Annotation and Matching of First-Class Agent Interaction Protocols

Tim Miller
Department of Computer Science
University of Liverpool
Liverpool, L69 7ZF, UK
tim@csc.liv.ac.uk

Peter McBurney
Department of Computer Science,
University of Liverpool
Liverpool, L69 7ZF, UK
p.j.mcburney@csc.liv.ac.uk

ABSTRACT

Many practitioners view agent interaction protocols as rigid specifications that are defined *a priori*, and hard-code their agents with a set of protocols known at design time — an unnecessary restriction for intelligent and adaptive agents. To achieve the full potential of multi-agent systems, we believe that it is important that multi-agent interaction protocols are treated as *first-class* computational entities in systems. That is, they exist at runtime in systems as entities that can be referenced, inspected, composed, invoked and shared, rather than as abstractions that emerge from the behaviour of the participants. Using first-class protocols, a goal-directed agent can assess a library of protocols at runtime to determine which protocols best achieve a particular goal. In this paper, we present three methods for *annotating* protocols with their outcomes, and *matching* protocols using these annotations so that an agent can quickly and correctly find the protocols in its library that achieve a given goal, and discuss the advantages and disadvantages of each of these methods.

Keywords: multi-agent systems, agent interaction, first-class protocols, annotation, matching

1. INTRODUCTION

In the distributed environments of multi-agent systems, coordination between agents is important for agents to achieve their goals. Interaction protocols are seen as a promising approach to coordination in multi-agent systems. However, rather than view interaction protocols as rigid specifications that are defined *a priori*, with agents being hard-coded to follow the protocol rules — a restriction that is out of place with the vision of agents being intelligent and adaptive — we believe it is important that agent interaction protocols are first-class computational entities, allowing agents to select, reference, share, compose, invoke and inspect protocols at runtime. Such an approach would allow agents to assess which protocols achieve their goals, and to learn the rules and effect of new protocols.

In previous work, we proposed the $\mathcal{RASA}$ framework [7, 6], which regards protocols as first-class entities. These first-class protocols are documents that exist within a multi-agent system, in contrast to hard-coded protocols, which exist merely as abstractions that emerge from the messages sent by the participants. To promote decoupling of agents from the protocols they use, $\mathcal{RASA}$ contains a formal, executable language for protocol specification, about which agents can reason to determine the rules and outcomes of protocols.

A major goal of research into first-class protocols is for agents to maintain a library of interaction protocols, and to be able to select the protocol that best suits the goals that it wants to achieve at a given time and in a given environment. For this, agents must be able to quickly and correctly determine the outcomes that can result for an interaction protocol, and compare protocols in their library. In this paper, we present methods for *annotating* a protocol with its possible outcomes, so that it does not have to determine the outcomes each time it is trying to find a suitable protocol, and for *matching* a protocol that achieves a given goal, using the annotations. Emphasis is placed on protocols specified in the $\mathcal{RASA}$ protocol language, but such ideas would be applicable to other protocol languages with operators similar to $\mathcal{RASA}$'s.

2. THE $\mathcal{RASA}$ FRAMEWORK

The $\mathcal{RASA}$ framework was designed to allow us to represent and reason about first-class protocols, and investigate the types of statements we can make about them. Our idea, along with other researchers working in this area, is to have goal-directed agents with access to libraries of first-class protocols. If an agent would like to interact with another to achieve a particular goal, it can search its protocol library to find the protocol that best achieves the goal.

The $\mathcal{RASA}$ specification language was designed as an example of the minimal operators that would be required for a successful first-class protocol specification language. First presented in [7], along with its operational semantics, the language uses constraint languages and process algebra to specify interaction protocols. In this section, we briefly present this language, and a logic for reasoning about protocols specified in this language.

2.1 Modelling Information

Communication in multi-agent systems is performed across a *universe of discourse*. Agents send messages expressing particular properties about the universe. We assume that these messages refer to *variables*, which represent the parts of the universe that have changing values, and use other *tokens* to represent relations, functions, and constants to specify the properties of these variables and how they relate to each other.

Rather than devise a new language for expressing information, or using an existing language, we take the approach that any constraint language can be used to model the universe of discourse, provided that it has a few basic constants, operators and properties. This allows us to express and study a wider variety of protocols, such as those that use description logics, constraint programming languages, or even predicate and modal logics. It also permits us to use different mechanisms for defining protocol meaning, such as norms and commitments.

Definition 2.1. *Cylindric constraint system.* We assume that the underlying communication language fits the definition of a *cylindric constraint system* proposed by De Boer *et al.* [1]. They define a cylindric constraint system as a complete algebraic lattice, $\langle C, \sqsupseteq, \sqcup, \text{true}, \text{false}, Var, \exists \rangle$. In this structure, C is the set of atomic propositions in the language, for example $X = 1$, $\sqsupseteq$ is an entailment operator, true and false are the least and greatest elements of C respectively, $\sqcup$ is the least upper bound operator, Var is a countable set of variables, and $\exists$ is an operator for hiding variables. The entailment operator defines a partial order over the elements in the lattice, such that $c \sqsupseteq d$ means that the information in d can be derived from c.

A constraint is one of the following: an atomic proposition, c, for example, $X = 1$, where X is a variable; a conjunction, $\phi \sqcup \psi$, where ϕ and ψ are constraints; or $\exists_x \phi$, where ϕ is a constraint and $x \in Var$. We extend this notation by allowing negation on the right of an entailment operator, for example, $\phi \sqsupseteq \neg\psi$ is true if and only if $\phi \sqsupseteq \psi$ is not. Other propositional operators are then defined from these, for example, $\phi \vee \psi \mathrel{\widehat{=}} \neg(\neg\phi \wedge \neg\psi)$ and $\phi \rightarrow \psi \mathrel{\widehat{=}} \neg\phi \vee \psi$. We will continue to use the meta-variables ϕ and ψ to refer to constraints throughout this paper. We also use $vars(\phi)$ to refer to the free variables that occur in ϕ; that is, the variables referenced in ϕ that are not hidden using $\exists$.

