Amongst First-Class Protocols

Jarred McGinnis[1] and Tim Miller[2]

[1] Department of Computer Science
Royal Holloway, University of London, Egham, Surrey TW20 0EX
jarred@cs.rhul.ac.uk
[2] Department of Computer Science
University of Liverpool, Liverpool, L69 7ZF
tim@csc.liv.ac.uk

Abstract. The ubiquity of our increasingly distributed and complex computing environments have necessitated the development of programming approaches and paradigms that can automatically manage the numerous tasks and processes involved. Hence, research into agency and multi-agent systems are of more and more interest as an automation solution. Coordination becomes a central issue in these environments. The most promising approach is the use of interaction protocols. Interaction protocols specify the interaction or social norms for the participating agents. However the orthodoxy see protocols as rigid specifications that are defined *a priori*. A recent development in this field of research is the specification of protocols that are treated as first-class computational entities. This paper explores the most prominent approaches and compares them.

1 Introduction

Research into multi-agent systems aims to promote autonomy and intelligence into software agents. Intelligent agents should be able to interact socially with other agents, and adapt their behaviour to changing conditions. Despite this, research into interaction in multi-agent systems is focused mainly on the documentation of interaction protocols *a priori*. We identify three significant disadvantages with this approach: 1) it strongly couples agents with the protocols they use — something which is unanimously discouraged in software engineering — therefore requiring agent code be changed with every change in a protocol; 2) agents can only interact using protocols that are known at design time, a restriction that seems out of place with the goals of agents being intelligent and adaptive; and 3) agents cannot compose protocols at runtime to bring about more complex interactions, therefore restricting them to protocols that have been specified by human designers — again, this seems out of place with the goals of agents being intelligent and adaptive. An important corollary of these points is that the protocol is internalised within the individual agents. There is no possibility to communicate, inspect or verify the protocols by the agent or others. Not to mention a repetition of effort for each agent engineer as each agent must be encoded with the same protocol.

Recent research into multi-agent system protocols has begun to focus on *first-class protocols*, which were first defined in [11]. By first-class, we mean that a protocol exists as a computational entity in a multi-agent system that can be shared between agents, referenced, invoked, and composed, in contrast to hard-coded protocols, which exist merely as abstractions that emerge from the messages sent by the participants. Agents in the system read the definition of a protocol to learn its rules and meanings, and then structure their interaction from this definition.

Authors of such work envisage systems in which agents have access to libraries of protocols. Agents can search through these libraries at runtime to find protocols that best suit the goal they are trying to achieve, and can share these protocol specifications with possible future participants. If no single protocol is suitable for the agent, runtime composition of these may offer an alternative.

This paper explores the current state-of-the-art in first-class protocol languages, and compares these. We look at the following approaches:

1. In Section 3, we explore *commitment machines* [17], a socially-centric approach that uses commitment to define the meaning of messages and protocols. We review three different approaches [4, 5, 18], all of which share the common feature of using Yolum and Singh's commitment machines framework [17].
2. In Section 4, we explore a normative approach to defining protocols [2], which using obligations, permissions, and prohibitions to specify protocols.
3. In Section 5, we explore the Lightweight Coordination Calculus (LCC), a protocol language described in [13] based on process algebra and logic programming.
4. In Section 6, we explore the $\mathcal{RASA}$ language [11], which combines process algebra and constraint languages.
5. In Section 7, we explore an approach [3] that extends Petri Nets to specify message sequencing and message meaning.

In Section 8, we compare and contrast the above approaches, outlining the relative advantages and strengths of the approaches, and in Section 9, we briefly present some approaches that resemble first-class protocol languages, and discuss why they are not.

2 First-Class Protocols — A Definition

Our notion of first-class protocol is comparable to the notion of first-class object/entity in programming languages [15]. That is, a first-class protocol is a referencable, sharable, manipulable entity that exists as a runtime value in a multi-agent system. From the definition of a first-class protocol, participating agents should be able to inspect the definition to learn the rules and effects of the protocol by knowing only the syntax and semantics of the language, and the ontology used to describe rules and effects.

