A Mechanism for Discovering Semantic Relationships among Agent Communication Protocols

Idoia Berges · Jesús Bermúdez · Alfredo Goñi · Arantza Illarramendi

Received: date / Accepted: date

Abstract One relevant aspect in the development of the Semantic Web framework is the achievement of a real inter-agent communication capability at the semantic level. Agents should be able to communicate with each other freely using different communication protocols, constituted by communication acts.

For that scenario, we introduce in this paper an efficient mechanism that presents the following main features:

- It promotes the description of the communication acts of protocols as classes that belong to a communication acts ontology, and associates to those acts a social commitment semantics formalized through predicates in the Event Calculus.
- It is sustained on the idea that different protocols can be compared semantically by looking to the set of fluents associated to each branch of the protocols. Those sets are generated using Semantic Web technology rules.
- It discovers the following types of protocol relationships: equivalence, specialization, restriction, prefix, suffix, infix and complement_to_infix.

Keywords Agents and the Semantic Web · Ontologies for agent systems · Agent communication protocols · Agent communication semantics

I. Berges
University of the Basque Country, Paseo Manuel de Lardizabal, 1, Donostia-San Sebastián, Spain
Tel.: +34-943015109
E-mail: idoia-berges@ikasle.ehu.es

J. Bermúdez
E-mail: jesus.bermudez@ehu.es

A. Goñi
E-mail: alfredo@ehu.es

A. Illarramendi
E-mail: a.illarramendi@ehu.es

1 Introduction

In the scenario promoted by the emerging Web, administrators of existing Information Systems distributed along the Internet are encouraged to provide the functionalities of those systems through agents that represent them. The underlying idea is to get real interoperation among those Information Systems in order to enlarge the benefits that users can get from the Web by increasing machines' processable tasks.

In general, communication among agents is based on the interchange of messages, which in this context are also known as communication acts. However, since Information Systems are developed independently, they incorporate different classes of communication acts as their Agent Communication Language (ACL) to the point that they do not understand each other. Moreover, protocols play an integral role in agents communication. A protocol specifies the rules of interaction between agents by restricting the range of allowed follow-up communication acts for each agent at any stage during a communicative interaction.

The importance of using communication protocols is widely recognized. However, that does not mean that all the agents should use the same universal protocols. In our opinion different communication protocols should coexist and every agent should have the opportunity of selecting the one that better fulfills its needs. Thus, one possible procedure that the administrators of the Information Systems could follow in order to implement the agents that will represent their systems is the following: They should first select, from a repository of standard protocols (more than one repositories may exist), those protocols that best suit the goals of their agents. Sometimes a single protocol will be enough, while other times it will be necessary to design a protocol as a composition of other protocols. Then, the selected protocols may be customized before they are embedded into the agents. This flexibility at the time of choosing and customizing communication protocols may have a drawback for a real communication among agents: it is likely that two protocols that have been designed for the same goal have different structure. Therefore, a reasoning process over the protocols embedded in the agents will be necessary to discover relationships –such as equivalence and restriction– between them.

In this paper we present a mechanism that discovers semantic relationships among protocols focusing on three main aspects: 1. The semantic representation of the main elements of the protocols, that is, of communication acts; 2. The semantic representation of the protocols' branches using Semantic Web Technology; and 3. The definitions of a set of semantic relationships between protocols.

Concerning the first point, communication acts that constitute protocols are described as classes that belong to a communication acts ontology, which we have developed, called CommOnt (see more details about the ontology in (Bermúdez et al, 2007)). The use of that ontology favours both the explicit representation of the meaning of the communication acts and the customization of existing standard protocols by allowing the use of particular communication acts that can be defined as specializations of existing standard communication acts. We have adopted the so called *social approach* (Singh, 1998, 2000) for expressing the intended semantics of those communication acts. According to the social approach, when agents interact they become involved in social commitments or obligations to each other.

With respect to the second point, one fundamental step of the mechanism is to extract the semantics associated to each protocol's branches. This step is sustained on the idea that the semantics of each branch is represented by a set of predicates generated

after the whole branch is analyzed. During this step Semantic Web technology rules are used.

Finally, a semantically founded set of basic relationships between protocols is defined, which later can be composed to express more complex relationships. Our mechanism is able to discover those complex semantic relationships taking into account the previously mentioned protocol analysis.

In summary, the main contributions of the mechanism presented in this paper are:

- It favours a flexible interoperation among agents of heterogeneous Information Systems that use different communication protocols and therefore avoids the need for prior agreement on how the messages are interchanged.
- It facilitates the customization of standard communication protocols by means of specialized communication acts that belong to specific ACL of Information Systems. The particular communication acts are described in an ontology.
- It provides a basis to reason about protocol relationships founded on the following basic relations: equivalence, specialization, restriction, prefix, suffix, infix and complement_to_infix. Moreover, notice that our approach allows the comparison between protocols in terms of the intended semantics of communication acts that appear in those protocols.

The rest of the paper is organized as follows: Section 2 provides background first on the communication ontology, which contains terms that correspond to communication acts that appear in the protocols, and next on the semantics associated to those acts. Section 3 explains how protocols and their semantics are described in our proposal and section 4 introduces the mechanism that makes a semantic analysis of protocols. Section 5 presents the formal definitions of a collection of semantic relationships that the mechanism can discover. A scenario where the proposed mechanism is applied is shown in section 6. Section 7 discusses different related works, and the conclusions appear in the last section.

2 Communication acts: how to represent their definitions and semantics

Among the different models proposed for representing protocols one which stands out is that of State Transition Systems (STS).

Definition 1 A *State Transition System* is a tuple (S, s_0, L, T, F), where S is a finite set of states, $s_0 \in S$ is the initial state, L is a finite set of labels, $T \subseteq S \times L \times S$ is a set of transitions and $F \subseteq S$ is a set of final states.

In our proposal we use STSs where transitions are labeled with communication act classes described in a communication acts ontology called CommOnt. That is to say, the set of labels L is a set of class names taken from that ontology. CommOnt is a OWL-DL ontology, therefore its term descriptions are founded on a Description Logic (DL). DLs are well designed for expressing conceptual knowledge and representing static structural knowledge. However, protocols exhibit a dynamic aspect due to the effect of communication acts. Thus, we have decided to represent the effects of actions through predicates in the Event Calculus(**?**).

In a nutshell, the CommOnt ontology describes structural and hierarchical aspects of communication acts and a knowledge base of Event Calculus predicates specifies

the effects of those communication acts. Terms that appear in the Event Calculus predicates come from names of classes and properties in CommOnt.

Reasoning over the effects of communication acts is achieved through an encoding of Event Calculus predicates and rules into SWRL (Semantic Web Rule Language)(**?**). SWRL is an extension of OWL-DL axioms to include Horn-like rules. Therefore, that encoding provides an executable mechanism that integrates satisfactorily the dual semantic representation of communication acts.

In the following three subsections we present respectively the main features of the CommOnt ontology, the dynamic semantics associated to communication acts, and an explanation of how these formalisms interact.

2.1 Main features of the CommOnt ontology

The goal of the CommOnt ontology is to facilitate the interoperation among agents that belong to different Information Systems. The leading categories of that ontology are: first, *communication acts* that are used for interaction by *actors* and that have different purposes and deal with different kinds of contents; and second, *contents*, which are the sentences included in the communication acts.

The main design criteria adopted for the communication acts category of the CommOnt ontology is to follow the *speech acts* theory (Austin, 1962), a linguistic theory that is recognized as the principal source of inspiration for designing the most familiar standard agent communication languages. Following that theory, every communication act is the sender's expression of an attitude toward some possibly complex proposition. A sender performs a communication act, which is expressed by a coded message, and is directed to a receiver. Therefore, a communication act has two main components. First, the attitude of the sender, which is called the *illocutionary force* (F), that expresses social interactions such as informing, requesting or promising. And second, the *propositional content* (p), which is the subject of the attitude. In CommOnt this *F(p)* framework is followed, and different kinds of illocutionary forces and contents leading to different classes of communication acts are described.

CommOnt is divided into three interrelated layers: *upper*, *standards* and *applications*, that group communication acts at different levels of abstraction. CommOnt terminology is described using OWL-DL, the description logic profile of the Web Ontology Language OWL(OWL, 2008). Therefore, communication acts among agents that commit to CommOnt have an abstract representation as individuals of classes that are specializations of a shared universal class of communication acts.

In the upper layer –according to Austin's speech acts theory– five upper classes of communication acts corresponding to *Assertives*, *Directives*, *Commissives*, *Expressives* and *Declaratives* are specified. In addition, the top class `CommunicationAct`[1] is defined, which represents the universal class of communication acts. Every particular communication act is an individual of this class. Moreover, some properties can be associated to this class. The most important properties of `CommunicationAct` are the content and the actors who send and receive the communication act. We show next the description of the class `CommunicationAct` in CommOnt:

$$\texttt{CommunicationAct} \sqsubseteq \texttt{=1 hasSender} \sqcap \forall \texttt{hasSender.Actor} \sqcap \forall \texttt{hasReceiver.Actor} \sqcap \forall \texttt{hasContent.Content}$$

[1] `This type` style refers to terms specified in the ontology.

There are some other properties related to the context of a communication act such as the conversation in which it is inserted or a link to the domain ontology that includes the terms used in the content, but those details are out of the scope of this paper. In addition to the principal classes `Assertive`, `Directive`, `Commissive`, `Expressive` and `Declarative`, some other interesting subclasses are also defined. For example:

$$\begin{aligned}
\texttt{Request} &\sqsubseteq \texttt{Directive} \sqcap \exists \texttt{hasContent.Command} \\
\texttt{Accept} &\sqsubseteq \texttt{Declarative} \\
\texttt{Inquiry} &\sqsubseteq \texttt{Directive} \sqcap \exists \texttt{hasContent.ReportAct} \\
\texttt{Responsive} &\sqsubseteq \texttt{Assertive} \sqcap \exists \texttt{inReplyTo.(Request} \sqcup \texttt{Inquiry)} \\
\texttt{Inform} &\sqsubseteq \texttt{Assertive}
\end{aligned}$$

A standards layer extends the upper layer of the ontology with specific terms that represent classes of communication acts of general purpose agent communication languages, like those from KQML(KQM, 1993) or FIPA-ACL(FIPA, 2005). Although the semantic framework of those agent communication languages may differ from the semantic framework adopted in CommOnt, in our opinion enough basic concepts and principles are shared to such an extent that a commitment to ontological relationships can be undertaken in the context of the interoperation of Information Systems.

With respect to FIPA-ACL, we can observe that it proposes four primitive communicative acts: *Confirm*, *Disconfirm*, *Inform* and *Request*. The terms `FIPA-Confirm`, `FIPA-Disconfirm`, `FIPA-Inform` and `FIPA-Request` are used to represent them as classes in CommOnt. Furthermore, the rest of the FIPA communicative acts are derived from these mentioned four primitives. Following some examples of descriptions can be found:

$$\begin{aligned}
\texttt{FIPA-Request} &\sqsubseteq \texttt{Request} \\
\texttt{FIPA-Query-If} &\sqsubseteq \texttt{Inquiry} \sqcap \texttt{FIPA-Request} \sqcap \texttt{=1 hasContent} \sqcap \\
&\quad \forall \texttt{hasContent.InformIf} \\
\texttt{FIPA-Inform} &\sqsubseteq \texttt{Inform}
\end{aligned}$$

Analogously, communication acts from KQML can be analyzed and the corresponding terms can be specified in CommOnt. It is vital for the interoperability aim to have the ability to specify ontological relationships among classes of different standards. For example:

$$\begin{aligned}
\texttt{KQML-Ask-If} &\equiv \texttt{FIPA-Query-If} \\
\texttt{KQML-Tell} &\equiv \texttt{FIPA-Inform} \\
\texttt{KQML-Achieve} &\equiv \texttt{FIPA-Request} \sqcap \exists \texttt{hasContent.Achieve}
\end{aligned}$$

Finally, it is often the case that every Information System uses a limited collection of communication acts that constitutes its particular agent communication language. The applications layer reflects the terms describing communication acts used in such particular Information Systems. The applications layer of the CommOnt ontology provides a framework for the description of the nuances of such communication acts. Some of these communication acts can be defined as specializations of existing classes in the standards layer and some others as specializations of upper layer classes. For example:

$$
\begin{aligned}
\texttt{A-MedicineModify} &\equiv \texttt{Request} \sqcap \texttt{=1 hasContent} \sqcap \\
&\quad \forall \texttt{hasContent.(Overwrite} \sqcap \exists \texttt{hasSubject.Medicine)} \\
\texttt{A-MedicineModify} &\sqsubseteq \texttt{FIPA-Request} \\
\texttt{A-VitalSignInform} &\sqsubseteq \texttt{Responsive} \sqcap \texttt{=1 hasContent} \sqcap \\
&\quad \forall \texttt{hasContent.(Proposition} \sqcap \exists \texttt{hasSubject.VitalSignData)} \sqcap \\
&\quad \texttt{=1 inReplyTo} \sqcap \forall \texttt{inReplyTo.VitalSignQuery} \\
\texttt{A-VitalSignInform} &\sqsubseteq \texttt{FIPA-Inform}
\end{aligned}
$$

Interoperation between agents of two systems that use different kinds of communication acts will proceed through those upper and standard layer classes.

In summary, CommOnt provides a terminology for communication acts and formal term relationships of equivalence and subsumption that allow to reason for interoperability purposes.

2.2 Dynamic semantics associated to communication acts

Formal semantics based on mental concepts such as *beliefs*, *desires* and *intentions* has been associated to communication acts. However, that option has been criticized for the approach (Singh, 1998) as well as for its analytical difficulties (Wooldridge, 2000). By contrast, we have adopted the so called social approach (Singh, 2000; Venkatraman and Singh, 1999; Fornara and Colombetti, 2002) to express the dynamic semantics of communication acts described in the CommOnt ontology. According to the social approach, when agents interact they become involved in social commitments or obligations to each other. Those commitments are public, and therefore they are suitable for an objective and verifiable semantics of agent interaction.

