$, but the sequence above is only one of many sequences we might have chosen for this set of peers. We might have chosen different roles (*e.g.* by buying the $canonS500$ at auction rather than at a shop) or different peers for the same roles (*e.g.* maybe peer m could take the role of a shopkeeper as well as or instead of its role as a manufacturer). We might have chosen to interact with only one peer in the same role each time (*e.g.* by shopping for all the cameras with peer s). The best choices of roles and peers are likely to depend on factors not expressed in the interaction model - for example, peer a might be unreliable; or peer s might give better service to some using it more frequently; or peers s and m may conspire (through separate communication channels) not to supply the same sources.

The task of a matchmaker is, by tackling problems like those above, to make the right choices of peer identifiers and roles as an interaction model unfolds over time. Given that the things that make or break an interaction often are task/domain specific in ways that cannot be analysed in detail, matchmaking algorithms may have to rely a great deal on empirical data describing successes or failures of previous interactions. This is analogous to the situation on the conventional Worldwide Web, where mass browsing behaviours continually influence the ranking of pages. Imagine, instead, that we want to rank choices of peers to involve in appropriate roles at a given stage of an interaction model's execution. Figure 8 defines a basic matchmaker capable of performing this task. A more extensive discussion (and more sophisticated matchmaking based on this principle) appears in [31].

The matchmaker of Figure 8 is an extension of the clause expansion rewrite rules of Figure 7. To each rewrite rule is added a parameter, Δ, that contains the set of peers that have been involved in closed parts of the clause, prior to the rewrite currently being applied (the predicate $closed$ of arity 2 collects the appropriate peers, following the closed part of a clause similarly to $closed$ of arity 1 which we defined in

$$R :: B \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} A :: E \quad \text{if } B \xrightarrow{R,M_i,M_o,\mathcal{S},O,\{R\}\cup\Delta} E$$

$$A_1 \text{ or } A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} E \quad \text{if } \neg closed(A_2) \wedge A_1 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} E$$

$$A_1 \text{ or } A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} E \quad \text{if } \neg closed(A_1) \wedge A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} E$$

$$A_1 \text{ then } A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} E \text{ then } A_2 \quad \text{if } A_1 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} E$$

$$A_1 \text{ then } A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta} A_1 \text{ then } E \quad \text{if } closed(A_1,\Delta_1) \wedge A_2 \xrightarrow{R_i,M_i,M_o,\mathcal{S},O,\Delta\cup\Delta_1} E$$

$$C \leftarrow M \Leftarrow A \xrightarrow{R_i,M_i,M_i-\{m(R_i,M \Leftarrow A)\},\mathcal{S},\emptyset,\Delta} c(M \Leftarrow A) \quad \text{if } m(R_i, M \Leftarrow A) \in M_i \wedge satisfy(C)$$

$$M \Rightarrow A \leftarrow C \xrightarrow{R_i,M_i,M_o,\mathcal{S},\{m(R_i,M \Rightarrow A)\},\Delta} c(M \Rightarrow A) \quad \text{if } satisfied(\mathcal{S},C) \wedge coalesce(\Delta,A)$$

$$null \leftarrow C \xrightarrow{R_i,M_i,M_o,\mathcal{S},\emptyset,\Delta} c(null) \quad \text{if } satisfied(\mathcal{S},C)$$

$$a(R,I) \leftarrow C \xrightarrow{R_i,M_i,M_o,\mathcal{S},\emptyset,\Delta} a(R,I) :: B \quad \text{if } clause(\mathcal{S}, a(R,I) :: B) \wedge satisfied(\mathcal{S},C)$$

$$\begin{array}{l} closed(c(M \Leftarrow A),\{A\}) \\ closed(c(M \Rightarrow A),\{A\}) \\ closed(A \text{ then } B, \Delta_1 \cup \Delta_2) \leftarrow closed(A,\Delta_1) \wedge closed(B,\Delta_2) \\ closed(X :: D, \Delta) \leftarrow closed(D,\Delta) \end{array} \tag{14}$$

$$coalesce(\Delta, a(R,X)) \leftarrow \neg var(X) \vee (var(X) \wedge X = sel(\{(X', P_p, P_n, N) | coalition(\Delta, a(R,X'), P_p, P_n, N)\}) \tag{15}$$

$$coalition(\Delta, A, P_p, P_n, N) \leftarrow P_p = \frac{card(\{E|(event(A,E) \wedge success(E) \wedge co(\Delta,E)\}))}{card(\{E|co(\Delta,E)\})} \tag{16}$$

$$P_n = \frac{card(\{E|(event(A,E) \wedge failure(E) \wedge co(\Delta,E)\}))}{card(\{E|co(\Delta,E)\})}$$
$$N = card(\{E|co(\Delta,E)\})$$

$$co(\Delta,E) \leftarrow (\exists A.A \in \Delta \wedge event(A,E) \wedge \neg(\exists R,X,X'.a(R,X) \in \Delta \wedge event(a(R,X'),E) \wedge X \neq X') \tag{17}$$

Where: Δ is a set of the peers ($a(R,X)$) appearing in the clause along the path of rewrites (above).
$var(X)$ is true when X is a variable.
$card(S)$ returns the cardinality of set S.
$sel(S_x)$ returns a peer identifier, X, from an element, (X, P_p, P_n), of S_x selected according to the values of P_p and P_n
See Figure 7 for definitions of other terms.

Fig. 8. A basic event-based matchmaker

Figure 7). The set, Δ, is needed in the seventh rewrite rule which deals with sending a message out from the peer. At this point the identity of the peer, A, may not be known so the predicate $coalesce(\Delta, A)$ ensures that an identifier is assigned. Expression 15 attempts to find an identifier for the peer if its identifier, X, is a variable. It does this by selecting the best option from a set of candidates, each of the form (X', P_p, P_n, N) where: X' is an identifier; P_p is the proportion of previous interactions in which X' was part of a successful coalition with at least some of the peers in Δ; P_n is the proportion of such interactions where the coalition was unsuccessful; and N is the total number of appropriate coalitions. The selection function, sel, could take different forms depending on the application but typically would attempt to maximise P_p while minimising P_n. Expression 16 generates values for P_p and P_n for each appropriate instance, A of a peer, based on cached records of interaction events. An interaction event is recorded in the form $event(a(R, X), E)$ where $a(R, X)$ records the role and identifier of the peer and E is a unique identifier for the event. For instance, in our earlier shopping example there would be a unique event identifier for the sequence of roles undertaken and an $event$ definition for each role associated with that event (so if the event identifier was $e243$ then there would be an $event(a(purchaser([canonS500, olympusE300, canonEOS1]), b), e243)$ and so on).

To demonstrate matchmaking in this event-driven style, suppose that our automated camera purchaser is following the interaction model given in expression 12 and has already performed the part of the interaction needed to buy the first camera in our list (the $canonS500$) from a peer, s, using the interaction model of Figure 2. The state of the interaction (described as a partially expanded model in the LCC style) is given in expression 18 below.

$$
\begin{array}{l}
a(purchaser([canonS500, olympusE300, canonEOS1]), b) :: \\
\quad a(buy\_item(canonS500), b) :: \\
\quad\quad a(buyer(canonS500), b)\ then \\
\quad\quad\quad ask(canonS500) \Rightarrow a(shopkeeper, s)\ then \\
\quad\quad\quad price(canonS500, 249) \Leftarrow a(shopkeeper, s)\ then \\
\quad\quad\quad buy(canonS500, 249) \Rightarrow a(shopkeeper, s)\ then \\
\quad\quad\quad sold(canonS500, 249) \Leftarrow a(shopkeeper, s) \\
\quad a(purchaser([olympusE300, canonEOS1]), b) :: \\
\quad\quad a(buy\_item(olympusE300), b) :: \\
\quad\quad\quad (a(buyer(olympusE300), b)\ or\ a(seeker(olympusE300), b)\ or \\
\quad\quad\quad\quad a(customer(olympusE300), b))
\end{array}
\tag{18}
$$

The choice at the end of expression 18 means that our $purchaser$ peer now has to choose whether to become a $buyer$ peer again or to be a $seeker$ or a $customer$. This will require it to choose a model from either Figure 2, Figure 3 or Figure 4. This, inturn, will require it to identify either a $shopkeeper$, an $auctioneer$ or a $manufacturer$ (respectively) with which to interact when following its chosen interaction model. Suppose that our purchaser has access to the following results of previous interactions:

$$\begin{array}{lll} event(a(buyer(olympusE300),b),e1) & event(a(shopkeeper,s1),e1) & failure(e1) \\ event(a(seeker(olympusE300),b),e2) & event(a(auctioneer,a1),e2) & success(e2) \\ event(a(customer(olympusE300),b),e3) & event(a(manufacturer,m1),e3) & success(e3) \\ event(a(seeker(olympusE300),b),e4) & event(a(auctioneer,a1),e4) & failure(e4) \\ event(a(buyer(canonEOS1),b),e5) & event(a(shopkeeper,s1),e5) & success(e5) \end{array} \quad (19)$$

Applying the method described in Figure 8, the contextual set of peers, Δ, from expression 18 is:

$$\begin{array}{l} \{\, a(purchaser([canonS500, olympusE300, canonEOS1]),b) \\ \;\; a(buy\_item(canonS500),b) \\ \;\; a(buyer(canonS500),b) \\ \;\; a(shopkeeper,s) \\ \;\; a(purchaser([olympusE300, canonEOS1]),b) \\ \;\; a(buy\_item(olympusE300),b) \quad \} \end{array}$$

and we can generate the following instances for $coalition(\Delta, A, P_p, P_n)$ via expression 16 of Figure 8:

$$\begin{array}{lll} \text{For} & a(buyer(olympusE300),b) & : coalition(\Delta, a(shopkeeper,s1), e1, 0, 1, 1) \\ \text{For} & a(seeker(olympusE300),b) & : coalition(\Delta, a(auctioneer,a1), 0.5, 0.5, 2) \\ \text{For} & a(customer(olympusE300),b) & : coalition(\Delta, a(manufacturer,m1), 1, 0, 1) \end{array}$$

Our selection function (sel in expression 15 of Figure 8) must then choose which of the three options above is more likely to give a successful outcome. This is not clear cut because sample sizes vary as well as the proportion of successes to failures. It is possible, however, to rate the auctioneer or the manufacturer as the most likely to succeed, given the earlier events.