To this end, we define four properties that constitute a first-class protocol language:

– *Formal:* The language must be formal to eliminate that possibility of ambiguity in the meaning of protocols, to allow agents to reason about them using their machinery, and to allow agents to pass and store the protocol definitions as values.
– *Meaningful:* The meaning of messages must be specified by the protocol, rather than simply specifying arbitrary communication actions whose semantics are defined outside the scope of the document. Otherwise, one may encounter a communicative action of which they do not know the definition, rendering the protocol useless.
– *Inspectable/executable:* Agents must be able to reason about the protocols at runtime in order to derive the rules and meaning of the protocol, so that they can determine the messages they will send that best achieve their goals, and compare the rules and effects of different protocols.
– *Dynamically composable:* If an agent does not have access to a protocol that helps to achieve its goals, then it should be able to compose new protocols that do at runtime, possibly from existing protocols. This new protocol must also form a first-class protocol in its own right.

This definition of first-class protocol eliminates many of the protocol specification languages that have been presented in the literature. We emphasise here that first-class does not equal *global*. By global, we mean languages that specify the protocol from a global view of the interaction, rather than from the view of the individual participants. Therefore, languages such as AgentUML and FSM-based languages are not first-class, as is commonly commented, even though they are global. AgentUML is not meaningful (although one could adapt it quite easily to make it meaningful), and the composability at runtime could also be difficult, if possible at all. FSM approaches could also add meaning, but the authors are not aware of any current FSM approaches that are executable and support dynamic composition.

3 Commitment Machines

Yolum and Singh [17] present commitment machines, which are used to formally represent the social relationships that exists between autonomous agents in the form of commitments. A conditional commitment for debtor a to bring about condition q when p is satisfied is represented using the constraint $CC(a, b, p, q)$, in which b is the creditor of the commitment. Non-conditional commitments (or base-level commitments) are written $C(a, b, p)$, which is equivalent to $CC(a, b, true, p)$. In commitment machines, agents participating in a protocol create and manipulate commitments as a result of sending particular messages. Agents are programmed to understand the meaning of commitments, and can therefore reason about protocols whose meaning is specified using commitment machines.

Winikoff [16] presents a mapping from commitment machines to an abstract programming language for agents called the Simple Abstract Agent Programming Language (SAAPL). This approach is somewhat different to our idea of first-class protocol languages in that the agents do not inspect protocols to decide their course of action, but are instead implemented as a mapping from the commitment machine into a SAAPL program.

Several approaches have used the idea of commitment machines for specifying first-class protocols. In this section, we present these approaches.

3.1 Commitment Machines in the Event Calculus

Yolum and Singh [18] use the Event Calculus for specifying commitment machines. The Event Calculus is a logical language for specifying at which time-points actions occur, and the effect that those actions have. Yolum and Singh's approach uses the Event Calculus to specify message sending as actions, and the effect that message sending has is specified as the creation or manipulation of commitments.

Two predicates in the Event Calculus are the most used for specifying protocols. The $Happens$ predicate specifies that an event, e, happens at the time t, written $Happens(e, t)$. The $HoldsAt$ predicate specifies that a property, p, holds at time t, written $HoldsAt(p, t)$. The set of time points is a partially ordered set, with the relation $<$, therefore, one can specify the occurrence of messages using $Happens$, and order them using $<$. For example,

$$Happens(m_1, t_1) \wedge Happens(m_2, t_2) \wedge t_1 < t_2$$

specifies that the message m_1 occurs before the message m_2. To specify further that the sending of m_2 commits agent a to perform p for agent b, we add to the above predicate, the following:

$$HoldsAt(t_2, C(a, b, p)).$$

Yolum and Singh present a set of axioms relating communicative acts and commitments, discuss the use of an abductive planner for agents to plan their execution paths.

3.2 Commitment Machines in OWL-P

Desai *et al.* present OWL-P [4], an ontology used for modelling protocols — specifically business protocols —, which is encoded in the OWL web ontology language. The ontology defines concepts such as message, protocol, roles, proposition, and commitment. Commitments are specified as discussed above, and therefore, a protocol specified using OWL-P is a commitment machine. An additional ontology is presented for protocol composition — that is, composing protocols that achieve a single business goal into protocols that achieve multiple business goals. The axioms that define the composition must be specified by the protocol designer themselves.

3.3 Commitment Machines in MAD-P

Desai and Singh [5] present MAD-P, an extension of the C+ language. The MAD-P approach is similar to that of the Event Calculus approach discussed in Section 3.1. Message passing is specified as actions occurring at particular time points, and the meaning of message passing is specified using commitments, with the relation **causes**, linking particular messages to their meaning, for example,

$$m_1 \textbf{ causes } cancel(C(a, b, p)).$$

Sequencing of messages is specified using the **before** relation, used in the context

$$m_1 \textbf{ before } m_2$$

meaning that message m_1 occurs before message m_2.