Definition 2 A *base-level commitment* $\mathsf{C}(x, y, p)$ is a ternary relation that represents a commitment made by x (the *debtor*) to y (the *creditor*) to bring about a certain proposition p.

For example, the base-level commitment $\mathsf{C}(Customer, Merchant, buyGoods)$ indicates the commitment made by the Customer agent to the Merchant agent to buy some goods.

Moreover, sometimes an agent accepts a commitment only if a certain condition holds or, interestingly, only when a certain commitment is made by another agent. This is called a conditional commitment.

Definition 3 A *conditional commitment* $\mathsf{CC}(x, y, p, q)$ is a quaternary relation that represents that if the condition p is brought out, x will be committed to y to bring about the proposition q.

For example, $\mathsf{CC}(Customer, Merchant, priceGoods(<35\$), buyGoods)$ indicates the commitment made by the Customer agent to the Merchant agent to buy some goods if the price of the goods is less than 35$.

The formalism we use for reasoning about commitments is based on the Event Calculus, which is a logic-based formalism for representing actions and their effects. The basic ontology of the Event Calculus comprises *events*, *fluents* and *time points*: *events* correspond to *actions* in our context; *fluents* are predicates whose truth value

may change over time. Event Calculus includes predicates for saying what happens and when (*Happens*), for describing the initial situation (*Initially*), for describing the effects of actions (*Initiates* and *Terminates*), and for saying what fluents hold at what times (*HoldsAt*). See (Shanahan, 1999) for more explanations. In our context, commitments (base-level and conditional) can be considered fluents, and semantics of communication acts can be expressed with predicates in the Event Calculus.

We show some predicates that describe the semantics associated to the classes of communication acts `Request`, `Accept` and `Responsive`, which appear in the upper level of CommOnt and whose descriptions have been shown in the previous subsection.

- *Initiates(Request(s, r, P),* CC*(r, s, accept(r, s, P), P), t).*
  A Request from s to r produces the effect of generating a conditional commitment which expresses that if the receiver r accepts the demand, it will be commited to the proposition P in the content of the communication act.
- *Initiates(Accept(s, r, P), accept(s, r, P), t).*
  The sending of an Accept produces the effect of generating the accept fluent.
- *Terminates(Responsive(s, r, P, RA),*C*(s, r, RA), t).*
  Terminates(Responsive(s, r, P, RA),CC*(s, r, accept(s, r, RA), RA), t).*
  Initiates(Responsive(s, r, P, RA), P)
  By sending a message of the class Responsive, the commitment (either base-level or conditional) of the sender s towards the receiver r to bring about proposition *RA* ceases to hold, and moreover, the fluent P is initiated.

The semantics is determined by the fluents that are initiated or terminated as a result of sending a message of that class from a sender to a receiver. In summary, communication acts have a dual semantic representation: The description in CommOnt of their structure and of their hierarchical relationships and then some Event Calculus predicates which specify their effects. The set of fluents that hold at a moment describes the state of the interaction.

Furthermore, some rules are needed to capture the dynamics of commitments. Commitments are a type of fluent, typically put in force by communication acts, that become inoperative after the appearance of other fluents. In the following rules, *e(x)* represents an event caused by *x*. The first rule declares that when a debtor of a commitment that is in force causes an event that initiates the committed proposition, the commitment ceases to hold.

Rule 1: *HoldsAt(*C*(x, y, p), t)* $\wedge$ *Happens(e(x), t)* $\wedge$ *Initiates(e(x), p, t)* $\rightarrow$ *Terminates(e(x),*C*(x, y, p), t).*

The second rule declares that a conditional commitment that is in force disappears and generates a base-level commitment when the announced condition is brought out by the creditor.

Rule 2: *HoldsAt(*CC*(x, y, c, p), t)* $\wedge$ *Happens(e(y), t)* $\wedge$ *Initiates(e(y), c, t)* $\rightarrow$ *Initiates(e(y),*C*(x, y, p), t)* $\wedge$ *Terminates(e(y),*CC*(x, y, c, p), t).*

2.3 Encoding of the interaction of the dual semantic representation

SWRL is a combination of OWL-DL with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language(**?**). We use SWRL as a formalism to integrate the dual semantic representation of communication acts. CommOnt axioms are expressed

in OWL-DL and can be therefore managed naturally in SWRL. Moreover, Event Calculus predicates and rules can be sistematically encoded in SWRL using a reification technique. Foremost, for each type of event and type of fluent, a corresponding ontology class must be present (in the T-Box). Any particular existence of event or fluent will be represented as an individual of the corresponding class along with its valued property statements (in the A-Box). For example, action *Request(s, r, P)* is encoded as an individual x in class `Request` and with *s, r* and P as values for its corresponding object properties: `hasSender`, `hasReceiver` and `hasContent` (specified in CommOnt). Analogously, a fluent *accept(r, s, P)* may be represented with the following assertions about individuals: *Acceptance(a)*, *hasSignatory(a,r)*, *hasAddressee(a,s)*, and *hasObject(a,P)*.

Then, two object properties: *initiates* and *terminates*, representing the predicates *Initiates* and *Terminates*, respectively, are also specified (in the T-Box) and will be used to relate an instance of communication act with the fluents it initiates or terminates. Finally, the aforementioned rules –that capture the dynamic aspects– , as well as effects of the application of communication acts, can be encoded with SWRL rules. For instance, the predicate *Initiates(Request(s, r, P),* CC*(r, s, accept(r, s, P), P), t)* can be encoded as follows[2]:

Request(x) $\wedge$ *hasSender(x,s)* $\wedge$ *hasReceiver(x,r)* $\wedge$ *hasContent(x,p)* $\wedge$ *hasCommit(x,c)* $\wedge$ *isConditionedTo(c,a)* $\rightarrow$ *initiates(x,c)* $\wedge$ *hasDebtor(c,r)* $\wedge$ *hasCreditor(c,s)* $\wedge$ *hascondition(c,p)* $\wedge$ *Acceptance(a)* $\wedge$ *hasSignatory(a,r)* $\wedge$ *hasAddressee(a,s)* $\wedge$ *hasObject(a,p)*

Several implementations of SWRL are being developed. Pellet(**?**) is a reasoner which implements a direct tableau algorithm for a DL-safe rules extension to OWL-DL. Pellet interprets SWRL using the DL-Safe rules notion, which means that rules will be only applied to named individuals in the ontology. Fortunately, the desired interpretation for our rules agrees with the DL-safe rules notion, and therefore Pellet has been our choice. A different implementation approach could consist on translating SWRL into First Order Logic and using a First Order theorem prover for reasoning.

3 Representations of the semantics of protocols

Similarly to the representation of communication acts, presented in the previous section, we propose a dual representation for protocols. On the one hand, we define a structure-based representation, using OWL-DL descriptions; on the other hand, we define a fluent-based semantics of protocols.

3.1 OWL-DL description of protocols

As we have already mentioned, we use STS as models for representing protocols. More specifically, in this paper we restrict our work to deterministic STS without cycles. In order to represent those models using OWL-DL, we have defined four different classes: `Protocol`, `State`, `Transition` and `Fluent`, which respectively represent protocols, states, transitions in protocols, and fluents associated to states. With these four

[2] We are aware that in the human readable syntax of SWRL, variables are prefixed with a question mark (e.g. *?x*). However, for the sake of visual clarity, the question mark has been removed from all the SWRL variables in this paper

classes and some (specialized) subclasses we are able to represent enough structure in order to fully describe the components of individual instances of our protocols.

We model those class descriptions with the following guidelines. Fluents are associated to states where they hold[3]:

$$\exists \texttt{hasFluent} \sqsubseteq \texttt{State} \;\; ; \;\; \exists \texttt{hasFluent}^- \sqsubseteq \texttt{Fluent}$$

Transitions get out of states and every transition is labelled with the communication act that is sent by that event, and it is associated to the state reached by that transition:

$$\begin{aligned}
\exists \texttt{hasTransition} &\sqsubseteq \texttt{State} \;\; ; \;\; \exists \texttt{hasTransition}^- \sqsubseteq \texttt{Transition} \\
\texttt{Transition} &\equiv \texttt{=1 hasCommAct} \sqcap \texttt{=1 hasNextState} \\
\exists \texttt{hasCommAct} &\sqsubseteq \texttt{Transition} \;\; ; \;\; \exists \texttt{hasCommAct}^- \sqsubseteq \texttt{CommunicationAct} \\
\exists \texttt{hasNextState} &\sqsubseteq \texttt{Transition} \;\; ; \;\; \exists \texttt{hasNextState}^- \sqsubseteq \texttt{State}
\end{aligned}$$

A protocol is an individual of the class `Protocol` and it is determined by the properties of its initial state, due to our conceptual modeling of states and transitions.

$$\begin{aligned}
\texttt{Protocol} &\equiv \exists \texttt{hasInitialState.State} \\
\exists \texttt{hasInitialState} &\sqsubseteq \texttt{Protocol} \;\; ; \;\; \exists \texttt{hasInitialState}^- \sqsubseteq \texttt{State}
\end{aligned}$$

Some other interesting subclasses are specified in order to describe the elements that compose our protocols:

$$\begin{aligned}
\texttt{FinalState} &\sqsubseteq \texttt{State} \\
\texttt{Commitment} &\sqsubseteq \texttt{Fluent} \sqcap \;\texttt{=1 hasDebtor} \sqcap \forall \texttt{hasDebtor.Actor} \sqcap \texttt{=1 hasCreditor} \sqcap \\
&\quad \forall \texttt{hasCreditor.Actor} \sqcap \texttt{=1 hasCondition} \sqcap \forall \texttt{hasCondition.Fluent} \\
\texttt{ConditionalCommitment} &\sqsubseteq \texttt{Fluent} \sqcap \;\texttt{=1 hasDebtor} \sqcap \forall \texttt{hasDebtor.Actor} \sqcap \texttt{=1 hasCreditor} \sqcap \\
&\quad \forall \texttt{hasCreditor.Actor} \sqcap \texttt{=1 hasCondition} \sqcap \forall \texttt{hasCondition.Fluent} \sqcap \\
&\quad \texttt{=1 isConditionedTo} \sqcap \forall \texttt{isConditionedTo.Fluent}
\end{aligned}$$

We are conscious that alternative descriptions may be considered, but our conceptual modeling is disposed to favor the rule encoding of dynamic aspects of a protocol, as it will be shown in the next subsection.

3.2 Fluent-based semantics of protocols

One of the most common approaches for comparing two protocols involves the discovery of structural relationships between them. However, we believe that dealing only with structural relationships is too rigid if a flexible interoperation among agents that use different standard protocols is to be promoted. For that reason, we propose to obtain an additional description of protocols, represented by their final states and the fluents that hold at those final states. In order to do so, we propose to consider the following definitions.

[3] $\exists \texttt{hasFluent} \sqsubseteq \texttt{State}$ and $\exists \texttt{hasFluent}^- \sqsubseteq \texttt{Fluent}$ mean that the class `State` and the class `Fluent` are respectively the domain and range of the property `hasFluent`.

Definition 4 Let $\mathcal{WR}$ be the set of SWRL rules that encode Rule 1 and Rule 2 presented in section 3.2. Let G and G' be sets of fluents; and let (s, l, s') be a transition in a STS. $G : <(s, l, s')> \vdash_{\mathcal{WR}} G'$ is a *transition derivation*, which means that, in the context of $\mathcal{WR}$, if G is the set of fluents that hold at state s, when transition (s, l, s') happens, G' is the set of fluents that hold at state s'.

Transition derivations represent the dynamics of a protocol. When a transition is accomplished some fluents may become true, others may become false, and the rest remain unchanged.

The encoding of $G : <(s, l, s')> \vdash_{\mathcal{WR}} G'$ into SWRL rules is done by taking into account our OWL-DL descriptions of STS presented in the previous subsection. Two main rules have been defined: On the one hand, the *fluent attachment rule* attaches to state s' the fluents initiated as a result of sending the communication act l:

Transition(t) ∧ hasNextState(t,s') ∧ hasCommAct(t,l) ∧ initiates(l,f) → hasFluent(s',f)

On the other hand, the *fluent transmission rule* transfers the fluents that hold in state s and that must also hold in state s' because the act of sending the communication act l has no effect on them:

hasFluent(s, f) ∧ hasTransition(s, t) ∧ hasNextState(t, s') → hasFluent(s', f)

Definition 5 A *branch* of a protocol $P = (S, s_0, L, T, F)$ is a sequence of transitions from T, $<(s_{i-1}, l_i, s_i)>_{i=1..n}$, that begins in the initial state s_0 and ends in a final state $s_n \in F$. We denote $\Omega(P)$ to the set of branches of protocol P.

Let $B = <(s_{i-1}, l_i, s_i)>_{i=1..n}$ be a branch, then $G_0 : B \vdash_{\mathcal{WR}} G_n$ means that the sets of fluents G_i $i = 1..n$ exist such that $G_{i-1} : <(s_{i-1}, l_i, s_i)> \vdash_{\mathcal{WR}} G_i$

Definition 6 If B is a branch of protocol $P(S, s_0, L, T, F)$, G_0 is the set of fluents that hold in s_0 and $G_0 : B \vdash_{\mathcal{WR}} G_n$, then G_n is a *protocol trace*. We denote $\mathcal{T}(B)$ to the final set of fluents generated by B. That is to say, $\mathcal{T}(B) = G_n$.

Notice that protocol traces are defined in terms of the semantics of communication acts, taking into account the content of the communication and not only the type of communication. By contrast, many other related works (see section 7) consider only communication acts as atomic acts without considering their content or their semantics.

From our viewpoint, the semantics of a protocol is determined by the traces of the protocol. That is to say, from a set-theoretical approach $\{\mathcal{T}(B) |\ B \in \Omega(P)\}$ is an interpretation of protocol P.