5 Maintaining an Interaction Context

When many peers interact we must make sure that the knowledge they share is consistent to the extent necessary for reliable interaction. This does not of course, require consistency across the totality of knowledge possessed by the peers - only the knowledge germane to the interaction. The general problem of attaining consistent common knowledge is known to be intractable (see for example [22]) so the engineering aim is to avoid, reduce or tolerate this theoretical worst case. The interaction model used in LCC identifies the points of contact between peers' knowledge and the interaction - these are the constraints associated with messages and roles. The knowledge to which these connections are made can be from two sources:

Devolved to the appropriate peers: so that the choice of which axioms and inference procedures are used to satisfy a constraint is an issue that is private and internal to the peer concerned. In this case there is no way of knowing whether one peer's constraint solving knowledge is consistent with another peer's internal knowledge.

Retained with the LCC interaction model: so the axioms used to satisfy a constraint are visible at the same level as the interaction model and the inference procedures may also be standardised and retained with the model. In this case we can identify the knowledge relevant to the interaction and, if an appropriate consistency checking mechanism is available, we can apply it as we would to a traditional knowledge base.

In practise it is necessary to balance retention of knowledge with an interaction model (and the control that permits) against devolution to private peers (with the autonomy that allows). The way an engineer decides on this balance is, as usual, by studying the domain of application. Where it is essential that constraints are satisfied in a standardised way then axioms and inference methods are retained with the interaction model. Where it is essential that peers autonomously satisfy constraints then they must be given that responsibility. What makes this more subtle than traditional software engineering is that axioms retained by the interaction model can be used to supply knowledge hitherto unavailable to the (otherwise autonomous) peers using the model. The remainder of this section demonstrates this using a standard example.

A classic logical puzzle involves a standardised form of interaction between a group of people, each of which has an attribute which cannot be determined except by observing the behaviour of the others. This puzzle appears in different variants (including the "cheating husbands", "cheating wives" and "wise men" puzzles) but here we use the "muddy children" variant attributed to [6]. A paraphrased version of the puzzle is this:

> A number of daughters have got mud on their foreheads. No child can see the mud on her own forehead but each can see all the others' foreheads. Their father tells them that at least one of them is muddy. He then asks them, repeatedly, whether any of them (without conversing) can prove that she is muddy. Assuming these children are clever logical reasoners, what happens?

The answer is that the children who are muddy will be able to prove this is so after the father has repeated the question $n-1$ times, where n is the number of muddy children. The proof of this is inductive: for $n = 1$ the muddy child sees everyone else is clean so knows she is muddy; for $n = 2$ the first time the question is asked the muddy children can see $(n-1) = 1$ other muddy child and, from the fact that no other child answered "yes", knows that she also must be muddy so answers "yes" next time; similarly for each $n > 2$.

The important features of this example for our purposes are that: the interaction is essential to the peers acquiring the appropriate knowledge; the interaction must be synchronised (otherwise the inductive proof doesn't hold); and, once the "trick" of induction on the number of cycles of questioning is understood, the mechanism allowing each peer to decide depends only on remembering the number of cycles. Figure 9 gives a LCC interaction model that allows a group of peers to solve the muddy children puzzle. Notice that it is not our intention to unravel the semantics of such problems (as has been done in, for example, [22]). Our aim is to show how to solve this problem simply.

To demonstrate how the interaction model of Figure 9 works, suppose that we have two peers, $a1$ and $a2$, and that $a1$ knows $is\_muddy(a2)$ while $a2$ knows $is\_muddy(a1)$. This knowledge is private to the peers concerned, and neither peer knows that it is itself

$$a(coordinator(Cs, N), X) :: \left(\begin{array}{l} a(collector(Cs, Cs, N, Cs'), X) \leftarrow not(all\_known(Cs'))\ then \\ a(coordinator(Cs', N1), X) \leftarrow N1 = N + 1 \end{array}\right) or \quad (20)$$
$$null \leftarrow all\_known(Cs')$$

$$a(collector(Cs, Rs, N, Cs'), X) :: \left(\begin{array}{l} poll(N, Rs) \Rightarrow a(child, Y) \leftarrow select(k(Y, Rp), Cs, Cr)\ then \\ Cs' = \{k(Y, R)\} \cup Cr' \leftarrow reply(R) \Leftarrow a(child, Y)\ then \\ a(collector(Cr, Rs, N, Cr'), X) \end{array}\right) or \quad (21)$$
$$null \leftarrow Cs = [] \ and \ Cs' = []$$

$$a(child, X) :: poll(N, Rs) \Leftarrow a(collector(Cs, Rs, N, Cs'), X)\ then \left(\begin{array}{l} reply(muddy) \Rightarrow a(collector(Cs, Rs, N, Cs'), X) \leftarrow muddy(N, Rs)\ or \\ reply(clean) \Rightarrow a(collector(Cs, Rs, N, Cs'), X) \leftarrow clean(N, Rs)\ or \\ reply(unknown) \Rightarrow a(collector(Cs, Rs, N, Cs'), X) \leftarrow unknown(N, Rs) \end{array}\right) then\ a(child, X) \quad (22)$$

$$muddy(N, Rs) \leftarrow Nk = card(\{Y | k(Y, muddy) \in Rs\})\ and\ Nm = card(\{Y' | is\_muddy(Y')\})\ and\ Nk = 0\ and\ N \geq Nm \quad (23)$$

$$clean(N, Rs) \leftarrow Nk = card(\{Y | k(Y, muddy) \in Rs\})\ and\ Nk > 0 \quad (24)$$

$$unknown(N, Rs) \leftarrow Nk = card(\{Y | k(Y, muddy) \in Rs\})\ and\ Nm = card(\{Y' | is\_muddy(Y')\})\ and\ Nk = 0\ and\ N < Nm \quad (25)$$

Where: $all\_known(Cs)$ denotes that each element of Cs is known to be either muddy or clean.
$muddy(N, Rs)$ is true if the peer is muddy at cycle N given the previous response set Rs.
$clean(N, Rs)$ is true if the peer is clean.
$unknown(N, Rs)$ is true if the peer can't yet decide whether it is muddy or clean.
$is\_muddy(Y)$ denotes that the peer knows (another) peer Y to be muddy.
$card(S)$ returns the cardinality of set S.

Fig. 9. A muddy children LCC model

muddy. In order to work out whether they are muddy, one of the peers (let us choose $a1$) must assume the role of $coordinator(Cs, N)$ where $Cs = \{k(a1, unknown), k(a2, unknown)\}$ is the set of peers with their initial knowledge about their muddiness and $N = 1$ is the cycle number. This is analogous to the role of the father in the original puzzle and it could be performed by a third peer rather than by $a1$ or $a2$ but here we choose to let $a1$ be both coordinator and child. The coordinator role is recursive over N; on each cycle performing a round of polling for each child to find out its current answer. Each child has the obligation to reply when polled - its answer being either

$muddy$, $clean$ or $unknown$ depending on whether it can satisfy axiom 23, 24 or 25. These are examples of axioms that it makes sense to retain with the interaction model because it is critical that all peers make this calculation in the same way and it is not guaranteed that each peer would possess the appropriate knowledge. By contrast the knowledge about which peers are known by a given peer to be muddy is assumed to be private to that peer so that is not retained with the model.

Interaction models do not solve the general problem of attaining common knowledge in a distributed system - no method appears capable of that. They do, however, give a way of identifying knowledge that must be shared and provide a basis for partitioning shared and private constraint solving knowledge.

6 Making Interactions Less Brittle

One of the ways in which our method differs from traditional programming is that execution of the clauses of an interaction model can happen across different machines, therefore satisfaction of constraints associated with a particular role in an interaction model is done in ignorance of constraints imposed by other peers in the interaction. Often a peer has a choice about how it satisfies constraints (binding interaction model variables in so doing) and if it makes the wrong choice relative to other peers' constraints then the interaction as a whole may fail. In this sense, interaction models can be more "brittle" than conventional programs. Since the messages physically sent during the interaction cannot be un-sent, the only way of bringing the interaction back on track is to reason about the interaction model. It is essential, however, that such reasoning does not invalidate the interaction model - in other words it should only make a given interaction model more likely to succeed as intended, not change the declarative meaning of the model. We shall consider two ways of doing this:

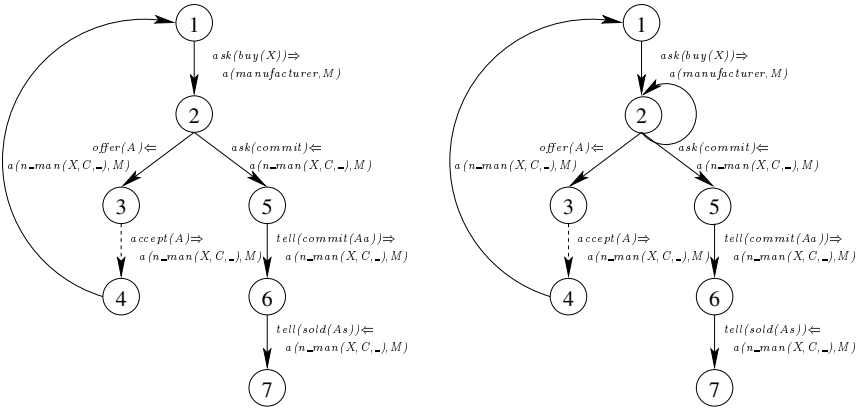
Allowing peers to set constraint ranges, rather than specific values, for variables shared in the interaction model - thus avoiding unnecessary early commitment.

Task/domain-specific adaptation of the interaction model during the process of interaction, making limited forms of "model patching" possible.

Before discussing these we study in a little more depth the programming issues raised by committed choice in LCC interaction models. Our interaction models allow choice via the *or* operators in model clauses (see Figure 1). For example, the clause:

$$a(r1, X) :: (m1 \Rightarrow a(r2, Y)\ then\ D_1)\ or\ (m2 \Rightarrow a(r2, Y))\ then\ D_2)$$

where D_1 and D_2 are some further definitions necessary for X to complete role $r1$, allows X to choose the state sequence commencing $m1 \Rightarrow a(r2, Y)$ or the sequence commencing $m2 \Rightarrow a(r2, Y)$. In some conventional logic programming languages (such as Prolog) this choice is not a commitment - if one of the options chosen fails then we may still complete the role by backtracking; rolling back the state of the clause to where it was before the choice was made, and choosing the alternative option. This



(a) Original customer's protocol

(b) Modified customer's protocol

Fig. 10. Customer's dialogue tree of the manufacturer scenario

is not possible in our situation because interacting peers may have (privately) altered their internal state, so for example $a(r2, Y)$ might have made some private internal commitments in response to message $m1$ that are not observable by $a(r1, X)$, which means that simply rolling back the state of the interaction model does not necessarily bring the interaction back to an identical state. Our choices are therefore committed choices. To make this issue concrete, let us return to our running example.

