Chapter XI
Modelling Interactions via Commitments and Expectations

Paolo Torroni
University of Bologna, Italy

Federico Chesani
University of Bologna, Italy

Pınar Yolum
Boğaziçi University, Turkey

Marco Gavanelli
University of Ferrara, Italy

Munindar P. Singh
North Carolina State University, USA

Evelina Lamma
University of Ferrara, Italy

Marco Alberti
University of Ferrara, Italy

Paola Mello
University of Bologna, Italy

ABSTRACT

Organizational models often rely on two assumptions: openness and heterogeneity. This is, for instance, the case with organizations consisting of individuals whose behaviour is unpredictable, whose internal structure is unknown, and who do not necessarily share common goals, desires, or intentions. This fact has motivated the adoption of social-based approaches to modelling interaction in organizational models. The idea of social semantics is to abstract away from the agent internals and provide a social meaning to agent message exchanges. In this chapter, we present and discuss two declarative, social semantic approaches for modelling interaction. The first one takes a state-oriented perspective, and models interaction in terms of commitments. The second one adopts a rule-oriented perspective, and models interaction in terms of logical formulae expressing expectations about agent interaction. We use a simple interaction protocol taken from the e-commerce domain to present the functioning and features of the commitment- and expectation-based approaches, and to discuss various forms of reasoning and verification that they accommodate, and how organizational modelling can benefit from them.

Copyright © 2009, IGI Global, distributing in print or electronic forms without written permission of IGI Global is prohibited.

INTRODUCTION

Organizations can be seen as sets of entities regulated by mechanisms of social order and created by more or less autonomous actors to achieve common goals. If we consider open agent societies from an organizational point of view, we can identify a number of basic elements that design methodologies for agent societies should account for. These include formalisms for the description, construction and control of normative elements such as roles, norms and social goals, and mechanisms to formalize the expected outcomes of roles and to describe *interaction* in order to *verify* the overall behaviour of the society.

Interaction is one of the main elements of multi-agent organizational models, since it is the main—if not the only—way for agents to coordinate with one another (see the chapter "Modelling dimensions for agent organizations" by Coutinho *et al.* for more information). In the literature, multi-agent communication has been the subject of a vast research activity addressing *semantic* and *engineering* aspects of multi-agent organizations. Two main approaches have emerged. In the early days of multi-agent research, a seemingly promising way to model agent interaction was largely inspired by Grice's and Searle's speech acts theory. This is now called the mentalistic approach since its focus is on the minds of the individuals participating in the interaction. Agent mental states would give motivation to message exchange, which in turn would affect the mental states of those participating in the exchange. This idea has been behind prominent Agent Communication Language (ACL) proposals such as KQML and FIPA-ACL. However, it became apparent that a semantics of agent communication based on mental states would necessarily impose significant restrictions on the architecture and operational behaviour of interacting parties, while making it difficult, at the same time, for an external observer to understand to what extent a message exchange would conform to such a semantics (Singh 1998).

Social approaches to agent communication seek to overcome these shortcomings and quickly gained large popularity. The idea of social semantics is to abstract away from the agent internals and provide a social meaning to agent message exchange. In other words, interaction is not motivated by the effect it may have on the mind of the agent, but instead on its visible effects on the agent's social environment. Besides paving the way to the development of a number of techniques aimed to make agent interaction verifiable by external observers, social semantics have proven to be a viable approach to accommodate truly open agent societies, since they do not pose restrictions of any sort on the nature and architecture of interacting parties. These are key factors that made social semantics much more widely adopted than mentalistic approaches, especially in the context of organizational models.

The second aspect we mentioned relates to multi-agent systems engineering and design. Again, social semantics of agent interaction has been successfully applied to Multi-Agent System (MAS) design, both with respect to methodologies and formal reasoning about models. Current Agent-Oriented Software Engineering methodologies—see for example Gaia (Zambonelli *et al.* 2003), MaSE (DeLoach *et al.* 2001), Prometheus (Padgham & Winikoff, 2004) and the Hermes methodology presented earlier on in this book (see chapter "Hermes: A pragmatic approach to flexible and robust agent interaction design and implementation," by Cheong & Winikoff)—include, at some design stage, modelling of actions, events, roles, normative relations and interaction protocols. It is possible to give a social semantics to these elements of a MAS by modelling them or their effect in terms of socially meaningful concepts. Modelling greatly benefits from the social semantics being declarative. A declarative, as opposed to procedural, semantics specifies what actions should be brought out in an interaction, rather than how they are brought out, and by doing so it helps to focus on the aims of the interactions and thus avoid designing unnecessarily over-constrained interaction patterns and modalities, as pointed out by Yolum & Singh (2002b).

Social semantics can be approached in two ways: by taking a state-oriented perspective, or a rule-oriented perspective. The main elements of these two approaches are, respectively, commitments and expectations. They both provide a formal, conceptually well-founded basis for modelling the interaction of agents while respecting their autonomy.

Commitments

The main motivation behind the commitment-based semantics of agent interaction, as opposed to, e.g., using AUML interaction protocol diagrams (Bauer *et al.* 2001), comes from the observation that to ensure flexibility in handling opportunities and exploiting opportunities, the agents should be able to deviate from rigid scripts. Representing the commitments that the agents have to one another and specifying constraints on their interactions in terms of commitments provides a principled basis for agent interactions. From a MAS modelling perspective, a role can be modelled by a set of commitments. For example, a seller in an online market may be understood as committing to its price quotes and a buyer may be understood as committing to paying for goods received. Commitments also serve as a natural tool to resolve design ambiguities. In general, the designers of the individual agents would be different from the designers of the roles in the multiagent system. For example, an online market would only specify the roles of buyer, seller, and broker and their associated commitments, but individual participants may, and would often prefer to, use differently implemented agents to play these roles. Finally, the formal semantics enables verification of conformance and reasoning about the MAS specifications to define core interaction patterns and build on them by reuse, refinement, and composition.

Central to the whole approach is the idea of manipulation of commitments, specifically, their creation, discharge, delegation, assignment, cancellation, and release. Commitments are stateful objects that change in time as events occur. Time and events are, therefore, essential elements of commitment conditions. The commitments present in the system and their state capture the state of the overall system, as obtained through the socially relevant actions performed thus far.

Commitments capture the social meaning of what agents have achieved so far, and can be used to judge their conformance and as a basis to reason about possible future evolutions of their interaction. An advantage of this approach is that the evolution of commitment state can be easily expressed using a finite state machine, which in turn has a direct implementation. Such an implementation can be used to create and manipulate commitments at run-time and directly verify whether the system is in a conformant state or not. Moreover, from the commitments, both designers and concerned agents can identify pending obligations and, in case of violation, the responsibilities of the debtors.

Commitments can be given various underlying semantics, depending on their intended use, such as on the modelling aspects that one wants to capture and on the types of verification that one wants to achieve. Some examples are temporal logics (linear or branching), nonmonotonic causal logic, and the event calculus, all of which have been applied to symbolically represent commitment machines.