Desai and Singh present a set of axioms for composing new protocols from existing protocols.

3.4 Discussion

The approaches outlined in this section are all different ways of specifying commitment machines. We note the following properties of all of these approaches:

- Protocols are specified from a global rather than local perspective.
- Message sequencing is specified in a declarative manner, rather than an algebraic/operational manner.
- The languages used for specifying the message sequences and the meaning of messages are the same. That is, the language, for example, the Event Calculus, is used to specify the order in which messages can occur, as well as the preconditions and effects of messages.
- All of the approaches assume some form of state — mainly the existence of commitments between agents.
- The OWL-P and MAD-P approaches support protocol composition, however, composition axioms are at a different level to protocol specification. The Event Calculus approach presented in [18] does not discuss composition, but it seems likely that composition axioms could be defined.

4 Normative Systems

Artikis *et al.* [2] propose an approach similar to commitment machines, especially the commitment machines based on C+, which Artikis *et al.* also use, however, the approach implements normative constraints rather than social commitments.

The normative approach distinguishes *valid* behaviour — behaviour that an agent had the power to perform at the time — from *invalid* behaviour — anything else. Interaction protocols are specified as actions, in which, for an agent Ag, and an action Act, Ag is *permitted* to perform Act, written $Permitted(Ag, Act)$, *prohibited* to perform *Act*, written $\neg Permitted(Ag, Act)$, and *obliged* to perform *Act*, written $Obliged(Ag, Act)$.

Similar to the commitment machines approach, agents create and manipulate norms as a result of sending messages, thus giving meaning to the protocol using norms. The *Causal Calculator* is used in [2] to execute the specifications.

We note the following properties of this approach

– Protocols are specified from a global rather than local perspective.
– Message sequencing is specified in a declarative manner, rather than an algebraic/operational manner.
– The language used for specifying the message sequences and the meaning of messages are the same.
– There is some form of state — mainly the norms associated with actions.
– Artikis *et al.* do not discuss composition, but it seems likely that composition axioms could be defined.

5 Lightweight Coordination Calculus

The Lightweight Coordination Calculus (LCC) [13] is a process-algebra based language for first-class protocol specification. A protocol consists of a protocol definition, and a set of axioms, K, keeping track of the common information known to all participants. A protocol definition consists of a set of at least two agent clauses, $A^{\{n\}}$, with each clause defining the agents from a local participant's view. An agent clause is defined using the format $\mathbf{agent}(R, Id) ::= op$, in which R is a role name, Id is an agent identifier, and op is an operation. Operations define the protocol that an agent must adhere to, and their syntax is defined as follows:

$$\begin{array}{lll}
op \in \text{Operation} :: & \text{no op} & \\
 & \mid\ (M \Rightarrow \mathbf{agent}(R, Id)) \leftarrow \psi & \text{(Send)} \\
 & \mid\ \psi \leftarrow (M \Leftarrow \mathbf{agent}(R, Id)) & \text{(Receive)} \\
 & \mid\ op1\ \texttt{then}\ op2 & \text{(Sequence)} \\
 & \mid\ op1\ \texttt{or}\ op2 & \text{(Choice)} \\
 & \mid\ \mathbf{agent}(R, Id) & \text{(Substitution)} \\
M \in \text{Message} \ :: & \langle m, \mathcal{P} \rangle &
\end{array}$$

We briefly discuss this definition. 'no op' is an empty operation, meaning that the agent does nothing. The send and receive operations define the sending of a message M to the agent defined by clause $\mathbf{agent}(R, Id)$, provided that the proposition ψ is satisfiable from the common knowledge, K, and the receiving of a message M from the agent $\mathbf{agent}(R, Id)$, which results in ψ being added to the common knowledge, K. Omitting $\psi \leftarrow$ and $\leftarrow \psi$ is equivalent to specifying that ψ is true. A message is defined as a tuple $\langle m, \mathcal{P} \rangle$, in which m is the message content, and $\mathcal{P}$ is the protocol definition (written using the LCC language) that remains to be executed and the axioms of common knowledge. Composition of protocols is defined using the composition operators, `then` and `or`, which represent sequential composition (the left operation must occur before the right), and choice (one and only one operation should occur). Finally, one can reference the name of an agent class $\mathbf{agent}(R, Id)$, and the corresponding definition (if it exists) is substituted for $\mathbf{agent}(R, Id)$. Intuitively, this represents an agent adopting the role R.