We can visualise a the possible sequences of messages sent or received when performing a role in interaction as a graph, an example of which is given in Figure 10 for the manufacturer's interaction model from Figure 4. Two peers are involved in this model, the customer and manufacturer. The dialogue tree of the customer is given in Figure 10(a). Nodes in the figure are states in the interaction (from the perspective of the customer). Solid arcs in the figure represent successful steps in the execution of the interaction model while the dashed arrow (from node 3) represents a possible failure that could occur if the customer could not accept the offer A proposed by the manufacturer. If that happens then our basic interaction model expansion mechanism, defined in Figure 7, will be unable to complete the interaction because from node 2 the only state other than the failing state (3) is node 5 but to reach it requires a different message to be received than the one actually received.

One proposed solution is to allow the peers to backtrack and try different offers. The interaction model should then be modified to allow the action of sending/receiving different offers possible. On the customer's side, the modification is represented in Figure 10(b). When the customer's model is at state 2, the customer can either receive an offer, or receive a message asking it to commit, or it can remain at state 2. This third and final option is added so that if any constraint failure occurs anywhere along the first two paths and the customer backtracks, then it will be able to receive another *offer* message.

The following is the modified customer's interaction model. A parallel modification would be required to the manufacturer's model.

$$
\begin{array}{l}
a(n\_cus(X, M, Aa), C) :: \\
\quad \left(\begin{array}{l} offer(A) \Leftarrow a(n\_man(X, C, \_), M)\ then \\ accept(A) \Rightarrow a(n\_man(X, C, \_), M) \leftarrow acceptable(A)\ then \\ a(n\_cus(X, M, [att(A)|Aa]), C) \end{array} \right) or \\
\quad \left(\begin{array}{l} ask(commit) \Leftarrow a(n\_man(X, C, \_), M)\ then \\ tell(commit(Aa)) \Rightarrow a(n\_man(X, C, \_), M) \leftarrow acceptable(A)\ then \\ tell(sold(Aa)) \Leftarrow a(n\_man(X, C, \_), M) \end{array} \right) or \\
\quad a(n\_cus(X, M, Aa), C)
\end{array}
$$

This is a pragmatic fix but it makes the interaction model more complex (hence harder to read and design) and it also changes the semantics of the model, since we have introduced an additional recursive option for the n_cus role that requires no interaction. Our only reason for changing the model in this way is to re-gain flexibility lost because of committed choice.

One way of regaining a form of backtracking for our interaction models is described in [41]. This involves an extension to the labelling used to denote closure of explored parts of the model, so that we can label parts of it as failed, then extending the model expansion rules (from Figure 7) to force (via failure messages) re-opening of previously closed sequences in the model when failure is detected. Although this gives a partial solution to the problem of backtracking, it does not address the problem that (invisible from the level of the interaction model) individual peers may have made internal commitments that prevent them "rolling back" their state to be consistent with backtracking at the model level. For this reason, it is interesting to explore (in Sections 6.1 and 6.2) ways of adding flexibility to interaction models without backtracking.

6.1 Brittleness through Variable Binding: Constraint Relaxation

The LCC language ensures coherent interaction between peers by imposing constraints relating to the message they send and receive in their chosen roles. The clauses of an interaction model are arranged so that, although the constraints on each role are independent of others, the ensemble of clauses operates to give the desired overall behaviour. For instance, the manufacturer interaction model of Figure 4 places two constraints on each attribute A in the set of attributes As: the first ($available(A)$) is a condition on the peer in the role of negotiating manufacturer sending the message $offer(A)$, and second ($acceptable(A)$) is a condition on the peer in the role of negotiating customer sending the message $accept(A)$ in reply. By (separately) satisfying $available(A)$ and $acceptable(A)$ the peers mutually constrain the attribute A.

In [23] we described how the basic clause expansion mechanism of LCC has been extended to preserve the ranges of finite-domain on variables. This allows peers to restrict rather than simply instantiate these constraints when interacting, thus allowing a less rigid interaction. For instance, applying this to our initial example of mutual finite-domain constraints in Figure 4, if the range of values permitted by the manufacturer

for A by $available(A)$ is $\{32, 64, 128\}$, while the range of values permitted by the customer for A by $acceptable(A)$ is $\{greater\ than\ 32\}$, then were we to use finite-domain constraint solver, a constraint space of $\{64, 128\}$ is obtained – a range that would be attached to the variable returned in the $accept(X)$ message.

An important aspect of the interaction model between the manufacturer and customer roles defined in Figure 4 is the message passing that communicates the attributes of the digital camera to be purchased. This dialogue can continue only as long as there exists a match between the finite-domain ranges of attribute values offered by the negotiating manufacturer with those required by the negotiating customer. To illustrate this point, consider the following example.

Assuming that the customer agreed to accept a memory size of 64, then the following statements describe the knowledge and constraints private to the manufacturer and customer respectively, concerning the price of the digital camera to be negotiated:

Manufacturer: $available(price(P)) \leftarrow P = 244$
Customer: $acceptable(price(P)) \leftarrow P \leq 250$

Upon negotiating these mutual constraints via the defined interaction model, the value for price that meets the manufacturer's offer, and also the customer's requirement will be in the range: $244 \leq price(P) \leq 250$. Depending on the peer's strategies (e.g. choosing the maximum value within the agreed range, *etc.*), the final price can be assigned to a value within this agreed range. To support this we need a means of propagating constraint ranges across the interaction.

Similarly to our construction of expressions 1 and 2, for ontology mapping in Section 3, it is straightforward to propagate range constraints through our state transitions by (in expression 26) identifying the initial set, V, of variables (each with its range constraint) in the initial state, $\mathcal{S}$ and then threading this set of variable ranges through the state transition sequence (expression 27). Prior to each transition step the relation $apply\_ranges(V_i, \mathcal{S}_p, S'_p)$ applies the range constraints, V_i, to the corresponding variables in the peer state $\mathcal{S}_p$ to give the range restricted state S'_p. After each transition step the relation $update\_ranges(\mathcal{S}''_p, V_i, V_n)$ identifies each variable in V_i that has been restricted in the new peer state $\mathcal{S}''_p$ and adds the new range restrictions to produce V_n.

$$\sigma(p, G_p) \quad \leftrightarrow \quad \Omega \overset{p}{\ni} \langle \mathcal{S}, V \rangle \ \wedge \ i(\mathcal{S}, \phi, V, \mathcal{S}_f, V_f) \ \wedge \ k_p(\mathcal{S}_f) \vdash G_p \tag{26}$$

$$i(\mathcal{S}, M_i, V_i, \mathcal{S}_f, V_f) \quad \leftrightarrow \quad \mathcal{S} = \mathcal{S}_f \ \vee \begin{pmatrix} \mathcal{S} \overset{s}{\supseteq} \mathcal{S}_p \wedge \\ apply\_ranges(V_i, \mathcal{S}_p, S'_p) \wedge \\ \mathcal{S}'_p \xrightarrow{M'_i, \mathcal{S}, M_n} \mathcal{S}''_p \wedge \\ update\_ranges(\mathcal{S}''_p, V_i, V_n) \wedge \\ \mathcal{S}''_p \overset{s}{\cup} \mathcal{S} = \mathcal{S}' \wedge \\ i(\mathcal{S}', M_n, V_n, \mathcal{S}_f, V_f) \end{pmatrix} \tag{27}$$

Simply propagating variable range constraints defends against only limited forms of brittleness, however. Suppose that, instead of allowing memory size to be in the range

$\{64, 128\}$, the customer required a memory size of 128, then the following, consequent local constraints on price would break the interaction model:

Manufacturer: $available(price(P)) \leftarrow P = 308$
Customer: $acceptable(price(P)) \leftarrow P \leq 250$

In this situation, no match is found between the customer's expected price and the one that can be offered by the manufacturer. Rather than terminating the dialogue at this stage, we might reduce brittleness of this nature by including a constraint relaxation mechanism that allows manufacturer and customer to negotiate further by relaxing the specified mutual constraint on the value of the attribute. This issue is explored more fully in [24] so we do not expand on the theme here. Constraint relaxation only is possible, however, if the peers participating in the interaction are cognitively and socially flexible to the degree they can identify and fully or partially satisfy the constraints with which they are confronted. Such peers must be able to reason about their constraints and involve other peers in this reasoning process. Interaction models (like those of LCC) provide a framework for individual peers to analyse the constraints pertinent to them and to propagate constraints across the interaction.

6.2 Brittleness through Inflexible Sequencing: Interaction Model Adaptation

When tackling the brittleness problem in Section 6.1 the issue is to ensure that a given, fixed interaction model has more chance of concluding successfully. LCC models also break, however, when their designers have not predicted an interaction that needs to occur. Consider for example the model of Figure 2 in which a buyer asks for some item; receives a price; offers to buy the item; and its sale is confirmed. This is one way of purchasing but we could imagine more complex situations - for instance when the shopkeeper does not have exactly the requested item for sale but offers an alternative product. We could, of course, write a new model to cover this eventuality but this will lead us either to write many models (and still have the task of predicting the right one to use) or to write complex models that are costly to design and hard to test for errors. One way to reduce this problem is to allow limited forms of task/domain-specific synthesis of models, with the synthesised components being used according to need in the interaction.

Automated synthesis of LCC models has many similarities to traditional synthesis of relational/functional programs and to process/plan synthesis. It is not possible here to survey this large and diverse area. Instead we use an important lesson from traditional methods: that the problem of synthesis is greatly simplified when we limit ourselves to specific tasks and domains. This narrowing of focus is natural for interaction models which are devised with a task and domain in mind. To demonstrate this we develop an example based on the interaction model of Figure 2.