Expectations

Social expectations have a similar motivation but a different purpose. Similar to commitments, the idea is to provide formal tools to specify and verify the good behaviour of an open MAS, based on the externally observable agent behaviour. Differently from commitments, expectations are not objects with a state, but abstract entities that model the possible events that would make the system evolve in

conformance with a set of protocols. The main idea of expectation-based semantics is that in open and dynamic MAS, a large share of information is unknown: either because it is not specified or it is private knowledge, or—especially in the case of events—because it has not (yet) been observed by anyone or is still to occur. To capture the unknown, the reasoning conducted to manipulate expectations is intrinsically hypothetical, and the social semantics of expectations is based on abductive reasoning.

To model MAS interaction, expectation-based semantics specifies the links between the observed events and the possibly observable, yet unknown, events. It does so by setting constraints on the model, called social integrity constraints (SICs). (Integrity) constraints are ubiquitous in organizational modelling theory and they are at the core of other approaches presented in this book (see, for instance, the chapter "A formal framework for modeling and analysis of organizations" by Popova & Sharpanskykh). SICs are declarative representations of invariants, which can be used to specify the set of "good" execution traces, also called histories. Expectations are a general notion, to model events of any sort, and do not necessarily refer to specific roles. Events are expected to happen or not to happen, independently of the responsibility of the agents involved in the production or avoidance of such events. There are no explicit roles of debtors and creditors. However, it is possible to represent attributes of actions of any sort, such as the sender and recipient of a message, and finite domains and CLP constraints (Jaffar & Maher, 1994) over action elements, such as the content of messages. Time is also modelled as a variable with a finite domain. Using SICs, it is possible to represent typical patterns of interaction, such as alternatives, parallel execution, sequences, mutual exclusions, and deadlines. Finally, like in commitment-based approaches, the formal semantics enables verification of conformance and reasoning on the MAS specifications, to define core interaction patterns and build on them by reuse, refinement, and composition.

Indeed, one of the main features of the expectation-based semantics is the ability to reason at run-time on the evolution of a system in terms of happening events. Such a feature enables monitoring and run-time verification of the agent behaviour's conformance to protocols and can be used to indicate possible (future or past) sequences of events that would ensure overall conformance. Although expectations—differently from commitments—are not linked to the notion of responsibility, they do have a possible reading in relation with the deontic concepts of obligation, prohibition, fulfillment, and violation. As such, they are suited to model institutional facts: an essential element of organizations which will be discussed in detail in the following parts of this book (in particular, see chapters "Specifying artificial institutions in the event calculus," by Fornara & Colombetti, and "Dynamic specifications for norm-governed computational systems," by Artikis *et al.*).

The formal semantics of SICs is given in a computational logic framework called SCIFF. Such a semantics is based on abduction: expectations are modelled as abducibles and SICs are represented as integrity constraints. The SCIFF operational semantics is defined by an abductive proof procedure. Verification is performed by the proof procedure operating directly on the specification. At every point in time, expectations define the possible evolutions of the externally visible MAS behaviour that would ensure overall system conformance to protocols. Since there can be many ways to achieve a conformant behaviour, every event that adds to the previously observed ones does not in general produce a single set of expectations, but a number of alternative sets, and it can rule out other sets by identifying inconsistencies (e.g., events that are expected both to happen and not to happen).

Commitment-based and expectation-based semantics are complementary modelling paradigms, capturing two important perspectives on social interaction, and they can both be beneficial to the design, specification, and verification of agent organizations. The objective of this chapter is thus to provide an overview of the social semantics of agent interaction using commitments and expectations and to motivate its adoption in the context of organizational modelling.

In the next section, we provide some definitions and a brief literature review, to demonstrate the topicality of social semantics in a broad sense. Then we will elaborate more on the inspiring principles that motivate the adoption of social approaches in various areas, with an emphasis on methodological and verification issues that provide further motivation to the work presented here. In the following section, we will present the NetBill protocol: a running example to use throughout the chapter. We will then review two bodies of work, dealing with social semantics in two different ways: Yolum and Singh's commitment-based semantics and the SCIFF framework for expectation-based specification and verification of agent interaction. In the concluding sections, we discuss implementation and resources related to the presented approaches and conclude by briefly describing research directions, opportunities, and new trends, and the application of social semantics to new emerging areas such as the Semantic Web services and business process management.

BACKGROUND

Traditionally, finite state machines and Petri Nets have been the main formalisms for specifying network protocol. These formalisms are advantageous mainly because they are easy to be followed by the protocol participants. These formalisms are useful in enumerating the allowed sequences of messages, but do not capture the semantics of the messages that are exchanged. Hence, these formalisms leave no opportunity for the enacting agents to reason about the protocol. Obviously, this is not acceptable in open multiagent systems, where agents are autonomous and can and will decide how they will carry out their interactions. Thus, *declarative* approaches are needed to specify the meaning of messages and to provide agents means to reason about their messages.

Another important aspect of protocols is verification. Guerin & Pitt (2002) distinguish three possible types of verification, depending on the available information about the protocol players. A first type aims to *verify that an agent will always comply.* This type of verification can be performed at design time: given a representation of the agent, by means of some proof technique (such as *model checking*), it is possible to prove that the agent will always exhibit the desired behaviour. This kind of verification requires knowledge on the agent internals or specifications. A whole body of research is devoted to agent verification; a good introduction to it is given by Bordini *et al.* (2004).

More directly related to organizational modelling are the other two types of protocol verification: *verify compliance by observation* and *verify protocol properties*. The goal of the former is to check that the behaviour being observed is compliant with some specification. It does not require any knowledge about the internals, but only the observability of the agent behaviour. Since it is based on observations, this type of verification can be performed at run-time (or, if key events are logged, after execution). This type of verification is of the uttermost importance in real-life systems, whose heterogeneity and complexity are such that protocols must allow participants adequate freedom in order to enable an effective, open organization: too many strict rules could result in over-constrained protocols of little use.

The last type of verification instead can be performed at design time, and aims to prove that some property will hold for all the interactions that correctly follow the protocol. This type of verification is again crucial in organizations defining many complex protocols, where it would be difficult (if not impossible) to manually verify that a given protocol guarantees a given property. Protocol specification languages for agent organizations should offer (or at least support) tools for expressing formal properties, and verifying them.

The formalisms that we present in this chapter mainly accommodate compliance by observation and design-time verification of interaction properties. Later in this book, Viganò & Colombetti present alternative logic-based tools to verify organizations ruled by institutions, with an emphasis on reasoning on normative constructs (see chapter "Model checking agent organizations regulated by institutions").

Running Example: The NetBill Transaction Protocol

NetBill, proposed by Cox *et al.* (1995), is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software programs, journal articles or multimedia files. The protocol rules transactions between two agents: a Merchant (*MR*), and a Customer (*CT*).