4 Steps to discover relationships between protocols

In this section we present the main steps that our proposal follows in order to compare two protocols and determine the semantic relationships between them. Since the semantics of a protocol is determined by its traces, in order to perform the comparison we need to deal with the branches of the protocol. We identify three main steps:

1. *Separation of each protocol into branches:* each protocol is separated into all the branches that can be generated from the initial state to a final state.
2. *Generation of protocol traces:* the protocol trace that corresponds to each branch of each protocol is calculated.

```
Algorithm 3.3.1: branchGeneration(P)
1  s0 ← getInitialState(P)
2  branches ← createVector()
3  partialBranches ← createVector()
4  partialBranches.add(createPartialBranch(s0))
5  while not partialBranches.isEmpty() do
6      partialBranch ← partialBranches.getElement()
7      lastState ← partialBranch.lastElement()
8      transitions ← getTransitions(lastState)
9      for t in transitions do
10         ns ← getNextState(t)
11         p ← partialBranch.clone().add(ns)
12         partialBranches.add(p)
12         if isFinalState(ns) then
13             branches.add(p)
14 return branches
```

Fig. 1 Algorithm: Generation of branches

3. *Pairing up branches:* the branches of one of the protocols are compared to the branches of the other protocol.

Moreover, once the previous steps have been concluded, it will be discovered whether any semantic relationship can be established between both protocols . In section 5 a formal definition of those semantic relationships is presented.

Now we will develop these steps more thoroughly.

4.1 Separation of the protocol into branches

In the first step, the protocol is separated into all the branches that can be generated from the initial state to a final state.

Algorithm *branchGeneration* (see Fig. 1) shows how to do that. In line 2, a global vector (`branches`) is created, which will be used to store the different branches of a protocol P after they have been calculated. In line 3 another vector (`partialBranches`) is created. This vector will be used as an auxiliary element to store the partial branches that are generated in the process of calculating the complete branches. This last vector is initialized, in line 4, with a partial branch that contains only the initial state of the protocol (s0). Taking into account the transitions of protocol P, this partial branch is repeatedly modified by adding new states to it (lines 5 to 13). Moreover, once one of the partial branches is completed (i.e. a final state has been reached), it is added to the global vector `branches` (lines 12-13), which will be returned once all the partial branches have been completed. An example can be found in Fig. 2.

4.2 Generation of protocol traces

Given a branch B, with a simulation of its corresponding $G_0 : B \vdash_{\mathcal{WR}} G_n$, the trace $\mathcal{T}(B)$ is calculated. During that simulation the SWRL rules that encode the semantics of the communication acts (see sections 2.3 and 3.2) that appear in the branch are applied. Then, the set of fluents that hold at the final state of each branch is the corresponding trace. That set represents the effects of the protocol branch.

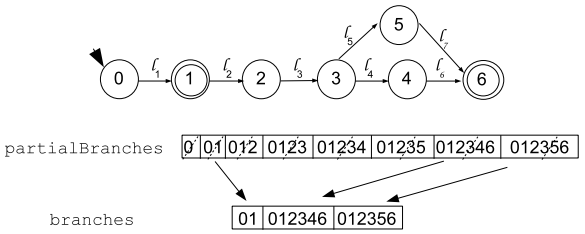


Fig. 2 Separation of branches

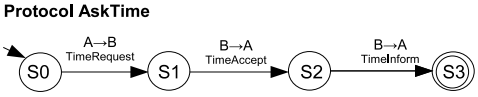


Fig. 3 Protocol diagram

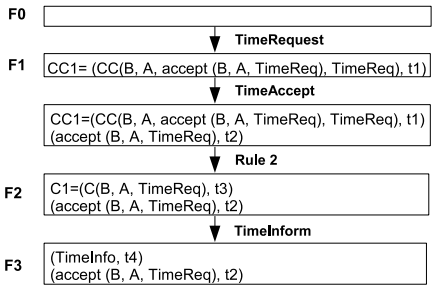


Fig. 4 Protocol fluents

For example, let us take protocol AskTime in Fig. 3. By using this protocol, agent A intends to get from agent B information about the time. So, in the first transition, agent A requests about the time. Then, agent B sends a message accepting to give the information about the time, and finally, in the third transition, agent B sends the requested information. The descriptions in the CommOnt ontology of the communication acts that appear in the protocol AskTime are the following:

$$\texttt{TimeRequest} \equiv \texttt{Request} \sqcap \texttt{=1 hasContent} \sqcap \forall \texttt{hasContent.TimeReq}$$
$$\texttt{TimeAccept} \equiv \texttt{Accept} \sqcap \texttt{=1 hasContent} \sqcap \forall \texttt{hasContent.TimeReq}$$
$$\texttt{TimeInform} \equiv \texttt{Responsive} \sqcap \texttt{=1 hasContent} \sqcap \forall \texttt{hasContent.TimeInfo} \sqcap \texttt{=1 inReplyTo} \sqcap \forall \texttt{inReplyTo.TimeRequest}$$

In Fig. 4 we show which fluents are associated to the states of the protocol and how they vary as a consequence of the communication acts that are sent and the rules described in section 3.2. We depart from a situation where the set of fluents is empty (F0). The impact the communication acts of the protocol have over the set of fluents that hold at each moment is explained next:

`TimeRequest: A→B`

When the `TimeRequest` message is sent, the conditional commitment `CC1` is initiated (F1), which states that if agent B accepts to give information about the time, then it will be committed to do so. This happens because according to the definitions in the CommOnt ontology, `TimeRequest` is a subclass of `Request`, and as a result, the absatract predicate *Initiates(Request(s, r, P), CC(r, s, accept(r,s,P), P), t)* can be applied. More precisely, in this case the predicate is instantiated as *Initiates(Request(A, B, TimeReq), CC(B, A, accept(B,A,TimeReq), TimeReq), t_1)*. The following SWRL rule, which encodes the abstract predicate, can be executed at this moment:

Request(x) ∧ hasSender(x,s) ∧ hasReceiver(x,r) ∧ hasContent(x,p) ∧ hasCommit(x,c) ∧ isConditionedTo(c,a) → initiates(x,c) ∧ hasDebtor(c,r) ∧ hasCreditor(c,s) ∧ hascondition(c,p) ∧ Acceptance(a) ∧ hasSignatory(a,r) ∧ hasAddressee(a,s) ∧ hasObject(a,p)

This rule initiates the aforementioned conditional commitment `CC1`, and due to the fluent attatchment rule (section 3.2), `CC1` is attached to state S1, and thus it holds in F1.

`TimeAccept: B→A`

Then, agent B agrees to respond by sending the `TimeAccept` message. Due to the predicate *Initiates(Accept(s,r,P), accept(s,r,P), t)* – instantiated in this case as *Initiates(Accept(B,A,TimeReq), accept(B,A,TimeReq), t_2)*– and the fact that `TimeAccept` is a subclass of `Accept`, the rule

Accept(x) ∧ hasSender(x,s) ∧ hasReceiver(x,r) ∧ hasContent(x,p) ∧ hasCommit(x,c) → initiates(x,c) ∧ Acceptance(c) ∧ hasSignatory(c,s) ∧ hasAddressee(c,r) ∧ hasObject(a,p)

is executed and consequently, the fluent *accept(B, A, TimeReq)* is initiated. At this point, Rule 2 (see section 2.2) can be applied because *accept(B,A,TimeReq)* is the condition of the conditional commitment `CC1`. As a consequence, `CC1` is terminated and the base commitment `C1` is initiated and attached to state S2 due to the aforementioned fluent attachment rule (F2). We would like to remark that, since SWRL does not offer any primitive for retraction, the deletion of the fluent `CC1` is performed using the methods of the OWL-API(`?`) that allow the manipulation of the ABox.

`TimeInform: B→A`

Finally, agent B sends the `TimeInform` message. Since `TimeInform` is a subclass of `Responsive` and because of the predicate *Initiates(Responsive(s,r,P, RA), P, t)*, – instantiated as *Initiates(Responsive(B,A,TimeInfo, TimeReq), TimeInfo, t_4)*– the fluent *TimeInfo*, is initiated. Moreover, `C1` is terminated due to the predicate *Terminates(Responsive(s,r,P, RA), C(s,r,RA), t)*, instantiated as *Terminates(Responsive(B, A, TimeInfo, TimeReq), C(B,A,TimeReq), t_4)*. In our environment, these predicates have been encoded with the rule:

Responsive(x) ∧ hasContent(x,p) ∧ inReplyTo(x,irt) ∧ initiates(irt, f) → terminates(x,f) ∧ initiates(x, p)

In addition, due to the fluent transmission rule, the fluent *accept(B, A, TimeReq)* is transferred from state S2 to state S3. So, at this point (F3) we can say that the fluents that hold at the final state of the protocol are (*accept(B, A, TimeReq)*, t_2) and (*TimeInfo*, t_4) and so, the protocol trace [(*accept(B, A, TimeReq)*, t_2), (*TimeInfo*, t_4)] is *generated* for the branch. As our proposal does not make use of the *time* parameter, it will be omitted in the following analysis.

4.3 Pairing up branches from two different protocols

Once protocol traces have been generated, it is necessary to establish relationships between branches of the two protocols $P1$ and $P2$ being compared. In order to compare two branches, we compare their protocol traces, and thus, their fluents. To do so, our mechanism first evaluates the cartesian product of branches $\Omega(P1) \times \Omega(P2)$, taking into account the following considerations:

Given $B1 \in \Omega(P1)$ and $B2 \in \Omega(P2)$, four separate cases may occur when comparing two fluents $t1 \in \mathcal{T}(B1)$ and $t2 \in \mathcal{T}(B2)$:

(*eq*) $t1$ and $t2$ are *equivalent*: *msc(t1)* $\equiv$ *msc(t2)*, being *msc(t)* the most specific concept of a fluent t in regard to an ontology of fluents. In this case we could see those fluents as *clones*, their names being their sole difference.

(*g1*) $t1$ is *more general* than $t2$: *msc(t2)* $\sqsubseteq$ *msc(t1)* and *msc(t2)* $\not\equiv$ *msc(t1)*.

(*g2*) $t2$ is *more general* than $t1$: *msc(t1)* $\sqsubseteq$ *msc(t2)* and *msc(t1)* $\not\equiv$ *msc(t2)*.

(*in*) $t1$ and $t2$ are *incomparable*: *msc(t1)* $\not\sqsubseteq$ *msc(t2)* and *msc(t2)* $\not\sqsubseteq$ *msc(t1)* (They have no subsumption relation between each other).

We are aware that due to the heterogeneity of the protocols, it may happen that the fluents to be compared come from different ontologies. In that case, ontology mediation techniques should be applied, but this aspect is out of the scope of this paper and it has been thoroughly studied in the specialized literature (**??**).

It may happen that a fluent exists in $\mathcal{T}(B1)$ which is *incomparable* with any fluent in $\mathcal{T}(B2)$, and a fluent in $\mathcal{T}(B2)$ which is *incomparable* with any fluent in $\mathcal{T}(B1)$; then, we say that the pairing up of $B1$ and $B2$ is not *feasible*. This means that $B1$ generates some fluent that is not generated by $B2$ and viceversa.

Definition 7 $(B1, B2) \in \Omega(P1) \times \Omega(P2)$ is a *feasible pair* iff $[\forall t1 \in B1.\ \exists t2 \in B2.\ t1$ and $t2$ satisfy $(eq), (g1)$ or $(g2)] \vee [\forall t2 \in B2.\ \exists t1 \in B1.\ t1$ and $t2$ satisfy $(eq), (g1)$ or $(g2)]$

We are not interested in unfeasible pairs of branches because, as it will be shown in section 5, the relationships we are interested in are those where all the fluents in at least one of the traces are related to some fluent in the other. We look for a total mapping of one trace into the other. The intuition behind this idea is that we try to find cases where a sort of substitution of protocols is admissible due to its semantic coincidence.

Each feasible pair $(Bi, Bj) \in \Omega(P1) \times \Omega(P2)$ receives a 4-place tuple *valuation* $(x_0^{ij}, x_1^{ij}, x_2^{ij}, x_3^{ij})$ where:

- x_0^{ij} is the number of pairs of fluents that fulfill case *(eq)*.
- x_1^{ij} is the number of pairs of fluents that fulfill case *(g1)*.
- x_2^{ij} is the number of pairs of fluents that fulfill case *(g2)*.
- x_3^{ij} is the exceeding number of fluents: $x_3^{ij} = |\#\mathcal{T}(B1) - \#\mathcal{T}(B2)|$, where $\#\mathcal{T}(B1)$ and $\#\mathcal{T}(B2)$ are the number of fluents of $\mathcal{T}(B1)$ and $\mathcal{T}(B2)$ respectively.

Then, we have defined a metric by means of the following function, which yields a value in $[0, 1]$ for every 4-place tuple:

$$f(x_0, x_1, x_2, x_3) = \frac{x_0 + max(x_1, x_2)}{\sum_{i=0}^{3} x_i}$$

Our interest is to quantify as many related fluents as possible within a pair of branches, regardless of whether their relationship is of equivalence or subsumption (that is why x_0 is given the same weight as $max(x_1, x_2)$ in the formula). Division by $\sum_{i=0}^{3} x_i$ assures the function value does not go beyond 1.

Definition 8 Given two sets of branches $\Omega(P1)$ and $\Omega(P2)$, a *matching* π is a subset of feasible pairs from $\Omega(P1) \times \Omega(P2)$ such that every branch from the smallest set is paired up with a different branch from the greatest set. (π is the graph of an injective map).