In order to synthesise models similar to those of Figure 2 we need to have some specification of our functional requirements. To make synthesis straightforward, we shall describe these in a domain-specific language close to the original model (remember we

$$\begin{aligned}
&\longrightarrow t(request(X), A1 \stackrel{*}{\Rightarrow} A2)\\
t(request(X), A1 \stackrel{*}{\Rightarrow} A2) &\longrightarrow t(request(X), A1 \stackrel{*}{\Rightarrow} A2) \wedge\\
&\quad \Diamond t(describe(X, D), A1 \stackrel{*}{\Leftarrow} A2)\\
t(request(X), A1 \stackrel{*}{\Rightarrow} A2) &\longrightarrow t(request(X), A1 \stackrel{*}{\Rightarrow} A2) \wedge\\
&\quad \Diamond t(alternative(X, Y), A1 \stackrel{*}{\Leftarrow} A2)\\
t(alternative(X, Y), A1 \stackrel{*}{\Leftarrow} A2) &\longrightarrow t(alternative(X, Y), A1 \stackrel{*}{\Leftarrow} A2) \wedge\\
&\quad \Diamond t(request(Y), A1 \stackrel{*}{\Rightarrow} A2)\\
t(describe(X, D), A1 \stackrel{*}{\Leftarrow} A2) &\longrightarrow t(describe(X, D), A1 \stackrel{*}{\Leftarrow} A2) \wedge\\
&\quad \Diamond t(propose(X, P), A1 \stackrel{*}{\Rightarrow} A2)\\
t(describe(X, D), A1 \stackrel{*}{\Leftarrow} A2) &\longrightarrow t(describe(X, D), A1 \stackrel{*}{\Leftarrow} A2) \wedge\\
&\quad \Diamond t(describe(X, D'), A1 \stackrel{*}{\Leftarrow} A2)\\
t(propose(X, P), A1 \stackrel{*}{\Rightarrow} A2) &\longrightarrow t(propose(X, P), A1 \stackrel{*}{\Rightarrow} A2) \wedge\\
&\quad \Diamond t(confirm(X, P), A1 \stackrel{*}{\Leftarrow} A2)
\end{aligned}$$

$$\begin{aligned}
t(T, A1 \stackrel{*}{\Rightarrow} A2) &\longrightarrow (m(A1, T \Rightarrow A2) \wedge m(A2, T \Leftarrow A1))\\
t(T, A1 \stackrel{*}{\Leftarrow} A2) &\longrightarrow (m(A1, T \Leftarrow A2 \wedge m(A2, T \Rightarrow A1))
\end{aligned}$$

Where: $T1 \longrightarrow T2$ gives a permitted refinement of term $T1$ to term $T2$.
$\Diamond P$ denotes that expression P will be true at some future time.
$t(E, D)$ specifies a message interchange of type E between two peers, where D is either of the form $A1 \stackrel{*}{\Rightarrow} A2$, denoting a message sent from $A1$ to $A2$ or of the form $A1 \stackrel{*}{\Leftarrow} A2$, denoting a message sent from $A2$ to $A1$.

Domain terms: $alternative(X, Y)$ when Y is a product offered as a substitute for X.
$confirm(X, P)$ when the transaction is confirmed for X at price P.
$describe(X, D)$ when D describes X.
$propose(X, P)$ when price P is suggested for X.
$request(X)$ when a product of type X is requested.

Fig. 11. Rewrites for synthesis of a restricted, task-specific specification

only want to add some limited flexibility, not solve the general problem of model synthesis). In our language we have chosen to use five domain-specific terms that describe the type of interaction of message passing events in the model (listed at bottom of Figure 11). We then specify how these may be combined, starting from an initial $request$ and describing how this can be extended via rewrites that add additional message passing behaviours. The syntax of the rewrites is described in Figure 11 but the intuition is that each rewrite extends some existing segment of message passing specification with an additional behaviour that may occur in future from the existing segment (the normal $\Diamond$ modal operator is used to denote an expression that must be true at some future time). For example, the second rewrite of Figure 11 says that a request from peer $A1$ to peer $A2$ can be extended with a subsequent description of X sent by $A2$ to $A1$. The operators $\stackrel{*}{\Rightarrow}$ and $\stackrel{*}{\Leftarrow}$, used to indicate the direction of message passing between peers, can be refined into the LCC message passing operators for individual peers using the two

rules in the centre of Figure 11. Using all of these refinement rules, we can synthesise specifications of interaction model behaviour such as:

$$
\begin{array}{l}
t(request(X), A1 \stackrel{*}{\Rightarrow} A2) \wedge \\
\Diamond \left(\begin{array}{l}
t(alternative(X,Y), A1 \stackrel{*}{\Leftarrow} A2) \wedge \\
\Diamond \left(\begin{array}{l}
t(request(Y), A1 \stackrel{*}{\Rightarrow} A2) \wedge \\
\Diamond \left(\begin{array}{l}
t(describe(Y,D), A1 \stackrel{*}{\Leftarrow} A2) \wedge \\
\Diamond \left(\begin{array}{l}
t(propose(Y,P), A1 \stackrel{*}{\Rightarrow} A2) \wedge \\
\Diamond \Big(t(confirm(Y,P), A1 \stackrel{*}{\Leftarrow} A2) \Big)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
\tag{28}
$$

If we then add additional refinements from the general domain terms of Figure 11 to the more specific terms used in the interaction model of Figure 2 as follows:

$$
\begin{array}{rl}
request(X) & \longrightarrow ask(X) \\
describe(X,D) & \longrightarrow price(X,D) \\
propose(X,P) & \longrightarrow buy(X,P) \\
confirm(X,P) & \longrightarrow sold(X,P) \\
alternative(X,Y) & \longrightarrow similar\_product(X,Y)
\end{array}
$$

then we can further refine the specification of expression 28 using these and the refinements message passing from Figure 11 to give the specification:

$$
\begin{array}{l}
(m(A1, ask(X) \Rightarrow A2) \wedge m(A2, ask(X) \Leftarrow A1)) \wedge \\
\Diamond \left(\begin{array}{l}
(m(A1, similar\_product(X,Y) \Leftarrow A2) \wedge m(A2, similar\_product(X,Y) \Rightarrow A1)) \wedge \\
\Diamond \left(\begin{array}{l}
(m(A1, ask(Y) \Rightarrow A2) \wedge m(A2, ask(Y) \Leftarrow A1)) \wedge \\
\Diamond \left(\begin{array}{l}
(m(A1, price(Y,D) \Leftarrow A2) \wedge m(A2, price(Y,D) \Rightarrow A1)) \wedge \\
\Diamond \left(\begin{array}{l}
(m(A1, buy(Y,P) \Rightarrow A2) \wedge m(A2, buy(Y,P) \Leftarrow A1)) \wedge \\
\Diamond \big((m(A1, sold(Y,P) \Leftarrow A2) \wedge m(A2, sold(Y,P) \Rightarrow A1)) \big)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
\tag{29}
$$

Since the specification above describes a sequence of pairs of messages exchanged between peers $A1$ and $A2$ it is straightforward to "unzip" this sequence into the send and receive components of each exchange, providing the definition for a LCC model.

$$
\begin{array}{ll}
A1 :: & \\
 & ask(X) \Rightarrow A2 \; then \\
 & similar\_product(X,Y) \Leftarrow A2 \; then \\
 & ask(Y) \Rightarrow A2 \; then \\
 & price(Y,D) \Leftarrow A2 \; then \\
 & buy(Y,P) \Rightarrow A2 \; then \\
 & sold(Y,P) \Leftarrow A2 \\
 & \\
A2 :: & \\
 & ask(X) \Leftarrow A1 \; then \\
 & similar\_product(X,Y) \Rightarrow A1 \; then \\
 & ask(Y) \Leftarrow A1 \; then \\
 & price(Y,D) \Rightarrow A1 \; then \\
 & buy(Y,P) \Leftarrow A1 \; then \\
 & sold(Y,P) \Rightarrow A1
\end{array}
$$

To arrive at this interaction model we have made numerous assumptions in order to make the task of synthesis easy. We assumed a narrow domain so that we had a small range of specifications to deal with, thus simplifying the task of writing the message passing specification. We assumed that only two peers were involved, simplifying both the specification and its refinement. We assumed a total ordering on the messages interchanged (with no interleaving between interchanges) giving an easy translation from temporal message sequence to interaction model. We would not, of course, expect these assumptions to hold for every domain - they are merely an example of assumptions that can usefully be made to simplify engineering in one specific domain. The hope for practical application of synthesis methods is that domains amenable to similar treatments occur commonly in practise.

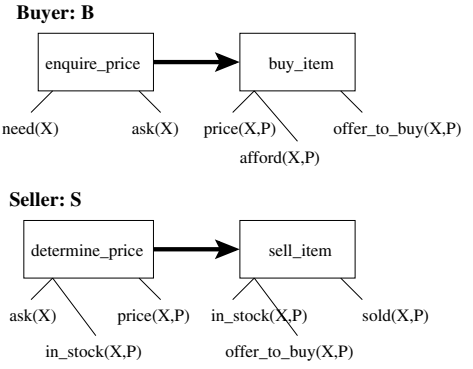
7 Satisfying Requirements on the Interaction Process

Interaction models, like traditional programs, are written in order to control computations that satisfy requirements in some domain of application. In our scenario of Section 1.1, for example, Sarah has various interaction requirements: she wants by the end to purchase a camera; before then she wants to know the price of that camera; and ideally she would like the lowest price possible. These are all functional requirements (they can be judged with respect to the computation itself) and we shall confine ourselves to this class of requirements although we recognise that, as for traditional programs, non-functional requirements also are important (for example, Sarah might want to increase her stature in her company by buying cheap cameras but our interaction model says nothing about company politics).

There are many ways in which to express requirements but we shall focus on process-based requirements modelling, in particular business process modelling, because this is (arguably) the most popular style of requirements description for Web services. There are two ways of viewing these sorts of models:

- As specifications for the temporal requirements that must be satisfied by a business process when it is performed. In this case the business process model need not be executable.
- As a structure that may be interpreted in order to perform the business process. In this case there must exist an interpretation mechanism to drive the necessary computations from the business process model.

We demonstrate these views using an elementary process modelling language. Real process modelling languages (for example BPEL4WS [3]) are more complex but similar principles apply to them (see [32] for details of our methods applied to BPEL4WS). In our elementary language we shall assume that only two (given) peers interact to implement a process and that the process is described using a set of actions of the form $activity(I, X, Cp, Cr)$, where I identifies the peer that performs the activity; X is a descriptive name for the activity; Cp is the set of axioms forming a precondition for the activity; and Cr is the set of axioms established as a consequence of the activity. The expression $process(\{I, I'\}, A)$ relates the two peer identifiers I and I' with the set of activities A. Figure 12 gives a process model which one might write for the shopping



$$
\begin{array}{l}
process(\{B,S\},\\
\quad \{activity(B, enquire\_price, \{need(X)\}, \{ask(X)\}),\\
\quad\ activity(B, buy\_item, \{price(X,P), afford(X,P)\}, \{offer\_to\_buy(X,P)\}),\\
\quad\ activity(S, determine\_price, \{ask(X), in\_stock(X,P)\}, \{price(X,P)\}),\\
\quad\ activity(S, sell\_item, \{offer\_to\_buy(X,P), in\_stock(X,P)\}, \{sold(X,P)\})\ \}\ )
\end{array}
\tag{30}
$$

Fig. 12. Elementary process model for the shopping service

model that we introduced in Figure 2 of Section 2.2. The diagram at the top of the figure depicts the process: boxes are activities with preconditions to the left of each box and postconditions to the right. The $process$ definition appears underneath, with the meaning of terms contained in activities being understood analogously to the meaning of messages and constraints in the model of Figure 2.