2.2 Modelling Protocols

The $\mathcal{RASA}$ protocol specification language is based on process algebras, and resembles languages such as CSP [4]. However, we add the notion of *state* to the language. State is useful, because it allows us to build up the meaning of protocols compositionally, for example, the effect of sending two messages sequentially is the effect of sending the second in the state that results after sending the first. The final outcome of the protocol is the end state.

Definition 2.2. A *protocol specification* is a collection of protocol definitions of the format $N(x, \ldots, y) \mathrel{\widehat{=}} \pi$, in which $N, x, \ldots, y \in Var$, and π represents a protocol.

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

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

Protocols are defined using the two types of atomic protocol, and algebraic operators for building up compound protocols from these. The first atomic action/protocol is the empty action: $\psi \rightarrow \epsilon$. This specifies that if the precondition ψ is provable from the current state, then no message sending is required.

The second atomic protocol is message sending, $\psi \xrightarrow{c(i,j).\phi_m} \psi'$. This is read as follows: if the precondition, ψ, is provable (using $\sqsupseteq$) from the current state, then the agent i is permitted to send the message ϕ_m to agent j. The effect of this message on the state is specified by the postcondition, ψ'. Omitting the prefix $c(i,j)$ from a message template implies that ϕ_m is an action other than a protocol. In this paper, we omit (i,j) when we do not care who the sender and receiver of the message is. We use the notion of *inertia* in calculating the new state from the postcondition; that is, any variables in ψ' are constrained by ψ' in the new state, and any other variables in the state are left unchanged. We allow agents to send the message ϕ'_m, such that $\phi'_m \sqsupseteq \phi_m$, so that agents can further constrain the values of the messages; thus, ϕ_m is only a template of the message. For example, consider the following atomic protocol:

$$X \geq Bid \xrightarrow{c.bid(X)} Bid = X.$$

in which the sender is bidding on an item, and Bid and X are variables. As part of the interaction, the sender would like to instantiate X to its actual bid, for example, to 10. Therefore, it would send the message $bid(X) \sqcup X = 10$, which constrains the message template by adding further information: that its bid is 10. If the pre-state is ϕ, then the post-state is calculated by taking $\psi' \sqcup \phi'_m$, and conjoining it with $\exists_z \phi$, where $z = vars(\psi' \sqcup \phi_m)$.

Compound protocols can be built up from these atomic protocols. If π_1 and π_2 are two protocols, then the following are also protocols: the protocol $\pi_1; \pi_2$, which represents sequential concatenation, such that π_1 is executed, followed by π_2; the protocol $\pi_1 \cup \pi_2$, which represents a choice between π_1 and π_2; and the protocol $\mathbf{var}_x^{\psi} \cdot \pi_1$, which is a protocol the same as π_1, except that a local variable x is available over the scope of π_1, but with the constraints ψ on x remaining unchanged throughout that scope. Any variable x already in the state is out of scope until π_1 finishes executing. In addition, $\mathcal{RASA}$ supports the referencing of protocols via their names. That is, for a protocol definition $N(x) \mathrel{\widehat{=}} \pi_1$, one can reference this from within another protocol using $N(y)$, where $y \in Var$.

Using such a definition, one can express protocols as sets of possible interactions, in which interactions are sequences of triples containing legal pre-states, messages, and post-states. The meaning of a protocol is derived from the post-states; that is, the messages themselves do not specify meaning, but are use only as a way to communicate information and constrain the post-states.

2.3 Reasoning about Protocols

$\mathcal{RASA}$ defines a logic for reasoning about protocols. By logic, we mean a syntax, semantics, and proof system. The logic is concerned with protocol outcomes; that is, the state of the protocol after it is executed. For this reason, we have adapted a version of propositional dynamic logic [3] to tailor it to the $\mathcal{RASA}$ specification language, and derived a proof system that corresponds to the system for dynamic logic.

The syntax for a proposition in this logic is defined using the following grammar, assuming that ϕ_0 is a constraint in the underlying constraint language:

$$\phi \ ::= \ \phi_0 \mid \phi \wedge \phi \mid \neg\phi \mid [\pi]\phi$$

A formal semantics for this logic has been defined in [6]. Here, we discuss this briefly. Each proposition is evaluated under a state (that is, a constraint). If ψ_0 is this state, then ϕ_0 is true if and only if $\psi_0 \sqsupseteq \phi_0$. $\phi \wedge \psi$ and $\neg\phi$ are defined as conjunction and negation respectively. The interesting operator, $[\pi]\phi$, which is found in propositional dynamic logic, has the meaning that ϕ holds for every possible outcome of the protocol π. That is, no matter which interaction path is taken in the protocol π, the proposition ϕ will hold after the protocol has executed.

As we did with the underlying constraint language, we define other propositional operators. We also define another operator from dynamic logic: $\langle\pi\rangle\phi$, which is the dual of $[\pi]\phi$, and means that ϕ holds in at least one possible outcome of the protocol π. This is defined as shorthand for $\neg[\pi]\neg\phi$. That is, ϕ holds in at least one end state of π if and only if $\neg\phi$ does not hold in all of them. We use subscripts on the Greek letters ϕ and ψ to indicate something that is strictly a constraint; that is, ϕ_0 is a constraint, while ϕ can be a constraint or a dynamic logic proposition.

We use this logic to match protocols, as well as to define rules for deriving annotations on protocols (Sections 5 and 6). Using such a logic is beneficial to us because it has a sound and complete proof system [6], which we use to prove that our annotation rules are correct, and to establish the soundness and correctness of our methods.

3. PROTOCOLS, GOALS, AND MATCHES

Before we continue with our presentation, we first define what it means for a protocol to "match" a goal.

Given a goal, ϕ_G, a matching protocol is a protocol, π, in an agent's protocol library, such that, from some initial state, ψ_I, the following holds:

$$\psi_I \rightarrow \langle\pi\rangle\phi_G.$$

That is, from the initial state, at least one outcome of the protocol entails the goal.

