Data Mining Agent Conversations: A Qualitative Approach to Multiagent Systems Analysis

Emilio Serrano [a] Michael Rovatsos [b] Juan A. Botía [a]

[a] *Universidad de Murcia, Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain*

[b] *School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, United Kingdom*

Abstract

This paper presents a novel method for analysing the behaviour of multiagent systems on the basis of the semantically rich information provided by agent communication languages and interaction protocols specified at the knowledge level. More low-level communication mechanisms only allow for a *quantitative* analysis of the occurrence of message types, the frequency of message sequences, and the empirical distributions of parameter values. Quite differently, the semantics of languages and protocols in multiagent systems can help to extract *qualitative* properties of observed conversations among agents. This can be achieved by interpreting the logical constraints associated with protocol execution paths or individual messages as the *context* of an observed interaction, and using them as features of learning samples. The contexts "mined" from such analyses, or *context models*, can then be used for various tasks, e.g. for predicting others' future responses (useful when trying to make strategic communication decisions to achieve a particular outcome), to support ontological alignment (by comparing the properties of logical constraints attached to messages across participating agents), or to assess the trustworthiness of agents (by verifying the logical coherence of their behaviour). This paper details a formal approach that describes our notion of context models in multiagent conversations, an implementation of this approach in a practical tool for mining qualitative context models, and experimental results to illustrate its use and utility.

Key words:
Agent communication languages, interaction protocols, interaction analysis, data mining, agent-oriented software engineering

Email addresses: emilioserra@um.es (Emilio Serrano), mrovatso@inf.ed.ac.uk (Michael Rovatsos), juanbot@um.es (Juan A. Botía).

Preprint submitted to Information Sciences *Received: date / Accepted: date*

1 Introduction

One of the cornerstones of agent technology is the loose coupling between agents achieved by introducing standardized high-level agent communication languages (ACLs, e.g. FIPA-ACL [6]) and interaction protocols. As opposed to low-level interaction mechanisms for computer systems (like those used in traditional distributed computing), these advanced languages and protocols attempt to capture shared meaning for messages exchanged in multiagent systems. This helps to ensure that, despite the heterogeneity among individual agents who cannot observe each other's internal states, some level of interoperability can be achieved in practice, so that large-scale open multiagent systems can be implemented in the real world.

At the same time, when it comes to *analysing* agent-based systems, the openness of these systems limits the available data to what goes on *among* agents (i.e. observations of message exchanges, at least if we assume that the observer does not have access to all internal details of all agents in the system [15]). However, the structure and "knowledge-level" assumptions captured in ACLs and interaction protocols is semantically rich and can be used to partially compensate for the loss of transparency caused by agent-level encapsulation, which makes the mental states of an agent opaque to others.

As an example of this, consider a message $\texttt{inform}(A, B, X)$ with the usual meaning that agent A informs B of a fact X (where X is taken from some domain ontology or "content language"). The use of this message type is usually tied to preconditions like $(Bel\ A\ X)$ stating that A in fact believes X to be true. While B is unable to verify whether this is *actually* the case (or A is lying/has a different interpretation of the Bel modality), the use of the message entitles B to operate under the assumption that $(Bel\ A\ X)$ is true for A. For example, if B contested X, it would be unreasonable for a protocol to allow A to state that she never claimed X. Therefore, at a *pragmatic* level, any semantic "annotations" (pre- and post-conditions) of messages that an agent is uttering can be used as assumptions about that agent's mental state (or, e.g. in commitment-based semantics [7], about their perception of a social state).

Quite surprisingly, this aspect of data analysis has been overlooked in the existing literature (see section 2). Existing approaches remain at the *quantitative* level, i.e. any measurements they take are based on assessing the observed values of some attributes of the interaction. A binary distinction between "interaction was successful or not" is often employed, sometimes also a measurement of the quality of different attributes along numerical scales, e.g. speed, price, reliability etc. While in non-agent scenarios this may be the only kind of data that is available, if one focuses only on quantitative analysis, a lot of additional structural information goes "to waste" in a sense when considering

ACL-based multiagent system interactions.

The contribution of this paper is to fill this gap by exploring the use of data mining techniques over semantically rich interaction protocols defined using typical ACLs. By using semantic elements of protocols as features of interaction traces, which are available as data samples from past interactions, we can inductively derive what we call *context models* i.e. logical theories that capture regularities in previously observed interactions. Note that the protocol definition or *protocol model* needs to include semantic annotations to allow the approach presented in this paper to build an effective context model automatically, and, in this paper, we will make the assumption that protocol specifications include such annotations.

Context models, which essentially capture generalised information about the conditions under which a protocol reaches a certain outcome, can be used for various purposes: (1) to make predictions about future behaviour (e.g. under what circumstances a peer is likely to deliver a product of reasonable quality); (2) to infer the definitions other agents use when validating logical constraints during an interaction (e.g. when acceptance of a certain type of offer indicates the range of a variable X for which the predicate $acceptable(X)$ holds true for an agent); and (3) to analyse the reliability and trustworthiness of agents based on the logical coherence of their utterances (e.g. if an agent has been observed to suggest that both $P(o)$ and $\neg P(o)$ are true of the same object o). Moreover, by simply grouping the data from interactions with several agents together and analysing it with a data mining algorithm, it is easy to generalise over individual "theories" held by them to develop a more global picture of the views held by an entire set of agents.

The remainder of the paper is structured as follows: After reviewing related work in section 2, we define the formal machinery that is needed to develop qualitative data mining methods for interaction protocols in section 3. Sections 4 and 5 discuss an implemented system for qualitative data mining over interaction protocols, and empirical results obtained in a case study, respectively. Section 6 concludes.

2 Related work

Most existing work on run-time agent systems analysis typically addresses the testing, debugging, validation and verification of these systems [17].

Regarding the analysis of *single* agents, the literature is heavily influenced by the BDI (Beliefs, Desires, Intentions) model. Sudeikat et al [20] propose to check the consistency of beliefs, the achievement of goals and the process of

plan execution in the JADEX agent platform. A similar analysis is enabled by the Agent Factory framework [5] which offers run-time views of the agent's internal state through its agent viewing tool. The INGENIAS platform [9] also provides run-time mental state inspection and testing besides static checking of the specification.

As far as *interaction* analysis is concerned, many agent development frameworks offer graphical tools that can be used for visual inspection, and ways of filtering messages to reduce the amount of data shown (e.g. [11]). As the amount of data increases and manual inspection becomes impractical, automated analysis methods are needed. Here, existing approaches focus on verification of observed interactions against a predefined model, and on manual debugging based on identification of incorrect protocol executions. Padgham et al [13], for example, propose methods for translating protocols specified in AUML to a Petri-net based formalism, and to report errors when protocol specifications are not followed correctly during execution. In a similar vein, Chopra et al [4] have recently formalised the semantic relationship between agents and protocols in order to verify whether a protocol supports the achievement of particular agent goals and whether the specification of the participating agents supports the satisfaction of particular commitments. These contributions provide verification mechanisms that can be used by the designer at runtime. With a stronger focus on design-time analysis, Wooldridge et al [22] propose a language for the design and automatic verification of multi-agent systems called MABLE. This imperative programming language uses the SPIN model checker to automatically verify properties of multiagent systems.

Despite their important contributions, the main limitation of these approaches is that they can only verify the correctness of protocol executions based on observed interactions, but they cannot derive any *additional* knowledge about the emergent behaviour of the system apart from whether agents are behaving correctly or not.

There is also work that focuses on measuring the performance of a multiagent system in terms of the frequency of message types or message patterns [15]. These measurements are used to identify problems in the system, or for deriving probabilistic models of communication behaviour in terms of so-called *expectation networks* [12]. These networks provide compact tree-like stochastic models of the ways in which protocols are used in an agent society. While they make use of logical constraints as conditions along the branches of these trees and generalise over ground instances by using variables in message structures, the analysis still mainly concentrates on regularities in the "surface structure" of execution traces. That is, while the inferred models attempt to generalise over different runs to capture summary information in a concise way, they do not analyse the *semantic* properties of the observed interactions. A similar ap-

proach that makes use of data mining techniques is that suggested by Serrano et al [16], who have used it in the context of testing and debugging agent-based systems. This approach uses data mining techniques to identify groups of agents which behave in a similar manner, and to identify the most important agents within an agent society, but it also focuses on a purely quantitative analysis of the system.

Some recent work in the area of ontology mapping [1,2] is closer in spirit to our work. There, hypotheses about the possible meanings of unknown terms used by the other agent are filtered by using structural information about the protocols. This is achieved by either looking at the ontological relationships between candidate concepts based on the terms that appear earlier on in the same dialogue or in previous dialogues, or by reasoning about the overall syntactic structure of the protocol. However, this kind of reasoning is only used to resolve ontological conflict, and not to infer more general emergent properties of interactions like the ones we are interested in.