From the process model in Figure 12 we can infer some of the temporal constraints on interaction models enacting the business process - for instance: that for items needed that are in stock we eventually know a price; or that if an item can be afforded then it is sold.

$$
\begin{array}{r}
need(X) \ \wedge\ shop(S) \ \wedge\ in\_stock(X,P) \rightarrow \Diamond price(X,P)\\
afford(X,P) \rightarrow \Diamond sold(X,P)
\end{array}
$$

Perhaps the most obvious engineering strategy is now to test interaction models to see if they satisfy temporal requirements like those above - raising confidence that those models passing such tests are compliant with the required business process. For example, we could test whether the model of Figure 2 satisfies the temporal requirements given above. This is traditional requirements engineering re-applied so we shall not dwell on it here. There is, however, a more novel approach to ensuring compliance with business process models: write an interpreter for the models in the LCC language itself.

Figure 13 gives an interpreter for our elementary process modelling language. The key feature of this interaction model is that it takes as an argument (in $initiator(P)$ of expression 31) an entire process model, P (in our running example, this is the process

$$a(initiator(P), I) :: interpret(P, K) \Rightarrow A \leftarrow step(I, P, \phi, K)) \; then \; a(interpreter, I) \tag{31}$$

$$a(interpreter, I) :: step(I, P, K, Kn) \leftarrow interpret(P, K) \Leftarrow a(R, I') \; then \; interpret(P, Kn) \Rightarrow a(interpreter, I') \leftarrow then \; a(interpreter, I) \tag{32}$$

$$step(I, process(S, A), K, K \cup Cr) \leftarrow I \in S \wedge activity(I, X, Cp, Cr) \in A \wedge satisfy(Cp, K) \tag{33}$$

Where: P is a process definition of the form $process(S, A)$.
S is a set of peer identifiers.
A is a set of activity definitions of the form $activity(I, X, Cp, Cr)$.
I is a peer identifier.
X is the name of the activity.
Cp is the set of axioms forming a precondition for the activity.
Cr is the set of axioms established as a consequence of the activity.
$step(I, P, K, Kn)$ defines a single step in the execution of a process, P, by peer I given shared knowledge K and generating the extended knowledge set Kn.
ϕ is the empty set of initial shared knowledge.
$satisfy(Cp, K)$ is true when the conjunctive set of propositions, Cp is satisfiable from shared knowledge K.

Fig. 13. A LCC interaction model used to define an interpreter for process models in the notation of Figure 12

definition of Figure 12). The model of expressions 31 to 33 then refers to P in order to determine the messages sent between the two peers involved. This approach is similar to meta-interpretation in logic programming languages, except in this case the meta-interpretation is being done by the interaction model.

By approaching the coordination problem in this way we allow languages used in domains of application to be interpreted directly, rather than having to translate them into equivalent LCC code. This makes it easier to gain confidence (through proof, testing or inspection) that the requirements associated with the domain model are satisfied. It also makes it easier to trace problems in interaction back to the domain model.

8 Building Interaction Models More Reliably

Normally people do not want, having chosen an interaction model, to find at some subsequent point in the interaction that the goal it was supposed to achieve is not attainable, or the way in which it is obtained is inappropriate to the problem in hand. To prevent this we need some explicit description of what the interaction model is (and that we have with LCC) combined with analytical tools that check whether it has the properties we desire. Put more formally, if we have an interaction model (such as the one in Figure 2) we wish to know whether a given form of enactment (such as the one defined in expressions 1 and 2) will generate at least one sequence (for example in the form

shown in expression 3) such that each desired property holds and no sequence in which an undesirable property holds.

There are many different ways in which the correct outcome may fail to be reached. For convenience, we will overlook the possibility of the participants failing to obey the interaction model. We will also ignore issues of fault-tolerance, such as the failure of participants, network failures, or lost messages. These issues are outside the scope of the paper. Instead, we will focus on the case where the interaction model is correctly followed, but the desired outcome is not reached. In this case, the problem lies in specification of the interaction model, and not in the implementation of the external peers.

The design of an interaction model to solve a particular goal is a non-trivial task. The key difficulty lies in the nature of the peers being coordinated. The process of coordinating the external peers requires us to specify a complex concurrent system, involving the interactions of asynchronous entities. Concurrency introduces *non-determinism* into the system which gives rise to a large number of potential problems, such as synchronisation, fairness, and deadlocks. It is difficult, even for an experienced designer, to obtain a good intuition for the behaviour of a concurrent interaction model, primarily due to the large number of possible interleavings which can occur. Traditional debugging and simulation techniques cannot readily explore all of the possible behaviours of such systems, and therefore significant problems can remain undiscovered.

The prediction of the outcome in the presence of concurrency is typically accomplished by the application of formal verification techniques to the specification. In particular, the use of formal *model-checking* techniques [12] appears to hold a great deal of promise in the verification of interaction models. The appeal of model-checking is that it is an automated process, unlike theorem proving, though most model-checking is limited to finite-state systems. A model checker normally performs an exhaustive search of the state space of a system to determine if a particular property holds. Given sufficient resources, the procedure will always terminate with a yes/no answer. Model checking has been applied with considerable success in the verification of concurrent hardware systems, and it is increasingly being used as a tool for verifying concurrent software systems, including multi-agent systems [7, 9, 60].

It should be noted that the issues of outcome prediction are not simply an artifact of the use of complex interaction specifications, such as LCC models. Rather, the source of these problems is the need to coordinate complex asynchronous entities. In particular, the issues that we have highlighted occur even when our interactions are specified by simple linear plans. With asynchronous processes, the linear plans will be evaluated concurrently, and the individual plan actions will be interleaved. We will now sketch how the problems which arise in such concurrent systems can be detected by the use of model checking techniques.

We have used the SPIN model checker [25] as the basis for our verification. This model checker has been in development for many years and includes a large number of techniques for improving the efficiency of the model checking, e.g. partial-order reduction, and state-space compression. SPIN accepts design specifications in its own language PROMELA (PROcess MEta-LAnguage), and verifies correctness claims specified as Linear Temporal Logic (LTL) formula.

To perform model checking on the specification, we require an encoding of the specification into a form suitable for model checking. In [54] we define an encoding of the MAP agent interaction language, which is similar to LCC, into PROMELA. A similar technique has been defined for the AgentSpeak language [9]. In AgentSpeak, coordination is specified using the Belief-Desire-Intention (BDI) model. To illustrate the encoding process here, we sketch a translation from LCC into a state-based model in Figure 14. This figure illustrates the encoding of the main language constructs of LCC. The *e* label signifies an empty state, and *!* denotes logical negation. The encoding process is applied recursively to an LCC model. The outcome will be a state graph for each role in the model.

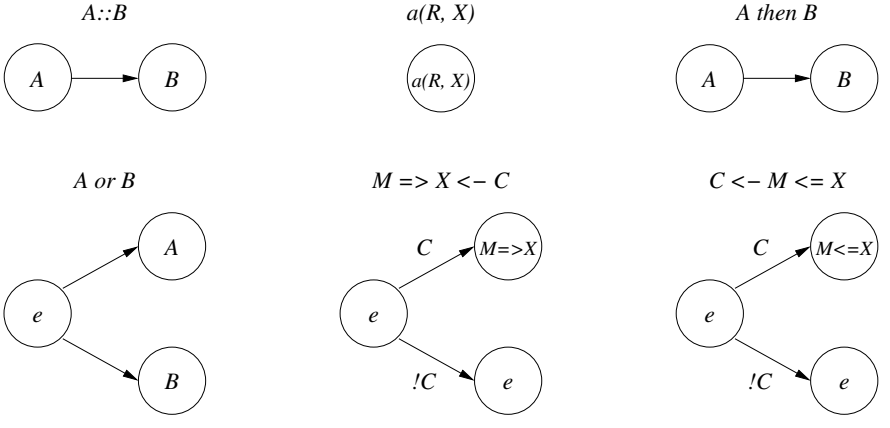


Fig. 14. State Space Encoding

The key feature of the encoding process for LCC is the treatment of the constraints. We make the observation that the purpose of a constraint is to impose a true/false decision on a model, and the purpose of the model checking process is to detect errors in the model and not in the constraints. Thus, based on these observations we can replace a constraint with a pair of states, one of which signifies that the constraint is true, and the other false. The exhaustive nature of the model checking process will mean that all possible behaviours of the interaction model will be explored. In other words, the model checker will explore all consequences for the model where the constraint was true, and all consequences where the constraint was false. Thus, we do not need to evaluate the actual constraints during the model checking process.