We define a protocol to be *executable* when it contains at least one possible path of interaction. We note that for an executable protocol, π, the following holds:

$$[\pi]\phi_G \rightarrow \langle\pi\rangle\phi_G.$$

That is, if ϕ_G holds for all outcomes, of which there is at least one (because it is executable), then it must hold for at least one outcome. From this, we can say that for an executable protocol, π, and goal ϕ_G, if $[\pi]\phi_G$, then π is a match for ϕ_G.

We explicitly distinguish this from the former type of match by calling the latter a *strong match*, and the former a *weak match*. Clearly, all strong matching protocols are also weak matching protocols.

The motivation for the work in this paper is the need for agents to be able to identify, at runtime and for a given goal, which protocols in its library achieve that goal. In this paper, we assume that an agent will want to retrieve all matching protocols, weak or strong, and then deliberate over which one is the most suitable at that point using some additional criteria.

4. MATCHING PROTOCOLS VIA PROOF

Using the logic discussed in Section 2.3, we can define a straightforward method for identifying whether a protocol achieves a given goal. For a goal ϕ_G, and an initial state, ϕ_I, an agent can find a weak match by proving the proposition $\psi_I \rightarrow \langle\pi\rangle\phi_G$, and a strong match by proving $\psi_I \rightarrow [\pi]\phi_G$.

However, there is an obvious downside to this approach for agents whose goals change regularly: performing such a calculation for every goal and every protocol is costly if an agent's goals continue to change. A proof of the form $[\pi]\phi_G$ is costly to perform. Caching the proof for further matching could offer some improved efficiency, but the caching is valid only for the initial state ψ_I, so a proof would have to be performed any time the initial state changed. For an agent whose goals remain static, or have a slow rate of change, this approach could be quite useful, but for other agents, the approach is impractical, and we prefer an approach that does not require an agent to discharge these proofs so often.

An additional downside of this approach is that it does not (necessarily) annotate protocols, therefore, it does not provide agents with any guidance as to the interactions that best achieve their goals. There is no restriction that would prevent agents from doing so, however, as with matching protocols via proof, the annotations will only be relevant to the specific goals and initial states.

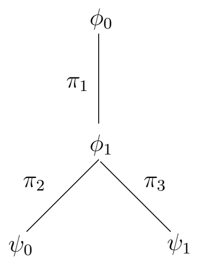
5. ANNOTATING AND MATCHING PROTOCOLS VIA DERIVATION

A key reason for deriving the logic for $\mathcal{RASA}$ is to annotate protocols with their outcomes to help an agent form a strategy at runtime. It is clear that agents could annotate protocols with meta-information, such as from where the protocol came, how much it trusts the protocol, *et cetera*, but in this paper, we are concerned only with annotations that can be derived from the protocol specification itself. Although this information can be derived at runtime, the overhead of calculating outcomes is impractical for more than a handful of protocols; therefore, we want to reduce the amount of calculation for agents at runtime as much as possible.

We express annotations using the logic presented in Section 2.3, for example, $[\pi]\phi$ is an annotation on protocol π specifying that ϕ holds in all outcomes, and for all initial states. However, we introduce one strict condition: outcomes must be specified using the underlying constraint language, not the dynamic logic. That is, for an annotation $[\pi]\phi$, we want ϕ to be a constraint — not to contain any expressions using $[\ ]$ or $\langle\rangle$. We assume that the goals of agents are specified using (or at least translated to) the constraint language. If ϕ contains a dynamic logic operator, the agent can still use this annotation, but it will need to derive the corresponding constraint that satisfies ϕ, which is an unnecessary overhead.

Our goal is to annotate, for each protocol in a protocol library, not only the outcomes that it achieves, but the outcomes of the sub-protocols that make up this protocol. For example, Figure 1 shows an abstract syntax tree of a protocol with ϕ_0 being the starting state, and the annotations that we may want to derive. This protocol contains two paths: (1) π_1 followed by π_2; and (2) π_1 followed by π_3. From the annotations, an agent could calculate that the entire protocol achieves ψ_0 for some, but not all, of its paths, and that ψ_0 is achieved by the path $\pi_1; \pi_2$.

We use the term "annotation" because the formula are related to the nodes of the abstract syntax trees, and would perhaps be attached to the definition of the protocol. For



$$\begin{array}{lll}
\phi_1 & : & [\pi_2]\psi_0 \\
 & & [\pi_3]\psi_1 \\
\\
\phi_0 & : & [\pi_1]\phi_1 \\
 & & [\pi_1;\pi_2]\psi_0 \\
 & & [\pi_1;\pi_3]\psi_1 \\
 & & \langle\pi_1;(\pi_2\cup\pi_3)\rangle\psi_0 \\
 & & \langle\pi_1;(\pi_2\cup\pi_3)\rangle\psi_1
\end{array}$$

Figure 1: Protocol and its annotations

the example in Figure 1, the first two annotations are on the node ϕ_1, while the rest are on the node ϕ_0.

In this section, we present a method for annotating protocols with their outcomes, and then matching protocols based on these annotations.

5.1 Annotating Protocols

Instead of proving annotations for the specific goals of an agent, it may be more efficient if the agent derives annotations directly from the protocol definition. It would not be possible to annotate the protocol with every constraint that could hold in an end state, so instead, we document the outcomes as *maximal postconditions*. By maximal, we mean a constraint such that every end state satisfies that constraint, and that it is satisfied only by those end states. Therefore, every end state would entail the maximal postcondition. We also document the maximal postconditions of the sub-protocols that make up a compound protocol, to help agents choose between multiple protocols that each achieve a goal, and to help them reason about which paths of interaction best achieve their goals.

The advantage of this approach is that each annotation needs to be calculated only once, and it remains valid for the lifetime of the system/protocol. In fact, annotations can be added to protocols and shared between agents, therefore not requiring each agent to derive them.

To derive annotations, we specify a set of *annotation rules*. When receiving a new protocol, an agent applies these rules to the protocol, and its sub-protocols, adding the annotations. Annotation rules are specified as theorems in the $\mathcal{RASA}$ logic. Each rule is of the form $\phi \wedge \phi' \wedge \ldots \rightarrow \psi$, resembling a Horn clause, and should be read: *if ϕ,ϕ',... can be derived as maximal postcondition annotations, then ψ is a maximal postcondition annotation.* Each of the annotation rules in this section has been proved using the axiom system defined in [6].