Constraints can fortify or clarify semantics of the protocols. Those occurring on the left of the '$\leftarrow$' are postconditions and those occurring on the right are preconditions. For example, an agent receiving a protocol with the constraint to believe a proposition s upon being informed of s can infer that the agent sending the protocol has a particular semantic interpretation of the act of informing other agents of propositions. This operation, $(M \Rightarrow \mathbf{agent}(R, Id)) \leftarrow \psi$, is understood to mean that message M is being sent to the agent defined as $\mathbf{agent}(R, Id)$ on the condition that ψ is satisfiable. This operation, $\psi \leftarrow (M \Leftarrow \mathbf{agent}(R, Id))$, means that once the message M is received from agent $\mathbf{agent}(R, Id)$, ψ holds. These together represent the meaning of the sending and receiving of individual messages. The meaning of a composite protocol is derived from the meaning of the messages that comprise it.

The properties of the LCC language are summarised below.

- LCC is based on the process calculus, CSP, a formal model for modelling concurrent systems. This makes LCC well suited as a language for interaction protocols and the concurrency found in multi-agent systems.
- There is long pedigree of process calculi for use as a high-level description of interactions. Besides facilitating human readability, there is a wealth of research to draw upon and apply to the field of agent coordination.
- Protocol specifications in LCC are local, rather than global.
- Although the framework provides the representation of the trace of messages occurring, there is no explicit labelling of states.
- The language for specifying message sequencing is independent of the underlying communication language.
- Constraints are declarations, not definitions.
- Designed to have a light-weight engineering requirement.
- Requires a meta-level operations to be composable.

6 $\mathcal{RASA}$

The $\mathcal{RASA}$ [11] language, part of the larger $\mathcal{RASA}$ framework, combines constraints and process algebra to model interaction protocols as first-class entities. The process algebra in the language is used to specify the sequencing of messages in a protocol, while the underlying constraint language is used to describe the meaning of messages and the message content. The meaning of entire protocols can be compositionally determined from combining the two.

Similar to LCC, $\mathcal{RASA}$'s protocol specification language resembles that of many process algebras, and in fact, the $\mathcal{RASA}$ syntax and semantics were influenced by LCC.

Let ϕ represent constraints defined in the constraint language, c communication channels, N protocol names, and x a sequence of variables. Protocol definitions adhere to the following grammar.

$$\pi ::= \phi \to \epsilon \mid \phi \xrightarrow{c(i,j).\phi} \phi \mid \pi ; \pi \mid \pi \cup \pi \mid N(x) \mid \mathbf{var}_x^{\phi} \cdot \pi$$

We use π as a meta-variable to refer to protocols; subscripts and superscripts are used to denote distinct meta-variables. $\phi \to \epsilon$ represents the empty protocol, in which no message is sent and there is no change to the protocol state, but only if ϕ holds in the current state. A protocol of the format $\phi \xrightarrow{c(i,j).\phi_m} \phi'$ is an atomic protocol. It represents that i can send the constraint ϕ_m to j over channel c only if the precondition ϕ holds in the current state, in which i and j are values in the constraint language. After the message is sent, the new state of the protocol is updated using the postcondition ϕ'. This is used to specify meaning of protocols: the precondition represents a rule for a protocol because ϕ_m can only be sent if this precondition is true; and the postcondition represents the effect that sending ϕ_m has on the state.

The protocol $\pi_1 ; \pi_2$ denotes the sequential composition of two protocols, such that all of protocol π_1 is executed, then protocol π_2. The protocol $\pi_1 \cup \pi_2$ denotes a choice of two protocols. $N(x)$ denotes a reference to a protocol π named $N(y)$, with variables y renamed to x, such that any occurrence of N is equivalent to its definition, π. The protocol $\mathbf{var}_x^{\phi} \cdot \pi$ denotes the declaration of a local variable x, with the constraints ϕ on x. The scope of x is limited to the protocol π, and the constraints on x do not change throughout its scope; that is, x is a constant.

A *protocol specification* is defined as a set of definitions of the form:

$$N(y_1, \ldots, y_n) \mathrel{\widehat{=}} \pi$$

in which $N, y_1, \ldots, y_n$ are variable names from the underlying constraint language, and π is a protocol definition. Protocol definitions can reference other protocols in the specification using their names.