To illustrate the model checking process, we present a state-space encoding of the shop model in Figure 15. We have removed the redundant *e* states from the graphs, and we have abbreviated *shopkeeper* to *shop*. The state space defines all possible behaviours of the model. In this case, the model is linear, and we can examine the state space by hand. However, models that contain iteration and choice will rapidly expand the state space, and formal model checking is required. We can clearly see that the model leaves some behaviours undefined, indicated by the remaining *e* states in the graphs. These states occur as the shop model does not define what should be done when a constraint

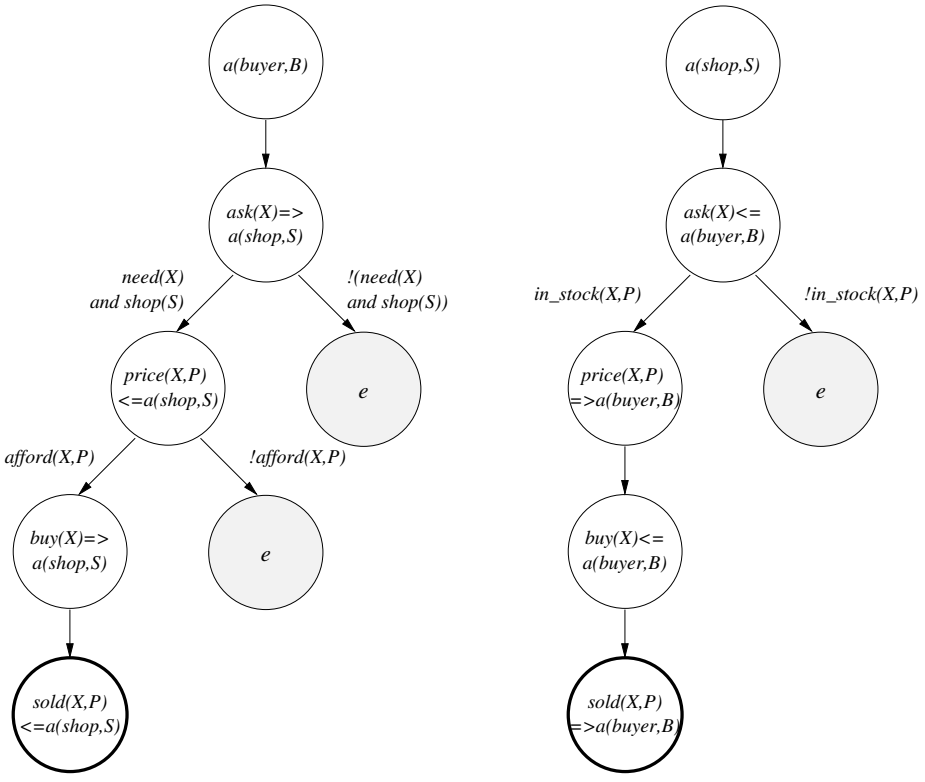


Fig. 15. Shop Model States

cannot be satisfied. For example, if the buyer cannot afford the item, the model will terminate prematurely. In many cases, we will want to avoid this possibility, and so we would amend the model with additional behaviours and recheck the state space.

Our initial model checking experiments with LCC have focused principally on the *termination* of the interaction models. This is an important consideration in the design of models, as we do not normally want to define models that cannot conclude. Non-termination can occur as a result of many different issues such as deadlocks, livelocks, infinite recursion, and message synchronisation errors. Furthermore, we may also wish to ensure that models do not simply terminate due to failure within the model, as in our shop example. The termination condition is the most straightforward to verify by model checking. Given that progress is a requirement in almost every concurrent system, the SPIN model checker automatically verifies this property by default. The termination condition states that every process eventually reaches a valid end state. This can be expressed as the following LTL formula, where `end1` is the end state for the first process, and `end2` is the end state for the second process, etc: $\Box(\ \Diamond(\texttt{end1}\ \wedge\ \texttt{end2}\ \wedge\ \texttt{end3}\ \wedge\ \cdots\ ))$. For our shop example, we may define the property $\Box(\Diamond(sold(X,P)))$ to ensure that the item is always sold to the buyer.

One of the main pragmatic issues associated with model checking is producing a state space that is sufficiently small to be checking with the available resources. Hence, it frequently is necessary to use abstraction techniques, such as we have done for the constraints, and to make simplifying assumptions. Other researchers have also considered this problem. For example, [10] proposes a program-slicing technique to improve the efficiency of the model checking process. Model checking is also restricted to finite models, and therefore we must ensure that our models are bounded. Nonetheless, our initial experiments with this approach have proved promising [56].

The encoding which we have outlined here is designed to perform *automatic* checking of LCC models. This makes the system suitable for use by non-experts who do not need to understand the model checking process. However, our approach places restrictions on the kinds of properties of the models that we can check. In particular, we cannot automatically verify properties which are specific to the domain of the model. For example, to verify that the highest bidder always wins in an auction model. Our current research is aimed at extending the range of properties that can be checked with model checking. For this, we need to retain the constraints in the model, and we must define additional formulae over these constraints. This should result in a greater ability to predict the outcome of our LCC models.

9 Implementation: The Current OpenKnowledge System

The ideas described in Sections 3 to 8 are part of the foundation for the OpenKnowledge peer to peer knowledge sharing system, for which a prototype has been built. By building the OpenKnowledge system[14, 50], we aim to demonstrate that sharing interaction models at very low cost to consumers and suppliers is possible. The novelty of the OK system lies in (1) the interaction centric approach, where interactions are published and efficiently stored in a P2P network, (2) decoupling interactions and roles from the services that execute these roles, and (3) a distributed way of finding coordinators that coordinate IMs. Within the design of the system we try to address the (unavoidable) tasks of ontology mapping, query routing, reputation management, dynamic peer recruitment etc. This system is completely distributed using P2P technology (not discussed here).

Each peer that participates in the OK system must run a piece of code that we call the *OpenKnowledge Kernel* [13] enabling the basic functionality of finding these interaction models and the code or peers to run the services implementing the roles in the IMs. We call these services *OK components (OKCs)*. The IMs together with the OKCs and/or the peers running the OKCs are efficiently stored and retrieved in a P2P network which we call the *OK Discovery service*. Additionally, due to the fact that the tasks are formally described, the OK system offers the functionality to select a peer to coordinate a task by executing the IM, selecting peers running the desired OKCs to fulfill a role and recruiting alternative peers in case of failure. The users of the OK-system can *publish* IMs, write interfaces to services, and subscribe these interfaces to play roles in the IMs. The system helps these users by providing tools to ease re-use of existing IMs or by helping connect two services via mappings in case the output of one does not match the input of the other. A reputation management mechanism is under development to help the user in selecting IMs, OKCs, Coordinators and peers running OKCs.

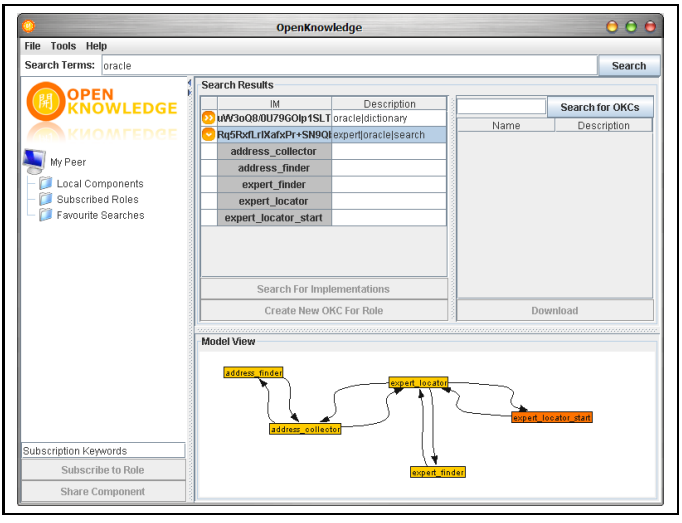


Fig. 16. Part of the Open Knowledge GUI

The first prototype provides the basic user interface shown in Figure 16) with the following functionality: (1) using the GUI, a user can wrap a piece of JAVA code into an OKC, (2) a user can share it by publishing it to the (distributed) discovery service, (3) a user search for interaction models by typing keywords (note that IMs are currently annotated by keywords), (4) a user can decide to subscribe to an OKC, meaning that it listens to function calls implementing the role for the linked IM, and finally (5) the discovery service selects a Coordinator to execute the IM when all roles are initiated by peers (the coordinator first will ask all peers with whom they want to play the interaction, after that it selects the optimal solution and if possible starts the interaction). New peers can subscribe for a running interaction, and in case of failing peers, they may be selected to take over. Figure 17 summarizes the architecture of the OK system.

To relate the possibilities of the system to sections in the system, we again use the scenario from Section 1.1.

- **Ontology matching**
  Her ontology for describing a camera purchase had to be matched to those of available services which is described in Section 3.

  The matching service implements this requirement by (1) mapping via ontologies terms from her query with the (currently term) descriptions of the interaction models and (2) mapping the capabilities of the peers with the role descriptions in the IMs.
- **Recommendations**
  Her recommendation service had to know which services might best be able to interact and enable them to do so (Sections 2 and 4).

  For this we implemented the Discovery Service which efficiently stores OKCs, IMs, peer subscriptions to OKCs and Coordinators. Together with the Mapping

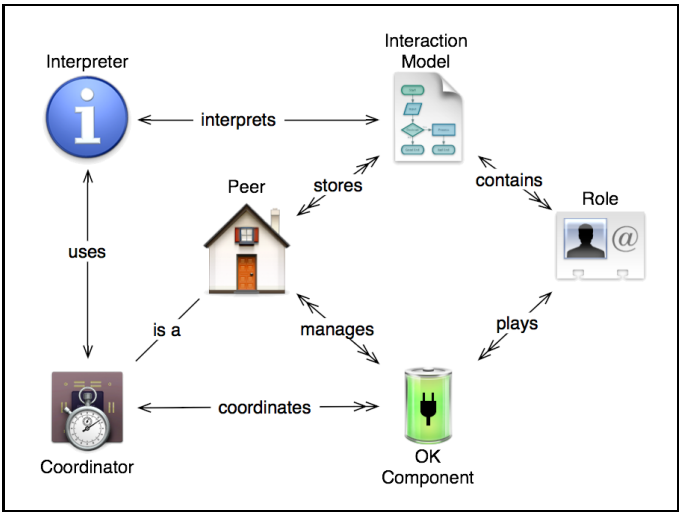


Fig. 17. Open Knowledge Architecture

Service and the Reputation strategies, the results can be ranked in order to facilitate the recommendation service. The peers that want to perform some tasks, such as buying a book or selling them, search for published interaction models for the task, and then advertise their intention of interpreting one of its roles to the discovery service for the specific task. For example, a bookseller will subscribe to perform the role of vendor in a purchase interaction, specifying that the topic is "books,novels,texts", while a peer searching a book will subscribe as customer, for a task described similarly (for example, just "book"). When all the roles are filled, the discovery service matches the peers that subscribed for the same or similar tasks (for example, "books,novels,texts" and "book").

– **Coordination**
When they interact the contextual knowledge needed to interact reliably should propagate to the appropriate services as part of the interaction (Section 5).

When the discovery service has enough subscriptions for an interaction model, it selects a coordinator to execute the IM. The coordinator sends a message to each subscribed peer containing relevant contextual information about the other peers. In this way, peers can communicate back to the coordinator their preferences about with whom they want to interact and not. Once all the roles in the IM are fulfilled by OKCs, the coordinator starts parsing the LCC in a centralised manner. When a constraint is encountered while parsing, the coordinator sends a message to the OKC fulfilling the role that must solve the constraint. This message contains the information associated to the constraint, and the interaction state. Upon receiving the constraint solving request message from the coordinator, the OKC will execute the code that solves the given constraint and return a message to the coordinator with the modified state. This process goes on repeatedly until the IM terminates.