A NetBill server is used to deal with financial issues such as those related to credit card accounts of customer and merchant. In this example, we focus on the NetBill protocol version designed for non zero-priced goods, and do not consider the variants that deal with zero-priced goods. A typical protocol run is composed of three phases:

1. **Price Negotiation.** The customer requests a quote for some goods identified by a Product ID (*PrID*):
   priceRequest(PrID)
   and the merchant replies with the requested price (*Quote*):
   priceQuote(PrID, Quote)
2. **Goods Delivery.** The customer requests the goods:
   goodsRequest(PrID, Quote)
   and the merchant delivers it in an encrypted format:
   delivery(crypt(PrID, Key), Quote)
3. **Payment.** The *Customer* issues an Electronic Payment Order (EPO) to the merchant, for the amount agreed for the goods:
   payment(epo(Customer, crypt(PrID, Key), Quote))
   the merchant appends the decryption *Key* for the goods to the EPO, signs the pair and forwards it to the NetBill server:
   endorsedEPO(epo(Customer, crypt(PrID, Key), Quote), Key, MR)
   the NetBill server deals with the actual money transfer and returns the result to the merchant:
   signedResult(Customer, PrID, Price, Key)
   who will, in turn, send a receipt for the goods and the decryption key to the customer:
   receipt(PrID, Price, Key).

The Customer can withdraw from the transaction until she has issued the EPO message *payment*. The Merchant can withdraw from the transaction until she has issued the *endorsedEPO* message.

COMMITMENT-BASED SEMANTICS

Commitments are made from one agent to another agent to bring about a certain property (Castelfranchi, 1995; Singh, 1991; Singh, 1999). Commitments result from communicative actions. That is, agents create commitments and manipulate them through the protocol they follow. We can differentiate

between two types of commitments: unconditional and conditional. An unconditional commitment is denoted as *C(x, y, p)* and means that the debtor *x* commits to creditor *y* to bring about condition *p* (Singh, 1999). A conditional commitment is denoted as *CC(x, y, p, q)* and means that if the condition *p* is satisfied, then *x* will be committed to bring about condition *q*. Conditional commitments are useful when a party wants to commit only if a certain condition holds or only if the other party is also willing to make a commitment.

Let us consider some example commitment from the NetBill protocol. The unconditional commitment,

C(merchant, customer, receipt(PrID, Price, K))

means that the merchant commits to sending the receipt for the given product. The conditional commitment,

CC(merchant, customer, payment(epo(customer, crypt(PrID, K), Quote)), receipt(PrID, Price, K))

specifies that the merchant commits to sending the receipt if the customer pays the money.

Singh (1999) defines six operations on commitments. In the following, *x*, *y*, *z* denote agents, and *c*, *c'* denote commitments.

1. *create(x, c)* establishes the commitment *c*. This operation can only be performed by the debtor of the commitment.
2. *discharge(x, c)* resolves the commitment *c*. The discharge operation can only be performed by the debtor of the commitment to mean that the commitment has successfully been carried out. Discharging a commitment terminates that commitment.
3. *cancel(x, c)* cancels the commitment *c*. The cancel operation is performed by the debtor of the commitment. Usually, the cancellation of a commitment is accompanied by the creation of another commitment to compensate for the cancellation.
4. *release(y, c)* releases the debtor from the commitment *c*. It can be performed by the creditor to mean that the debtor is no longer obliged to carry out his commitment.
5. *assign(y, z, c)* assigns a new agent as the creditor of the commitment. More specifically, the creditor of the commitment *c* may assign a new creditor *z* to enable it to benefit from the commitment. Operationally, commitment *c* is eliminated and a new commitment *c'* is created for which *z* is the creditor.
6. *delegate(x, z, c)* is performed by the debtor of commitment *c* to replace itself with another agent *z* so that *z* becomes responsible to carry out the commitment. Similar to the previous operation, commitment *c* is eliminated, and a new commitment *c'* is created in which *z* is the debtor.

The creation and manipulation of commitments is handled via the above operations. In addition to these operations, reasoning rules on commitments capture the operational semantics of our approach. Some of these operations require additional domain knowledge to reason about. For example, canceling a commitment may be constrained differently in each domain. Or, delegating a commitment to a third party may require agreements between agents. We abstract from these details to focus on the general approach. The reasoning rules we provide here only pertain to the create and discharge operations and the conditional commitments.

Discharge Axiom: A commitment is no longer in force if the condition committed to holds.

The following axiom captures how a conditional commitment is resolved based on the temporal ordering of the commitments it refers to.

Progress Axiom: When the conditional commitment *CC(x, y, p, q)* holds, if *p* becomes true, then the original commitment no longer relevant. Instead, a new commitment is created, to reflect that the debtor *x* is now committed to bring about *q*. Conversely, if *q* occurs when the conditional commitment *CC(x, y, p, q)* holds, the original commitment is terminated and no other commitment is created.

Modelling the NetBill Protocol

Rather than exploring the entire NetBill protocol, let us view a representative part of the protocol to explain how a commitment protocol is specified and executed. The two basic roles in the NetBill protocol are merchant (*MR*) and customer (*CT*). The commitments that exist in the NetBill protocol are inherently conditional. That is, both the customer and the merchant promise to bring about certain conditions if the other party also commits to bring about certain conditions.

The following abbreviations capture the conditional commitments that exist in the NetBill protocol.

- *accept(i, m)* abbreviates *CC(CT, MR, goods(i), pay(m))*, which means that the customer commits to paying amount *m* if he receives the goods *i*
- *promiseGoods(i, m)* abbreviates *CC(MR, CT, accept(i, m),goods(i))*, which means that the merchant commits to sending the goods if the customer promises to paying the agreed amount
- *promiseReceipt(i, m)* abbreviates *CC(MR, CT, pay(m), receipt(i))*, which means that the merchant commits to sending the receipt if the customer pays the agreed-upon amount
- *offer(i, m)* abbreviates *(promiseGoods(i, m)* $\wedge$ *promiseReceipt(i, m))*

These commitments are created by exchange of messages. That is, each message corresponds to an operation on commitments. By exchanging messages, participants manipulate their commitments. The following lists the messages in the NetBill protocol and the commitments they create:

- *priceQuote(PrID, Quote): promiseGoods(PrID, Quote)*, and *promiseReceipt(PrID, Quote)*
- *goodsRequest(PrID, Quote): accept(PrID, Quote)*
- *delivery(crypt(PrID, Key), Quote): promiseReceipt(PrID, Quote)*

In addition to creating the above commitments, the messages also bring about certain propositions. The following lists the messages and the propositions that they realize:

- *delivery(crypt(PrID, Key), Quote): goods(crypt(PrID, Key))*
- *payment(epo(C, crypt(PrID, K), Quote)): pay(Quote)*
- *receipt(PrID, Price): receipt(PrID)*

Executing the NetBill Protocol

Commitment protocol specification can either be used at run time to reason about the actions (Yolum & Singh, 2002b) or can be compiled into a finite state machine at compile time (Yolum & Singh, 2002a; Chopra & Singh, 2004). If it is used at run time, then agents working from the same commitment protocol specification can reason about the protocol logically and each can choose actions appropriate for its current situation. This is especially useful when exceptions arise since the agents can find alternatives to complete their interactions. A useful method for generating alternatives is planning. Since an agent knows its current state and its desired state, it can apply a planning algorithm to derive the actions that need to be executed to reach a goal state. This way, if the agent moves to an unexpected state (as a result of an exception), it can still construct plans to reach a final state. For example, assume that in the NetBill protocol, a customer wants to buy an item without learning the price first. Current NetBill specification requires the customer to ask the price first, hence will not allow this scenario. However, in the commitment-based specification, agents are not restricted to specific sequences of actions. Any agent can start the protocol, and at whatever state holds then. Hence, the customer can send the *goodsRequest* action above and thereby make a conditional commitment to pay if the merchant delivers the goods and promises the receipt. The merchant can then reason about the rest of the protocol to determine its actions, e.g., that if it wants to sell the item then it needs to send the goods, and so on.