All these contributions disregard semantic elements of interactions, such as the assessment of constraints used by the agents, which cause a concrete protocol execution to take a specific course. Also, they fail to induce compact theories about the ways in which interaction is unfolding in a system. As Symeonidis et al [21] argue, transferring such knowledge extracted using data mining into newly created agents could be useful for agent design.

3 Qualitative approach to multiagent systems analysis

3.1 Protocol models

To describe our overall approach, it is necessary to formalise what kinds of agent interaction protocols it assumes. While the implementation of the approach uses a specific protocol specification language (see section 4), it is sufficient for the general framework to represent protocols in a very general way. Our approach considers protocols as graphs whose

- nodes are speech-act like message placeholders,
- edges define transitions among messages, and
- edges are labelled with logical constraints and conditions.

A path in the protocol (or graph) defines a sequence of messages (nodes) where the constraints (logical conditions along path edges) must be satisfied to make the path admissible.

More formally, we define a *protocol model* as a graph $G = (V, E)$ where each node $v \in V$ is labelled with a message $m(v) = q(X, Y, Z)$ with performative q (a string) and sender/receiver/content variables X, Y, and Z. Each edge is labelled with a (conjunctive) list of (say, n) constraints

$$c(e) = \{c_1(t_1, \ldots, t_{k_1}), \ldots, c_n(t_1, \ldots t_{k_n})\} \tag{1}$$

where each constraint $c_i(\ldots)$ has arity k_i, head c_i and arguments t_j which may contain constants, functions or variables (in general the label of an edge could be an arbitrary formula $\phi \in \mathcal{L}$ of a first-order logical language $\mathcal{L}$). All variables that occur in such constraints are implicitly universally quantified. We also assume that all outgoing edges of a node result in messages with distinct performatives, i.e. for all $(v, v') \in E$, $(v, v'') \in E$

$$(m(v') = q(\ldots) \wedge m(v'') = q(\ldots)) \Rightarrow v' = v'' \tag{2}$$

so that each observed message sequence corresponds to (at most) one path in G by virtue of its performatives. The requirement for performatives to be different is needed in order to be able to distinguish different paths in the protocol model. If only propositional content was different, this would be modelled as an attribute of the message, and could be generalised over different cases.

Figure 1 shows an example protocol model in this generic format for illustration purposes. In this negotiation protocol model agent A specified the terms T (e.g. product, quality) of a desired product, and requests T from agent B. The initial response from B depends on availability: if the terms T cannot be satisfied, A and B go through an iterative process of negotiating new terms for the item, which is determined by the definitions of the *keepNegotiating*, *altAcceptable*, and *alternative* predicates the agents use. In case of acceptance, the negotiation process is repeated to agree on a price P for the product. Edge constraints are annotated with the variable representing the agent that has to validate them (subscript A or B). Different out-edges represent XOR if constraints are mutually exclusive, and OR else. For brevity, we will use some additional (redundant) shorthand notation c_i/m_j for constraints and messages in this example below.

3.2 Context of a protocol model

Next, we need to define a notion of context in a conversation based on the semantics of the protocol models. For this, the *semantics* of a protocol model G can be defined as follows: Consider a finite path

$$\pi = v_1 \stackrel{e_1}{\rightarrow} v_2 \stackrel{e_2}{\rightarrow} \ldots \stackrel{e_{n-1}}{\rightarrow} v_n \tag{3}$$

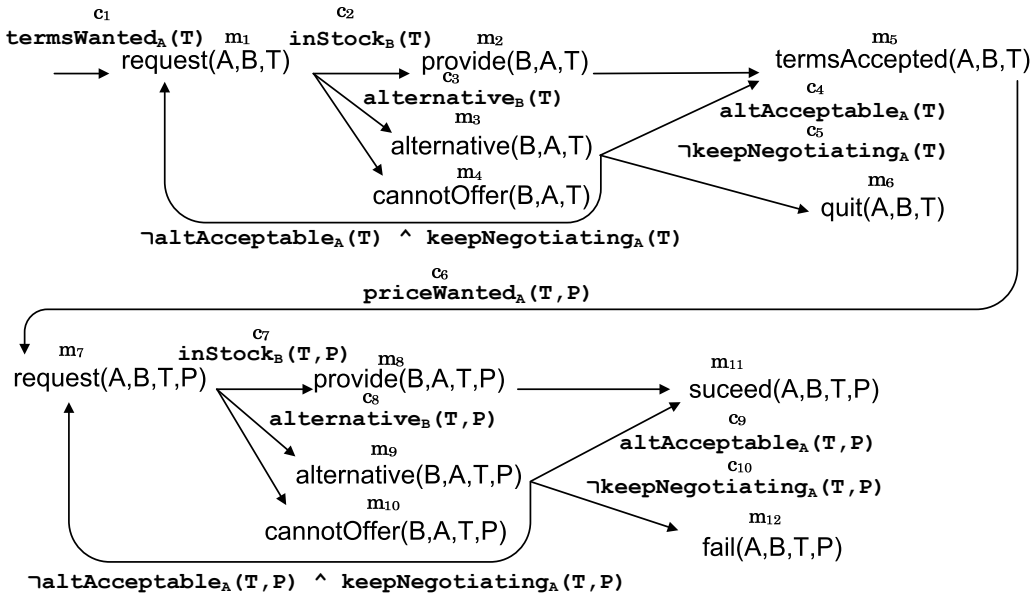


Fig. 1. A simple negotiation protocol model

in the graph G (which may include unfolding of cycles, assuming that all variable names are replaced by fresh variables whenever a node is encountered another time). If $\mathbf{m} = \langle m_1, \ldots, m_n \rangle$ are the (ground, i.e. variable-free) messages perceived in a run, define $G(\mathbf{m}) = \langle \pi, \theta \rangle$ as a function that returns: (1) the (unique) path π that can be traced in G following the observed messages; and (2) θ, the most general unifier of the set

$$\{m_1, \ldots, m_n\} \cup \{m(v_i) | 1 \leq i \leq n\}. \tag{4}$$

In other words, the pair $\langle \pi, \theta \rangle$ returns the path and variable substitution that the message sequence $\mathbf{m}$ corresponds to in protocol model G.

With this, we can define the *context* of $\mathbf{m}$ as

$$C(G(\mathbf{m})) = \bigwedge_{i=1}^{n-1} c(e_i)\theta \tag{5}$$

where $\mathbf{m} = \langle m_1, \ldots, m_n \rangle$, $G(\mathbf{m}) = \langle \pi, \theta \rangle$, and c are constraints associated with each edge as defined in equation 1.

In other words, the resulting conjunction of edge constraints happened in an execution of a protocol model is true if: (1) we determine the unifier θ that makes the perceived messages m_i and the protocol model labels $m(v_i)$ match; and (2) we apply θ as a substitution to the variables of the constraints along the edges of the path π that match the observed message sequence. Any variables that remain free after application of θ are implicitly existentially quantified.

As an example, consider an execution of the protocol described in figure 1 that has ended with the message *cannotOffer* (or m_4) without iterating in the loop of the protocol model. In this case, the observed path is $\pi = \stackrel{c_1}{\rightarrow} m_1 \stackrel{\neg c_2 \wedge \neg c_3}{\rightarrow} m_4$. Therefore, the context of the execution are the observed ground terms (for A, B, and T) which make the constraints we encounter along this path true (c_1, $\neg c_2$ and $\neg c_3$).

Note that the ground terms may contain variable symbols when these are allowed in the content language, e.g. an agent may ask another to return the value of 3+5 with the message "$calc\_request(a_1, a_2, 3 + 5 = ?X)$", but we assume that in this case terms like $?X$, supposed to denote a non-ground value, are, in fact, ground objects of the content language and should not be confused with (meta-)variables of the protocol description language (i.e. they will not be affected by the substitutions we consider here).

3.3 Context models and mining protocol executions

The basic method for applying data mining methods to interaction protocols as specified above is as follows: Consider a protocol model G, and message sequences $\mathbf{m}$ obtained from past executions of G. Any such sequence can be translated to a pair $G(\mathbf{m}) = \langle \pi, \theta \rangle$ (we assume that only sequences allowed by G occur; if not, G can be modified on the fly to accommodate unexpected messages by adding extra constraint-free edges and message nodes). Assuming that a set of such substitution-annotated paths is used as a training data set D, an inductive learning algorithm $L : \mathcal{D} \rightarrow \mathcal{H}$ can be used to map any concrete data set $D \subseteq \mathcal{D}$, where $\mathcal{D}$ is the set of all possible observations, to a learning hypothesis $h \in \mathcal{H}$ taken from the hypothesis space of the machine learning algorithm in question [10].