Our aim is to obtain the matching that maximizes the sum of the metric applied to the valuation of its pairs. Formally, we look for the matching π that

$$\text{maximizes} \sum_{(B_i, B_j) \in \pi} f(x_0^{ij}, x_1^{ij}, x_2^{ij}, x_3^{ij})$$

In the case that two or more matchings $\pi_k (k \in 1 \dots n)$ that maximize the sum above exist, we priorize the equivalence relationships between fluents over the subsumption relationships. Then, the best matching is any of the set $\{\pi_k : (k \in 1 \dots n)\}$ such that

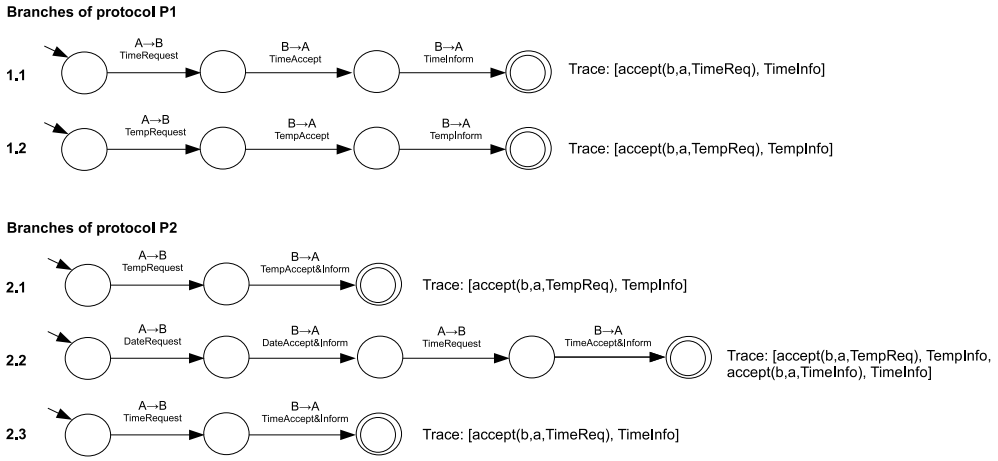
$$\text{maximizes} \sum_{(B_i, B_j) \in \pi_k} g(x_0^{ij}, x_1^{ij}, x_2^{ij}, x_3^{ij}), \quad \text{with } g(x_0, x_1, x_2, x_3) = \frac{x_0}{\sum_{i=0}^{3} x_i}$$

In Fig. 5 we present an example of the process of pairing up branches. On the one hand, in protocol $P1$ agent A requests and later receives information about the time (branch 1.1) or about the temperature (branch 1.2). Three transitions are necessary in each of the branches to reach the final state of the protocol. On the other hand protocol $P2$ is composed of three branches: Branch 2.1 has the same semantics as branch 1.2 but in this case only two transitions are needed, since with the `TempAccept&Inform` communication act, agent B both accepts and responds to the request. In branch 2.2, agent A requests for information about the date and the time and gets the respective responses (two transitions for each of the parameters). Finally, information about the time is requested in branch 2.3, also in two steps. The table in Fig. 5 registers the valuation and metric value of the feasible pairs from $\Omega(P1) \times \Omega(P2)$. When a pair is not feasible we represent it with an `X` in the corresponding cell.

5 Protocol relationships

Once the mechanism described in the previous section has finished the analysis of branches of protocols, the next step consists on discovering the semantic relationships between those protocols. This section presents the collection of semantic relationships we are considering.

Notice that the exploitation of those relationships increases the possibilities of achieving successful communications among agents in open and dynamic environments where heterogeneity issues can appear at different levels. For instance, communication between two agents that use distinct protocols is nowadays impossible unless a priory



	Branch 2.1	Branch 2.2	Branch 2.3
Branch 1.1	X	(2,0,0,2) f(2,0,0,2)=2/4=0.5	(2,0,0,0) f(2,0,0,0)= 2/2=1
Branch 1.2	(2,0,0,0) f(2,0,0,0)= 2/2=1	X	X

Best pairing:
Branch 1.1 with Branch 2.3
Branch 1.2 with Branch 2.1

Fig. 5 Pairing up branches of different protocols

agreement is considered. However, if an equivalence relationship among subparts of those protocols were discovered, those agents could at least have a chance to communicate partially (they would only take into account the related subparts), because the semantics associated to those subparts would be the same and therefore the agents would be prepared to reply to the same kind of requirements. Firstly, we present the definitions of relationships between two protocol branches, followed by the definitions of relationships between protocols.

5.1 Relationships between branches

Let A, B be two protocol branches, and $\mathcal{T}(A)$, $\mathcal{T}(B)$ be the protocol traces generated by branches A and B respectively.

Definition 9 Branch A is a *equivalent* to branch B ($A{=_b}B$) if there exists a bijective function $\phi : \mathcal{T}(A) \rightarrow \mathcal{T}(B)$ such that $\forall t \in \mathcal{T}(A).\ msc(t) \equiv msc(\phi(t))$.

Definition 10 Branch A is a *specialization* of branch B ($A{\ll_b}B$) if there exists a bijective function $\phi : \mathcal{T}(A) \rightarrow \mathcal{T}(B)$ such that $\forall t \in \mathcal{T}(A).\ msc(t) \sqsubseteq msc(\phi(t))$.

We denote $Prune[B/s_k]$ to the branch that results from pruning branch B from state s_k onwards. Given a branch $B =< (s_{i-1}, l_i, s_i) >_{i=1..n}$, and $1 \leq k \leq n$, $Prune[B/s_k] = < (s_{i-1}, l_i, s_i) >_{i=1..k}$.

Definition 11 Branch A is a *prefix* of branch B ($A{=}_{pre}B$) if there exists a state s_k in B such that $A{=}_b Prune[\mathrm{B}/s_k]$.

Branch A is a *specialized-prefix* of branch B ($A{\ll}_{pre}B$) if there exists a state s_k in B such that $A{\ll}_b Prune[\mathrm{B}/s_k]$.

Let $s_k (1 \leq k < n)$ a state in the definition of branch B. We denote $ChangeInit[B/s_k]$ to the new branch that results from modifying the definition of branch B in such a way that s_k becomes the initial state of the new branch: $ChangeInit[B/s_k]{=}{<}(s_{i-1}, l_i, s_i){>}_{i=k+1..n}$

Definition 12 Branch A is a *suffix* of branch B ($A{=}_{suf}B$) if there exists a state s_k in B such that $A{=}_b ChangeInit[\mathrm{B}/s_k]$.

Branch A is a *specialized-suffix* of branch B ($A{\ll}_{suf}B$) if there exists a state s_k in B such that $A{\ll}_b ChangeInit[\mathrm{B}/s_k]$.

Definition 13 Branch A is an *infix* of branch B ($A{=}_{inf}B$) if there exists a state s_k in B such that $A{=}_{pre} ChangeInit[\mathrm{B}/s_k]$ (i.e. A is a prefix of a suffix of B).

Branch A is an *specialized-infix* of branch B ($A{\ll}_{inf}B$) if there exists a state s_k in B such that $A{\ll}_{pre} ChangeInit[\mathrm{B}/s_k]$.

Definition 14 Branch A is an *complement_to_infix* of branch B ($A{=}_c B$) if there exists a state s_k in A such that $Prune[\mathrm{A}/s_k]{=}_{pre}B$ and $ChangeInit[\mathrm{A}/s_k]{=}_{suf}B$.

Branch A is an *specialized-complement_to_infix* of branch B ($A{\ll}_c B$) if there exists a state s_k in A such that $Prune[\mathrm{A}/s_k]{\ll}_{pre}B$ and $ChangeInit[\mathrm{A}/s_k]{\ll}_{suf}B$.

Taking into account the previous definitions we have developed a reasoning service that decides if a branch is a prefix, a suffix, an infix or a complement_to_infix of another branch, either in an equivalent or specialization sense.

5.2 Relationships between protocols

Protocols are constituted by branches, and therefore we use the definitions in the previous subsection to define the relationships between protocols.

Definition 15 Protocol P is *equivalent* to protocol Q ($P[\mathcal{E}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(P).\ A{=}_b\phi(A)$.

Definition 16 Protocol P is a *specialization* of protocol Q ($P[\mathcal{Z}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(P).\ A \ll_b \phi(A)$.

Sometimes, a protocol is defined by restrictions on the allowable communication acts at some states of a general protocol.

Definition 17 Let $\Omega'(Q)$ a proper subset of $\Omega(Q)$ ($\Omega'(Q) \subset \Omega(Q)$). Protocol P is a *restriction* of protocol Q ($P[\mathcal{R}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega'(Q)$ such that $\forall A \in \Omega(P).\ A{=}_b\phi(A)$.

Definition 18 Protocol P is a *prefix* of protocol Q ($P[\mathcal{P}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(P).\ A{=}_{pre}\phi(A)$.

Definition 19 Protocol P is a *suffix* of protocol Q ($P[\mathcal{S}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(P).\ A{=}_{suf}\phi(A)$.

Definition 20 Protocol P is an *infix* of protocol Q ($P[\mathcal{I}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(P).\ A{=_{inf}}\phi(A)$.

Definition 21 Protocol P is a *complement_to_infix* of protocol Q ($P[\mathcal{C}]Q$) if there exists a bijective function $\phi : \Omega(P) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(P).\ A{=_c}\phi(A)$.

We denote the composition of relations by sequencing the names of the relations. For instance, the relationship $P[\mathcal{Z}\mathcal{R}\mathcal{P}]Q$ means that protocol P is a specialization of a restriction of a prefix of protocol Q.

Following, we present first a list of relevant properties regarding the previous relationships and then some proofs of those properties.

5.2.1 Properties of the protocol relationships

Let P and Q be two protocols. Then, the following properties can be highlighted:

1. $\mathcal{P}$, $\mathcal{S}$, $\mathcal{I}$, $\mathcal{C}$, $\mathcal{Z}$ and $\mathcal{E}$ are reflexive.
2. $\mathcal{R}$ is irreflexive.
3. $\mathcal{P}$, $\mathcal{S}$, $\mathcal{I}$, $\mathcal{C}$, $\mathcal{Z}$, $\mathcal{E}$ and $\mathcal{R}$ are transitive.
4. $\forall \mathcal{X} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}, \mathcal{C}, \mathcal{E}, \mathcal{Z}\}.\ [\mathcal{X}\mathcal{R}] = [\mathcal{R}\mathcal{X}]$.
5. $\forall \mathcal{X} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}, \mathcal{C}\}.\ \forall \mathcal{Y} \in \{\mathcal{E}, \mathcal{Z}\}.\ [\mathcal{Y}\mathcal{X}] \Rightarrow [\mathcal{X}\mathcal{Y}]$ but $[\mathcal{X}\mathcal{Y}] \not\Rightarrow [\mathcal{Y}\mathcal{X}]$.
6. $\forall \mathcal{X} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}\}.\ \forall \mathcal{Y} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}\}.\ (\mathcal{X} \neq \mathcal{Y} \rightarrow\ (P[\mathcal{X}\mathcal{Y}]Q \Leftrightarrow\ P[\mathcal{I}]Q))$.

5.2.2 Proofs

Next, we provide proofs for some of the properties listed above. Remaining proofs can be figured out accordingly.

Property 1 : $\mathcal{P}$, $\mathcal{S}$, $\mathcal{I}$, $\mathcal{C}$, $\mathcal{Z}$ and $\mathcal{E}$ are reflexive.

- $\mathcal{P}$ is reflexive.
  Proof: We need to prove that $P[\mathcal{P}]P$.
  It suffices to define ϕ as the identity function, and obviously, $\forall A \in \Omega(P).\ A =_{pre} A$, because $A =_b Prune[A/s_n]$, where s_n is the final state of A.

- $\mathcal{Z}$ is reflexive.
  Proof: We need to prove that $P[\mathcal{Z}]P$.
  It suffices to define ϕ as the identity function, and obviously $\forall A \in \Omega(P),\ A \ll_b A$, because for the identity function $id : \mathcal{T}(A) \rightarrow \mathcal{T}(A)$, $\forall t \in \mathcal{T}(A).\ msc(t) \sqsubseteq msc(id(t))$.

Property 2 : $\mathcal{R}$ is irreflexive.

- *Proof:* We need to prove that $P[\mathcal{R}]P$ is false. The set $\Omega(P)$ is finite, therefore it is impossible to define a bijective function from $\Omega(P)$ to a proper subset $\Omega'(P) \subset \Omega(P)$.

Property 3 : $\mathcal{P}$, $\mathcal{S}$, $\mathcal{I}$, $\mathcal{C}$, $\mathcal{Z}$, $\mathcal{E}$ and $\mathcal{R}$ are transitive.

– $\mathcal{Z}$ is transitive.
Proof: We need to prove that $P[\mathcal{Z}\mathcal{Z}]Q \Rightarrow P[\mathcal{Z}]Q$.
$P[\mathcal{Z}\mathcal{Z}]Q$ means that there exists a protocol M such that $P[\mathcal{Z}]M$ and $M[\mathcal{Z}]Q$. Looking at the definition of $\mathcal{Z}$, $M[\mathcal{Z}]Q$ means that there exists a bijective function $\phi : \Omega(M) \rightarrow \Omega(\mathrm{Q})$ such that $\forall A \in \Omega(M).\ A \ll_b \phi(A)$. Moreover, $P[\mathcal{Z}]M$ means that there exists a bijective function $\phi' : \Omega(P) \rightarrow \Omega(M)$ such that $\forall A \in \Omega(P).\ A \ll_b \phi'(A)$. Then, the bijective function $\phi \circ \phi' : \Omega(P) \rightarrow \Omega(Q)$ satisfies the needed properties since $\forall A \in \Omega(P)\ A \ll_b \phi'(A) \ll_b \phi \circ \phi'(A)$ and $\ll_b$ is transitive.

– $\mathcal{P}$ is transitive.
Proof: We need to prove that $P[\mathcal{P}\mathcal{P}]Q \Rightarrow P[\mathcal{P}]Q$.
$P[\mathcal{P}\mathcal{P}]Q$ means that there exists a protocol M such that $P[\mathcal{P}]M$ and $M[\mathcal{P}]Q$. Looking at the definition of $\mathcal{P}$, $M[\mathcal{P}]Q$ means that there exists a bijective function $\phi : \Omega(M) \rightarrow \Omega(Q)$ such that $\forall A \in \Omega(M).\ \exists sq_i \in Q.\ A{=_b}Prune[\phi(A)/sq_i]$. Moreover, $P[\mathcal{P}]M$ means that there exists a bijective function $\phi' : \Omega(P) \rightarrow \Omega(M)$ such that $\forall B \in \Omega(P).\ \exists sm_i \in M.\ B{=_b}Prune[\phi'(B)/sm_i]$. We ask for a small abuse of notation, denoting $\phi(sm_i)$ to the state in Q corresponding to sm_i after applying ϕ to branch $\phi'(B)$. Then, the bijective function $\phi \circ \phi' : \Omega(P) \rightarrow \Omega(Q)$ is such that $\forall B \in \Omega(P).\ \exists \phi(sm_i) \in Q.\ B{=_b}Prune[\phi \circ \phi'(B)/\phi(sm_i)]$.