– **Dynamic recruitment**
When the services are being coordinated then it might be necessary to reconcile their various constraints in order to avoid breaking the collaboration (Section 6).

It could happen that some peers fail or don't meet the expectations of Sarah, then either automatically or manually some other peers can be selected that (either at the beginning or during the interaction) have subscribed to the roles for which another peer needs to be found. Eventually, the discovery service acknowledges the coordinator of the running interaction with new peers. Also, during the interaction, the discovery service may be queried to return peers that subscribed to the role however with other constraints.

– **Feedback round**
Before she started, Sarah might have wanted to be reassured that the interaction conforms to the requirements of her business process (Section 7) and that her interaction was reliable (Section 8).

As stated previously, before any interaction starts, the coordinator sends a message to each individual peer with the credentials and constraints of the other peers that want to play a role in the interaction. In this way the user can check which ones suit the desired characteristics.

Summarizing, the OK-system tries to implement the methodology described in this paper. Currently we have a first version running, providing the basic functionality as described above (user interface, code wrapping, publishing of OKCs and IMs together with efficient and fully distributed storage and retrieval, coordinator selection and execution of the IMs. What is still missing and where we currently are working on, is the mapping mechanisms, reputation mechanisms and dynamic recruitment of peers.

10 Related Views of Coordination

The need for coordination is prevalent and the drive for large, complex, distributed systems maintains a pressure to improve the way in which coordination is programmed. The focus of our review below is on methods that can (without too much stretch of the imagination) be considered programming, so for example we ignore methods which have been used to specify interaction-related problems but for which no means of practical computation has been provided. By the same token we do not focus upon explorations of relevant foundational theories of interaction and coordination - for example those originating in theoretical computing science (such as [20, 38, 53, 58]), planning (such as [21]) and multi-agent systems theory (such as [52, 61]). Even with this limit there remain many related approaches so our review describes categories of system with illustrative examples selected from the many available.

10.1 Restricted Languages and Specialist Infrastructures

We have presented a generic language for coordination. By choosing a more specific language in which to express desired patterns of coordination, however, it may be possible to build tools that exploit those language restrictions in order to build a limited variety of interactions with less effort. This follows the tradition of domain specific

synthesis of traditional programs (such as [49] in ecological modelling; [34] in astrophysics) and in visual languages used to describe the structure of process and workflow systems.

Examples of this in an Internet setting are workflow editing and enactment systems to support scientific computing over large data sets - for instance the Taverna [40] and Kepler [2] workflow systems. These systems aim to support scientists (who have little knowledge of the intricacies of computing on computational grid architectures) with a high level visual language for designing experiments expressed as workflows, and then executing these in a manner that allows the sorts of provenance attribution and runtime monitoring that their communities of scientists demand. These systems are effective because they constrain the task and domain. Neither Kepler nor Taverna is intended as a programming language - they are languages for Grid service connection and provide graphical interfaces for this purpose. Although it is possible to invent a visual language that can be generically used for specifying logic-based programs (see [1] for an example) there is no evidence that a generic visualisation is easier for human communities of practise to understand than the mathematical representation from which it originated - hence to need to specialise by task and/or domain.

10.2 Coordination Via Finite State Models

Throughout this paper we have taken a view of computation that emphasises process rather than state. There is, however, a strong interaction between process and state so there are deep similarities between LCC and interaction models described by finite state machines. An example is the Islander system [16]. The framework for describing agent interactions in Islander relies upon a (finite) set of state identifiers representing the possible stages in the interaction. Agents operating within this framework must be allocated roles and may enter or leave states depending on the locutions (via message passing) that they have performed. In order to structure the description, states are grouped into scenes. An institution is then defined by a set of scenes and a set of connections between scenes with constraints determining whether agents may move across these connections. A scene is defined as a collection of the following sets: roles; state identifiers; an initial state identifier; final state identifiers; access state identifiers for each role; exit state identifiers for each role; and cardinality constraints on agents per role.

Systems like Islander have been used in what is essentially a client-server mode, where interacting peers must connect through a central server that enforces whatever synchronisation and sequencing the chosen interaction model requires. This is a very different way of using interaction models from that described using LCC but there is nothing to prevent state machine models from being used with the LCC style of deployment (or vice versa). A more fundamental difference is in the way models are described which, for LCC, is in the style of a declarative programming language.

10.3 Coordination Controlled by Local Constraints

Policy languages, such as those described in [27], are a means of specifying requirements imposed locally by a peer as conditions for interacting with other peers in different contexts. Such specifications are useful because they provide a way of determining

some of the constraints on interaction in advance of actually interacting. Re-interpreted from our viewpoint, this sets constraints locally that could interact with the global constraints set by an interaction model, affecting the likelihood that it would succeed. In this view, policies are complementary to our approach but they offer an additional opportunity (and problem) not discussed elsewhere in this paper. Policies allow the possibility of making better guesses about the appropriateness of peers to perform roles required by interaction models but, to take advantage of this, it is necessary to have rapid and automatic ways of checking the satisfiability of local constraints with respect to (partially complete) interaction models. In [42] we describe early, encouraging results in applying a form of model checking to this problem.

10.4 Coordination Via Shared Task or Service Specifications

In this paper we have assumed that entire interaction models can be shared and this is the basis for coordination between peers as well as other features, such as matchmaking, desirable in open peer-to-peer environments. An alternative view, promoted by specification languages such as WSMO [18] and automated by systems such as IRS [39], is that only task specifications should be shared between peers. These task specifications differ from interaction models in that they do not describe the course of the interaction, only the outcome that is desired by the peer posting the task. It is then necessary for tasks posted by peers to be connected to other peers capable of perhaps performing those tasks. This is facilitated by special purpose components known as mediators, that relate task specifications to problem solver specifications posted by peers that wish to perform tasks. Programming of interactions independently of peers (our focus in this paper) is not the aim of this sort of system, in which control of interaction remains local to each peer. Instead, standardisation of task and problem solver specifications (using a specification language originating in UPML [17]) is used to make tasks more flexibly shared.

OWL-S is a domain-independent OWL ontology for describing web services such that they can be reasoned about by users and middleware. With [35], a service is specified by three facets: the service *profile*, service *model* model, and service *grounding*. The profile specifies what the service does and who provides it. The functionality is described by typing inputs and outputs of the service, using concepts in some domain-specific ontology external to OWL-S, the intention being that the ontology can be used to find those services most closely resembling the requested one. For instance, a photographer looking to buy a new 35mm SLR film camera might happily accept a digital 35mm SLR (closely related in the ontology), but would be less satisfied to be given a 35mm point-and-shoot film camera, which would be ontologically more distant. The service model describes how the service operates, by means of atomic processes and a workflow-like language to combine those processes into composite ones. The grounding describes how to map from this high-level description to the low-level implementation, that is, how to invoke the service.

OWL-S is a language so does not, itself, specify any matchmaking process or infrastructure: service discovery and matchmaker querying, while enabled by design, are not defined. Others, however, have built inference systems that use OWL-S. Perhaps the best known matchmaker system is Semantic MatchMaker [45]. Here, service discovery

is achieved by inserting OWL-S descriptions into user-defined fields in the UDDI record [43]. A semantic matchmaker is responsible for interpreting this to find matches between clients and providers. More troublesome is the lack of provision in OWL-S for changing or replacing the executing process. To resolve this 'broker paradox' [44], Semantic MatchMaker introduces an 'exec' primitive to indicate that a new, brokered, process should be substituted in place of the executing one which negotiated the matchmaking. The broker paradox is not a problem in LCC, since we can dynamically alter the current interaction model, or instruct a participant to execute a new one.

10.5 Coordination as Logic Programming

LCC can be viewed as an unusual from of logic programming in which the subgoals of clauses are message passing subgoals or role changes. This, we believe, is a strength of the approach because there exists a large body of engineers trained in this form of computational logic. Other efforts, however, have produced different solutions to coordination that also draw inspitation from logic programming. The Go! language [11], for example, provides a multi-threaded environment in which agents are coordinated via a shared memory store of beliefs, desires and intentions. This is a form of agent oriented programming, using a standardised architecture, and therefore local agent beliefs (rather than interaction models as in LCC) are the anchor point for coordination. Perhaps closer to LCC is the work being done on modelling multi-agent coordination using logic programs, for example in [4] where the Event Calculus is used to specify and analyse social constraints between agents (the motivation for this being similar to that of [37]). For a logic programming view of LCC see [47].

11 Conclusion: What Is the Simplest Thing That Could Possibly Work?

In Section 3 to 7 we discussed six problems encountered when automated reasoning systems must interact in large, open, distributed systems. These problems are: ontology alignment; coalition formation; outcome prediction; maintaining shared knowledge; respecting local constraints; and relating interaction to process requirements. Our contribution is to provide a novel integrative view across all these problems by turning control of interaction into a declarative programming problem, thus bringing it within scope of the existing tools and techniques. This is not to say that any of our six problems are entirely solved in this way (they are, arguably, too general to be solved definitively) but by making models of interaction explicit, via the LCC language of Section 2.1, we are able to tackle aspects of each of the problems that are difficult to address using conventional declarative or agent oriented programming.

What is the simplest thing that could possibly work? The LCC language of Section 2 is pared down to a minimal set of concepts that we have argued are essential to describe intearctions in an executable form. Ontology matching using the interaction-specific method described in Section 3 need not involve more than each individual peer maintaining lists of matches between expressions that are useful specifically for the interactions in which it is engaged. The event based matchmaker of Section 4 relies on

counting successes and failures generated from the underlying interaction mechanism so this is no more complex from a user's point of view than page ranking in the current Web. Interaction context is maintained, in Section 5, by using parameters to roles in the interaction model. Getting this right can be a sophisticated task for the designer of an interaction model but those using interaction models need not be aware of this sophistication (just like the muddy children in our example needn't have been aware that the interaction model in which they were involved was constructed in a devious way). One simple reaction to brittleness of interaction (our topic in Section 6) is to endure it as a fact of life in the same way as we tolerate brittleness in conventional Web services. If this proves too brittle, however, our declarative style of modelling allows us to increase flexibility in some respects (via constraint handling and adaptation) without requiring individual peers to become significantly more sophisticated. All of the methods described in these sections could be made more complex as needs demand but none of them require great sophistication from individual users of the peer to peer system.