The extension proposed in this paper to this very general view is to *augment* the learning data by the logical context of the data samples, i.e. to include the logical formula $C(G(\mathbf{m}))$ in the data samples, which can be directly inferred using the logical constraints provided by the definition of G. An "outcome" variable must also be added to the training data to use classification algorithms. For example, $outcome \in \{S, F, N\}$ to denote *S*uccessful completion of the protocol model, *F*ailure to complete or a *N*eutral outcome. In this way, the context model *CM*, which is obtained by using data mining techniques over this augmented learning data, is capable of linking a specific context observed in the execution of a protocol model with the result of the protocol. In other words, the data mining task involved in obtaining a context model is a classification problem where the class attributes are (1) labels reflecting the outcome of a protocol, or (2) constraints occurring within the protocol. The context model obtained in both cases must relate the messages sent in

the protocol, their content, and the context that made them possible to the output observed. As explained in the introduction, the goal of obtaining such a model is to perform a number of qualitative analysis tasks such as making predictions about future behaviour, inferring the constraint definitions other agents use, or assessing the reliability and trustworthiness of agents based on the logical coherence of their utterances.

Determining the most suitable learning algorithm for a particular context mining task is beyond the scope of this paper, and our method does not depend on the use of a specific algorithm. Also, we do not aim to improve the underlying machine learning algorithms employed to obtain context models. For example, figure 2 shows the context model for the constraint $altAcceptable_A$ of the protocol described in figure 1 when used in a car trading system (section 5 describes this experiment). In this case, a decision tree algorithm has been employed to learn a specific constraint in the protocol.

```
 1 persons = 2: F (158)
 2 persons = 4: F (158)
 3 persons = more
 4 |   lug_boot = small
 5 |   |   doors = 2: F (8)
 6 |   |   doors = 3: F (7)
 7 |   |   doors = 4: F (8)
 8 |   |   doors = 5-more: T (105)
 9 |   lug_boot = med
10 |   |   doors = 2: F (13)
11 |   |   doors = 3: F (8)
12 |   |   doors = 4: F (13)
13 |   |   doors = 5-more: T (120)
14 |   lug_boot = big: T (402)
```

Fig. 2. Context model of the $altAcceptable_A$ constraint, obtained using the J48 decision tree algorithm after 1000 negotiations using the protocol described in figure 1. The notation `a =v : T/F` denotes that "if `a` has value `v` the target predicate has value `T/F`". Every leaf includes the number of instances classified under a certain path in parentheses.

In general, the result of the constraints (but not the arguments of these constraints) should be removed when the learning algorithm tries to predict the "outcome" value. Without this modification, a learning algorithm applied over the context of the protocol described in figure 1 would quickly learn that $altAcceptable_A(T, P) \rightarrow outcome = S$, or that

$$\neg altAcceptable_A(T, P) \wedge \neg keepNegotiating_A(T, P) \rightarrow outcome = F.$$

This information is explicit in the definition of the protocol. Moreover, such trivial inferences may hinder the learning technique from relating the details

of constraint argument values to the general outcome of the protocol. On the other hand, samples that include these truth values can be used to predict the satisfaction of these constraints for given argument values. This could be useful when the objective is not to learn the overall outcome of the protocol, but definitions of constraints themselves (as figure 2 shows). Section 5 includes experiments that illustrate the use of both strategies.

3.4 Capturing data to build context models

Due to the nature of multiagent interaction, additional design decisions have to be made before standard data mining machinery can be used, which are related to the details of how exactly training data is constructed from raw protocol execution traces. This is discussed in the following sections.

3.4.1 Training data with multiple agents

The first issue that arises in defining the datasets to be used for protocol mining is how to deal with the presence of multiple agents. In principle, treating all messages and context models that occur in observed interactions as features of learning samples with corresponding variable instantiations as feature values amounts to an attempt to derive globally valid interaction patterns. This learning strategy implies that a shared theory regarding logical constraints and a shared ontological understanding of all terms used in communication exists among agents. In many cases, however, one may want to infer definitions of constraints or behavioural patterns that are specific to an agent, or a group of agents.

To be able to make these distinctions, we need a method for filtering data obtained from protocol executions according to individual agents or group of agents. Assume an assignment $\sigma : \mathit{Var} \rightarrow \mathit{Ag}$ where Var is the set of all variables occurring as sender/receiver variables in nodes of the graph, and Ag the set of agent names. Then for any agent $a \in \mathit{Ag}$, $V_\sigma(a)$ are the nodes that correspond to messages sent by agent a under role assignment σ, and $E_\sigma(a)$ are the *incoming* edges to those messages (formally, $E_\sigma(a) = \{(v, v') \in E | v' \in V_\sigma(a)\}$). We generalise these notions to $V_\sigma(A)/E_\sigma(A)$ for $A \subseteq \mathit{Ag}$ by taking the union over the respective sets for a set of agents A.

The "maximally cautious" form of data filtering in this setting is to reduce the path π of every sample to those nodes and edges that pertain to the learning agent a_i, including only contextual information $C(G_{a_i}(\mathbf{m}))$ from the restricted graph $G_{a_i} = \langle V_\sigma(a_i), E_\sigma(a_i) \rangle$) that a_i can safely verify herself along π (so that all logical constraints verified by other agents are dropped). Note, however, that the path π and substitution θ used in the learning sample are still

based on the full graph, as the observed messages were objectively perceived, i.e. $G(\mathbf{m}) = \langle \pi, \theta \rangle$. This procedure is safe against both deliberate misuse of constraints and messages in the protocol specification by the other agent(s) *and* (non-malicious) divergence between the meaning constraints have for the different parties involved. But it also discards a lot of information that could be useful in learning a predictive logical theory about the circumstances under which different interaction outcomes are achieved.

At the other end of the spectrum, if a_i fully trusts the other agent(s) and can safely assume that all agents' ontologies and logical theories are fully aligned, it can use the entire path information as part of each learning sample, assuming that the definitions of constraints are common to all agents *and* that every agent verifies the constraints reliably and honestly.

These filtering approaches can be generalised to make arbitrary distinctions between subsets of agents whose messages and edge constraints are pooled together when generating feature-value pairs in learning samples.

3.4.2 Training data with paths, loops, and variables

Typical machine learning and data mining algorithms assume a fixed number of attributes (features) and values. In the case of protocol mining, as the attributes will be logical constraints and messages along a path π, and their values will be specified by the ground terms observed as instances of their variables as given by θ, a number of issues arise that require further design decisions to be made.

Firstly, when collecting different paths to be added in the training data set D, their labels (messages/constraints) may differ. The use of "unknown" values for unused variables is valid, although it may give misleading results. In many practical cases, it will be more appropriate to create a different data set D_π for each observed path π avoiding this problem. Finally, at a domain-specific level, one can merge data across different paths into a single set while only observing a fixed set of certain messages and constraints, potentially even ignoring different messages along the path.

Secondly, many common interaction protocols (e.g. negotiation protocols like auction and bargaining protocols) involve iterations of sub-sequences that can be repeated an arbitrary number of times. The existence of a loop in a protocol means that variables occurring in a logical constraint or message used in the loop can have several constants as ground instantiations in the same execution $\{g_1, g_2, ..., g_n\}$, where n is the number of iterations in the loop which may vary for different samples in D. A possible strategy is storing N "copies" of each variable, where N is the maximum number of iterations observed in a protocol. In every path with fewer iterations than N, the remaining variables

will have to be assigned a value of "unknown". The redundancy introduced by this strategy can be avoided considering only the first/last ground term g_1/g_n for a specific variable V in a loop. In many cases, such as most kind of negotiations, this is enough since intermediate steps are less important for the outcome of the interaction. Selecting one of these strategies or a different one depends on the specific protocol model, its purpose and the context. For example, in some bidding systems, the second highest bidder would win. In that case, the lowest bid, the second highest and the highest bid are of interest and should be recorded before building a context model.

3.5 Analysing interactions using context models

As explained in section 3.3, the context models introduced in this paper have the primary purpose of making predictions about future behaviour. These predictions allow agents or developers to design strategies to enhance interactions in multi-agent systems. Although these predictions and strategies are dependent on the specific interaction protocol, this section offers an example of strategy for the protocol displayed in figure 1. The strategy is designed from the point of view of the customer (A in the protocol) and aims at choosing a good seller for the requested product.

According to this strategy, A performs the following processing steps: (1) compute a context model CM relating the interaction outcome to terms T, price P and seller B; (2) produce tuples with the desired values for terms T, price P and the expected outcome; each of these tuples will differ in the ground value B, one for each known seller; (3) use CM to classify the data obtained in (2). If the predicted outcome for a tuple matches the output desired by the customer, the ground term for B in this tuple is selected as an agent to interact with; (4) after interacting, if the prediction of CM does not match the observed interaction, re-build CM from scratch to include more recent information in the training data; (5) if a seller was not chosen, update T and P according to the customer's preferences, and repeat from (1) for a pre-defined number of attempts before choosing a seller randomly.