However, if agents are not equipped with tools to reason about a commitment protocol specification, then the commitment protocols can be compiled into a finite state machine. Finite state machines are easy to execute. As a result of the compilation, agents only need to follow the transitions in the finite state machine.

Verifying Commitment Protocols

Verifying the compliance of protocols at run time means checking if agents follow the protocol by their actions. In terms of commitment protocols, an agent follows the protocol if it discharges its unconditional commitments. By observing the commitments operations, an agent can decide whether agents comply with a given commitment protocol. The observing agent can be a dedicated agent that keeps track of all the messages in the system. Conversely, each agent in the system may track the commitments in which it is involved and verify that the corresponding agents comply (Venkatraman & Singh, 1999). The main idea underlying this type of verification is to compose a trace of the current protocol execution and compare this to the protocol specification. One successful way of performing this comparison is with model checking at run time. By representing the protocol specification and the protocol execution in temporal logic, one can use existing model checking tools to find out if a particular execution is a legal outcome of the protocol.

Commitment-based semantics of protocols makes it possible to verify the properties at design time (Yolum, 2007). More specifically, given a commitment-based protocol, one can check important properties such as effectiveness, consistency, and robustness. The *effectiveness* of a commitment protocol captures if a given specification allows continual progress to reach a desired state. The *consistency* of a protocol shows whether the protocol can yield conflicting computations. Ideally, the participants of a protocol should not be able to execute actions that lead the protocol to enter an inconsistent state. By studying the effects of allowed commitment operations, one can infer whether the protocol is consistent. The *robustness* of a protocol shows how well the protocol tolerates exceptions. That is, if the protocol

can be carried out in alternative ways, then the protocol will be more likely to succeed in completing the necessary transactions. One can study the robustness of protocols, by investigating how well it offers alternative execution paths.

All these aspects of commitment protocols can be studied by first representing a protocol in terms of a *commitment graph* in which the nodes represent possible commitments in the protocol and the edges represent the operations between the commitments. Using a commitment graph, one can check whether infinite loops can occur such that the effective progress of the protocol is endangered or existing operations are enough to ensure that the commitments created by the protocol can be discharged. Algorithms for detecting these properties have been implemented in a tool to study organizations (Shah Nawaz, 2007).

EXPECTATION-BASED SEMANTICS

Social expectations are a logic tool to give semantics to agent interactions. Intuitively, a set of expectations represents a possible evolution of the agent system towards a correct situation. In general, many possible evolutions of the MAS are coherent with its semantics, all of them being conformant. An expectation-based system should not commit to a predefined evolution, but provide a set of possible hypotheses on the future correct evolution of the system. For this reason, it is quite natural to give expectations a hypotheses-based semantics, such as an *abductive* semantics.

Abduction is one of the fundamental forms of inference identified by the philosopher Peirce (Hartshorne & Weiss, 1965), and is naturally embedded in logic programming as abductive logic programming (ALP) (Kakas *et al.* 1993). An abductive logic program includes a knowledge base *KB*, that is, a set of clauses that may contain predicates without definition, whose truth cannot be proven, but only *assumed.* Such predicates are called *abducibles* and are typically identified syntactically. The truth of a sentence, or a goal, *G* is decided as *G* being a logic consequence (indicated with the symbol $\models$, that stands for logic entailment) of the knowledge base and a set of assumptions:

$$KB \cup \Delta \models G$$

However, hypotheses cannot be assumed freely, because, depending on the application domain, some combinations of hypotheses should be forbidden, e.g., because they contradict each other. Thus, a set of *integrity constraints* typically restricts the combinations of predicates that can be abduced at the same time. Integrity constraints (IC) must all be entailed at all times by the union of the *KB* and the set of assumptions being devised:

$$KB \cup \Delta \models IC$$

The KB defines the structural description of the society. It includes for example the rules for joining and leaving the society, the definition and assignment of roles, and the organizational rules. Therefore, it has strong similarities with the organizational level identified by Dignum *et al.* (2002). The goal (*G*) can be considered as a "global goal" of the agent organization, external to the goal of each individual agent (Dignum & Dignum, 2007). In SCIFF, expectations are mapped on to abducible literals. Expec-

tations can be positive, as in **E**(*p,t*), meaning that it is expected that an event *p* happens at some time *t*. They can also represent the need to avoid inconsistent behaviour, by stating that an action *p* should not be performed: **EN**(*p,t*) is a negative expectation saying that *p* should not happen at any time *t*.

In order for a MAS to comply with a specification, the events it produces must match the expectations. To this end, a literal **H**(*p,t*) is used to model an event *p* occurring at time *t*. To achieve a compliant history of events, if **E**(*p,t*) (resp. **EN**(*p,t*)) is in the set of expectations, then **H**(*p,t*) should be (resp. not be) in the set of events.

Integrity constraints serve, as is usual in ALP, to avoid combinations of assumptions that do not make sense. But they also have two important purposes for modelling interaction: from a declarative perspective, they associate actual events and expectations; from an operational perspective, they introduce forward reasoning into the machinery of resolution-based logic programming. For example, the integrity constraint

$$\mathbf{H}(p,T) \rightarrow \mathbf{E}(r,T_1) \wedge T_1 \leq T+5 \\ \vee\ \mathbf{EN}(s,T_2)$$

states that if an event *p* happens at some time *T*, then an event *r* is expected to happen within 5 time units, or else no event *s* should happen at any time. Operationally, specifications written in SCIFF are interpreted and executed in a corresponding proof-procedure, which monitors the evolution of happened events and processes them on-the-fly, providing as output sets of expectations. In the example above, upon the occurrence of event *p* at time 1 (i.e., **H**(*p*,1)), the forward evaluation of the IC generates two alterative sets of expectations: $\{\mathbf{E}(r,T_1) \wedge T_1 < 6\}$ and $\{\mathbf{EN}(s,T_2)\}$. Such sets grow monotonically during the evolution of the interaction: as more events happen, they are recorded in a set called *history*, and are matched with the expected behaviour. In case of matching, the expectations are *fulfilled*, otherwise they are *violated*.