5.1.1 Global Annotations

The most straightforward of the rules is that if a protocol π achieves ϕ_0 for all outcomes, and has at least one outcome, then it must achieve ϕ_0 for at least one outcome.

$$[\pi]\phi_0 \wedge \langle\pi\rangle\text{true} \quad\rightarrow\quad \langle\pi\rangle\phi_0$$

This is similar to the D axiom found in many modal logics. Note that, for a protocol, π, that is not executable, for example, a protocol whose precondition is false, the proposition $[\pi]\phi$ will hold for any ϕ; that is, for all end states, of which there are none, ϕ holds. Therefore, if false is provable at all end states, either there are no end states, or all end states are equivalent to false, which is represented as $\neg[\pi]\neg$false, and is equivalent to $\langle\pi\rangle$true from the definition of $\langle\ \rangle$. Therefore, we annotate with $\langle\pi\rangle\phi_0$ only if $\langle\pi\rangle$true.

5.1.2 Annotating $\psi_0 \rightarrow \epsilon$

An empty protocol receives only one annotation:

$$[\psi_0 \rightarrow \epsilon]\psi_0$$

This rule contains no premise. An empty protocol does not change the state, therefore, the postcondition is anything that is true before the execution of the protocol. In the case of the maximal postcondition, the only information that we can derive is that the precondition must hold for the protocol to execute, therefore, the maximal postcondition is the precondition ψ_0.

5.1.3 Annotating $\psi_0 \xrightarrow{c.\phi_m} \psi_0'$

An atomic protocol receives only one annotation:

$$\begin{array}{l}
\langle\psi_0 \xrightarrow{c.\phi_m} \psi_0'\rangle\text{true} \quad\rightarrow \\
\qquad [\psi_0 \xrightarrow{c.\phi_m} \psi_0'](\psi_0' \sqcup \phi_m \sqcup \exists_{vars(\psi_0' \sqcup \phi_m)}\psi_0)
\end{array}$$

The premise of this rule insists that the protocol is executable. The maximal postcondition of an atomic protocol $\psi_0 \xrightarrow{c.\phi_m} \psi_0'$ is the constraint that corresponds to the $\psi_0' \sqcup \phi_m$ (the constraint that specifies the postcondition), conjoined with the constraints on variables from the precondition ψ_0 that have not been changed by the protocol; that is, those that are not free in ψ_0' or ϕ_m.

5.1.4 Annotating $\pi_1;\pi_2$

Two annotation rules are associated with sequentially composed protocols:

$$[\pi_1][\pi_2]\phi_0 \quad\rightarrow\quad [\pi_1;\pi_2]\phi_0$$

$$\langle\pi_1\rangle\langle\pi_2\rangle\phi_0 \quad\rightarrow\quad \langle\pi_1;\pi_2\rangle\phi_0$$

The first says that if, for every end state of π_1, the maximal postcondition of π_2 is ϕ_0, then the maximal postcondition of $\pi_1;\pi_2$ is also ϕ_0. The second is similar, but for the $\langle\rangle$ operator. Note that, as a result of the global annotation rule specified in Section 5.1.1, an agent will also derive the annotation $\langle\pi_1;\pi_2\rangle\phi_0$ from $[\pi_1]\langle\pi_2\rangle\phi_0$ or $\langle\pi_1\rangle[\pi_2]\phi_0$.

To derive $[\pi_1][\pi_2]\phi_0$ (and similarly for $\langle\rangle$), one must derive the maximal postcondition of π_2 under the initial state that is the maximal postcondition of π_1. That is, for the annotation $[\pi_1]\psi_0$, derive the maximal postcondition for $\psi_0 \rightarrow [\pi_2]\phi_0$. This can be expressed as the following rule.

$$[\pi_1]\psi_0 \wedge \psi_0 \rightarrow [\pi_2]\phi_0 \quad\rightarrow\quad [\pi_1;\pi_2]\phi_0$$

$$\langle\pi_1\rangle\psi_0 \wedge \psi_0 \rightarrow \langle\pi_2\rangle\phi_0 \quad\rightarrow\quad \langle\pi_1;\pi_2\rangle\phi_0$$

These rules are read differently to others, because it is unlikely that there will be annotations $\psi_0 \rightarrow [\pi_2]\phi_0$ or $\psi_0 \rightarrow \langle\pi_2\rangle\phi_0$. In these cases, the rules are read that if $[\pi_1]\psi_0$ (respectively $\langle\pi_1\rangle\psi_0$) is an annotation, and then calculate the maximal postcondition, ϕ_0, of π_2 under the state ψ_0. ϕ_0 is then the maximal postcondition of $\pi_1;\pi_2$.

5.1.5 Annotating $\pi_1 \cup \pi_2$

Choice protocols are the most difficult to annotate because they offer more than one path, each with a maximal postcondition, and our method must derive information that covers all of these paths. We propose the following three rules, each which is straightforward to prove.

$$\begin{array}{lcl}\langle\pi_1\rangle\phi_0 & \rightarrow & \langle\pi_1\cup\pi_2\rangle\phi_0\\ \langle\pi_2\rangle\phi_0 & \rightarrow & \langle\pi_1\cup\pi_2\rangle\phi_0\\ [\pi_1]\phi_0\wedge[\pi_2]\psi_0 & \rightarrow & [\pi_1\cup\pi_2](\phi_0\vee\psi_0)\end{array}$$

However, the final rule above is not adequate as an annotation. Recall from the start of this section, that we restrict the annotations to propositions of the form $[\pi]\phi_0$, in which ϕ_0 is a constraint. Constraint stores cannot hold negations or disjunctions, and as a result, the application of this rule is non-trivial, and in many cases, deriving the maximal postcondition is not possible.