The reader may have already noted several properties of $\mathcal{RASA}$:

- The use of a process algebra inherits many of the benefits stated in Section 5.
- Protocol specifications in $\mathcal{RASA}$ are global, rather than local.
- The language for specifying message sequencing is algebraic, rather than declarative; a design decision which was made to simplify protocol composition — especially runtime composition.
- The underlying language for specifying meaning is declarative.
- The language for specifying message sequencing is independent of the underlying communication language.
- $\mathcal{RASA}$ specifications maintain a *state*, which is not explicitly sent in messages (unless this is specified as part of the messages themselves).
- The operators for protocol composition have the same syntax and semantics at all levels of dialog. That is, atomic protocols are protocols in their own right, and composing them together brings about compound protocols, which can be further composed using the same operators.

7 Petri Nets

De Silva *et al.* [3] have experimented with specification of first-class interaction protocols using Petri Nets. Petri Nets are graph structures with additional annotations. The approach proposed by De Silva *et al.* models protocols by representing the arcs of a Petri Net as possible messages, and the nodes as states between messages. Petri Nets were chosen rather than similar approaches such as finite state automata due to their ability to model concurrency.

In addition to representing messages, the approach enables the specification of *internal actions*, also using Petri Nets, which specify the actions other than message sending that agents can use. These actions include the functions an agent should execute, which variables to update after a transition, and which conditions the agent must test before sending a message. As such, these actions are used to specify the rules of the protocol, and the meanings of messages.

A local-view approach is taken in the modelling of the protocols, although it is straightforward to see how Petri Nets could also be used to specify a global view. For the local view, four types of actions (two external and two internal) are available for specifying protocols: the internal actions, *Send* and *Recv*, for sending and receiving messages respectively; and the external actions, *Action*, for reading and writing variables and executing functions, and *Pred*, which are boolean functions. These are defined as *templates*, and each must adhere to the following format:

Send[Sender,Performative,Receiver,Content]
Recv[Receive,Performative,Sender,Content]
Action[Label,Type,Act,Args]
Pred[Boolean]

The templates for *Send*, *Recv*, and *Pred* are straightforward to follow. For *Action*, *Label* is a unique label identifying the action, *Type* is *execute* for functions *read* or *write* for variables, and *Args* specifies the arguments to the function, or the values for the variables.

De Silva *et al.* do not discuss the language that is used to specify the message content, the boolean functions, or the arguments, though from examples in [3], one infers that they use some form of propositional logic. Because there appears to be no restriction on the language, it seems reasonable to say that any language capable of expressing boolean expressions could be used.

We note the following properties of this approach:

- Protocol specifications are local rather than global.
- The language for specifying message sequencing is operational, rather than declarative.
- Specifying the meaning of messages is done using a declarative language.
- Petri Nets for specifying message sequencing are independent of the underlying communication language.
- It appears that the specification must maintain state, although there is not discussion of this by De Silva *et al.* .

8 Comparisons

The approaches to first-class protocols described in the previous sections share the properties of being formally defined, meaningful, inspectable, executable and dynamically composable. However there are issues of design in which they differ. The point of this comparison is not to declare one approach as the winner but to highlight the advantages and disadvantage each. No disadvantage should be considered fatal, but merely a consideration that must be taken. It is unlikely any first-class protocol language will be the panacea to all the ills of agent communication. By highlighting the issues and differences, it is hoped that the system designer can make an informed decision when choosing to take advantage of the first-class protocol approach.

8.1 Declarative vs Algebraic/operational

Singh [14] and Winikoff [16] both present good arguments for the benefits of declaratively specifying protocols, stating that this allows for a more flexible interaction. They argue that specifying *what* rather than *how* gives permits a more flexible approach to interaction. For example, one can specify that three events, a, b, and c, occur, and that b must occur before c, but with no other constraints. The interacting agents are free to choose the sequence of these messages as long as they obey the one constraint, which allows flexible interaction. For an algebraic language to specify this, one would likely have to specify all the possible sequences, which could lead to a larger expression. It is difficult to envisage an example that would be straightforward using a declarative language, but complex in an algebraic language, however, it is clear that a further level of abstraction provides the usual benefits associated with abstraction.

Another difference between the two approaches is the computational aspects. Adapting a declarative language would likely have the benefit that the language has tool support for automated reasoning and execution. While LCC and $\mathcal{RASA}$ are both executable if the agents can execute the underlying language, one must implement an agent to understand the process algebra in each. Furthermore, LCC and $\mathcal{RASA}$ were both designed to be quite generic, so there is no commitment to an underlying language, and tool support would be difficult to provide without committing to a particular underlying language. However, computationally, the declarative approach would be more demanding. Calculating the set of possible dialogs is straightforward in algebraic languages: simply traverse the tree that is formed by the definition. Using a declarative language, one would have to solve the paths as a constraint, which, for protocols of more than a few messages, could prove demanding. Winikoff [16] avoids this problem by implementing agents as a mapping from the protocols, however, Yolum and Singh's agents [18] reason by calculating all possible interactions.