The SCIFF proof-procedure is an extension of the IFF proof-procedure defined by Fung & Kowalski (1997). It is based on a rewriting system, which applies a set of transitions to exhaustion, generating a derivation tree. In each node of the tree, the status records the important information, such as happened events, pending, fulfilled and violated expectations, and active CLP constraints. According to the declarative semantics, the MAS evolution is considered to be compliant to the specification if, and only if, there is at least one leaf node of the tree whose expectations are fulfilled.

Compared to other abductive languages, SCIFF shows unique features: use of variables in abducibles, ability to reason on evolving histories, expectations, concepts of fulfillment and violation, and a capability to reason about both existentially and universally quantified variables, also involving CLP constraints and quantifier restrictions (Bürckert, 1994). An in-depth description of language and proof-procedure is given by Alberti *et al.* (2008).

Alongside SCIFF, a *generative* version of the proof-procedure, called g-SCIFF (Montali et al., 2008), can be used to perform an analysis of the formal properties of the protocols. It works by *generating* sets of compliant histories in a compact, intensional form. The set of generated histories can be then analysed and reasoned about formally.

Let us now show a possible SCIFF specification of the NetBill protocol and of one of its properties.

Box 1.

$[SIC_1]$ **H**(tell(MR, CT, *priceQuote*(*PrID*, *Quote*), ID), T)
→ **E**(tell(CT, MR, *priceRequest*(*PrID*), ID), T2) ∧ T2 < T.

$[SIC_2]$ **H**(tell(CT, MR, *goodsRequest*(*PrID*, *Quote*), ID), T)
→ **E**(tell(MR, CT, *priceQuote*(*PrID*, *Quote*), ID), Tpri) ∧ Tpri < T.

$[SIC_3]$ **H**(tell(MR, CT, *delivery*(*crypt*(*PrID*, K), *Quote*), ID), T)
→ **E**(tell(CT, MR, *goodsRequest*(*PrID*, *Quote*), ID), Treq) ∧ Treq < T.

$[SIC_4]$ **H**(tell(CT, MR, *payment*(*epo*(CT, *crypt*(*PrID*, K), *Quote*)), ID), T)
→ **E**(tell(MR, CT, *delivery*(*crypt*(*PrID*, K), *Quote*), ID), Tdel) ∧ Tdel < T.

$[SIC_5]$ **H**(tell(MR, netbill, *endorsedEPO*(*epo*(CT, *crypt*(PrID, K), Quote), K, MR), ID), T)
→ **E**(tell(CT, MR, *payment*(*epo*(C, *crypt*(*PrID*, K), *Quote*)), ID), Tpay) ∧ Tpay < T.

$[SIC_6]$ **H**(tell(netbill, MR, *signedResult*(CT, *PrID*, *Quote*, K), ID), Tsign) ∧ MR ! = netbill
→ **E**(tell(MR,netbill, *endorsedEPO*(*epo*(CT,*crypt*(*PrID*, K), *Quote*), K, M), ID),T) ∧ T < Tsign.

$[SIC_7]$ **H**(tell(MR, CT, *receipt*(*PrID*, *Quote*, K), ID), Ts)
→ **E**(tell(netbill, MR, *signedResult*(CT, *PrID*, *Quote*, K), ID), Tsign) ∧ Tsign < Ts.

$[SIC_8]$ **H**(tell(CT, MR, *payment*(*epo*(CT, *crypt*(*PrID*, K), *Quote*)), ID), T)
→ **E**(tell(MR, netbill, *endorsedEPO*(*epo*(CT, *crypt*(PrID, K), Quote), K, MR), ID), Tepo) ∧ T < Tepo ∧ MR ! = netbill.

$[SIC_9]$ **H**(tell(MR, netbill, *endorsedEPO*(*epo*(CT, *crypt*(*PrID*, K), *Quote*), K, MR), ID), T)
→ **E**(tell(netbill, MR, *signedResult*(CT, *PrID*, *Quote*, K), ID), Tsign) ∧ T < Tsign.

$[SIC_{10}]$ **H**(tell(netbill, MR, *signedResult*(CT, *PrID*, *Quote*, K), ID), Tsign)
→ **E**(tell(MR, CT, *receipt*(*PrID*, *Quote*, K), ID), Ts) ∧ Tsign < Ts.

NetBill Protocol Specification in SCIFF

We can model the NetBill protocol using the SICs shown in Box 1. We assume that the network layer is reliable and that transmission time is negligible, so that the times of sending and receiving can be considered to be the same.

$[SIC_1$-$SIC_7]$ are *backward* SICs, i.e., integrity constraints that state that if some set of events happens, then some other set of events is expected to have happened before. $[SIC_1]$, for example, imposes that if *MR* has sent a *priceQuote* message to *CT*, stating that *MR*'s quote for the goods identified by *PrID* is *Quote*, in the interaction identified by *ID*, then we expect that *CT* has sent to *MR* a *priceRequest* message for the same goods, in the course of the same interaction, at an earlier time.

$[SIC_8$-$SIC_{10}]$ instead are *forward* SICs. $[SIC_9]$ imposes that an *endorsedEPO* message from *MR* to the NetBill server be followed by a *signedResult* message, with the corresponding parameters. Note that we impose forward constraints only from the *payment* message onwards, because both parties (merchant and customer) can withdraw from the transaction during *Price Negotiation* and *Good Delivery*: hence the uttering of messages in the first part of the protocol does not lead to any expectation to utter further messages.

Forward and backward SICs have a different purpose. The former can be used to produce specifications in line with the philosophy of commitment-based semantics. They model a reactive behaviour and describe a step-by-step evolution of the interaction similarly to what we could do with a state machine. For example, the purpose of SIC_1 is to prevent unsolicited quotes (a *priceQuote* must always follow a *priceRequest*). If we deleted SIC_1 from the NetBill specification, we would enable unsolicited quotes. Conversely, if we replaced SIC_1 with the following forward SIC:

H(tell(CT, MR, *priceRequest*(*PrID*), ID), T)
→ **E**(tell(MR, CT, *priceQuote*(*PrID*, *Quote*), ID), T2) ∧ T < T2.

the semantics would be, instead, that a merchant must always respond to a *priceRequest*. Therefore, if we want to specify the protocol in such a way that agents can start at any time, skipping the previous steps, then forward SICs is what we want to use. Backward SICs instead can be used to give a protocol specification that more closely follows NetBill's original, rigid specification. Both types of SICs are well-suited to run-time verification. While each time there is a new communicative action forward SICs tell what should happen (or should be avoided) later, backward SICs instead tell what should have happened before for the system to be in a compliant state. Sometimes it is useful to have both SICs. This is the case, e.g., with SIC_5 and SIC_8. We can use SIC_5 for run-time verification and SIC_8 to specify the next message needed to keep the system compliant. Forward SICs can also be used to implement compliant agents, as it is shown by Alberti *et al.* (2006a).