The strategy explained above can be fully automated using our approach enables a customer to identify sellers who have not only engaged in successful interactions with the customer, but where these successful interactions were observed under specific, desirable negotiation terms. We will return to this example strategy in section 5 where we evaluate its utility more quantitatively.

```
1  a(participant, B) ::
2  request(X) <= a(initiator, A) then
3  (agree(X) => a(initiator, A) <- consider(X) then
4   (informDone(Y) => a(initiator, A) <- performed(X,Y))
5    or
6    (failure() => a(initiator, A))
7  )
8  or
9  (refuse()  => a(initiator, A))
```

Fig. 3. LLC implementation of the "participant" role in the FIPA Request Interaction Protocol

4 Implementation

ProtocolMiner is a prototypical tool which we have developed to implement qualitative context mining. ProtocolMiner is implemented as a plugin for the OpenKnowledge platform [19], a protocol specification and execution platform designed for large-scale heterogeneous multiagent systems. ProtocolMiner automates the registration and recovery of the training data for any protocol implemented and executed in OpenKnowledge. While the tool itself is designed for use by a human designer, an application programming interface (API) is also provided to allow agents to exploit knowledge extracted from past interactions.

We first briefly introduce the definition of semantically annotated protocols that ProtocolMiner is designed to operate on. This is followed by details about how the tool implements the approach described in section 3. Finally, the stages in the use of the tool that a user has to go through to perform qualitative protocol execution analysis are detailed.

4.1 Defining protocols and capturing data in ProtocolMiner

ProtocolMiner (and OpenKnowledge) uses a protocol definition language called the *Lightweight Coordination Calculus* or LCC [14], a language that uses declarative Prolog-like constraints in protocol specifications.

As an example, an implementation of the "participant" role in the FIPA Request Interaction Protocol [1] in LCC would have the format shown in figure 3. This specification defines a role `a(participant,B)` (binding the concrete agent identifier to variable `B` at runtime when the agent adopts the role) in

[1] See `www.fipa.org/specs/fipa00026/`

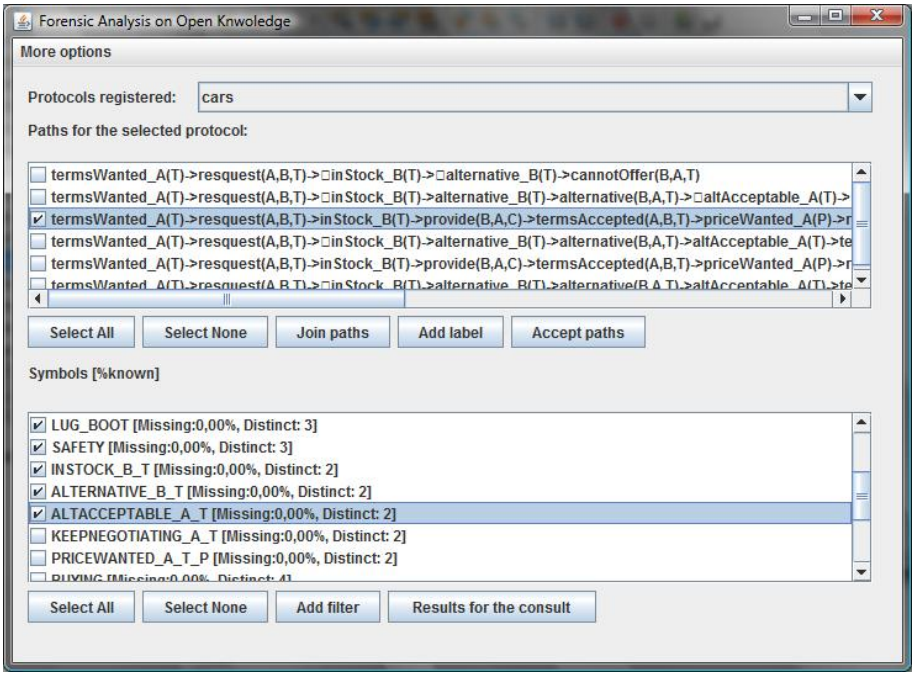


Fig. 4. ProtocolMiner user interface

terms of the message sequences allowed for that role. Incoming/outgoing messages are denoted by double arrows `<=` and `=>` from/to another role identifier, and guards (preconditions) on messages appear on the right hand side of a message exchange, prefixed by a single arrow `<-` (the language also allows for postconditions prefixed by `->` not used in the above example). Sequential concatenation, disjunction, and iteration are captured by keywords `then`, `or` and Prolog-like recursive calls of role clauses, respectively. In this specific example, the constraint $consider(X)$ is used to determine whether to `agree` or `refuse` the request for X.

Although ProtocolMiner has been implemented for use with LCC-based protocol specifications, other interaction platforms can be used as data sources as long as their specification mechanism can be translated to the protocol model graph structures defined in section 3.1.

4.2 Qualitative analysis in ProtocolMiner

This section shows how the tool presented implements the formal approach described in section 3. The ProtocolMiner user interface is shown in figure 4 after logging executions of the case study presented for evaluation purposes in the following section.

As explained above, ProtocolMiner allows a developer to define a protocol which meets the requirements specified in section 3.1 by using LCC language.

ProtocolMiner automatically logs the results of constraint validation by participating agents based on the values assigned to variables such as X and Y in the example shown in figure 3.

The strategies to prepare training data described in section 3.4 are also implemented in the tool. The ProtocolMiner GUI, shown in figure 4, allows a developer or agent to select the agents considered in the dataset through SQL expressions in order to undertake the different strategies detailed in section 3.4.1. For example, if the developer is only interested in agents a_1 and a_2, the *"add filter"* option can be used to introduce `A='a1' and B='a2'`. Additionally, the list of selected attributes in the dataset allows for arbitrary selections of subsets of constraints (or edges) to be considered by the data mining algorithm. The tool also enables a developer or agent to select the paths considered in the data set as described in the first part of section 3.4.2 by using *"accept paths"*. A label can be added to the accepted paths by means of the option *"add label"*. Moreover, several paths can be selected and merged to a single path using the *"join paths"* option. Furthermore, the tool can be configured to use one of several strategies to deal with loops discussed in section 3.4.2: (1) log all the values given to an attribute along a path (every value is stored in a new attribute), (2) log only the first and last values along a path, or (3) log only the last value given to an attribute at the end of a protocol execution. As explained above, selecting the correct strategy depends on the specific protocol model defined and its purpose (e.g. some bidding systems may require extracting not only the highest bidder but also the second highest).

Finally, ProtocolMiner also integrates the Weka [3] algorithms allowing the tool to use a large number of data mining techniques to obtain the context models described in section 3.3.

4.3 Phases in the use of ProtocolMiner

This section shows an overview of the interaction analysis workflow using ProtocolMiner, figure 5 depicts the main phases.

- Assuming agents that run on an OpenKnowledge platform, these share the OK kernel that provides the core infrastructure for distributed interaction, and which has been modified to provide the necessary interfaces to ProtocolMiner, which is integrated with the kernel. Phase 1 in the figure shows how the designer implements OK components [19] to support agent implementations provided by the same designer.
- Phase 2 shows the interactions that agents A_i, designed by the developer, contribute to in an open and distributed multi-agent system which also contains agents B_j provided by other parties.

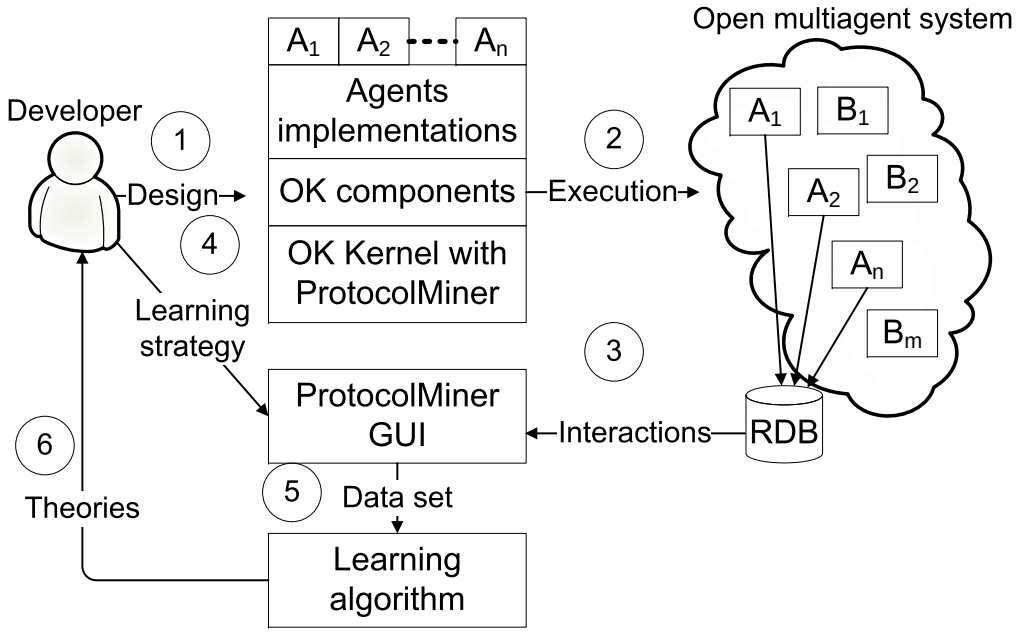


Fig. 5. ProtocolMiner use. Main stages are labelled 1-6.