The authors believe that a key benefit to using algebraic languages is the human readability. Despite the motivations behind first-class protocols being machine readable, it is clear that human designers will need to read and reason about these as well. An algebraic formalism is at a level that is more inline with the way humans think about interaction. One only has to look at existing work on dialogue games [9], abstract models of interaction [7], and token-based approaches such as AgentUML to see that operational-based approaches are the favoured approach for protocol specification and design. Even outside of computer science, instructions that are meant to be read by humans, such as recipes and installation instructions, are presented in a step-by-step manner. From the literature, it seems that declarative approaches (whether first-class or otherwise) are considerably more verbose than algebraic/operational approaches. This is not surprising, because one is specifying the semantics of sequential composition, choice, etc., each time they specify such a composition. As an example of the verbosity of declarative approaches, consider the model of the Contract-Net Protocol using Social Integrity Constraints in [1]. This model consists of 17 rules, which is verbose for such a straightforward protocol, especially as the messages contain no meaning.

Finally, we note that LCC, $\mathcal{RASA}$, and the Petri Nets approach all have the advantage of not mixing the communication language with the language for message sequencing. Adapting a declarative language for modelling interaction enforces the restriction that message meaning must be specified in that language. Considering that the meaning of the message is tied in with the message itself, this further implies that all communication would also be in this language — an unfortunate restriction. This reduces the application of the language and any protocols specified in it, as demonstrated by the commitment machines approach being implemented in three different languages, the Event Calculus [18], OWL-P [4], and MAD-P [5], all by the same research group.

8.2 Local vs Global

This dichotomy is between the perspective from which the protocols are defined. *Local* protocols define clauses with respect to the dialogical activities of a single actor. For agents to communicate they must each have a set of complementary protocols –e.g. For a message being sent in one protocol, there is a message being received in another. *Global* protocols are defined as one protocol for the actions of every participant.

There are advantages and disadvantages that must be considered with respect to the perspective used by the protocol language. Local protocols have the advantage of simplicity of use for the individual agents. They do not need to sift through the protocol to determine what roles and actions apply to them. However this can obscure the activities of dialogical partners. This shortcoming can be overcome, as is done in LCC, by sending all agent clauses to an individual agent and not just the clause it is meant to execute. Conversely, global representations give a more complete representation of the conversation space, but the agent will have protocol steps that are irrelevant to it. The choice of perspective is also influenced by the model of interaction that exists within the multi-agent system. For example mediated or managed interaction would need a global protocol. Whereas, peer-to-peer interactions would be facilitated by a local representation.

Regarding the approaches in this paper, LCC and the Petri Nets approach specify protocols from a local view, while the rest specifying protocols from a global view.

8.3 Composability

Dynamic protocols are a leap that few system designers are brave enough to take. There are a number of reasons for this. Most engineers take refuge with interaction protocols for the reliability and certainty they can provide their agent interactions. They forgive the rigidity and fragility for the safety they provide. However as the agent paradigm matures a few researchers have recognised the inevitability and benefits of protocols that can be composed automatically at run-time. There are inherit drawbacks to allowing composability such as issues of trust (e.g. who should be allowed to make changes). However, the inclusion of this functionality does not weaken any of the composable first-class protocol languages described. Indeed there are numerous frameworks where the modification of its first-class protocols is disallowed.

Although not explicitly explored, it is imaginable that a set of meta-rules could be defined over the normative approach and commitment machines to produce the composition. The same is true of the Petri Net approach. However at the current time, this has not been explored as far as the authors are aware.

LCC, although not initially designed for the purpose, has been shown suitable for dynamic composition of protocols using a number of approaches. Using dialogue games as a semantic model for composition, [10] composes the protocols for atomic dialogue games to create more complex games according to the rules of iteration, sequencing, parallelism and embedding proposed in [8]. Additionally, [10], using adjacency pairs, composition is done at the individual message level rather than for whole protocols. The $\mathcal{RASA}$ language, in reaction to dynamic LCC, was designed with dynamic composition as a fundamental feature of the language. As a consequence, there is a much more methodical application and execution of dynamic composition in this language.