Verifying Expectation-Based Protocols

Compliance to expectation-based protocols specified in the SCIFF language can be verified at run-time using the SCIFF proof-procedure. Design-time verification instead can be done using g-SCIFF.

A sample protocol property that can be verified using g-SCIFF is the following:

As long as the protocol is respected, the merchant receives the payment for some goods G if and only if the customer receives the goods G. (goods atomicity property, GAP)

To this end, we model payment by way of a *signedResult* message issued from the NetBill server, and goods receipt by way of two messages addressed to the customer, containing the encrypted goods (*delivery* message) and the encryption key (*receipt* message). Then, GAP can be expressed with a double implication:

H(tell(netbill, MR, signedResult(CT, PrID, Quote, K), ID), Tsign)
⇔ **H**(tell(MR, CT, delivery(crypt(PrID, K), Quote), ID), T)
∧ **H**(tell(MR, CT, receipt(PrID, Quote, K), ID), Ts)

and its validity can be automatically proven as shown by Chesani (2007). Design-time verification can address consistency to show that there exists at least one possible way to execute a protocol correctly. The existence of alternative protocol runs that bring the protocol to correct completion, which is related to the property of robustness mentioned earlier, is modelled via disjunctive SICs.

One of the advantages of the expectation-based approach is that there is a single language to specify protocols, verify their run-time execution and their properties at design-time. For run-time verification, SCIFF does not resort to model checkers, which can become quite costly in terms of memory use. Instead, SCIFF reasons on intensional state descriptions and it makes extensive use of a constraint solver. In this way, it is possible to model deadlines and, e.g., identify violations as soon as they expire without an expected event occurring.

SCIFF is embedded in the SOCS-SI interaction monitoring and verification system.[1] A protocol repository[2] contains a number of protocols modelled in SCIFF and some experimental results.

CONCLUSION

Mechanisms to describe and verify interaction are indispensable elements of engineering methodologies for agent societies. Commitment-based and expectation-based semantics are declarative approaches for specifying interaction protocols. Since both of these approaches are based on social semantics, they are verifiable without knowing the agents' internals.

Commitment protocols associate commitment operations with the messages of the agents so that by sending messages agents create or manipulate their commitments. Agents that follow these protocols can choose their actions by generating plans. This enables protocols to be executed flexibly. By maintaining the list of commitments in the system, an agent can verify whether others comply with the given protocol.

Expectation-based protocols use logic-based forward rules and a knowledge base to associate socially relevant events, such as agent messages, with expectations about what else should or should not happen. This enables protocols to be easily verified at run-time. It also enables reasoning with events together with other facts or hypotheses that can be made about the components and behaviour of an organization.

Expectations and commitments are important elements of organizational multi-agent systems. They exist outside of individual agents to specify and regulate the behaviour of an agent organization's members, and as such they could be used to express the organizational element inside a social artifact (Dignum & Dignum, 2007; Omicini *et al*. 2004).

The approaches presented in this chapter rely on the notions of openness and heterogeneity in MAS and are proposed to support organizational design and a positive integration of organizational and individual perspectives. Agents can reason upon the status of commitments and social expectations to plan and deliberate about future courses of actions. The fulfillment status of expectations and commitments can be verified to monitor the behaviour of a system and decide about reachability of goals in an organization-centric coordination perspective, while preserving the autonomy of the individuals.

While practical applications of agents to organizational modelling are being widely developed, work presented in this chapter answers to the need of formal theories to describe interaction. Interaction models often need to refer to other elements of the domain knowledge that are central in organizational models, such as roles and organizational facts. Formal theories should aim to bring together all these elements to accommodate design, monitoring, reasoning and verification. Previous work by Yolum & Singh (2002b) has shown how commitments can be embedded in a logical framework based on the event calculus. Expectations are an essential element of a general logical framework that integrates reactivity and rationality in the context of hypothetical reasoning and relying on the strengths of constraint-based

reasoning. This poses the interesting question of how to gain the most benefit from the combination of commitments and expectations in an organizational modelling context, and paves the way to new interesting lines of research.

FUTURE RESEARCH DIRECTIONS

Commitments and expectations have different strengths and weaknesses with respect to expressive power and verification capability. Therefore, an interesting future direction would be a thorough study of their differences, to understand if and how these languages can be mapped to one other and how can be used together for protocol execution, monitoring, and verification.

A promising research direction being undertaken using both approaches is the application of such paradigms to the domain of Service Oriented Computing, especially in connection with advanced cross-enterprise service engagements. MAS and services have much in common (Singh & Huhns, 2005). Both aim to facilitate the integration of new applications, overcoming difficulties due to platform, language, and architecture heterogeneity and component autonomy. And when we think of services as real-life business services rather than merely as technical (web) services, the benefits of MAS organization modelling become more apparent. Services must both support individual perspectives, expressed in terms of policies, and overall notions of correctness and compliance, expressed via protocols. Viewed in this manner, the above discussions of commitments and expectations—although cast in the terms of protocols—apply to *service composition* in its broadest sense.

A real-life service engagement involves both commitments and expectations. The commitments encode the high-level contractual relationships among the participants and the expectations encode the rules of interaction as well as de facto modes of behaviour. Further, we can identify the protocols as patterns of interaction, which can be used to succinctly and reusably specify service engagements. In this manner, the design, modelling, verification, and enactment of organizations of business partners may readily be built on the approaches surveyed in this chapter. Take for example two possible candidate roles for a service engagement that we want to design. Are these two profiles interchangeable, with respect to the task at hand? Interoperability checking helps us define a set of possible alternatives that we can later evaluate based on extra-logical criteria in the economy of the organization. The adoption of a social approach for specifying and verifying interaction enables us to declaratively specify a minimal set of constraints that guarantee a fruitful collaboration, while respecting the autonomy of the individuals and avoiding harmful over-specification.

We and our colleagues have already begun work along this research path and have obtained several promising results. Recently, Singh and colleagues have applied the concept of commitment-based protocols to the service oriented architecture and business process management contexts, by addressing the problem of business process adaptability (Desai *et al*. 2006a) and of protocol composition (Desai *et al*. 2006b). Alberti *et al.* (2006b, 2007a) and Chesani *et al.* (2007) have applied social expectations to the specification of individual services and choreographies, to the formal study of the conformance of services to choreographies and of the interoperability of services. Further, expectations have been used to study goal reachability in the context of service contracting (Alberti *et al*. 2007b).

Another interesting issue concerns the declarative semantics of the approaches. Chopra & Singh (2004) and Xing *et al.* (2003) show how commitments can be mapped to various underlying logic-based formalisms. In these logics, the proof of properties for the specified system is usually done via model

checking. Expectations, instead, adopt the abductive logic programming semantics. A future research direction is the investigation of how these different formalisms relate. In particular, in the context of organizational modelling and interaction verification, it is interesting to study the combination of model checking and proof-theoretic techniques to perform a variety of different verification tasks, such as conformance checking, static verification of properties, and interoperability between global and local models.