- In phase 3, interaction data is sent to a relational database by the ProtocolMiner code included in the OK kernel. This data basically includes information about paths taken in the protocol, messages sent and received with their argument values, and the truth values of constraints contained in the protocols. The values assigned to arguments of messages sent by agents B_j are trivially obtained from the reception of these messages by known agents. On the other hand, results and arguments of constraints evaluated by unknown agents are not directly available in the general case. Nonetheless, some of these values can be recovered by observing the evolution of the interaction in many cases allowing ProtocolMiner to register them. For example, in figure 1, if the *cannotOffer* message arrives to an agent with the ProtocolMiner tool, the tool infers that $alternative_B$ is false based on the protocol specification in LCC (see section 4).
- In phase 4, the designer specifies the options to build the training data, see section 3.4, determining which filters should be applied to the raw data.
- ProtocolMiner generates a dataset based on this filtering strategy (phase 5) and invokes a learning algorithm which returns a hypothesis regarding the target query based on the dataset (phase 6).
- These hypotheses, which ideally represent useful theories about hidden properties of the system, are used by the developer to modify agent design and the whole analysis cycle is repeated if needed when new interactions produce new data.

Note that the developer can include a pre-designed strategy for the first phase when designing agents that use the ProtocolMiner infrastructure, so that the remaining phases, which are fully automated, can be used by artificial agents.

For example, section 3.5 presents a procedure which allows agents to use context models to improve their interactions themselves, assuming that these agents are able to select a strategy to capture the data and a learning algorithm to build the context model.

5 Case study

To illustrate the usefulness of our approach, we have analysed data generated in a car selling domain, where agents negotiate over cars using the protocol shown in figure 1. We first describes the scenario and the decisions taken to generate context models following the formal approach presented in section 3. We then report on experiments that analyse the predictively ability of the context models derived for a specific constraint in an unknown agent. A second set of experiments is performed to show the capacity of context models to predict the overall outcome of a protocol model. Finally, experiments are conducted that show how interactions in this case are improved when agents' interaction decisions are informed by context models.

5.1 Description of the case study

In our example scenario, we use a well-known database for car evaluation [8] [2]. This database includes the technical characteristics and prices which are used in the system. More specifically, a potential customer (role A) is requesting offers from a car selling agent (role B) where T specifies the technical characteristics including number of doors, capacity in terms of persons to carry, the size of the luggage boot and the estimated safety of the car. For the interactions analysed in the case study, we assume that the values for car characteristics are given as a tuple $T = (doors, persons, lug\_boot, safety)$ where

- $doors \in \{2, 3, 4, 5\text{-}more\}$
- $persons \in \{2, 4, more\}$
- $lug\_boot \in \{small, med, big\}$
- $safety \in \{low, med, high\}$

After negotiating the car's technical features, the agents use the protocol to negotiate the price and maintenance terms (below we refer to these as "price terms"). Specifically, the potential customer (role A) requests price terms P from a car selling agent (role B) for the negotiated features T. Price terms are given as a tuple $P = (buying, maint)$ where

[2] Car Evaluation Data Set `http://archive.ics.uci.edu/ml/datasets/Car+Evaluation`.

- $buying \in \{v\text{-}high, high, med, low\}$
- $maint \in \{v\text{-}high, high, med, low\}$

We specify five customer agent preferences regarding T and P, which we call mental states (MS). These preferences are used to define how the constraints are evaluated by customers in the protocol.

- $MS_1(T) = (persons = more \wedge lug\_boot = big) \vee (doors = 5\text{-}more \wedge persons = more)$
- $MS_1(P) = (buying = low \wedge maint = low) \vee (buying = med \wedge maint = med)$
- $MS_2(T) = (doors = 5\text{-}more \wedge safety = high) \vee (doors = 4 \wedge safety = high)$
- $MS_2(P) = (maint = med) \vee (buying = med)$
- $MS_3(T) = (persons = more \wedge doors = 5\text{-}more) \vee (lug\_boot = big \wedge safety = high)$
- $MS_3(P) = (buying = low)$
- $MS_4(T) = (true)$
- $MS_4(P) = (buying = low \wedge maint = low)$
- $MS_5(T) = (door = 5\text{-}more \wedge persons = more) \vee (persons = more \wedge lug\_boot = big) \vee (safety = high)$
- $MS_5(P) = (buying = low \wedge maint = low) \vee (buying = low \wedge maint = med) \vee (buying = med \wedge maint = low)$

We define ten customer agents C_i, where $1 \leq i \leq 10$, with associated mental states $C_i := MS_{i \bmod 5}$. That is, agents C_1 and C_6 have mental state MS_1, C_2 and C_7 have mental state MS_2, and so on.

For the purposes of this case study, we assume that a single seller (S) is analysing the system evolution from its local point of view, aiming to predict the different outcomes of its interactions based on perceived regularities regarding the observed behaviour of the customers.

In the following sections, we outline how qualitative context mining is performed in this particular setting.

5.2 Decisions for the generation of context models

In converting raw sequences of message exchanges to training data samples (see section 3.4), we make the following design choices, which effectively imply the most simple and general data generation method implemented in ProtocolMiner.

Firstly, we consider the agent $B = S$ (S is the seller agent name) who performs the analysis to obtain knowledge about the others agents' (opaque) mental states. Therefore, the learning input is restricted to $V_c(A)$ and $E_c(A)$, where

c is the customer role in the protocol and A is any agent participating in this role (see section 3.4.1).

As far as variables occurring in constraints are concerned, we uniformly record all attributes contained in "terms" descriptions T and P, including a "?" (unknown) value for those not mentioned in a given execution trace. This is feasible in the given protocol model as the amount of unspecified data is manageable. The strategy to deal with loops is to only record the last value of every variable occurring in multiple iterations over the `alternative-request` subsequence for the technical or the price terms negotiation, as we are primarily interested in the final offer accepted or rejected by the customer (see section 3.4.2).

The given protocol allows for the following five different *final* messages: (1) m_4 = `cannotOffer`, (2) m_6 = `quit`, (3) m_{10} = `cannotOffer`, (4) m_{12} = `fail` and (5) m_{11} = `succeed`. All iterations of the `alternative-request` loop are merged into a single path model. Therefore, ProtocolMiner obtains the following five path models (constraint and message abbreviations as in figure 1):

- $\pi_1 = \overset{c_1}{\rightarrow} m_1 (\overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{\neg c_4 \wedge c_5}{\rightarrow} m_1)^* \overset{\neg c_2 \wedge \neg c_3}{\rightarrow} m_4$
- $\pi_2 = \overset{c_1}{\rightarrow} m_1 (\overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{\neg c_4 \wedge c_5}{\rightarrow} m_1)^* \overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{\neg c_4 \wedge \neg c_5}{\rightarrow} m_6$
- $\pi_3 = \overset{c_1}{\rightarrow} m_1 (((\overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{\neg c_4 \wedge c_5}{\rightarrow} m_1)^* \overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{c_4}{\rightarrow}) | (\overset{c_2}{\rightarrow} m_2 \rightarrow)) m_5 \overset{c_6}{\rightarrow} m_7 (\overset{\neg c_7 \wedge c_8}{\rightarrow} m_9 \overset{\neg c_9 \wedge c_{10}}{\rightarrow} m_7)^* \overset{\neg c_7 \wedge \neg c_8}{\rightarrow} m_{10}$
- $\pi_4 = \overset{c_1}{\rightarrow} m_1 (((\overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{\neg c_4 \wedge c_5}{\rightarrow} m_1)^* \overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{c_4}{\rightarrow}) | (\overset{c_2}{\rightarrow} m_2 \rightarrow)) m_5 \overset{c_6}{\rightarrow} m_7 (\overset{\neg c_7 \wedge c_8}{\rightarrow} m_9 \overset{\neg c_9 \wedge c_{10}}{\rightarrow} m_7)^* \overset{\neg c_7 \wedge c_8}{\rightarrow} m_9 \overset{\neg c_9 \wedge \neg c_{10}}{\rightarrow} m_{12}$
- $\pi_5 = \overset{c_1}{\rightarrow} m_1 (((\overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{\neg c_4 \wedge c_5}{\rightarrow} m_1)^* \overset{\neg c_2 \wedge c_3}{\rightarrow} m_3 \overset{c_4}{\rightarrow}) | (\overset{c_2}{\rightarrow} m_2 \rightarrow)) m_5 \overset{c_6}{\rightarrow} m_7 (((\overset{\neg c_7 \wedge c_8}{\rightarrow} m_9 \overset{\neg c_9 \wedge c_{10}}{\rightarrow} m_7)^* \overset{\neg c_7 \wedge c_8}{\rightarrow} m_9 \overset{c_9}{\rightarrow}) | (\overset{c_7}{\rightarrow} m_8 \rightarrow)) m_{11}$

Based on this information, we introduce an "outcome" variable $outcome \in \{S, F, N\}$ (see section 3.3) to denote *S*uccessful completion of the negotiation (path π_5), *F*ailure to complete (paths π_1 and π_2) or a *N*eutral outcome (the remaining paths). This classification is biased by the seller's point of view, as she is specifically interested in failures of the customer agent, not in own choices that led to premature termination of the negotiation procedure.