Despite this, we can still derive some information that is useful. For example, take the following choice protocol definition:

$$\begin{array}{lcl}N & \widehat{=} & x=1\xrightarrow{c.a(x)}x=y\sqcup x\in[0..7]\quad\cup\\ & & x=1\xrightarrow{c.b(x)}x\neq y\sqcup x\in[3..10]\end{array}$$

Despite the fact that these two postconditions are inconsistent with each other (because one contains $x=y$ and the other $x\neq y$), we can still derive common information. The maximal postcondition is $x\in[3..7]$, so we could derive the annotation $[N](x\in[3..7])$ — that is, we know that x will be in the range $[3..7]$ whichever path is taken. Such an annotation could prove useful for an agent.

However, except for the most trivial cases (e.g. where $\phi_0\sqsupseteq\psi_0$ or $\psi_0\sqsupseteq\phi_0$), calculating this is beyond the means of any constraint solver known to the authors, because constraint solvers are not designed to find the most general constraint store that is consistent with two unrelated, and possible inconsistent, constraints.

To find a solution such as the one above, we propose an approach in which an agent analyses different parts of the constraints. This does not necessarily find the best solution, but it can derive a constraint that satisfies parts of both ϕ_0 and ψ_0.

To do this, we consider the variables in the two constraints. Clearly, any maximal postcondition of a choice must only reference variables that are in both ϕ_0 and ψ_0 — any variables in only one will not be constrained in the maximal postcondition of the choice protocol. Therefore, what we aim to do is derive a set of constraints, each of which is relevant to only a subset of the variables. For example, if we take the two postconditions from above and hide y from both, then the constraint $\exists_y(x=y\sqcup x\in[0..7])\sqcup\exists_y(x\neq y\sqcup x\in[3..10])$ is reduced to the constraint $x\in[3..7]$, which is the maximal postcondition relative to x, so we can use this as an annotation. If we hide x instead, then the resulting constraint is unsatisfiable, so this is not considered as an annotation.

In this example, we examine the variables in ϕ_0 and ψ_0, and look at constraints that result from hiding some of these variables. It is not always useful to hide only one variable, otherwise we lose information about the constraints between variables. Instead, we hide different combinations of variables. To obtain the relationships between all variables, one can take an approach that, for every set of variables $Z\subset vars(\phi_0)\cap vars(\psi_0)$, check if the constraint $\exists_Z\phi_0\sqcup\exists_Z\psi_0$ is satisfiable — something which we hope is straightforward for any constraint solver to check. If this is satisfiable, then we add the annotation $[\pi_1\cup\pi_2](\exists_Z\phi_0\sqcup\exists_Z\psi_0)$.

We note that this approach is sound but not complete. That is, such an approach will always produce annotations that are correct, but they are not guaranteed to be annotations containing the maximal postcondition. For example, take the following definition:

$$P\quad\widehat{=}\quad x=1\xrightarrow{c.a(x)}x=1\quad\cup\quad x=1\xrightarrow{c.b(x)}x=2$$

Here, either $x=1$ or $x=2$ will hold in the outcome. Therefore, the proposition $[P]x\in[1..2]$ is valid, but our annotation rules fail to derive this, because $x=1\sqcup x=2$ is not satisfiable, and as a result, we derive only the annotations $\langle P\rangle x=1$ and $\langle P\rangle x=2$ (using the first two rules), even though the annotation $[P]x\in[1..2]$ is the annotation containing the maximal postcondition.

While this approach is sound and may prove useful, the approach for calculating it is somewhat undesirable, because for n variables, we have 2^n-1 different combinations to check, and a worst case of 2^n-1 annotations. For a large n, deriving annotations is costly, as is searching through annotations to match protocols. For large n, agents could selectively choose to calculate annotations based on the variables in their current goals. That is, if they generally have goals that related to certain variables in the system, then calculate only the annotations for those variables.

However, all is not lost in this derivation. Theorem 5.1 (Section 5.2.2) shows that the final rule does not need to be applied to find a suitable protocol, provided that the first two rules for choice protocols are applied. While not necessary, applying the above rule may reduce the runtime complexity of finding a suitable protocol. This is discussed further in Section 5.2.

5.1.6 Annotating $\mathbf{var}_x^{\psi_0}\cdot\pi$

Annotation of variable declarations is straightforward. If the maximal postcondition of the sub-protocol π is ϕ_0, then the maximal postcondition of $\mathbf{var}_x^{\psi_0}\cdot\pi$ is ϕ_0 with the value of x constrained to the same value as it is before execution. For this we introduce a new variable x_0 and constrain it to be equal to x. In the postcondition, we then constrain that x must be equal to x_0. We know that the constraints on x_0 have not been changed by π because it is a fresh variable and therefore not referenced in in π. Finally, the variable x_0 is hidden using the $\exists$ operator, because x_0 is not part of the maximal postcondition.

$$\begin{array}{lcl}\psi_0\rightarrow[\pi]\phi_0 & \rightarrow & x=x_0\rightarrow[\mathbf{var}_x^{\psi_0}\cdot\pi]\exists_{x_0}(x=x_0\sqcup\phi_0)\\ & & \text{where } x_0 \text{ is fresh}\end{array}$$

5.1.7 Annotating $\pi_1;(\pi_2\cup\pi_3)$

Protocols of the form $\pi_1;(\pi_2\cup\pi_3)$ (and $(\pi_1\cup\pi_2);\pi_3$) are special cases of protocols. These protocols form the basis of *paths* in the protocols, because the $\cup$ operator introduces a branching in the abstract syntax tree, and ; introduces a concatenation of protocol paths. Theories of business process modelling often refer to these as *or-splits* and *or-joins* respectively, because they represent the splitting and joining of single traces with multiple traces respectively.

We can derive important annotations from protocols of this format, mainly those that annotate the outcomes achieved by individual paths in protocols. This is important because it gives agents additional information for choosing protocols that achieve their goals, as well as choosing which paths best achieve their goal. The rules specified so far in this section fail to take into account these special cases.