– $\mathcal{R}$ is transitive.
Proof: We need to prove that $P[\mathcal{R}\mathcal{R}]Q \Rightarrow P[\mathcal{R}]Q$.
$P[\mathcal{R}\mathcal{R}]Q$ means that there exists a protocol M such that $P[\mathcal{R}]M$ and $M[\mathcal{R}]Q$. Looking at the definition of $\mathcal{R}$, $M[\mathcal{R}]Q$ means that there exists a bijective function $\phi : \Omega(M) \rightarrow \Omega'(Q)$ such that $\forall A \in \Omega(M).\ A =_b \phi(A)$. Moreover, $P[\mathcal{R}]M$ means that there exists a bijective function $\phi' : \Omega(P) \rightarrow \Omega'(M)$ such that $\forall B \in \Omega(P)\ B =_b \phi'(B)$. Then, the bijective function $\phi \circ \phi' : \Omega(P) \rightarrow \Omega''(Q) \subset \Omega(Q)$, where $\Omega''(Q) = \phi \circ \phi'(\Omega(P)) = \phi(\Omega'(M))$, is such that $\forall A \in \Omega(P).\ A{=_b}\phi \circ \phi'(A)$.

Property 4 : $\forall \mathcal{X} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}, \mathcal{C}, \mathcal{E}, \mathcal{Z}\}.\ [\mathcal{X}\mathcal{R}] = [\mathcal{R}\mathcal{X}]$.

– $[\mathcal{P}\mathcal{R}] = [\mathcal{R}\mathcal{P}]$.
Proof: First, we prove $P[\mathcal{P}\mathcal{R}]Q \Rightarrow P[\mathcal{R}\mathcal{P}]Q$.
Because of $P[\mathcal{P}\mathcal{R}]Q$, there exists a protocol M such that $P[\mathcal{P}]M$ and $M[\mathcal{R}]Q$. According to definitions of $\mathcal{P}$ and $\mathcal{R}$, $\exists \phi : \Omega(P) \rightarrow \Omega(M)$ bijective, such that $\forall A \in \Omega(P).\ A =_{pre} \phi(A)$ and $\exists \phi' : \Omega(M) \rightarrow \Omega'(Q) \subset \Omega(Q)$ bijective, such that $\forall A \in \Omega(M).\ A =_b \phi'(\mathrm{A})$.
We must prove that there exists a protocol M' such that $P[\mathcal{R}]M'$ and $M'[\mathcal{P}]Q$. Let us define $\Omega(\overline{M})$ as the set of branches of protocol Q that do not have a corresponding branch in M ($\Omega(\overline{M}) = \Omega(Q) - \Omega'(Q)$, notice that $\Omega(\overline{M}) \neq \emptyset$). Then, M' is a protocol with $\Omega(M') = \Omega(P) \cup \Omega(\overline{M})$.
In order to prove $M'[\mathcal{P}]Q$ we define the bijective function $\psi : \Omega(M') \rightarrow \Omega(Q)$
  – if $A \in \Omega(P)$ then $\psi(A) = \phi'(\phi(A))$.
  – if $A \notin \Omega(P)$ then $\psi(A) = A$.
Then, $\forall A \in \Omega(M').\ A =_{pre} \psi(A)$, since $A =_{pre} \phi(A) =_b \phi'(\phi(A))$ implies $A =_{pre} \phi'(\phi(A))$.
To prove $P[\mathcal{R}]M'$, we use the identity function $id : \Omega(P) \rightarrow \Omega(P) \subset \Omega(M')$ so that $\forall A \in \Omega(P).\ A =_b id(A)$.

Second, we prove $P[\mathcal{R}\mathcal{P}]Q \Rightarrow P[\mathcal{P}\mathcal{R}]Q$.

Because of $P[\mathcal{RP}]Q$, there exists a protocol M such that $P[\mathcal{R}]M$ and $M[\mathcal{P}]Q$. According to definitions of $\mathcal{R}$ and $\mathcal{P}$, $\exists\phi' : \Omega(P) \to \Omega'(M) \subset \Omega(M)$ bijective, such that $\forall A \in \Omega(P).\ A =_b \phi'(A)$ and $\exists\phi : \Omega(M) \to \Omega(Q)$ bijective, such that $\forall A \in \Omega(M).\ A =_{pre} \phi(A)$.
We must prove that there exists a protocol M' such that $P[\mathcal{P}]M'$ and $M'[\mathcal{R}]Q$.
We build a protocol M' in such a way that $\Omega(M') = \{\phi(\phi'(A)) \in \Omega(Q) \mid A \in \Omega(P)\}$. In order to prove $P[\mathcal{P}]M'$, we define the function $\psi : \Omega(P) \to \Omega(M')$ such that $\psi(A) = \phi(\phi'(A)$. Then, ψ is bijective and $\forall A \in \Omega(P).\ A =_b \phi'(A) =_{pre} \phi(\phi'(A)$, and consequently $A =_{pre} \psi(A)$.
Moreover, the identity function $id : \Omega(M') \to \Omega(M') \subset \Omega(Q)$ is bijective and satisfies the condition for $M'[\mathcal{R}]Q$, by construction of $\Omega(M')$.

– $[\mathcal{ZR}] = [\mathcal{RZ}]$.
Proof: First, we prove $P[\mathcal{ZR}]Q \Rightarrow P[\mathcal{RZ}]Q$.
Because of $P[\mathcal{ZR}]Q$, there exists a protocol M such that $P[\mathcal{Z}]M$ and $M[\mathcal{R}]Q$. According to definitions of $\mathcal{Z}$ and $\mathcal{R}$, $\exists\phi : \Omega(P) \to \Omega(M)$ bijective, such that $\forall A \in \Omega(P).\ A \ll_b \phi(A)$ and $\exists\phi' : \Omega(M) \to \Omega'(Q) \subset \Omega(Q)$ bijective, such that $\forall A \in \Omega(M).\ A =_b \phi'(\mathrm{A})$.
We must prove that there exists a protocol M' such that $P[\mathcal{R}]M'$ and $M'[\mathcal{Z}]Q$.
Let us define $\Omega(\overline{M})$ as the set of branches of protocol Q that do not have a corresponding branch in M ($\Omega(\overline{M}) = \Omega(Q) - \Omega'(Q)$). Then, M' is a protocol with $\Omega(M') = \Omega(P) \cup \Omega(\overline{M})$.
In order to prove $M'[\mathcal{Z}]Q$ we define the bijective function $\psi : \Omega(M') \to \Omega(Q)$
  – if $A \in \Omega(P)$ then $\psi(A) = \phi'(\phi(A))$.
  – if $A \notin \Omega(P)$ then $\psi(A) = A$.
Then, $\forall A \in \Omega(M').\ A \ll_b \psi(A)$, since $A \ll_b \phi(A) =_b \phi'(\phi(A))$ implies $A \ll_b \phi'(\phi(A))$.
Moreover, $P[\mathcal{R}]M'$ because the identity function $id : \Omega(P) \to \Omega(P) \subset \Omega(M')$ is bijective and satisfies the needed properties, by construction of $\Omega(M')$.
Second, we prove $P[\mathcal{RZ}]Q \Rightarrow P[\mathcal{ZR}]Q$.
Because of $P[\mathcal{RZ}]Q$, there exists a protocol M such that $P[\mathcal{R}]M$ and $M[\mathcal{Z}]Q$. According to definitions of $\mathcal{R}$ and $\mathcal{Z}$, $\exists\phi : \Omega(P) \to \Omega'(M) \subset \Omega(M)$ bijective, such that $\forall A \in \Omega(P).\ A =_b \phi(A)$ and $\exists\phi' : \Omega(M) \to \Omega(Q)$ bijective, such that $\forall A \in \Omega(M).\ A \ll_b \phi'(\mathrm{A})$.
We must prove that there exists a protocol M' such that $P[\mathcal{Z}]M'$ and $M'[\mathcal{R}]Q$.
Then we build a protocol M' with the branches of Q that have a corresponding branch in $\Omega(M)$ due to ϕ' ($\Omega(M') = \phi'(\Omega'(M))$).
In order to prove $P[\mathcal{Z}]M'$ we define the bijective function $\psi : \Omega(P) \to \Omega(M')$ with $\psi(A) = \phi'(\phi(A))$. Then, $\forall A \in \Omega(P).\ A \ll_b \psi(A)$, since $A =_b \phi(A) \ll_b \phi'(\phi(A))$ implies $A \ll_b \phi'(\phi(A))$.
Finally, $M'[\mathcal{R}]Q$ because the identity function $id : \Omega(M') \to \Omega(M') \subset \Omega(Q)$ is bijective and satisfies the needed properties, by construction of $\Omega(M')$.

Property 5 : $\forall\mathcal{X} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}, \mathcal{C}\}.\ \forall\mathcal{Y} \in \{\mathcal{E}, \mathcal{Z}\}.\ [\mathcal{YX}] \Rightarrow [\mathcal{XY}]$ but $[\mathcal{XY}] \not\Rightarrow [\mathcal{YX}]$.

– $P[\mathcal{ZP}]Q \Rightarrow P[\mathcal{PZ}]Q$.
Proof: Because of $P[\mathcal{ZP}]Q$, there exists a protocol M such that $P[\mathcal{Z}]M$ and $M[\mathcal{P}]Q$. According to the definitions of $\mathcal{Z}$ and $\mathcal{P}$, $\exists\phi : \Omega(P) \to \Omega(M)$ bijective, such that $\forall A \in \Omega(P).\ A \ll_b \phi(A)$ and $\exists\phi' : \Omega(M) \to \Omega(Q)$ bijective, such that $\forall B \in \Omega(M).\ B =_{pre} \phi'(B)$, which implies that $\forall B \in \Omega(M).\ \exists s_{\phi'(B)}.\ B =_b$

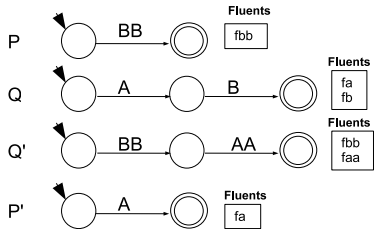


Fig. 6 Counter-example: $\mathcal{PZ} \not\Rightarrow \mathcal{ZP}$

$Prune[\phi'(B)/s_{\phi'(B)}]$ and exists $Suff(\phi'(B))$=$ChangeInit[\phi'(B)|s_{\phi'(B)}]$ (In other words, each branch of Q can be seen as a concatenation of two subbranches: $B{\cdot}Suff(\phi'(B))$). Let us define $\Omega(\overline{M}) = \{Suff(\phi'(B))|B \in \Omega(M)\}$.
We must prove that there exists a protocol M' such that $P[\mathcal{P}]M'$ and $M'[\mathcal{Z}]Q$.
Then we build a protocol M' with the branches formed by the concatenation of each branch in $\Omega(P)$ with its corresponding branch in $\Omega(\overline{M})$ ($\Omega(M') = \{A \cdot Suff(\phi' \circ \phi(A))|A \in \Omega(P)\}$).
In order to prove $P[\mathcal{P}]M'$ we define the bijective function $\psi : \Omega(P) \to \Omega(M')$ with $\psi(A) = A \cdot Suff(\phi' \circ \phi(A))$, which satisfies $A =_{pre} \psi(A)$ by construction.
In order to prove $M'[\mathcal{Z}]Q$, notice that every $B \in \Omega(M')$ is of the form $A \cdot Suff(\phi' \circ \phi(A))$ for some $A \in \Omega(P)$; then we define the bijective function $\psi : \Omega(M') \to \Omega(Q)$ with $\psi(A \cdot Suff(\phi' \circ \phi(A))) = \phi(A) \cdot Suff(\phi' \circ \phi(A))$, which satisfies $A \cdot Suff(\phi' \circ \phi(A)) \ll_b \phi(A) \cdot Suff(\phi' \circ \phi(A))$ by definition of ϕ.

– $P[\mathcal{PZ}]Q \not\Rightarrow P[\mathcal{ZP}]Q$.
We show a counter-example.
Because of $P[\mathcal{PZ}]Q$, there exists a protocol M such that $P[\mathcal{P}]M$ and $M[\mathcal{Z}]Q$.
Looking at Fig. 6, let us suppose that $msc(fbb) \sqsubseteq msc(fb)$, $msc(faa) \sqsubseteq msc(fa)$ and no other relationship exists between fluents, so Q' in the picture fulfills the requirements.
We must prove that there does not exist a protocol M' such that $P[\mathcal{Z}]M'$ and $M'[\mathcal{P}]Q$. Notice that any protocol M' such that $M'[\mathcal{P}]Q$, should satisfy $M'[\mathcal{E}]Q$ or $M'[\mathcal{E}]Q'$; but it is obvious that not $P[\mathcal{Z}]Q$ neither $P[\mathcal{Z}]Q'$.

Property 6 : $\forall \mathcal{X} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}\}.\ \forall \mathcal{Y} \in \{\mathcal{P}, \mathcal{S}, \mathcal{I}\}.\ (\mathcal{X} \neq \mathcal{Y} \to\ (P[\mathcal{XY}]Q \Leftrightarrow\ P[\mathcal{I}]Q))$.