These choices result in a data set with the following set of 15 attributes: values A, B, $doors(T)$, $persons(T)$, $lug\_boot(T)$, $safety(T)$, c_1, c_4, c_5, $buying(P)$, $maint(P)$, c_6, c_9, c_{10} (some constraints are not present due to a restriction to the seller's context model), and $outcome$. Here, we adopt a simplified notation $attr(T) = v$ to denote $attr = v$ in terms T. Omitting the attributes with unknown values, an example of a sample thus obtained from a tuple corresponding to π_1 in a transaction between C_1 and the seller agent could be

represented by the substitution

$$\theta = \{A/C_1, B/S, T/T_2, Outcome/F\}$$

and the context (see section 3.2) would be

$$termsWanted_{C_1}(T_1) \wedge \neg altAcceptable_{C_1}(T_2) \wedge \neg keepNegotiating_{C_1}(T_2)$$

where T_1 and T_2 are ground terms for the technical features of the car.

While determining the most suitable learning algorithms for a particular context mining task is beyond the scope of this paper, we experiment with three open source implementations of data mining techniques included in ProtocolMiner via the Weka tool [3]. These experiments use the default parameters in these techniques to illustrate the impact of using different algorithms in an exemplary way, and also to show that our approach does not depend on the use of a specific learning algorithm. More specifically, we use the *J48* decision tree algorithm (an implementation of the C4.5 algorithm), the *NNge* classification rules algorithm (Nearest neighbour like algorithm) and the *BayesNet* algorithm (a Bayesian network classifier) [3].

5.3 Accuracy of a context model for a constraint

In our first experiment, the seller tries to learn a context model for a single constraint, $c_4 = altAcceptable_A(T)$, and a single customer agent, C_1. As explained in section 3.3, the outcome attribute used in this case is satisfaction of the constraint (c_4). All remaining attributes that involve truth values of other constraints are removed from the samples (see section 3.3). Also, attributes which appear after c_4 (values for P) are removed from the samples, just like attributes with a single value across all samples, in this case $A = C_1$ and $B = S$. Notice that all this preprocessing can be automated after selecting which specific constraint to analyse.

The output of the J48 algorithm after 1000 protocol executions is shown in figure 2. The time taken to build the model (on a 2.4 Ghz 4GB RAM machine) is 0.01 seconds and a tree with 15 nodes is obtained as the hypothesised mental state that corresponds to the logical formula

$$\begin{aligned} altAcceptable_{C_1}(T) \Leftrightarrow & (persons(T) = more \wedge lug\_boot(T) = small \wedge doors(T) = 5\text{-}more) \vee \\ & (persons(T) = more \wedge lug\_boot(T) = med \wedge doors(T) = 5\text{-}more) \vee \\ & (persons(T) = more \wedge lug\_boot(T) = big) \end{aligned}$$

It is easy to verify that this is logically equivalent to $MS_1(T)$, i.e. the actual preferences of agent C_1 (see section 5.1).

The model generated with the NNge technique is shown in figure 6. Although the results of NNge are less legible than the ones of the decision tree, we obtain exactly the same results as before, a context model logically equivalent to the mental state.

Finally, a ten-fold cross-validation applied to a model obtained with the Bayes-Net method returns a 94.3% of correctly classified instances, a mean absolute error [3] of 0.07 and a relative absolute error of 16.31%.

These experiments demonstrate that good context models can be obtained for isolated constraints independently of the learning algorithm used (with a probabilistic Bayesian method naturally performing a bit worse than crisp methods in this case where the customer agent's behaviour is deterministic in terms of the constraint in question).

A possible strategy that the seller could follow to employ this context model for her own benefit is: generating all the acceptable options for C_1, ordering the list by a utility function for the seller (e.g. selling price), and offering an alternative based on this ranking. The seller might even choose to strategically lie about the availability of a car with a low value in the utility function if they can be sure that there are higher-value cars in the database that satisfy the inferred context model and which could be successfully offered to the customer. Of course this assumes extensive interaction with a single customer and a simple decision-making procedure on the customer's side, but the example illustrates the potential for using our method in practical agent reasoning.

5.4 Accuracy of a context model for a whole protocol model

In this experiment, a seller tries to learn a model for the overall outcome of the protocol. For this, we consider a ten-customer scenario with the five mental states described in section 5.1 and $outcome \in \{S, F, N\}$. To focus on

[3] The mean absolute error is calculated as follows:

$$\left[\sum_{i=1}^{n}\sum_{j=1}^{nc}\left[|actual(instance_i, class_j) - predicted(instance_i, class_j)|/nc\right]\right]/n$$

where n is the number of instances in the test set, nc the number of classes or outcome labels, *actual* returns 1 if an $instance_i$ has the class number $class_j$ and 0 in other case, and finally, *predicted* returns 1 if an instance $instance_i$ has been classified as the class number $class_j$ by the model and 0, else.

```
1   class F IF : doors in {2,3,4,5-more} ^ persons in {2} ^
        lug_boot in {small,med,big} ^ safety in {low,med,
       high}  (158)
2   class T IF : doors in {5-more} ^ persons in {more} ^
       lug_boot in {small,med,big} ^ safety in {low}  (134)
3   class T IF : doors in {5-more} ^ persons in {more} ^
       lug_boot in {small,med,big} ^ safety in {med,high}
       (252)
4   class F IF : doors in {3,4} ^ persons in {4,more} ^
       lug_boot in {small,med} ^ safety in {med,high}  (65)
5   class F IF : doors in {2,5-more} ^ persons in {4} ^
       lug_boot in {small,med,big} ^ safety in {low,med,
       high}  (68)
6   class T IF : doors in {2} ^ persons in {more} ^
       lug_boot in {big} ^ safety in {med,high}  (43)
7   class F IF : doors in {3,4} ^ persons in {4} ^ lug_boot
        in {small,med,big} ^ safety in {low}  (30)
8   class T IF : doors in {4} ^ persons in {more} ^
       lug_boot in {big} ^ safety in {med,high}  (49)
9   class T IF : doors in {3} ^ persons in {more} ^
       lug_boot in {big} ^ safety in {med,high}  (60)
10  class T IF : doors in {2} ^ persons in {more} ^
       lug_boot in {big} ^ safety in {low}  (23)
11  class F IF : doors in {2,3,4} ^ persons in {more} ^
       lug_boot in {small,med} ^ safety in {low}  (20)
12  class T IF : doors in {4} ^ persons in {more} ^
       lug_boot in {big} ^ safety in {low}  (28)
13  class F IF : doors in {3,4} ^ persons in {4} ^ lug_boot
        in {big} ^ safety in {med,high}  (22)
14  class T IF : doors in {3} ^ persons in {more} ^
       lug_boot in {big} ^ safety in {low}  (38)
15  class F IF : doors in {2} ^ persons in {more} ^
       lug_boot in {small,med} ^ safety in {med,high}  (10)
```

Fig. 6. NNge output for 1000 negotiations with customer C_1 modelling the $altAcceptable_A$ constraint. Every rule includes the number of instances classified in parentheses. `class T/F` denotes the truth value of the constraint.

the outcome, attributes of constraints results are removed from the samples in this experiment (as explained in section 3.3), and the single-value attribute $B = S$ is not included.

Figure 7 shows the average (across 100 repeated experiments) model accuracy, evaluated using cross-validation, for the first 10000 negotiations (top chart), as well as the time needed to perform the experiments (bottom chart).