From the authors' view, we believe that algebraic languages are far more suited for composition. In algebraic languages, the start and end of a protocol and its sub-protocols are straightforward to identify. We assert that this makes compositions easier to define, and their meaning more straightforward to calculate.

8.4 Top-down vs Bottom-up

The top-down vs bottom-up debate is merely an issue of taste. Nonetheless, we believe that it is an important point to note, and there are good reasons why the different approaches are taken. $\mathcal{RASA}$'s bottom-up approach is a consequence of it being purposefully designed for dynamic composition, and LCC's top-down design comes from its pedigree of trying to execute electronic institutions in a more peer to peer manner. The commitment machines approaches are declaratively specified, so they adopt neither approach.

8.5 State vs No state

LCC is the only language that does not specify the meaning of messages as the alteration of a state. Instead, LCC agents explicitly pass around any such information as part of each message. The benefits of this are clear: all participants are aware of any information they need, and there is no chance that the participants view of the state can become inconsistent with each other. We see no restriction in other approaches that would prevent protocol designers from enforcing that agents send the state with each message. However, the approach is more straightforward in LCC. Two obvious disadvantages of passing the state with every message is that there is an extra overhead, and that it becomes more laborious to model protocols in which the participants should have different information.

State is important for composition. To clearly define the meaning of a compound protocol, one must relate the meaning between its sub-protocols. It seems that some form state is the only way to do. Whether this in the form of a state itself, or whether it is information that is passed, as in LCC, does not appear to be important.

8.6 Expressiveness

A comparison of expressiveness is non-trivial, because no formal framework exists for comparing the expressiveness of protocol languages. However, we note some interesting aspects of the expressiveness of first-class protocol languages.

Regarding message sequencing, the languages are quite similar. Each of them specifies a set of possible interactions, in which each interaction is a sequence of messages with preconditions and meaning. The only aspect that we see as different is regarding parallel message sending. Versions of LCC and $\mathcal{RASA}$ define parallelism as the interleaving of messages, not as messages being sent at the same time. Declarative approaches do not suffer this restriction, because messages can be declared to be sent at the same time. For example, using the Event Calculus, one can specify that messages m_1 and m_2 are sent simultaneously:

$$Happens(m_1, t) \wedge Happens(m_2, t)$$

How this is enforced in the final system, and how straightforward it would be for agents to reason about such behaviour, is an implementation detail, and is out of the scope of this paper. In the Petri Net approach, the authors of authors of [3] state that they specifically use Petri Nets because of its ability to handle concurrent behaviour.

The second aspect of expressiveness relates to the expressiveness of messages and their meaning. The normative approach and the different commitment machine approaches can express any messages and meaning that can be expressed in the Event Calculus, OWL, or C+ respectively, and only those messages and meaning. In contrast, the $\mathcal{RASA}$, LCC, and Petri Nets approaches do not specify a particular underlying language. This provides a greater amount of flexibility in specifying meaning compared to the other approaches, because one can choose to model protocol meaning using different underlying languages, and are flexible enough to model commitment machines and norms.

However, the fact the one can under-specify the message sequencing implies that, for flexible protocol execution, the expressiveness of declarative languages are better suited than algebraic/operational languages.

8.7 Discussion

As a reader of this paper, one may be wishing to decide which first-class protocol approach they would should use. We refrain from making any definite recommendations because the approach used is dependent on the application in which the protocols will be used, and different engineers will have different views. However, we use the results in this section to highlight three aspects of these approaches that stand out:

Local vs. Global: For a peer-to-peer interactions with two-parties, we believe a local approach is best suited. For mediated interactions, or interactions with more than two parties, we believe a global approach is best suited.

Flexible interaction: Declarative approaches seem better suited for flexible interaction; that is, under-specifying the protocol, and having agents calculate the allowable sequences.

Runtime composition: We recommend an algebraic approach if runtime composition is an important theme in an application.

9 Other Approaches

Several other approaches exist for agent interaction specification that resemble first-class protocols, but which we do not consider to be first-class approaches. In this section, we briefly introduce some of the most closely related approaches, and discuss why they do not fall into the category of first-class protocol specification languages.

Social Integrity Constraints (SICs) [1] are rules specifying actual and expected behaviour. SICs are not consider as first-class protocols because there is no *meaning* to the messages. The rules specifies only which messages can occur, and the order they can occur in. In [1], the authors envisage systems in which participating agents inherently know the meaning of communicative actions, which first-class protocols aim to prevent. We believe that it would be possible for consequents of the rules could carry additional information that specified meaning, and agents could reason over this information. However, SICs do not include the notion of *state*, therefore, a new semantics would have to be specified, and would have to take into account message meaning, such as when two messages occur one after the other, but with conflicting outcomes.