We specify two annotation rules for protocols of this form:

$$\langle \pi_1; \pi_2 \rangle \phi_0 \quad \rightarrow \quad \langle \pi_1; (\pi_2 \cup \pi_3) \rangle \phi_0$$
$$\langle \pi_1; \pi_3 \rangle \phi_0 \quad \rightarrow \quad \langle \pi_1; (\pi_2 \cup \pi_3) \rangle \phi_0$$

These say that if the protocols $\pi_1; \pi_2$ or $\pi_1; \pi_3$ have at least one end state in which ϕ_0 is the maximal postcondition, then the composite protocol $\pi_1; (\pi_2 \cup \pi_3)$ also has at least one end state such that ϕ_0 holds. This is clear from the semantics of $\langle\rangle$, and is provable by showing that $\pi_1; (\pi_2 \cup \pi_3)$ is equivalent to $\pi_1; \pi_2 \cup \pi_1; \pi_3$, and using the annotation rules from Section 5.1.5.

Note that the annotations produced on the right hand side of the rules will be derived from the rules for choice and sequential composition, but the annotations in the premise will not. Therefore, these two rules exist solely to document that one must annotate $\pi_1; \pi_2$ and $\pi_1; \pi_3$.

We also note that protocols of the form $(\pi_1 \cup \pi_2); \pi_3$ have similar annotation rules, however, these should be clear to the reader, so they are omitted.

5.2 Using Derived Annotations

Agents use annotations to search for protocols that achieve their goals, and to guide them through the execution of a protocol. In this section, we present a method for determining whether an annotated protocol achieves a given goal. The process of selecting a protocol, should there be more than one such match, and the process of reasoning about interaction are more likely to be varied between different agent implementations, so they are not discussed here. However, the process of matching protocols is more straightforward and likely to follow a similar pattern between implementations.

If an agent has a goal, ϕ_G, then it must find a protocol that achieves ϕ_G. To do this, it could either search through annotations of protocols until it finds a protocol that satisfies its needs, or search through all protocols and then make a choice if multiple protocols achieve its needs. In this section, we focus only on the process of assessing whether a protocol achieves the goal — that is, matching protocols — assuming that the annotations have been derived using the rules from Section 5.1. This method is guaranteed to find a protocol, if one exists.

5.2.1 A First Attempt

For a goal ϕ_G and protocol π, take the annotations of π and then perform the following:

1. For every annotation of the format $[\pi]\phi_A$, test $\phi_A \sqsupseteq \phi_G$; that is, test whether the maximal postcondition satisfies the goal. If this entailment is successful, and the initial state under which the protocol will be executed satisfies its precondition, add π to the list of strong matching protocols.

2. If step 1 fails, for every annotation of the format $\langle\pi\rangle\phi_A$, test $\phi_A \sqsupseteq \phi_G$. If this entailment is successful, and the initial state under which the protocol will be executed satisfies its precondition, add π to the list of weak matching protocols.

This is a reasonable way to match protocols, however, it does not guarantee that an agent will find a protocol that satisfies its goal, even if one exists. For example, take a situation in which an agents goal is $x = 1$. We have an annotation $[\pi](x \in [1..5] \sqcup x = y)$, but the above process fails to match this because $x \in [1..5] \sqcup x = y$ does not entail $x = 1$. However, it may be the case that one end state satisfies $x = 1$, but because the annotations document only *maximal* postconditions, there is no annotation such that $x = 1$.

Clearly, adding annotations for every possible postcondition is at best, expensive, and at worst, impossible. Instead, we add an extra step to the process which is not expensive, and which guarantees that we find a matching protocol.

5.2.2 A Complete Approach

A complete approach requires us to assess the maximal postconditions in more detail. To do this, we perform an additional test for every annotation, while still performing the naive approach. For a goal ϕ_G and protocol π, take the annotations of $[\pi]\phi_A$ and $\langle\pi\rangle\phi_A$ and then perform the following:

1. Test whether ϕ_G and ϕ_A are consistent with each other; that is, $\phi_G \sqcup \phi_A \not\sqsupseteq$ false. If the goal and postcondition are consistent, then it may be that there is an outcome stronger than the maximal postcondition that satisfies our goal, as in the example above. If not, then π can never achieve the goal ϕ_G, so do not continue.

2. If step 1 succeeds, attempt to prove $\psi_I \rightarrow [\pi]\phi_G$, in which ψ_I is the initial state in which the protocol will be executed. If this is provable, add π to the list of strong matching protocols.

3. If step 1 succeeds and step 2 fails, attempt to prove $\psi_I \rightarrow \langle\pi\rangle\phi_G$, in which ψ_I is the initial state in which the protocol will be executed. If this is provable, add π to the list of weak matching protocols.

Using our example above, if an agent has a protocol with the maximal postcondition $x \in [1..5]$, and a goal $x = 1$, then it can calculate in a straightforward manner that $x = 1$ is consistent with $x \in [1..5]$. From here, it tries to prove if it is possible that $x = 1$ in all outcomes of the protocol: $[\pi]x = 1$. If this holds, then its possible for the agent to achieve its goals with this protocol. If not, the it tries to prove $\langle\pi\rangle x = 1$. The steps of proving $[\pi]x = 1$ (and $\langle\pi\rangle x = 1$) are in fact necessary, because it may be possible that the goal and maximal postconditions are consistent, but that the goal is not achieved by the protocol. For example, consider a case in which $x \in [1..5]$ is also the only constraint that holds for x in all outcomes of a protocol:

$$N \quad \widehat{=} \quad x = 0 \xrightarrow{c.a(y)} y = 1 \sqcup x \in [1..5]$$

Here, because the sender cannot constrain the value of x (it not being part of the message), the only postcondition is $y = 1 \sqcup x \in [1..5]$, so $x = 1$ is not achieved. However, consider the following alternate protocol, in which $y = 1$ is replaced with $y = x$ in the postcondition:

$$N \quad \widehat{=} \quad x = 0 \xrightarrow{c.a(y)} y = x \sqcup x \in [1..5]$$

An agent can constrain the value of x , and therefore $x = 1$ is a possible end state, and $\langle N\rangle x = 1$ could be proved.