The experiments demonstrate that accurate models can be built using contex-

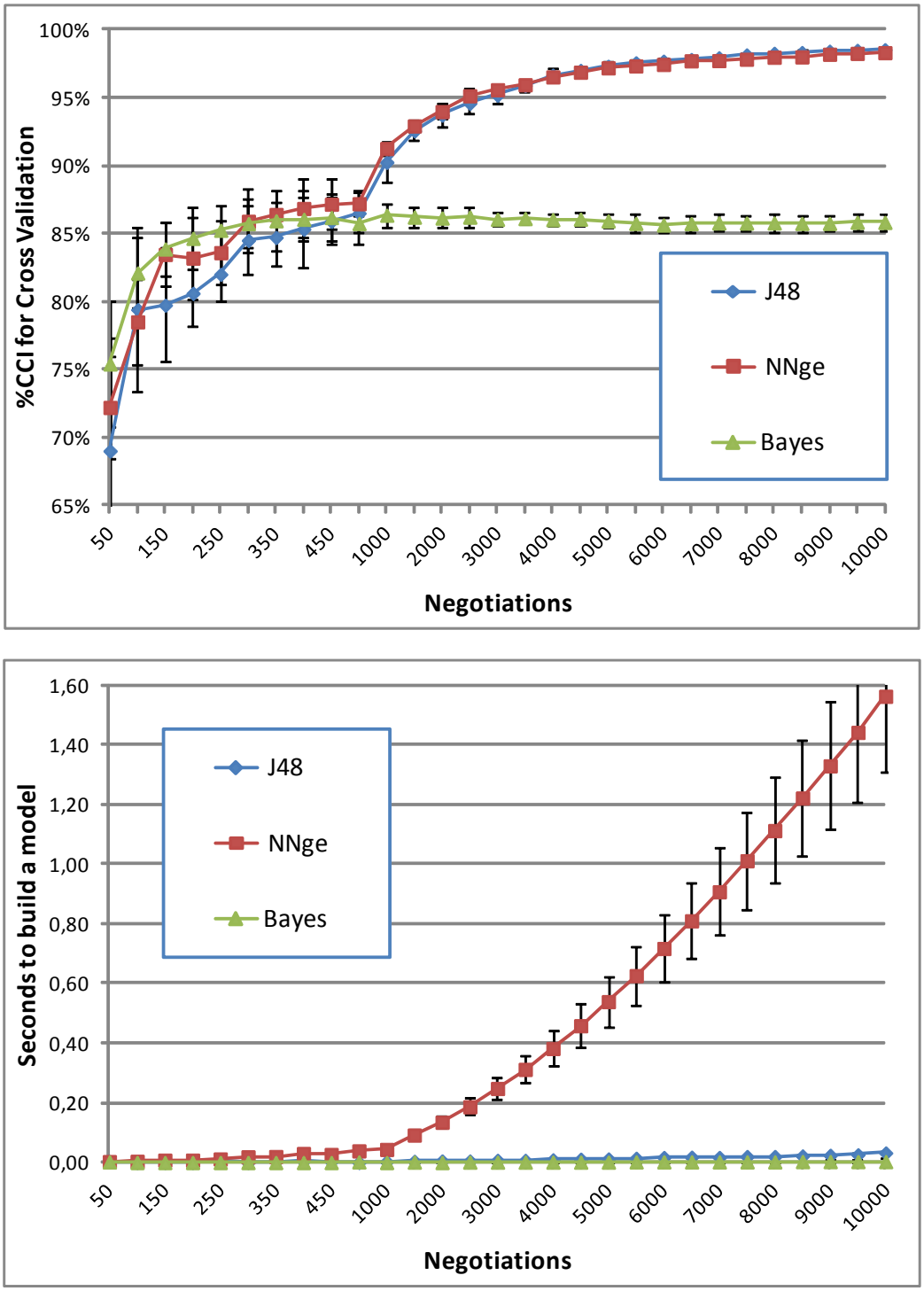


Fig. 7. Top chart: Average model accuracy (based on cross-validation) shown across total number of negotiation (100 experiments). Learning algorithms: J48, NNge and BayesNet. Bottom chart: Time to build each model (classifier), averaged over 100 experiments. Error bars show standard deviation.

tual information extracted from concrete executions of a protocol to predict its final outcome. More specifically, after an average of 200 total negotiations (i.e. 20 per customer), the models classified at least 80% of all instances correctly. Besides, the time needed to obtain these models can be considered reasonable even for agents that might use our method run-time. Even NNge, the slowest of the algorithms, builds a model for 10000 negotiations in less than 2 seconds.

Finally, the use of three different data mining techniques demonstrates that our method is independent of the algorithm, although the Bayes classifier clearly yields the lowest (by around 13%) performance. The decision tree model is the least reliable for small numbers of instances, and simultaneously, performs best for large datasets (it classifies 98.53% of all instances correctly).

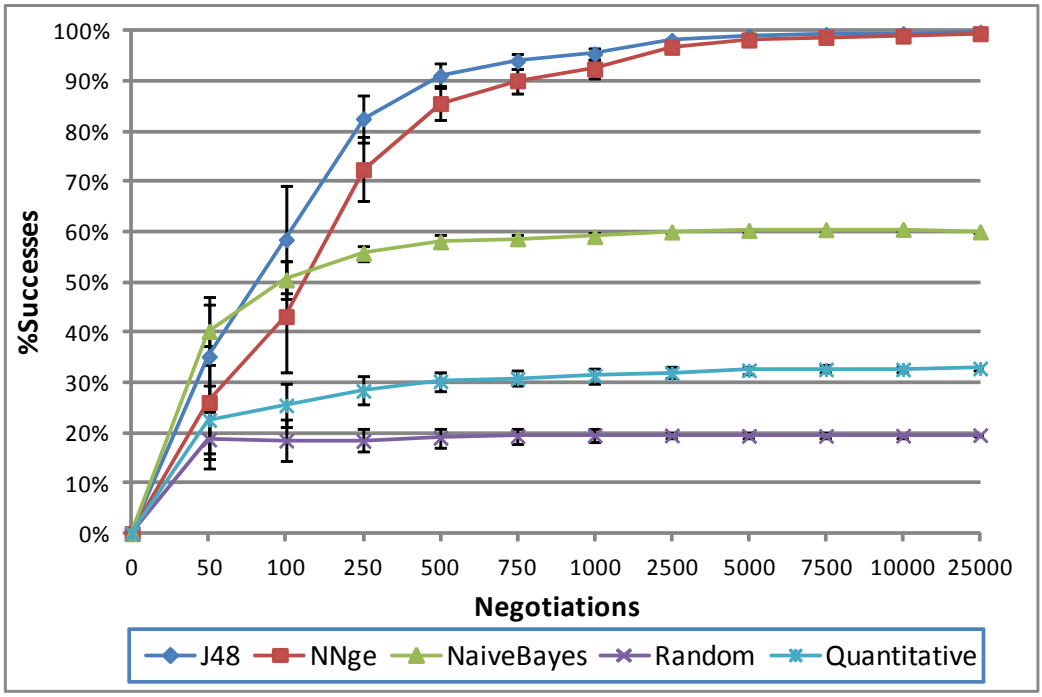
5.5 Enhancement of interactions by context models

In this section, we show how agents can use context models directly to improve their own performance in communicative exchanges with others. For this, we enable customers to use the models to choose a good seller for the concrete product they are looking for. Assume that we have three sellers, S_1, S_2 and S_3, who are able to offer products which satisfy the following mental state models:

- $MS_{S_1}(T, P) = (safety = med \wedge buying = low \wedge maint = low)$
- $MS_{S_2}(T, P) = (safety = high \wedge buying = med \wedge maint = low)$
- $MS_{S_3}(T, P) = (persons = more \wedge doors = 5\text{-}more \wedge lug\_boot = med \wedge buying = med \wedge maint = low)$

The customer agents follow strategy described in section 3.5 to improve their interactions using context models. We compare the prediction accuracy of this strategy against two alternative analysis strategies:

- *Random.* The seller is chosen randomly – this provides a baseline for the minimum performance that could be achieved without any use of context models. An optimal strategy is not included, as 100% success constitutes the upper bound of what can be achieved in this scenario (we ensure that there are always sellers in the system who can provide the requested items).
- *Quantitative.* The seller is chosen using a distance function based on the number of past successes and failures with them in the customer's personal experience. The function used is $D(s, f) = 1 - (1 + \frac{s}{2f+1})^{-1}$, where s is the number of successes and f the number of failures with a particular seller, and the seller to interact with is chosen with probability corresponding to $D(s, f)$ [18].