– $P[\mathcal{PS}]Q \Leftrightarrow P[\mathcal{I}]Q$. (A prefix of a suffix is an infix, and viceversa).
Proof: First we prove $[\mathcal{PS}] \Rightarrow [\mathcal{I}]$.
Because of $P[\mathcal{PS}]Q$, there exists a protocol M such that $P[\mathcal{P}]M$ and $M[\mathcal{S}]Q$.
According to the definitions of $\mathcal{P}$ and $\mathcal{S}$, $\exists\phi : \Omega(P) \to \Omega(M)$ bijective, such that $\forall A \in \Omega(P).\ A =_{pre} \phi(A)$ and $\exists\phi' : \Omega(M) \to \Omega(Q)$ bijective, such that $\forall B \in \Omega(M).\ B =_{suf} \phi'$(B).
Then, the bijective function $\psi : \Omega(P) \to \Omega(Q)$ with $\psi(A) = \phi'(\phi(A))$ satisfies $A =_{inf} \psi(A)$, since $\forall A \in \Omega(P).\ A =_{pre} \phi(A) =_{suf} \phi'(\phi(A))$, and due to definition of $=_{suf}$, there exists a state s_a in $\phi'(\phi(A))$ such that $A =_{pre} \phi(A) =_b ChangeInit[\phi'(\phi(A))/s_a]$, which implies that $A =_{pre} ChangeInit[\phi'(\phi(A))/s_a]$, which is the definition for $P[\mathcal{I}]Q$.
Next we prove $[\mathcal{I}] \Rightarrow [\mathcal{PS}]$.

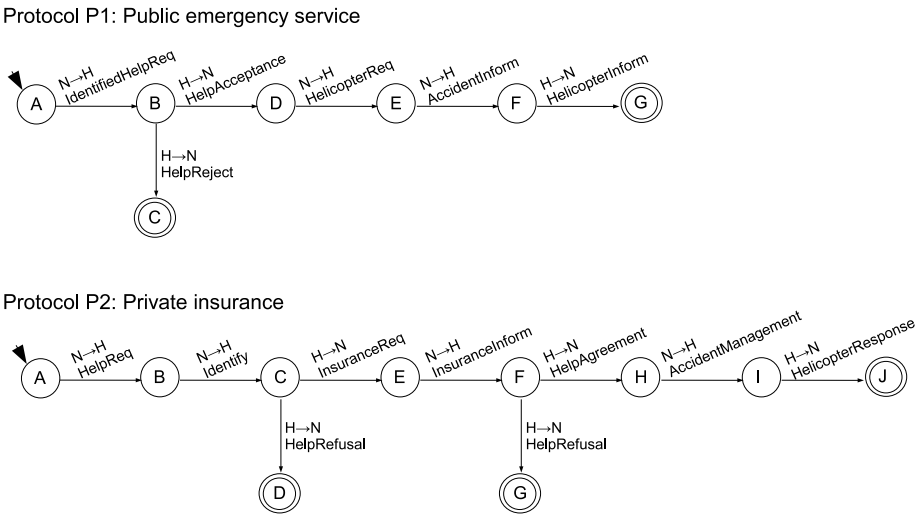


Fig. 7 One scenario of the proposed mechanism.

Because of $P[\mathcal{I}]Q$, $\exists\phi : \Omega(P) \rightarrow \Omega(Q)$ bijective, such that $\forall A \in \Omega(P)$. $A =_{inf} \phi(A)$, which means that $\forall A \in \Omega(P)$ there exists a state $s_{\phi(A)}$ in $\phi(A)$ such that $A =_{pre} \mathit{ChangeInit}[\phi(A)/s_{\phi(A)}] =_{suf} \phi(A)$. We must prove that there exists a protocol M such that $P[\mathcal{P}]M$ and $M[\mathcal{S}]Q$. We define $\Omega(M) = \{\mathit{ChangeInit}[\phi(A)/s_{\phi(A)}] \mid A \in \Omega(P)\}$. Then, the functions $\psi : \Omega(P) \rightarrow \Omega(M)$ with $\psi(A) = \mathit{ChangeInit}[\phi(A)/s_{\phi(A)}]$ and $\psi' : \Omega(M) \rightarrow \Omega(Q)$ with $\psi'(\mathit{ChangeInit}[\phi(A)/s_{\phi(A)}]) = \phi(A)$ are bijective and justify $P[\mathcal{P}]M$ and $M[\mathcal{S}]Q$.

To conclude this section, we want to point out that our implemented mechanism discovers complex composition relationships, like those presented in this section, between two given protocols. The following section presents an example.

6 One scenario of the proposed mechanism at work

The aim of this section is to show one scenario that illustrates the different steps that the proposed mechanism follows in order to discover the relationship between two different protocols that could be activated in case of a road accident. In both cases, the actors that interact are a nurse from the ambulance that covers the emergency and a doctor from the emergency staff from a hospital. Those actors are represented by software agents that belong to different Information Systems. Notice that those agents use protocols constituted by communication acts that are specific to each Information System, and which are a specialization of general communication act classes defined in the upper level of the CommOnt ontology.

On the one hand, protocol $P1$ in Fig. 7 illustrates the interaction with a public hospital. The protocol is initiated by a nurse from the ambulance, who sends a request (`IdentifiedHelpReq`) for help identifying herself within the message. Depending on the situation, the hospital staff can either reject (`HelpReject`) or accept the request and ask information about the accident (`HelpAcceptance`). In the latter case, then

Fig. 8 Description of the communication acts

the nurse requests a helicopter to take the injured to the hospital (`HelicopterReq`) and sends a message with the information about the accident (`AccidentInform`). Finally, the hospital staff responds whether it is possible or not to send a helicoper (`HelicopterInform`).

On the other hand, protocol $P2$ shows the interaction with a private hospital, where patients are required to have an insurance policy. In the first step, the nurse sends a request for help (`HelpReq`) followed by a message of self-identification (`Identify`). The hospital staff may either reject the request (`HelpRefusal`) or ask the nurse about the patient's insurance details (`InsuranceReq`), in which case the nurse informs about those details (`InsuranceInform`). After checking those details, the hospital staff decides whether the patient is eligible for help (`HelpAgreement`) or not (`HelpRefusal`). In the former case, then the nurse sends a message in which she gives information about the accident and requests a helicopter (`AccidentManagement`). Finally, the hospital staff responds whether it is possible or not to send a helicoper (`HelicopterResponse`).

It can be noticed that both protocols are quite different. In Fig. 8 the description and explanations of the communication acts that appear in the protocols are provided.

Next, the different steps presented in the previous sections (4.1, 4.2 and 4.3) are illustrated for the scenario under consideration.

Separation into branches: There are two different branches in protocol $P1$ and three different branches in protocol $P2$. For brevity, we represent a branch with the sequence of its states. Then, protocol $P1$ is separated into the branches $B1.1$=[A,B,C] and $B1.2$=[A,B,D,E,F,G] and protocol $P2$ into $B2.1$=[A,B,C,D], $B2.2$=[A,B,C,E,F,G] and $B2.3$=[A,B,C,E,F,H,I,J].

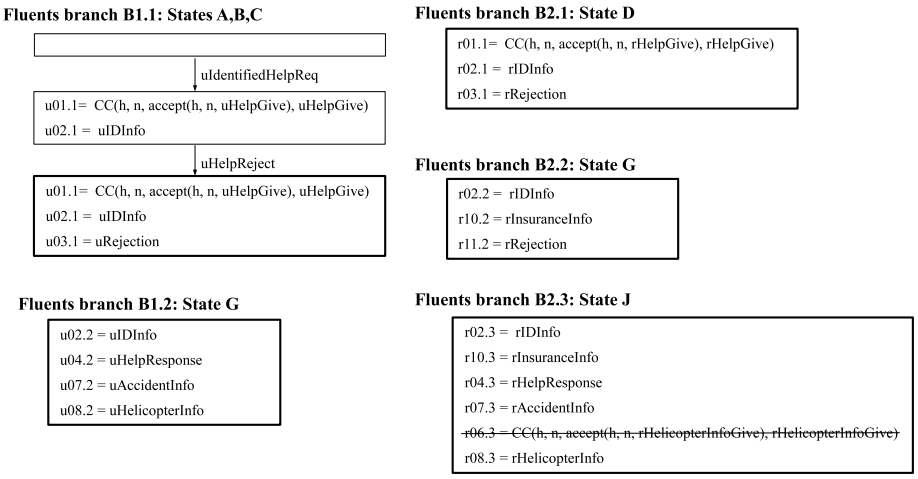


Fig. 9 Fluents generated by the branches

Generation of fluents: In Fig. 9 the fluents generated by each branch are shown. The fluents generated by protocol $P1$ and the instances of communication acts in that protocol are prefixed by u (public), while those generated by protocol $P2$ are prefixed by r (private). Below a detailed description of the generation process of the fluents in branch $B1.1$ is provided. The generation process of the remaining branches can be found in appendix A.

Branch B1.1

$< (A, uIdentifiedHelpReq, B) >$: Nurse n sends the message *uIdentifiedHelpReq* to hospital h.

Due to the following knowledge:

$$\texttt{uIdentifiedHelpReq} \in \texttt{IdentifiedHelpReq}$$
$$\texttt{IdentifiedHelpReq} \sqsubseteq \texttt{Request}$$
$$Initiates(Request(s,r,P), \mathsf{CC}(r,\ s,\ accept(r,s,P),\ P),\ t)$$

the fluent $u01.1 = CC(h,\ n,\ accept(h,n,uHelpGive),\ uHelpGive)$ is initiated.

Moreover, due to:

$$\texttt{uIdentifiedHelpReq} \in \texttt{IdentifiedHelpReq}$$
$$\texttt{IdentifiedHelpReq} \sqsubseteq \texttt{Inform} \sqsubseteq \texttt{Assertive}$$
$$Initiates(Assertive(s,r,P),\ P,\ t)$$

the fluent $u02.1 = uIDInfo$ is initiated. Both fluents are attached to state B due to the fluent attachment rule.

$< (B, uHelpReject, C) >$: Hospital h sends the message *uHelpReject* to nurse n.

Due to the following knowledge:

$$\texttt{uHelpReject} \in \texttt{HelpReject}$$
$$\texttt{HelpReject} \sqsubseteq \texttt{Reject}$$
$$Initiates(Reject(s,r,P),\ reject(s,r,P),\ t)$$

	B1.1 u01.1 / u02.1 / u03.1	B1.2 u02.2 / u04.2 / u07.2 / u08.2
B2.1 r01.1 r02.1 r03.1	(3,0,0,0) f(3,0,0,0)=3/3=1	X
B2.2 r02.2 r10.2 r11.2	X	X
B2.3 r02.3 r10.3 r04.3 r07.3 r08.3	X	(4,0,0,1) f(4,0,0,1)=4/5=0.8

Fig. 10 Branches comparison table

the fluent $u03.1$ =*reject(h, n, uHelpGive)* is generated and attached to state C due to the fluent attachment rule. Moreover, fluents $u01.1$ and $u02.1$ are transmitted from state B to state C due to the fluent transmission rule.

Comparison of branches: For the sake of the example, we have labelled the generated fluents with a code in the format $pX.Y$, where $p \in$ {u,r} refers to the protocol that generates the fluent, $X \in$ {01,...,n} is a key number for identifying fluents and $Y \in$ {1,2,3} is a number for distinguising fluents from different branches. In our coding, number X identifies a fluent, that is to say $r02.1$ is related to $u02.1$ via an (eq) relationship (see section 4.3). Taking the previous considerations into account, the branches comparison table displays the results in Fig. 10.

Remember that the tuple (4,0,0,1) indicates that four fluents in the trace of $B2.3$ and four fluents in the trace of $B1.2$ can be paired up through an equivalence relationship, none of the remaining fluents can be paired up through a generalization relationship and there is one exceeding fluent (due to the fact that the trace of $B2.3$ has five fluents and the trace of $B1.2$ has only four fluents).

Moreover, there are four unfeasible pairs: ($B1.2$, $B2.1$), ($B1.1$, $B2.2$), ($B1.2$, $B2.2$) and ($B1.1$, $B2.3$). For example, ($B1.1$, $B2.2$) is not feasible because in the trace of $B1.1$ there are some fluents ($u01.1$ and $u03.1$) which are incomparable with any other in the trace of $B2.2$, and in turn, in the trace of $B2.2$ there are other fluents ($r10.2$ and $r11.2$) which are incomparable with any other in the trace of $B1.1$.

The best pairing can be obtained pairing up branch $B1.1$ with branch $B2.1$ and branch $B1.2$ with $B2.3$ ($\pi = \{(B1.1, B2.1), (B1.2, B2.3)\}$). In addition, since all the fluents in $B1.1$ have been related through an equivalence relationship *(eq)* with those in $B2.1$ ((3,0,0,0) in the table) it can be said that those branches are equivalent ($B1.1 =_b B2.1$). Furthermore, due to the (4,0,0,1) valuation given to ($B1.2$, $B2.3$), it can be seen that $B1.2$ is somehow included in $B2.3$. Since the fluent that has not been paired up ($r10.3$) is generated in an inner state of $B2.3$, it results that $B1.2 =_c B2.3$.

In summary, two of the three branches of $P2$ can be related with those in $P1$ (notion of restriction $\mathcal{R}$), those branches are related via the complement_to_infix relationship $\mathcal{C}$ and the relationship between the compared fluents is the one of equivalence $\mathcal{E}$, so we can say that $P1[\mathcal{ECR}]P2$.

Therefore, our mechanism has discovered a semantic relationship between those two protocols and this is an important first step in order to deal with alternatives of successful communication of the considered agents.

For instance, assume that the ambulance service is ascribed to a private hospital organization and, therefore, the nurse software agent is initially adapted to use the $P2$ protocol. Moreover, the accident location recommends to ask for help to a public hospital, whose doctor software agent will be initially adapted to the $P1$ protocol. In particular, in our scenario, the nurse and doctor software agents (using $P2$ and $P1$ protocols, respectively) need some help to interoperate. The relationship discovered by our mechanism allows the system to recognize a chance of interoperability. The confrontation of both protocols has proved that the nurse agent could adopt protocol $P1$ to communicate with the doctor agent because $P1$ does not oblige the nurse agent to do any action against the goals of its default protocol $P2$. Moreover, the doctor agent would be unaware of such adaptation and could continue using its regular protocol $P1$.