Simplicity may also be achieved through familiarity, and here the issue is whether those who wish to describe (rather than only use) interactions would adopt any language other than the one with which they are already familiar. Section 7 demonstrates one way of bridging this gap by write interpreters for the community-specific languages in LCC. Writing each interpreter is complex, but done only once, while using it is then simple. Furthermore, some of the complexity of model design can be reduced by applying traditional methods of formal verification to interaction models, as we describe in section 8.

Acknowledgements

This work was supported by the UK EPSRC Advanced Knowledge Technologies Interdisciplinary Research Collaboration (GR/N15754/01) and by the EU OpenKnowledge project (FP6-027253).

References

1. Agusti, J., Puigsegur, J., Robertson, D.: A visual syntax for logic and logic programming. Journal of Visual Languages and Computing 9 (1998)
2. Altintas, I., Birnbaum, A., Baldridge, K., Sudholt, W., Miller, M., Amoreira, C., Potier, Y., Ludaescher, B.: A framework for the design and reuse of grid workflows. In: Herrero, P., Perez, M., Robles, V. (eds.) SAG 2004. LNCS, vol. 3458, pp. 120–133. Springer, Heidelberg (2005)
3. Andrews, A., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.1 (2003)
4. Artikis, A., Pitt, J., Sergot, M.: Animated specifications of computational societies. In: Castelfranchi, C., Lewis Johnson, W. (eds.) Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems, Bologna, Italy, July 15–19, 2002, pp. 1053–1061. Association for Computing Machinery (2002)
5. Barker, A., Mann, R.: Integration of multiagent systems to AstroGrid. In: Proceedings of Astronomical Data Analysis Software and Systems XV, European Space Astronomy Centre, Spain (2005)

6. Barwise, J.: Scenes and other situations. Journal of Philosophy 78(7), 369–397 (1981)
7. Benerecetti, M., Giunchiglia, F., Serafini, L.: Model checking multiagent systems. Journal of Logic and Computation 8(3), 401–423 (1998)
8. Besana, P., Robertson, D., Rovatsos, M.: Exploiting interaction contexts in p2p ontology mapping. In: 2nd International Workshop on Peer to Peer Knowledge Management, San Diego, California, USA. CEUR Workshop Proceedings, pp. 1613–1673 (July 2005); ISSN 1613-0073, online CEUR-WS.org/Vol-139/2.pdf
9. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking agentspeak. In: Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia. ACM Press, New York (2003)
10. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: State-space reduction techniques in agent verification. In: Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems, pp. 896–903. ACM Press, New York (2004)
11. Clark, K.L., McCabe, F.G.: Go! for multi-threaded deliberative agents. In: Leite, J.A., Omicini, A., Sterling, L., Torroni, P. (eds.) DALT 2003. LNCS (LNAI), vol. 2990, pp. 54–75. Springer, Heidelberg (2004)
12. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
13. de Pinninck, A.P., Dupplaw, D., Kotoulas, S., Siebes, R.: The openknowledge kernel. In: Proceedings of the IX CESSE conference, Vienna, Austria (2007)
14. de Pinninck, A.P., Dupplaw, D., Kotoulas, S., Siebes, R., Roberson, D., van Harmelen, F.: The architecture of the open-knowledge system. Technical report, Open-knowledge consortium (2006)
15. Decker, K., Sycara, K., Williamson, M.: Middle-agents for the internet. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan (1997)
16. Esteva, M., de la Cruz, D., Sierra, C.: Islander: an electronic institutions editor. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems, pp. 1045–1052 (2002)
17. Fensel, D., Benjamins, R., Motta, E., Wielinga, R.: A framework for knowledge system reuse. In: Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm, Sweden (1999)
18. Fensel, D., Bussler, C.: The web service modellign framework. Electronic commerce: Research and applications 1, 113–137 (2002)
19. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-match: an algorithm and an implementation of semantic match. In: Proceedings of the European Semantic Web Symposium, pp. 61–75 (2004)
20. Goldin, D., Smolka, S., Attie, P., Sonderegger, E.: Turing machines, transition systems and interaction. Information and Computation 194(2) (2004)
21. Grosz, B., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence 2 (1986)
22. Halpen, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM 37(3), 549–587 (1990)
23. Hassan, F., Robertson, D.: Constraint relaxation to reduce brittleness of distributed agent ppotocols. In: Proceedings of the ECAI Workshop on Coordination in Emergent Agent Societies, Valencia, Spain (2004)
24. Hassan, F., Robertson, D., Walton, C.: Addressing constraint failures in an agent interaction protocol. In: Proceedings of the 8th Pacific Rim International Workshop on Multi-Agent Systems, Kuala Lumpur, Malasia (2005)
25. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison Wesley, Reading (2003)
26. Jackson, D., Wing, J.: Lightweight formal methods. IEEE Computer (April 1996)

27. Kagal, L., Finin, T., Joshi, A.: A policy language for pervasive systems. In: Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (2003)
28. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. Knowledge Engineering Review (2003)
29. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web services choreography description language version 1.0, 2004. W3C Working Draft (October 12, 2004)
30. Klusch, M., Sycara, K.: Brokering and matchmaking for coordination of agent societies: a survey. In: Coordination of Internet agents: models, technologies, and applications, pp. 197–224. Springer, Heidelberg (2001)
31. Lambert, D., Robertson, D.: Matchmaking and brokering multi-party interactions using historical performance data. In: Fourth International Joint Conference on Autonomous Agents and Multi-agent Systems (2005)
32. Li, G., Chen-Burger, J., Robertson, D.: Mapping a business process model to a semantic web services model. In: Proceedings of the IEEE International Conference on Web Services, San Diego (2004)
33. Li, G., Robertson, D., Chen-Burger, J.: A novel approach for enacting distributed business workflow on a peer-to-peer platform. In: Proceedings of the IEEE Conference on E-Business Engineering, Beijing (2005)
34. Lowry, M., Philpot, A., Pressburger, T., Underwood, I.: A formal approach to domain-oriented software design environments. In: Proceedings of the 9th Knowledge-Based Software Engineering Conference, Monterey, California, pp. 48–57 (1994)
35. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: owl-s 1.1 (2004)
36. McGinnis, J., Robertson, D.: Realising agent dialogues with distributed protocols. In: van Eijk, R.M., Huget, M.-P., Dignum, F.P.M. (eds.) AC 2004. LNCS (LNAI), vol. 3396, pp. 106–119. Springer, Heidelberg (2005)
37. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning, pp. 482–493 (2002)
38. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, part I/II. Information and Computation 100(1), 1–77 (1992)
39. Motta, E., Domingue, J., Cabral, L., Gaspari, M.: Irs-ii: A framework and infrastructure for semantic web services. In: Proceedings of the Second International Semantic Web Conference, Florida, USA (2003)
40. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)
41. Osman, N.: Addressing Constraint Failures in Distributed Dialogue Protocols. Ph.D thesis, School of Informatics, University of Edinburgh, M.Sc Thesis (2003)
42. Osman, N., Robertson, D., Walton, C.: Run-time model checking of interaction and deontic models for multi-agent systems. In: Proceedings of the Third European Workshop on Multi-agent Systems, Brussels, Belgium (2005)
43. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Importing the semantic web. In: UDDI (2002)
44. Paolucci, M., Soudry, J., Srinivasan, N., Sycara, K.: A broker for OWL-S web services. In: Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services (2004)
45. Paulucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: Proceedings of the International Semantic Web Conference (2002)
46. Robertson, D.: A lightweight coordination calculus for agent social norms. In: Proceedings of Declarative Agent Languages and Technologies workshop at AAMAS, New York, USA (2004)

47. Robertson, D.: Multi-agent coordination as distributed logic programming. In: International Conference on Logic Programming, Sant-Malo, France (2004)
48. Robertson, D., Agusti, J.: Software Blueprints: Lightweight Uses of Logic in Conceptual Modelling. Addison Wesley/ACM Press (1999) ISBN 0201398192
49. Robertson, D., Bundy, A., Muetzelfeldt, R., Haggith, M., Uschold, M.: Eco-Logic: Logic-Based Approaches to Ecological Modelling. Logic Programming Series. MIT Press, Cambridge (1991)
50. Siebes, R., Dupplaw, D., Kotoulas, S., de Pinninck, A.P., Roberson, D., van Harmelen, F.: The functional description of the open-knowledge system. Technical report, Open-knowledge consortium (2006)
51. Smith, R.G.: The contract net protocol: high-level communication and control in a distributed problem solver. In: Distributed Artificial Intelligence, pp. 357–366. Morgan Kaufmann Publishers Inc., San Francisco (1988)
52. van der Hoek, W., Wooldridge, M.: On the logic of cooperation and propositional control. Artificial Intelligence 164(1-2) (2005)
53. van Leeuwen, J., Wiedermann, J.: A computational model of interaction in embedded systems. Technical Report UU-CS-02-2001, Dept. of Computer Science, University of Utrecht (2001)
54. Walton, C.: Model checking agent dialogues. In: Proceedings of the 2004 Workshop on Declarative Agent Languages and Technologies, New York, USA (2004)
55. Walton, C.: Model checking multi-agent web services. In: Proceedings of AAAI Spring Symposium on Semantic Web Services, California, USA (2004)
56. Walton, C.: Model checking multi-agent web services. In: Proceedings of the AAAI Spring Symposium on Semantic Web Services, Stanford, USA. AAAI, Menlo Park (2004)
57. Walton, C., Barker, A.: An agent-based e-science experiment builder. In: Proceedings of the 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid, Valencia, Spain (August 2004)
58. Wegner, P.: Why interaction is more powerful than algorithms. Communications of the ACM 40(5) (1997)
59. Wong, H., Sycara, K.: A taxonomy of middle-agents for the internet. In: Proceedings of the International Conference on Multi-agent Systems (2000)
60. Wooldridge, M., Fisher, M., Huget, M.P., Parsons, S.: Model checking multiagent systems with MABLE. In: Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy (2002)
61. Wooldridge, M., Jennings, N.R.: The cooperative problem solving process. Journal of Logic and Computation 9(4) (1999)
62. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.: Quality driven web services composition. In: Proceedings of the twelfth international conference on World Wide Web, pp. 411–421. ACM Press, New York (2003)
63. Zhang, Z., Zhang, C.: An improvement to matchmaking algorithms for middle agents. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, pp. 1340–1347. ACM Press, New York (2002)