ACKNOWLEDGMENT

This work has been partially funded by the following projects: MIUR-PRIN 2005-011293,[3] MIUR-FIRB TOCAI.it,[4] TÜBİTAK-105E073.

REFERENCES

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., & Mello, P. (2006a). A verifiable logic-based agent architecture. In F. Esposito, Z. W. Ras, D. Malerba, G. Semeraro (Eds.), *Proceedings of the 16th International Symposium on Foundations of Intelligent Systems, LNAI 4203* (pp. 188–197). Berlin, Germany: Springer.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Montali, M. (2006b). An abductive framework for a-priori verification of web services. In A. Bossi, M. J. Maher (Eds.), *Proceedings of the 8th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming* (pp 39-50). New York, NY: ACM Press.

Alberti, M., Chesani, F., Gavanelli ,M., Lamma, E., Mello, P., Montali, M., & Torroni, P. (2007a). A rule-based approach for reasoning about collaboration between smart web services. In M. Marchiori, J. Z. Pan, C. de Sainte Marie (Eds.), *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems, LNCS 4524* (pp. 279–288). Berlin, Germany: Springer.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., & Torroni, P. (2007b). Web service contracting: Specification and reasoning with SCIFF. In E. Franconi, M. Kifer, W. May (Eds.), *Proceedings of the 4th European Semantic Web Conference, LNAI 4519* (pp. 68–83). Berlin, Germany: Springer.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: The SCIFF Framework. *ACM Transactions on Computational Logic, 9*(4), Article 29.

Bauer, B., Müller, J. P., & Odell, J. (2001). Agent UML: A formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering, 11*(3), 207–230.

Bordini, R. H., Fisher, M., Visser, W., & Wooldridge, M. (2004). Model checking rational agents. *IEEE Intelligent Systems 19*(5), 46–52

Bürckert, H. J. (1994). A resolution principle for constrained logics. *Artificial Intelligence, 66*, 235–271.

Castelfranchi, C. (1995) Commitments: From individual intentions to groups and organizations. In V. R. Lesser, L. Gasser (Eds.): *Proceedings of the First International Conference on Multiagent Systems* (pp. 41–48). Cambridge, MA: The MIT Press.

Chesani, F. (2007). *Specification, execution and verification of interaction protocols.* Unpublished doctoral dissertation, University of Bologna, Department of Computer Engineering (DEIS), Italy.

Chesani, F., Mello, P., Montali, M., & Storari, S. (2007). Agent societies and service choreographies: A declarative approach to specification and verification. In M. Baldoni, C. Baroglio, V. Mascardi (Eds.), *Agents, Web-Service, and Ontologies: Integrated Methodologies. Proceedings of the International Workshop MALLOW-AWESOME'007. Durham, September 6th-7th, 2007.* Retrieved May 26, 2008, from http://awesome007.disi.unige.it/proceedings.html

Chopra, A. K., & Singh, M. P. (2004). Nonmonotonic commitment machines. *Advances in Agent Communication, International Workshop on Agent Communication Languages, LNCS 2922* (pp. 183–200). Berlin, Germany: Springer.

Cox, B., Tygar, J. C., & Sirbu, M. (1995). NetBill security and transaction protocol. In D. E. Geer Jr (Ed.): *Proceedings of the First USENIX Workshop on Electronic Commerce*. Retrieved on May 26, 2008, from http://www.usenix.org/publications/library/proceedings/ec95/cox.html

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

Desai, N., Chopra, A. K., & Singh, M. P. (2006a). Business process adaptations via protocols. In J. Zhao, M. Blake, P. Hung (Eds.), *Proceedings of the Third IEEE International Conference on Services Computing* (pp. 103–110). Washington, DC: IEEE Computer Society.

Desai N., Mallya, A. U. Chopra, A. K., & Singh, M. P. (2006b). OWL-P: A methodology for business process development. In M. Kolp, P. Bresciani, B. Henderson-Sellers, M. Winikoff (Eds.), *Agent-Oriented Information Systems III, 7th International Bi-Conference Workshop, LNCS 3529* (pp. 79–94). Berlin, Germany: Springer.

Dignum, V., & Dignum, F. (2007). Coordinating tasks in agent organization or: Can we ask you to read this paper? In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, E. Matson (Eds.): *Coordination, Organisations, Institutions and Norms in Agent Systems II, LNCS 4386* (pp. 32–47). Berlin, Germany: Springer.

Dignum, V., Meyer, J.-J. Ch., Wiegand, H., & Dignum, F. (2002). An organisational-oriented model for agent societies. In: G. Lindemann, D. Moldt, M. Paolucci, B. Yu (Eds.), *Proceedings of the RASTA Workshop at AAMAS'02, Communications Vol. 318* (pp. 31–50). Hamburg University, Faculty of Informatics, Germany.

Fung, T. H., & Kowalski, R. A. (1997). The IFF proof-procedure for abductive logic programming. *Journal of Logic Programming, 33*(2), 151–165.

Guerin, F., & Pitt, J. (2002) Proving properties of open agent systems. In: C. Castelfranchi, W. Lewis Johnson (Eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 557–558). New York, NY: ACM Press.

Hartshorne, C., & Weiss, P. (1965). *Collected papers of Charles Sanders Peirce, 1931–1958*. Cambridge, MA: Harvard University Press.

Jaffar, J., & Maher, M.J. (1994). Constraint logic programming: A survey. *Journal of Logic Programming, 19–20*, 503–582.

Kakas, A. C., Kowalski, R. A., & Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation, 2*(6), 719–770.

Montali, A., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., & Torroni, P. (2008). Verification from declarative specifications using Logic Programming. In M. Garcia de la Banda & E. Pontelli (Eds.), *Proceedings of the 24th International Conference on Logic Programming, LNCS*. Berlin, Germany: Springer.

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., & Tummolini, L. (2004). Coordination artifacts: Environment-based coordination for intelligent agents. In: C. Castelfranchi, W. Lewis Johnson (Eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 286–293). New York, NY: ACM Press.

Padgham, L., & Winikoff, M. (2004). *Developing intelligent systems: A practical guide*. Hoboken, NJ: John Wiley & Sons, Inc.

Shah Nawaz, S. (2007). *Commitment-based analysis of organizations: Dealing with inconsistencies.* Unpublished Master Thesis. Boğaziçi University, Department of Computer Engineering, Istanbul, Turkey.

Singh, M. P. (1991). Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels* (pp. 104–106). Menlo Park, CA: AAAI Press.

Singh, M. P. (1998). Agent communication languages: Rethinking the principles. *IEEE Computer, 31*(12), 40–47.

Singh, M. P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law, 7*, 97–113.

Singh, M. P., & Huhns., M. N. (2005). *Service-Oriented Computing***,** Hoboken, NJ: John Wiley & Sons, Inc.

Venkatraman, M., & Singh, M. P. (1999). Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems, 2*(3), 217–236.