7 Related Works

Different criteria can be considered for a classification of related works that can be found in the specialized literature. One significant criterion, with respect to protocol definitions, is that which distinguishes among those works that prevail in a semantic approach in contrast with others that only consider a structural one. Another interesting criterion is the consideration or not of formal relationships between protocols. A third criterion is if the works consider a notion of protocol composition.

We will firstly review the first criterion that is, works which introduce semantic considerations when defining protocols.

The closer work is (Mallya and Singh, 2007), where protocols are represented as transition systems where transitions are formalized as operations on commitments. Subsumption and equivalence of protocols are defined with respect to three state similarity funtions. We share some goals with that work, but in that paper there are no references to how protocol relationships are computed. In contrast, according to our proposal, protocol relationships can be computed by straightforward algorithms. It is worth mentioning that protocol relationships considered in that paper deserve study within our framework. Another work that consider protocol relationships is (Desai et al, 2005), where protocols are represented with a set of rules with terms obtained from an ontology. In particular they formalize protocols into the π-calculus; then, equivalence through bisimulation is the only relationship considered.

Among other works that also consider semantic aspects but do not consider the study of relationships between protocols, the works (Yolum and Singh, 2002), (Fornara and Colombetti, 2003) and (Kagal and Finin, 2007) can be mentioned. The first two are quite similar one to another. Both capture the semantics of communication acts through agents' commitments and represent communication protocols using a set of rules that operate on these commitments. Moreover, those rule sets can be compiled as Finite State Machines. In (Kagal and Finin, 2007), protocols are defined as a set of permissions and obligations of agents participating in the communication. They use an OWL ontology for defining the terms of the specification language, but their basic reasoning is made with an *ad hoc* reasoning engine. We share with the previous works their main goal of defining protocols in a general framework that allows them to be re-used.

One work that focuses on the problem of protocol composition is (Yolum and Singh, 2007). It introduces considerations of rationality on the enactment of protocols. Our proposal could be complemented with the ideas presented in that work.

We review secondly works that deal solely with structural aspects of protocols. Notice that we advocate for a more flexible approach dealing with semantics.

The work presented in (Montes-Rendón et al, 2006) describes a methodology that facilitates communication among heterogeneous negotiation agents based on the alignment of communication primitives. Finite State Machines are used as a model to represent negotiation protocols and as a base for aligning primitives. This is quite a rigid approach because communication will only be possible if protocols have such a similar structure that their communication primitives can be aligned two by two.

(d'Inverno et al, 1998) and (Mazouzi et al, 2002) use Finite State Machines and Petri nets, respectively, to represent protocols but without taking into account the meaning of the communication acts interchanged or considering relationships between protocols. The work of (Ryu et al, 2007) has an interesting approach to manage changes in business protocols. More precisely, it presents an extensive study on how to translate active instances from an old protocol to a new one, without violating several types of constraints. In order to do so, it compares the old protocol with the new one but only examines structural differences between them, which results in a more rigid approach that the one allowed by our mechanism, which considers semantic representation of protocols.

For the third criterion we issue the problem of determining if an agent's policy conforms to a protocol. This is a very relevant problem but not one which we are examining in this paper. Nevertheless, the topic is closely related to ours and it is worth mentioning here. In (Endriss et al, 2003), deterministic Finite State Machines are the abstract models for protocols, which are described by simple logic-based programs. Three levels of conformance are defined: weak, exhaustive and robust. They consider communication acts as atomic actions, in contrast to our semantic view. In (Baldoni et al, 2006) a nondeterministic Finite State Automata is used to support a notion of conformance that guarantees interoperability among agents which conform to a protocol. Their conformance notion considers the branching structure of policies and protocols and applies a simulation-based test. Communication acts are considered atomic actions, without considering their semantics. In (Chopra and Singh, 2006), a notion of conformance is defined and, moreover, it is proved orthogonal to their proposed notions of coverage and interoperability.

Apart from the previous classification, in the context of Web Services, state transition systems are used in (Bordeaux et al, 2004) for representing dynamic behaviour of services and defining some notions of compatibility and substitutability of services that can be easily translated to the context of compatibility of protocols. Relationships between their compatibility relations and our defined relationships deserve study. Finally, although agent technology and Web Services technology have been developed in separate ways, there exists a work (Greenwood and M.Lyell, 2007) which tries to consolidate their approaches into a common specification describing how to seamlessly interconnect FIPA compliant agent systems (FIPA, 2005) with W3C compliant Web Services. The purpose of specifying an infrastructure for integrating these two technologies is to provide a common means of allowing each to discover and invoke instances of the other.

8 Conclusions

In this paper we have explained a mechanism for discovering semantic relationships among agent communication protocols. The mechanism is based on the idea that different protocols can be compared semantically by looking to the set of fluents associated to the branches of protocols. That assumption favours a much more flexible comparison of protocols than the more traditional one based on comparing protocol structures. Through the paper we have shown first, one ontology that represents concepts related to communication acts that agents use to communicate with each other and which take part of the protocols. Secondly, we have concentrated on showing: how protocols are modelled, in our case using OWL-DL language, and the features of the mechanism that analyses each protocol by decomposing it into different branches, and by generating the semantic information associated to each branch. Thirdly, we have presented different relationship definitions that are managed by the mechanism which permit the identification of protocol relationships. Moreover, some properties of those relationships and proofs for them have been included. Finally, with an example we have illustrated the feasibility of the proposed mechanism which has been implemented using Java as a programming language and Pellet as a description logic reasoner. As a future work we are studying how to extend the mechanism presented in this paper so that protocols with cycles can be treated. Since our mechanism divides the protocol into its different branches (one for each of the possible runs) and the number of those possible different runs in a protocol that contains cycles is infinite, an exhaustive study of how to simplify the cases must be done. We are working currently towards an approach that analyzes the behaviour of the fluents that hold in the *exit* states of cycles (that is to say, how those fluents are affected by the fact of being in a cycle), with the purpose of establishing a pattern about that behaviour. Another future work studies the interest of adapting the proposed mechanism to contexts where Information Systems are represented through Web Services.

Acknowledgements The work of Idoia Berges is supported by a grant of the Basque Government (Programa de Formación de Investigadores del Departamento de Educación, Universidades e Investigación). This work is also supported by the Basque Country Government IT-427-07 and the Spanish Ministry of Education and Science TIN2007-68091-C02-01.

References

(1993) Specification of the KQML Agent-Communication Language. DARPA Knowledge Sharing Initiative, http://www.cs.umbc.edu/kqml/kqmlspec/spec.html

(2008) Web Ontology Language (OWL) Guide Version 1.0. http://www.w3.org/2001/sw/WebOnt/guide-src/Guide.html

Austin JL (ed) (1962) How to do things with words. Oxford University Press

Baldoni M, Baroglio C, Martelli A, Patti V (2006) A priori conformance verification for guaranteeing interoperability in open environments. In: ICSOC, pp 339–351

Bermúdez J, Goñi A, Illarramendi A, Bagüés MI (2007) Interoperation among agent-based information systems through a communication acts ontology. Inf Syst 32(8):1121–1144

Bordeaux L, Salaün G, Berardi D, Mecella M (2004) When are two web services compatible? In: TES, pp 15–28

Chopra AK, Singh MP (2006) Producing compliant interactions: Conformance, coverage, and interoperability. In: DALT, pp 1–15

Desai N, Mallya AU, Chopra AK, Singh MP (2005) Interaction protocols as design abstractions for business processes. IEEE Trans Softw Eng 31(12):1015–1027

d'Inverno M, Kinny D, Luck M (1998) Interaction protocols in agentis. In: In Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), pp 261–268

Endriss U, Maudet N, Sadri F, Toni F (2003) Logic-based agent communication protocols. In: Workshop on Agent Communication Languages, pp 91–107

FIPA (2005) FIPA communicative act library specification. Http://www.fipa.org/specs/fipa00037/SC00037J.html

Fornara N, Colombetti M (2002) Operational specification of a commitment-based agent communication language. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press, New York, NY, USA, pp 536–542

Fornara N, Colombetti M (2003) Defining interaction protocols using a commitment-based agent communication language. In: AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, ACM Press, New York, NY, USA, pp 520–527

Greenwood D, MLyell HS A Mallya (2007) The IEEE FIPA approach to integrating software agents and web services. In: International Conference on Autonomous Agents and Multiagent Systems AAMAS, Hawaii USA, pp 14–18

Kagal L, Finin T (2007) Modeling conversation policies using permissions and obligations. Autonomous Agents and Multi-Agent Systems 14(2):187–206

Mallya AU, Singh MP (2007) An algebra for commitment protocols. Autonomous Agents and Multi-Agent Systems 14(2):143–163

Mazouzi H, Seghrouchni AEF, Haddad S (2002) Open protocol design for complex interactions in multi-agent systems. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press, New York, NY, USA, pp 517–526

Montes-Rendón A, Bravo M, Velázquez-Hernández JC (2006) An ontology-based methodology for communicating negotiation agents over internet. In: Proceedings of the 2006 Joint Conference on Information Sciences, JCIS 2006, Kaohsiung, Taiwan

Ryu SH, Saint-Paul R, Benatallah B, Casati F (2007) A framework for managing the evolution of business protocols in web services. In: Conceptual Modelling 2007, Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia

Shanahan M (1999) The event calculus explained. Lecture Notes in Computer Science 1600:409–430

Singh MP (1998) Agent Communication Languages: Rethinking the Principles. IEEE Computer 31(12):40–47

Singh MP (2000) A social semantics for agent communication languages. In: Issues in Agent Communication, Springer-Verlag, pp 31–45

Venkatraman M, Singh MP (1999) Verifying compliance with commitment protocols. Autonomous Agents and Multi-Agent Systems 2(3):217–236

Wooldridge M (2000) Semantic Issues in the Verification of Agent Comunication Languages. Journal of Autonomous Agents and Multi-Agent Systems 3(1):9–31

Yolum P, Singh M (2007) Enacting protocols by commitment concession. In: International Conference on Autonomous Agents and Multiagent Systems AAMAS, Hawaii USA, pp 116–123

Yolum P, Singh MP (2002) Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press, pp 527–534

A Generation of fluents

Most important facts in the Knowledge Base:

(k1) `uIdentifiedHelpReq` $\in$ `IdentifiedHelpReq` $\sqsubseteq$ `Request`

(k2) `uIdentifiedHelpReq` $\in$ `IdentifiedHelpReq` $\sqsubseteq$ `Inform` $\sqsubseteq$ `Assertive`

(k3) `uHelpReject` $\in$ `HelpReject` $\sqsubseteq$ `Reject`

(k4) `uHelpAcceptance` $\in$ `HelpAcceptance` $\sqsubseteq$ `Responsive`

(k5) `uHelpAcceptance` $\in$ `HelpAcceptance` $\sqsubseteq$ `Inquiry`

(k6) `uHelicopterReq` $\in$ `HelicopterReq` $\sqsubseteq$ `Request`

(k7) `uAccidentInform` $\in$ `AccidentInform` $\sqsubseteq$ `Responsive`

(k8) `uHelicopterInform` $\in$ `HelicopterInform` $\sqsubseteq$ `Responsive`

(k9) `rHelpReq` $\in$ `HelpReq` $\sqsubseteq$ `Request`

(k10) `rIdentify` $\in$ `Identify` $\sqsubseteq$ `Inform` $\sqsubseteq$ `Assertive`

(k11) `rHelpRefusal` $\in$ `HelpRefusal` $\sqsubseteq$ `Reject`

(k12) `rInsuranceReq` $\in$ `InsuranceReq` $\sqsubseteq$ `Request`

(k13) `rInsuranceInform` $\in$ `InsuranceInform` $\sqsubseteq$ `Responsive`

(k14) `rHelpRefusal` $\in$ `HelpRefusal` $\sqsubseteq$ `Reject`

(k15) `rHelpAgreement` $\in$ `HelpAgreement` $\sqsubseteq$ `Responsive`

(k16) `rHelpAgreement` $\in$ `HelpAgreement` $\sqsubseteq$ `Inquiry`

(k17) `rAccidentManagement` $\in$ `AccidentManagement` $\sqsubseteq$ `Responsive`

(k18) `rAccidentManagement` $\in$ `AccidentManagement` $\sqsubseteq$ `Request`

(k19) `rHelicopterResponse` $\in$ `HelicopterResponse` $\sqsubseteq$ `Responsive`

Predicates that describe the semantics associated to the communication acts of the example:

(p1) *Initiates(Request(s, r, P)*, CC*(r, s, accept(r, s, P), P), t).*

(p2) *Initiates(Inquiry(s, r, P)*, CC*(r, s, accept(r, s, P), P), t).*

(p3) *Terminates(Responsive(s, r, P, RA)*,CC*(s, r, accept(s, r, RA), RA), t).*

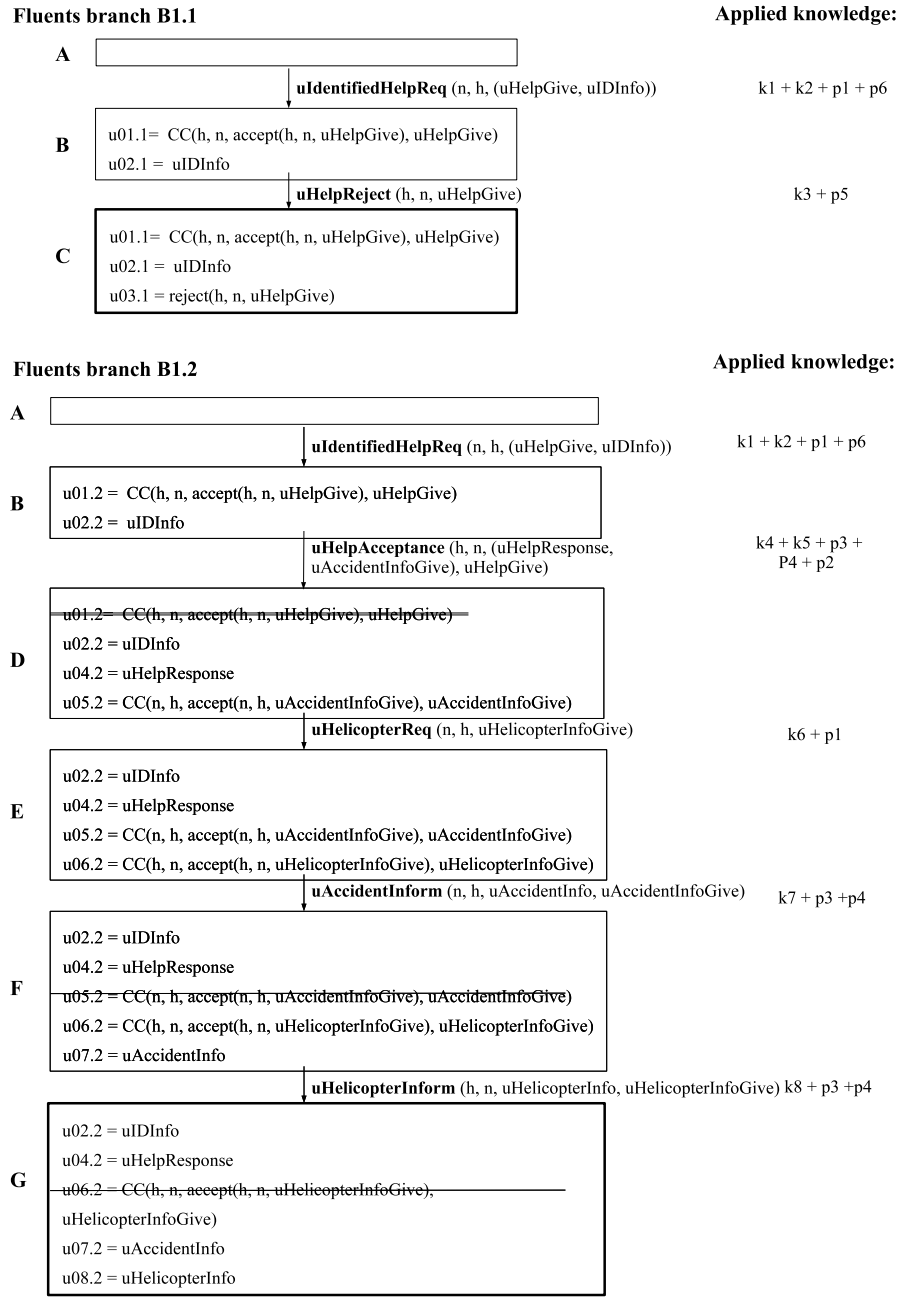
(p4) *Initiates(Responsive(s, r, P, RA), P)*

(p5) *Initiates(Reject(s, r, P), reject(s, r, P))*

(p6) *Initiates(Assertive(s, r, P), P)*

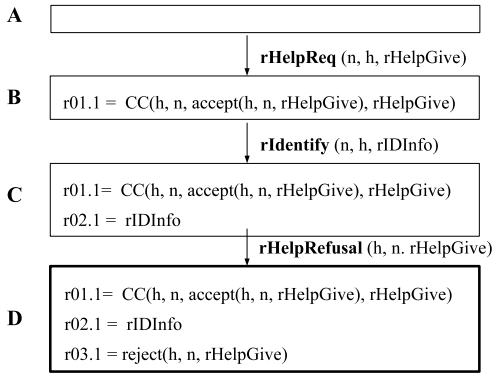
Next, a detailed description of the generation process of the trace of each branch is shown. Moreover, the knowledge that has been applied at each stage of the interaction is indicated. The *fluent attachment* and *fluent transmission* rules do not appear explicitly in this analysis but have been applied when necessary.

Appendix 1

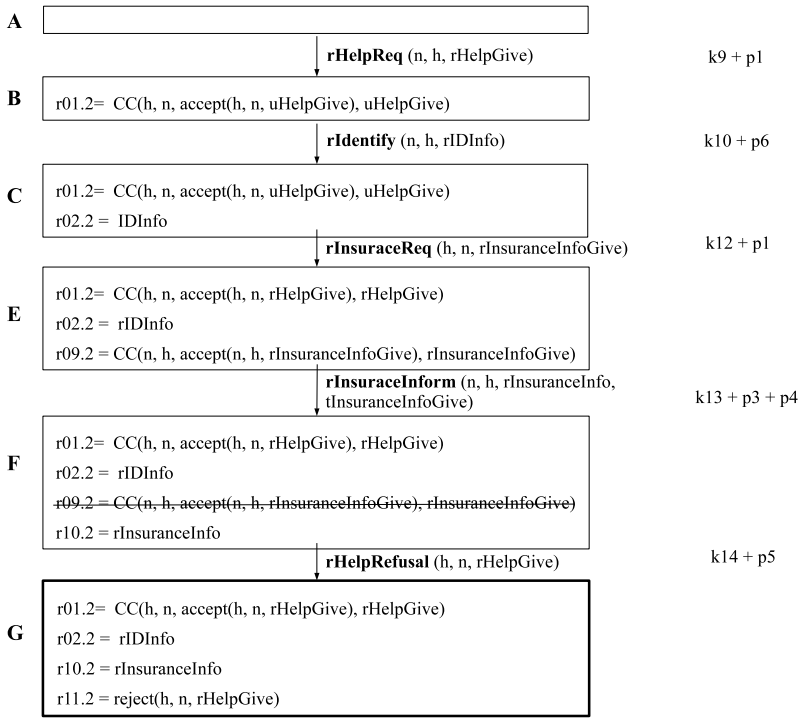

Fluents branch B1.1
Applied knowledge:
A
uIdentifiedHelpReq (n, h, (uHelpGive, uIDInfo))
k1 + k2 + p1 + p6
B
u01.1= CC(h, n, accept(h, n, uHelpGive), uHelpGive)
u02.1 = uIDInfo
uHelpReject (h, n, uHelpGive)
k3 + p5
C
u01.1= CC(h, n, accept(h, n, uHelpGive), uHelpGive)
u02.1 = uIDInfo
u03.1 = reject(h, n, uHelpGive)
Fluents branch B1.2
Applied knowledge:
A
uIdentifiedHelpReq (n, h, (uHelpGive, uIDInfo))
k1 + k2 + p1 + p6
B
u01.2 = CC(h, n, accept(h, n, uHelpGive), uHelpGive)
u02.2 = uIDInfo
uHelpAcceptance (h, n, (uHelpResponse, uAccidentInfoGive), uHelpGive)
k4 + k5 + p3 + P4 + p2
D
u01.2= CC(h, n, accept(h, n, uHelpGive), uHelpGive)
u02.2 = uIDInfo
u04.2 = uHelpResponse
u05.2 = CC(n, h, accept(n, h, uAccidentInfoGive), uAccidentInfoGive)
uHelicopterReq (n, h, uHelicopterInfoGive)
k6 + p1
E
u02.2 = uIDInfo
u04.2 = uHelpResponse
u05.2 = CC(n, h, accept(n, h, uAccidentInfoGive), uAccidentInfoGive)
u06.2 = CC(h, n, accept(h, n, uHelicopterInfoGive), uHelicopterInfoGive)
uAccidentInform (n, h, uAccidentInfo, uAccidentInfoGive)
k7 + p3 +p4
F
u02.2 = uIDInfo
u04.2 = uHelpResponse
u05.2 = CC(n, h, accept(n, h, uAccidentInfoGive), uAccidentInfoGive)
u06.2 = CC(h, n, accept(h, n, uHelicopterInfoGive), uHelicopterInfoGive)
u07.2 = uAccidentInfo
uHelicopterInform (h, n, uHelicopterInfo, uHelicopterInfoGive)
k8 + p3 +p4
G
u02.2 = uIDInfo
u04.2 = uHelpResponse
u06.2 = CC(h, n, accept(h, n, uHelicopterInfoGive),
uHelicopterInfoGive)
u07.2 = uAccidentInfo
u08.2 = uHelicopterInfo

Appendix 2

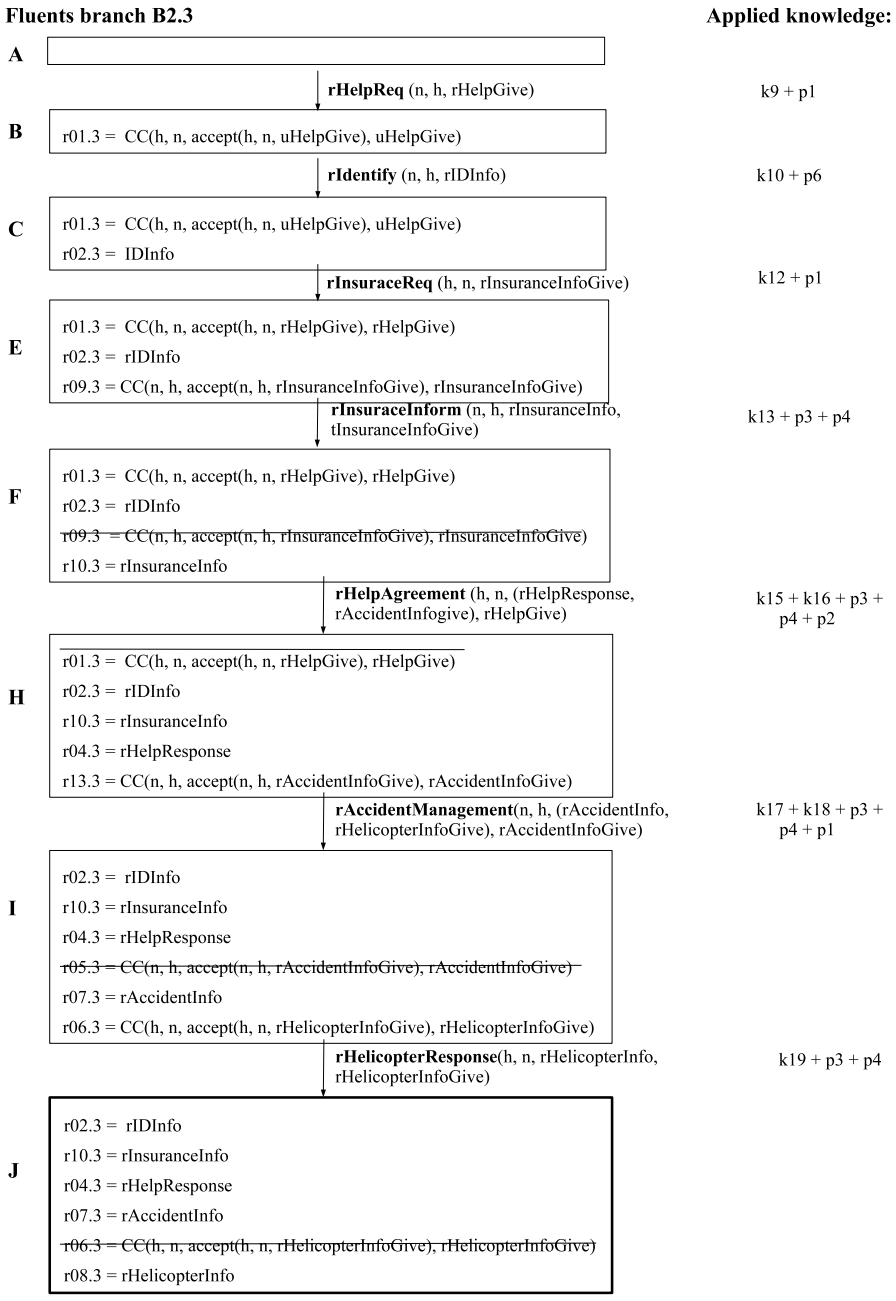
Fluents branch B2.1 **Applied knowledge:**



Fluents branch B2.2 **Applied knowledge:**



Appendix 3


Fluents branch B2.3
Applied knowledge:
A
rHelpReq (n, h, rHelpGive)
k9 + p1
B
r01.3 = CC(h, n, accept(h, n, uHelpGive), uHelpGive)
rIdentify (n, h, rIDInfo)
k10 + p6
C
r01.3 = CC(h, n, accept(h, n, uHelpGive), uHelpGive)
r02.3 = IDInfo
rInsuraceReq (h, n, rInsuranceInfoGive)
k12 + p1
E
r01.3 = CC(h, n, accept(h, n, rHelpGive), rHelpGive)
r02.3 = rIDInfo
r09.3 = CC(n, h, accept(n, h, rInsuranceInfoGive), rInsuranceInfoGive)
rInsuraceInform (n, h, rInsuranceInfo, tInsuranceInfoGive)
k13 + p3 + p4
F
r01.3 = CC(h, n, accept(h, n, rHelpGive), rHelpGive)
r02.3 = rIDInfo
r09.3 = CC(n, h, accept(n, h, rInsuranceInfoGive), rInsuranceInfoGive)
r10.3 = rInsuranceInfo
rHelpAgreement (h, n, (rHelpResponse, rAccidentInfogive), rHelpGive)
k15 + k16 + p3 + p4 + p2
H
r01.3 = CC(h, n, accept(h, n, rHelpGive), rHelpGive)
r02.3 = rIDInfo
r10.3 = rInsuranceInfo
r04.3 = rHelpResponse
r13.3 = CC(n, h, accept(n, h, rAccidentInfoGive), rAccidentInfoGive)
rAccidentManagement(n, h, (rAccidentInfo, rHelicopterInfoGive), rAccidentInfoGive)
k17 + k18 + p3 + p4 + p1
I
r02.3 = rIDInfo
r10.3 = rInsuranceInfo
r04.3 = rHelpResponse
r05.3 = CC(n, h, accept(n, h, rAccidentInfoGive), rAccidentInfoGive)
r07.3 = rAccidentInfo
r06.3 = CC(h, n, accept(h, n, rHelicopterInfoGive), rHelicopterInfoGive)
rHelicopterResponse(h, n, rHelicopterInfo, rHelicopterInfoGive)
k19 + p3 + p4
J
r02.3 = rIDInfo
r10.3 = rInsuranceInfo
r04.3 = rHelpResponse
r07.3 = rAccidentInfo
r06.3 = CC(h, n, accept(h, n, rHelicopterInfoGive), rHelicopterInfoGive)
r08.3 = rHelicopterInfo