We also note that attempting the proof $[\pi]\phi_G$ is beneficial, rather than only proving $\langle\pi\rangle\phi_G$. That is, it is possible that $\phi_A \not\sqsupseteq \phi_G$, but that all outcomes hold for ϕ_G, which may initially seem counter-intuitive. This is best demonstrated with the following example:

$$P \quad \widehat{=} \quad x = 1 \xrightarrow{c.a(x)} x = 1 \quad \cup \quad x = 1 \xrightarrow{c.b(x)} x = 2$$

Here, either $x = 1$ or $x = 2$ will hold in the outcome. Therefore, the proposition $[P]x \in [1..2]$ is valid, but, as discussed in Section 5.1, our annotation rules fail to derive this. As a result, the goal $x \in [1..2]$ would not be matched, but the proof $[P]x \in [1..2]$ would succeed.

Although the approach outlined in this section requires a proof to be performed at runtime, similar to the matching-via-proof method described in Section 4, checking whether or not the goal is consistent with the maximal postcondition before attempting the proof would eliminate the need to do these proofs for many protocols. This is advantageous because proofs in the dynamic logic are more computationally expensive than the matching rules, especially for large protocols.

THEOREM 5.1. *This annotation and matching method is sound and complete.*

PROOF. To prove this theorem, we need to demonstrate that any matched protocol achieves the goal, and that only protocols that achieve the goal are matched.

Soundness is straightforward to prove. For a goal ϕ_G, and annotations $[\pi]\phi_A$ and $\langle\pi\rangle\phi_A$, a match is noted when at least one of the following two conditions hold:

1. $\phi_A \sqsupseteq \phi_G$; or
2. $\phi_G \sqcup \phi_A \not\sqsupseteq$ false and $[\pi]\phi_G$ (respectively $\langle\pi\rangle\phi_G$).

Part (1) is trivially sound from modus ponens and the modal logic inference rule of necessitation; Part (2) is trivially sound from $[\pi]\phi_G$ (or $\langle\pi\rangle\phi_G$). Additionally, annotations are added using the rules from Section 5.1, which are valid theorems of our logic. Therefore, the method is sound.

Completeness is less straightforward to prove. For completeness, we must prove that, for any true formula $[\pi]\phi_G$ (or $\langle\pi\rangle\phi_G$), the rules defined in Section 5.1 will produce an annotation $[\pi]\phi_A$ (respectively $\langle\pi\rangle\phi_A$), such that either:

1. $\phi_A \sqsupseteq \phi_G$; or
2. $\phi_G \sqcup \phi_A \not\sqsupseteq$ false and $[\pi]\phi_G$ (respectively $\langle\pi\rangle\phi_G$).

This is proved via *reductio ad absurdum.* It is trivially valid that every valid protocol receives an annotation, and from the soundness proof, we know that each annotation is correct, therefore, we know there is a correct annotation $\langle\pi\rangle\phi_A$. We prove this only for the case of $\langle\pi\rangle\phi_A$, because from the global annotation rule, this will also be valid for $[\pi]\phi_A$, for executable π.

If our method is incomplete, then there exists ϕ_G such that $\langle\pi\rangle\phi_G$ holds, but that:

1. $\phi_A \not\sqsupseteq \phi_G$; and
2. $\phi_G \sqcup \phi_A \sqsupseteq$ false or $\neg\langle\pi\rangle\phi_G$.

That is, π is not matched as achieving ϕ_G, despite the fact that it is a valid postcondition. It suffices to prove that item 2 above is a contradiction. Firstly, $\neg\langle\pi\rangle\phi_G$ contradicts the assumption that ϕ_G is a valid postcondition.

Secondly, if ϕ_A is the maximal postcondition of at least one path in π, then it must be that any postcondition of that path is consistent with ϕ_A. It holds trivially from the annotation rules that every path in a protocol receives an annotation, therefore, it cannot be that each annotation on π is inconsistent with ϕ_G if ϕ_G is a valid postcondition.

If ϕ_A is not the maximal postcondition (recall from Section 5.1.5 that the final annotation rule for choice protocols does not necessarily derive the maximal postcondition), then it may be that the case that an annotation on ϕ_A is inconsistent with the goal ϕ_G. However, the third rule from Section 5.1.5 is not necessary for completeness, because the first two rules cover such a case: if $\langle\pi_1 \cup \pi_2\rangle\phi_G$ is true, then is must be that $\langle\pi_1\rangle\phi_G$ or $\langle\pi_2\rangle\phi_G$, one of which will be derived from the first two rules. Therefore, the other annotations rules defined in Section 5.1.5 will have annotated $\pi_1 \cup \pi_2$ with their maximal postconditions, one of which must be consistent with ϕ_G, and therefore the protocol will be matched. □

From this, we see that the final annotation rule defined in Section 5.1.5 is redundant. However, it may be useful because in some cases, it can prevent an agent from having to discharge proofs for propositions $[\pi]\phi_G$ and $\langle\pi\rangle\phi_G$. Whether the rule is used would clearly be a policy of individual agents.

Such a decision is specific to the strategy of the agents, not the protocol itself, and therefore it is out of scope of this paper.

6. ANNOTATING AND MATCHING PRE/POSTCONDITION MODELS

When using our framework, we often specify protocols as precondition/postcondition models; that is, models that specify a relationship between pre-states and post-states of protocols. The semantics of atomic protocols (and therefore compound protocols) does not support specifying postconditions as a relation with the pre-state. For example, consider a case in which we want to increment the integer value of a variable, x. The only way to specify this is to specify that the value of x is 1 greater than before the message was sent, which is not possible using an atomic protocol; the postcondition merely represents a constraint between state variables, with no way of referencing pre-state values. However, we use the variable declaration operator to simulate this behaviour:

$$N \quad \widehat{=} \quad \mathbf{var}_{x_0}^{x_0=x} \cdot x < 10 \xrightarrow{c.q(x)} x = x_0 + 1$$

Recall that the constraints placed on a locally declared variable are maintained throughout its entire scope. Therefore, the constraints on x_0 in the postcondition are that it equals the constraints on x in the pre-state. If we were to execute this message sending in the state $x = 1$, then the postcondition would resolve to the following: $\exists_x(x = 1 \sqcup x_0 = x) \sqcup x = x_0 + 1$. The only solution for this is $x_0 = 1 \sqcup x = 2$. The scope of the variable x_0 would end, and the post-state would be $x = 2$.