Xing J., & Singh, M. P. (2003). Engineering commitment-based multiagent systems: A temporal logic approach. In: J. S. Rosenschein, T. Sandholm, M. Wooldridge, M. Yokoo (Eds.), *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 891–898). New York, NY: ACM Press.

Yolum, P. (2007). Design time analysis of multiagent protocols. *Data and Knowledge Engineering, 63*, 137–154.

Yolum, P., & Singh, M. P. (2002a). Commitment machines. In J.-J. Ch. Meyer, M. Tambe (Eds.): *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages, LNAI 2333* (pp. 235–247). Berlin, Germany: Springer.

Yolum, P., & Singh, M.P. (2002b). Flexible protocol specification and execution: Applying event calculus planning using commitments. In: C. Castelfranchi, W. Lewis Johnson (Eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 527–534). New York, NY: ACM Press.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003) Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM), 12*(3), 317–370.

ADDITIONAL READING

On Multi-Agent Organizational Paradigms

Horling, B., & Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review, 19*, 281–316.

On Agent-Oriented Software Engineering Methodologies

Zambonelli, F., & Omicini, A. (2004). Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Sytems, 9*, 253–283.

Desai, N., Mallya, A.U., Chopra, A.K., & Singh, M.P. (2005). Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering, 31(12)*, 1015–1027.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., & Perini, A. (2004). Tropos: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems, 8(3)*, 203–236.

Bergenti, F., Gleizes, M.-P., & Zambonelli, F. (Eds.). *Methodologies and software engineering for agent systems: The agent-oriented software engineering handbook*. Boston, MA: Kluwer Academic Publishers.

On Commitments

Desai, N., Chopra, A.K., & Singh, M.P. (2007). Representing and reasoning about commitments in business processes. In A. Howe and R. Holt (Eds.): *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, (pp. 1328–1333). Menlo Park, CA: AAAI Press.

Flores, R.A., Pasquier, P., & Chaib-draa, B. (2007). Conversational semantics sustained by commitments. *Autonomous Agents and Multi-Agent Systems* 14(2), 165–186.

Fornara, N. & Colombetti, M. (2002) Operational specification of a commitment-based agent communication language. In: C. Castelfranchi, W. Lewis Johnson (Eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 535–542). New York, NY: ACM Press.

Singh, M.P. (2008) Semantical considerations on dialectical and practical commitments. In D. Fox, C. Gomes (Eds.): *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Yolum, P., & Singh, M.P. (2007). Enacting protocols by commitment concession. In E. H. Durfee, M. Yokoo, M. N. Huhns, O. Shehory (Eds.): *Proceedings of the Sixth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS).* (pp. 116–123). New York, NY: ACM Press.

Wan, F. & Singh, M.P. (2005). Formalizing and achieving multiparty agreements via commitments. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, M. Wooldridge (Eds.): *Proceedings of the Fourth International Joint Conference on Autonomous Agents and MultiAgent Systems*. (pp. 770–777). New York, NY: ACM Press.

Winikoff, M. (2007). Implementing Commitment-Based Interaction. In E. H. Durfee, M. Yokoo, M. N. Huhns, O. Shehory (Eds.): *Proceedings of the Sixth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS).* (pp. 868–875). New York, NY: ACM Press.

Winikoff, M., Liu, W., & Harland, J. (2005). Enhancing commitment machines. In J. A. Leite, A. Omicini, P. Torroni, P. Yolum (Eds.): *Proceedings of the Second International Workshop on Declarative Agent Languages and Technologies*, *LNAI 3476* (pp. 198–220). Berlin, Germany: Springer.

On Computational Logic-Based Agents

Fisher, M., Bordini, R.H., Hirsch, B., & Torroni, P. (2007). Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence, 23(1)*, 61–91.

Mascardi, V., Martelli, M., & Sterling, L. S. (2004). Logic based specification languages for

intelligent software agents. *Theory and Practice of Logic Programming, 4(4)*, 429–494.

Torroni, P. (2004). Computational logic in multi-agent systems: Recent advances and future directions. *Annals of Mathematics and Artificial Intelligence, 42(1–3)*, 293–305.

On the Logics Mentioned in the Chapter

Abductive Logic Programming

Kakas, A.C., Kowalski, R.A., & Toni, F. (1998). The role of abduction in logic programming. In D.M. Gabbay, C.J. Hogger and J.A. Robinson (Eds.): *Handbook of logic in artificial intelligence and logic programming, vol. 5* (pp. 235–324). Oxford, UK: Oxford University Press.

Constraint Logic Programming

Marriott, K., & Stuckey, P.J. (1998). *Programming with constraints: An introduction*. Cambridge, MA: The MIT Press.

Event Calculus

Kowalski, R., & Sergot, M. J. (1986). A logic-based calculus of events. *New Generation Computing, 4(1)*, 67–95.

Model Checking

Clarke, E.M., Grumberg, O., & Peled, D.A. (2000). *Model checking*. Cambridge, MA: The MIT Press.

Temporal Logic

Emerson, E.A. (1990). Temporal and modal logic. In J. van Leeuwen (Ed.): *Handbook of theoretical computer science, Part B* (pp. 995–1072). Amsterdam, The Netherlands: North-Holland.

Nonmonotonic Causal Theories

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence, 153(1–2)*, 49–104.

On Tools and Applications

Desai, N., Chopra, A.K., Arrott, M., Specht, B., & Singh, M.P. (2007). Engineering foreign exchange processes via commitment protocols. In L.-J. Zhang, W. van der Aalst, P. C. K. Hung (Eds.): *Proceedings of the Fourth IEEE International Conference on Services Computing* (pp. 514–521). Washington, DC: IEEE Computer Society.

KEY TERMS

Commitment: In simple terms, a directed obligation from one agent to another to bring about a particular condition. A commitment is open to manipulation from its participants.

Declarative Semantics: Association of meaning that specifies what rather than how. Communication with declarative semantics specifies what actions should be brought out in an interaction, rather than how they are brought out.

Expectation: An abstract entity that captures the possible events that would make a multiagent system conform to its requirements.

Execution Flexibility: Providing agents options in carrying out their interactions. Protocols that support execution flexibility allow agents to handle exceptions and take advantage of opportunities at run time.

Interaction Protocol: A set of rules that regulate the interactions between agents that work together.

Verification of Agent Compliance: Checking if agents that participate in a protocol follow the protocol rules.

Verification of Protocol Rules: Checking if protocol rules enable agents to carry out the protocol as desired. If protocol rules are specified incorrectly, possibly leading to deadlocks or livelocks, their verification should signal this.

ENDNOTES

1 http://lia.deis.unibo.it/research/socs_si/
2 http://lia.deis.unibo.it/research/socs_si/protocols.html
3 Specification and verification of agent interaction protocols. Project Home Page: http://www.ricercaitaliana.it/prin/dettaglio_prin_en-2005011293.htm
4 Knowledge-oriented technologies for enterprise aggregation in Internet. Project Home Page: http://www.dis.uniroma1.it/~tocai/