Fornaro and Colombetti [6] propose a method for defining the semantics of agent communication languages using commitments. Their notion of commitment is similar to that used in commitment machines, described in Section 3.

Though the motivation for social commitments is similar to first-class protocols, it is used to define the semantics of performative-based agent communication languages, rather than protocols. Composition is obtained using interaction diagrams, and the semantics of this composition is not defined formally.

10 Conclusions

The purpose of this paper was to bring more coherence to the emerging research on first-class protocols. By the comparison of the most prominent approaches, we tease out their commonalities and their differences. As this approach to agent communication inevitably attracts more interest it is conceivable that new languages will be developed, but no matter how exotic they will have the properties described in Section 2, as well as falling on one side or the other the comparison criteria.

Though we have focused on agent communication and first-class protocols for the expression of norms of agent societies, in addition to the advantages outline in the introduction, there is another point that should be stressed. The beauty of first-class protocols is that they are generically applicable. In [12], the authors show the use of $\mathcal{RASA}$ for hybrid interactions between agents and webservice for workflow execution in the e-science domain. This is due to the power and expressiveness of first-class protocols. Social semantics are no longer held internally. They are rewritten modularly and separate from the computational entities that would make use of them. Other research has shown how webservices can follow LCC protocols for determining message passing sequences without needing to understand the social semantics of the messages being sent.

Acknowledgements

The work presented in this paper was supported by the EU projects, ARGUGRID, (ArguGRID-IST-035200), and PIPS (EC-FP6-IST-507019).

References

1. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 72–78, New York, USA, 2004. ACM Press.
2. A. Artikis, M. Sergot, and J. Pitt. Specifying electronic societies with the Causal Calculator. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Agent-Oriented Software Engineering, 3rd International Workshop*, number 2585 in LNCS, pages 1–15. Springer, 2003.
3. L. P. de Silva, M. Winikoff, and W. Liu. Extending agents by transmitting protocols in open systems. In *Proceedings of the Challenges in Open Agent Systems Workshop*, Melbourne, Australia, 2003.

4. N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. OWL-P: A methodology for business process modeling and enactment. In *Workshop on Agent Oriented Information Systems*, pages 50–57, July 2005.
5. N. Desai and M. P. Singh. A modular action description language for protocol composition. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2007. (To appear).
6. N. Fornara and M. Colombetti. A commitment-based approach to agent communication. *Applied Artificial Intelligence*, 18(9–10):853–866, 2004.
7. M. W. Johnson, P. McBurney, and S. Parsons. A mathematical model of dialog. *Electronic Notes in Theoretical Computer Science*, 141(5):33–48, 2005.
8. P. McBurney and S. Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.
9. P. McBurney, R. van Eijk, S. Parsons, and L. Amgoud. A dialogue-game protocol for agent purchase negotiations. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(3):235–273, 2002.
10. J. McGinnis. *On the mutability of protocols*. Phd thesis, University of Edinburgh, Edinburgh, Scotland, 2006.
11. T. Miller and P. McBurney. Using constraints and process algebra for specification of first-class agent interaction protocols. In G. O'Hare, A. Ricci, M. O'Grady, and O. Dikenelli, editors, *Engineering Societies in the Agents World VII*, volume 4457 of *LNAI*, pages 245–264, 2007.
12. T. Miller, P. McBurney, J. McGinnis, and K. Stathis. First-class protocols for agent-based coordination of scientific instruments. In *5th International Workshop on Agent-based Computing for Enterprise Collaboration (ACEC) Agent-Oriented Workflows and Services*, 2007. to appear.
13. D. Robertson. Multi-agent coordination as distributed logic programming. In *Proceedings for International Conference on Logic Programming*, 2004.
14. M. P. Singh. A social semantics for agent communication languages. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany, 2000.
15. C. Strachey. Fundamental concepts in programming languages. *Higher-Order and Symbolic Computation*, 13(1):11–49, April 2000.
16. M. Winikoff. Implementing commitment-based interactions. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007. (To appear).
17. P. Yolum and M. P. Singh. Commitment machines. In J.-J. Ch. Meyer and M. Tambe, editors, *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages*, volume 2333 of *LNCS*, pages 235–247. Springer, 2002.
18. P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and AI*, 42(1–3):227–253, 2004.