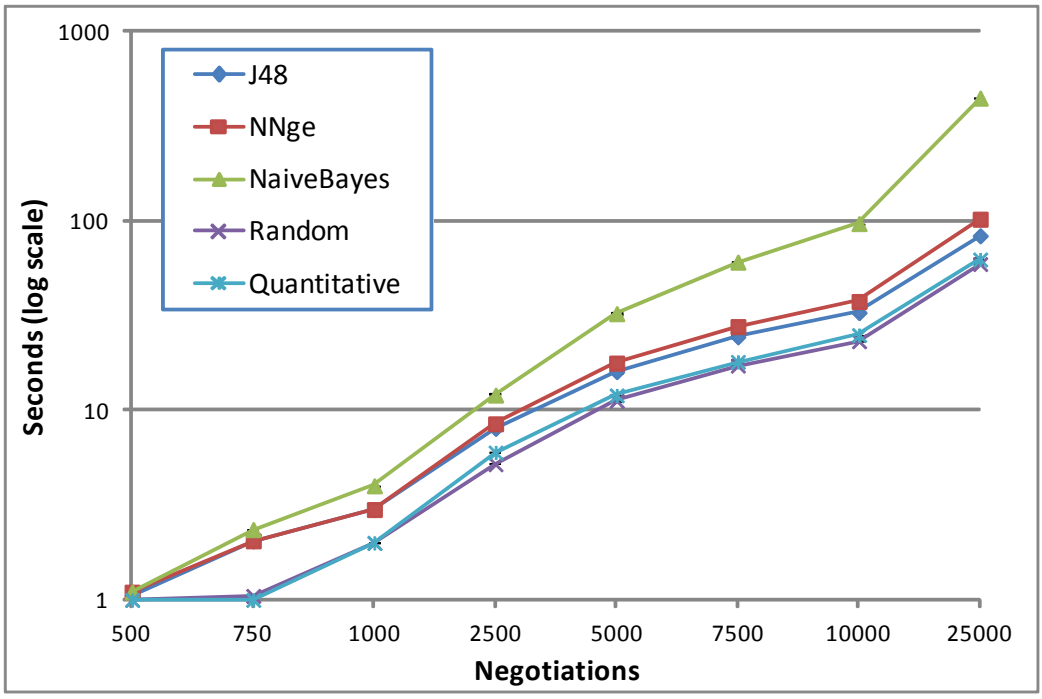


Fig. 8. Top chart: Average number of successful negotiations against number of total negotiations (100 experiments); error bars show standard deviation. Learning algorithms: J48, NNge and BayesNet. Bottom chart: Time taken to perform experiment per negotiation, log scale (averaged over 100 experiments); the standard deviation across experiments is negligible.

As figure 8 shows (top chart), after 100 negotiations (10 negotiations per customer) the use of context models greatly outperforms the random and quantitative strategies, with the use of decision trees converging faster to optimal performance than the other two learning techniques. However, later convergence of the NNge shows that it performs equally well in the long term, provided sufficient data becomes available. Even the worst learning technique in this case, the Bayes classifier, results in almost twice as many successful interactions than the quantitative approach.

The downside of this approach is an increased runtime, at least when, as above, context models are re-built every time a customer obtains a wrong prediction, which happens very often, while also the datasets over which the models are built increase over time. Figure 8 also shows (bottom chart) the time taken on average per negotiation, which reaches around 83 seconds after 25000 negotiations for J48 and 102 seconds for NNge. While this is clearly a drawback of our method, it is highly customisable in that the maximum amount of data processed or the frequency with which models are re-built can be adapted as suits the system designer (albeit at the cost of lower accuracy). The figure also shows that the lowest performance is reached using the Bayes classifier although the experiments in section 5.4 concluded that it was the fastest technique. The explanation for this is that its low accuracy (see figure 7) results in a constant need to re-build (see section 3.5), which implies a much higher overall computational effort than the other two techniques.

6 Conclusions and future work

In this paper, we have presented a novel mechanism to exploit *qualitative* information provided by high-level ACLs and interaction protocols, in which messages are associated with logical constraints, which can be used as "semantic" annotations of communication in a natural way. Our work was motivated by a lack in existing multiagent systems analysis methods which mostly ignore this rich source of contextual information when analysing run-time multiagent interactions. We presented a formal approach that allows us to make interaction data available for qualitative data mining using information about the shared protocol models as background knowledge.

The main advantage of using the additional structure available in agent-based communication protocols is that it allows for the use of data mining methods to infer qualitative information from observed message exchanges at a higher level than this is possible using traditional distributed systems protocol. We discussed different alternatives for dealing with the specific nature of agent interaction protocols when converting interaction experiences to training data,

addressing issues such as the presence of multiple agents, varying-length execution paths, and loops that are commonly present in common multiagent interaction protocols. Subsequently, we presented an implementation of our techniques with the ProtocolMiner tool, and a case study which hinted at the potential of applying data mining in multiagent systems.

In the future, we aim to apply our analysis methods to more real-world examples in order to extract guidelines for making appropriate choices when selecting training data extraction strategies and appropriate data mining algorithms. We would also like to explore the use of more advanced machine learning methods to learn logical theories of, for example, the internal ontological conceptualisations agents use, and to rate their competence and trustworthiness based on the knowledge they appear to have based on their interaction behaviour. We believe these to be promising practical avenues for addressing one of the fundamental problems of open systems, which is to be able to derive knowledge of the internal workings of other agents without being able to observe their internal state.

Acknowledgments

This research work is supported by the Spanish Ministry of Science and Innovation in the scope of the Research Projects TSI-020302-2010-129, TIN2011-28335-C02-02 and through the Fundación Séneca within the Program 04552/-GERM/06. Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain.

References

[1] M. Atencia and W. M. Schorlemmer. I-SSA: Interaction-Situated Semantic Alignment. In *OTM'08*, volume 5331 of *Lecture Notes in Computer Science*, pages 445–455. Springer, 2008.

[2] P. Besana and D. Robertson. Probabilistic Dialogue Models for Dynamic Ontology Mapping. In *URSW'08*, volume 5327 of *Lecture Notes in Computer Science*, pages 41–51. Springer, 2008.

[3] R. R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse. *Weka manual (3.7.1)*, June 2009. http://prdownloads.sourceforge.net/weka/WekaManual-3-7-1.pdf?download.

[4] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Reasoning about agents and protocols via goals and commitments. In *AAMAS'10*, pages 457–

464, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[5] R. W. Collier. Debugging Agents in Agent Factory. In *PROMAS'06*, pages 229–248, 2006.

[6] FIPA. FIPA Communicative Act Library Specification, 2002. Available under URL http://www.fipa.org/specs/fipa00037/.

[7] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In *AAMAS'02*, pages 536–542, Bologna, Italy, July 15–19 2002. ACM Press.

[8] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[9] J. J. Gómez-Sanz, J. Botia, E. Serrano, and J. Pavón. Testing and Debugging of MAS Interactions with INGENIAS. In *AOSE'08*, pages 199–212, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[11] D. T. Ndumu, H. S. Nwana, L. C. Lee, and J. C. Collis. Visualising and debugging distributed multi-agent systems. In *AGENTS'99*, pages 326–333, New York, NY, USA, 1999. ACM.

[12] M. Nickles, M. Rovatsos, and G. Weiss. Expectation-oriented modeling. *Engineering Applications of Artificial Intelligence*, 18(8):891–918, 2005.

[13] L. Padgham, M. Winikoff, and D. Poutakidis. Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173–190, 2005.

[14] D. Robertson. A lightweight coordination calculus for agent systems. In *DALT'04*, pages 183–197, 2004.

[15] M. Rovatsos, M. Schillo, K. Fischer, and G. Weiss. Indicators for Self-Diagnosis: Communication-based Performance Measures. In *MATES'03*, volume 2831 of *Lecture Notes in Artificial Intelligence*, pages 25–37, Berlin, Germany, 2003. Springer-Verlag.

[16] E. Serrano, J. J. Gómez-Sanz, J. A. Botia, and J. Pavón. Intelligent data analysis applied to debug complex software systems. *Neurocomputing*, 72(13-15):2785 – 2795, 2009.

[17] E. Serrano, A. Muñoz, and J. Botia. An approach to debug interactions in multi-agent system software tests. *Information Sciences*, 205(0):38 – 57, 2012.

[18] E. Serrano, A. Quirin, J. A. Botía, and O. Cordón. Debugging complex software systems by means of pathfinder networks. *Information Sciences*, 180(5):561–583, 2010.

[19] R. Siebes, D. Dupplaw, S. Kotoulas, A. P. De Pinninck, F. Van Harmelen, and D. Robertson. The OpenKnowledge system: an interaction-centered approach to knowledge sharing. In *OTM'07*, pages 381–390, Berlin, Heidelberg, 2007. Springer-Verlag.

[20] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf, and W. Renz. Validation of BDI Agents. In *PROMAS'06*, pages 185–200, 2006.

[21] A. L. Symeonidis, K. C. Chatzidimitriou, I. N. Athanasiadis, and P. A. Mitkas. Data mining for agent reasoning: A synergy for training intelligent agents. *Eng. Appl. Artif. Intell.*, 20(8):1097–1111, 2007.

[22] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In *AAMAS'02*, pages 952–959, New York, NY, USA, 2002. ACM.