Annotating protocols using the rules in Section 5.1 would annotate this correctly, however, we would lose all information about the relationship between the pre-state and post-state. That is, we would have an annotation $[\pi]x < 11$, which contains no information about the relationship between the pre-state and $x < 11$. To preserve this relationship, we propose annotating variable declarations in a different manner. Note, this approach is only applicable if each atomic protocol is surrounded by a variable declaration; that is, the precondition/postcondition notion is used throughout the entire protocol.

This new approach to annotation requires us to explicitly consider the pre-state in the annotation rules. If we label our pre-state as ψ_0, then rule is as follows:

$$x_0 = x \wedge \psi_0 \rightarrow [\pi]\phi_A \quad \rightarrow \\ [\mathbf{var}_{x_0}^{x_0=x} \cdot \pi]\exists_{x_0}(\exists_x(\psi_0 \sqcup x_0 = x) \sqcup \phi_A)$$

where $vars(\phi_A) = x \cup x_0$. Therefore, this rule says that if ϕ_A is the maximal postcondition of π, then the maximal postcondition of the variable declaration protocol, under the initial state ψ_0, is calculated by assigning the pre-state occurrences of each variable x to a local variable x_0, and hiding this pre-state occurrence. Conjoin this with the postcondition, which specifies the relationship between each x and its pre-state counterpart, x_0. Finally, the local variables are hidden, because they are out of scope. When the agent reads this annotation, it substitutes in the initial state for ψ_0, giving it the postcondition.

As an example, take the protocol from above that increments a variable x. The annotation would be the following:

$$[\mathbf{var}_{x_0}^{x_0=x} \cdot \pi]\exists_{x_0}(\exists_x(\psi_0 \sqcup x_0 = x) \sqcup x = x_0 + 1).$$

Substituting in the current state, for example, $x = 1$, will result in the constraint

$$\exists_{x_0}(\exists_x(x = 1 \sqcup x_0 = x) \sqcup x = x_0 + 1)$$

which simplifies to $\exists_{x_0}(x_0 = 1 \sqcup x = x_0 + 1)$, which in turn, simplifies to $x = 2$.

This is a neat and useful approach, however, we note that the above annotation is in fact meta-level, because ψ_0 in this case is a variable in the annotation, rather than just a meta-variable used to represent a constraint. Substituting in the current state for ψ_0 will give us the precondition/postcondition annotation. Unless the constraint solver supports constraints as variables, this substitution must be done before asking the constraint solver to provide a solution.

This approach overcomes many weaknesses of the previous approaches: it does not require the agent to perform entire proofs, therefore reducing the overhead of the matching-via-proof method from Section 4; and it is not a general annotation, overcoming the problem of having to perform dynamic logic proofs if the goal entails the end state of the annotation, as described in Section 5.1. The obvious disadvantage to using this approach is that it is restricted only to protocols that are modelled using the precondition/postcondition approach.

7. RELATED WORK

As far as the authors are aware, there has been no investigation into the annotation and matching of first-class protocols to date. While such a topic may seem similar to classical planning techniques, the methods outlined in this paper are quite different to classical planning altogether. The goal in this paper is to take an existing definition of a protocol, and to summarise its outcomes; whereas planning aims to derive plans that are similar to the protocols themselves.

Our concept of protocol libraries is similar to plan libraries, such as those found in the Procedural Reasoning System [2]. However, there are some notable differences. Firstly, plan postconditions are generally not derivable from the plan themselves, but instead form part of the definition, and the authors are unaware of any types of plans that use concepts similar to maximal postconditions. In addition, the matching of plans to goals is usually testing whether the plan postcondition unifies with the goal, which is different to our approach of matching, due to us using constraint languages rather than logical languages, and using maximal postconditions.

Specification matching for component-based software engineering has been explored in the past [8]. Motivation for specification matching is similar to our motivation for protocol matching: to find a component that satisfies a specification. These approaches are considerably different to ours. Firstly, they consider only pre- and postcondition models, and, more importantly, unlike our approach, they do not consider cases in which the goal specification (the equivalent of an agent's goal) is stronger than the postcondition, because specifications are not written using maximal postconditions.

There are many approaches to specification of first-class protocols. Space prevents us from a discussion of these, however, [5] gives detailed presentation of the state-of-the-art in this area, and presents the advantages and disadvantages the different approaches, including $\mathcal{RASA}$. We assert that the general idea of annotation and matching could be applied to protocols specified in these languages.

8. DISCUSSION AND OTHER WORK

In this paper, we have presented three methods for annotating and matching first-class protocols specified in $\mathcal{RASA}$. Each of these methods has advantages and disadvantages. The method that we believe is the most useful involves annotating a protocol with its maximal postcondition, and then using these annotations to match the protocols that an agent will test at runtime to find a protocol that achieves a given goal.

In related work, we are investigating how protocol libraries can be stored and searched for efficient protocol matching. We are also investigating several other interesting aspects of first-class protocols, such as runtime composition of protocols, which would permit agents to compose new protocols if no protocol can be found that achieves a certain goal. In future work, we plan to assess other techniques for protocol annotation and matching, such as analysing properties other than outcomes, for example, the number of participants. To develop and test these ideas, we plan a prototype implementation in which agents negotiate the exchange of information using protocols specified using the $\mathcal{RASA}$ framework.

9. REFERENCES

[1] F. S. De Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. *ACM Transactions on Programming Languages and Systems*, 19(5):685–725, 1997.

[2] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 677–682, 1987.

[3] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, Cambridge, MA, USA, 2000.

[4] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[5] J. McGinnis and T. Miller. Amongst first-class protocols. In *Engineering Societies in the Agents World*

VIII, LNAI, 2007. (To Appear).

[6] T. Miller and P. McBurney. Executable logic for reasoning and annotation of first-class agent interaction protocols. Technical Report ULCS-07-015, University of Liverpool, Department of Computer Science, 2007.

[7] 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.

[8] A. Zaremski and J. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.