An Approach for the Qualitative Analysis of Open Agent Conversations

An Approach for the Qualitative Analysis of Open Agent Conversations

Emilio Serrano[1], Michael Rovatsos[2], and Juan A. Botia[1]
[1] *University of Murcia* *, {emilioserra,juanbot}@um.es
[2] *University of Edinburgh,* michael.rovatsos@ed.ac.uk

Abstract

This paper presents an approach for the qualitative analysis of data obtained from past communicative interactions in an open multiagent system. Such qualitative analysis focuses on the use of high-level agent communication languages to infer theories about agents with mental states which are normally not accessible for the outside observer. The inference of these theories, or context models, is based on logging semantic data available from protocol execution traces and using this information as samples for the application of data mining algorithms. These context models can be applied both by system developers and agents themselves at run-time for various tasks, e.g. to predict future agent behaviour, to support the process of ontological alignment in communication, or to assess the trustworthiness of agents. An implementation of the approach presented is also given, the ProtocolMiner tool, which automates the building of context models from arbitrary protocol executions.

Keywords: Agent communication languages, interaction protocols, interaction analysis, data mining, agent-oriented software engineering.

1 Introduction

The interaction among autonomous, rational agents is one of the essential characteristics of a multiagent system (MAS). For this reason, most MAS platforms offer various mechanisms to support such interaction, including an infrastructure for sending and receiving messages, establishing conversations, following interaction protocols, sharing vocabularies using ontologies, etc.

Most of the analysis tools included in MAS development frameworks fall into two categories: (1) analysis of agents' mental states, and (2) analysis of the interactions among agents. The analysis of mental states usually assumes that the

*Acknowledgments: This research work is supported by the Spanish Ministry of Science and Innovation under the grant AP2007-04080 and 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.

agents' implementation is known and available. This constraint is very restrictive in open MAS that run on interoperable agent interaction platforms where different agents can be added to the system by different users at runtime. On the other hand, methods for interaction analysis often remain very superficial and address only fixed syntactic elements that can be observed in message exchanges (e.g. performative, sender and receiver, successful/unsuccessful completion of the protocol, etc). This paper covers these shortcomings employing the semantics of interactions to perform a *qualitative analysis* which is able to infer theories about agents with hidden mental states.

To appreciate the utility of a qualitative analysis, let us take the FIPA Contract-Net protocol as example [1]. This protocol describes an agent (the *Initiator*) who wants one or more agents (the *Participants*) to perform a task. In this protocol, there are a large number of analysis tasks that can be performed following a *quantitative analysis*: (1) obtaining the number of conversations in which each agent has participated; (2) conversations in which an error occurred in the flow of messages defined by the protocol; (3) number of agents which rejected a proposal; (4) number of tasks that a participant was unable to perform after committing to them; etc.

Although this quantitative analysis can be very useful for a developer, more interesting information is usually captured in the specific semantics of messages, i.e. *qualitative properties*. The developer of the Initiator agent, for example, may be interested in which tasks are usually rejected by agents or, more specifically, how the process used by a Participant to accept or reject a task is implemented (in the sense of a decision rule in the agent's reasoning mechanism). Of course, this information is hidden if the Participant agents' implementation is unknown by the Initiator. Nonetheless, considering a concrete past execution of the protocol, the Initiator agent can easily recognise if a particular task has been accepted or rejected by a specific Participant. What is more, after several executions of the protocol, the Initiator can generalise the individual cases to build a more general theory that explains participants' behaviours.

We call theories which allow a developer to perform a qualitative analysis *context models*. As illustrated in the example, these models correlate the status of logical constraints attached to interaction protocol specifications to perceived agent behaviour. In other words, they map the conditions under which a certain behaviour occurs to the resulting behaviour itself.

Construction of these context models is based on capturing regularities in previously observed interactions by using data mining techniques. Context models are able to reveal implicit causal relationships between states of the system and the reasoning and decision-making mechanisms of all agents involved. These models can be used for various purposes: (1) to make predictions about future behaviour; (2) to infer the definitions other agents apply when validating logical constraints during an interaction; and (3) to analyse the reliability and trustworthiness of agents based on the logical coherence of their utterances.

In addition to the definition, construction and use of the context models; the contribution of this paper is to present an approach to automatically generate these models and an implementation of this approach, the *ProtocolMiner* tool.

The remainder of the paper is structured as follows. After reviewing related

work in section 2, we introduce the formal approach in section 3. The ProtocolMiner tool is presented in section 4. Section 5 gives empirical results obtained in a case study. Finally, section 6 concludes.

2 Related work

Many tools for run-time multiagent systems analysis address the testing, debugging, validation or verification of these systems. For example, the *Tracer Tool* [9] provides a semi-automated solution for agent software understanding, using high-level agent concepts instead of detailled execution traces and programming data structures. The tool proposed in [14] for the JADEX agent platform can be used to verify the consistency of internal events and message declarations and to obtain an overall communication structure as a three-dimensional graph. The inspector tool [6] for the the *Agent Factory Agent Programming Language* (AFAPL) provides support for the inspection of the internal states of agents, and for monitoring the performance of the underlying agent components. Similarly, INGENIAS [8] provides a visual debugging tool to inspect agents' mental states. These approaches are, however, only suitable for systems in which the mental states of the agents can be inspected by the designer, i.e. effectively only for systems whose code has been disclosed *a priori*, or who have been designed by the user performing the analysis themselves.

In contrast to this, there are also methods aimed at design-time (static) analysis of multiagent systems properties such as MABLE [15]. This imperative programming language uses the SPIN model checker to automatically verify properties of the system. While valuable, these approaches can only verify certain properties of agent interactions (based on observed interactions or on design-time specifications). However, they cannot derive any *additional* and *explicit* knowledge about the emergent behaviour of the system apart from whether agents are behaving correctly or not.

The only exception to this is some work that has recently emerged in the area of ontology matching [3, 4]. In these contributions, hypotheses about the possible meanings of unknown terms used by the other agent are filtered based on structural knowledge of the protocols. This is achieved by either (1) looking at the ontological relationships between candidate concepts based on the terms that appear earlier in the same dialogue or in previous dialogues, or (2) by reasoning about the overall syntactic structure of the protocol. However, this kind of reasoning is only used to resolve ontological conflict. On the other hand, the approach presented in this paper is able to infer more general emergent properties of interactions.

The notable limitation of all (but the latter) existing work is that it does not consider semantic elements of interactions for analysis, e.g. the constraints used by the agents while they are executing protocols, and which cause a concrete protocol execution to unfold in a particular way. Also, they fail to induce compact, explicit theories about the ways in which interaction evolves in a system, and which could be useful for the design of adaptive agents. The following section describes the ProtocolMiner tool which is able to induce this kind of theories. MR: the next section describes the theory, not the tool.

3 Formal approach for a qualitative analysis

Beyond the presentation of the ProtocolMiner tool, which uses a specific protocol specification language, the analysis provided by this tool is based on a formal approach. This approach for qualitative interaction analysis is independent of the specific development platform used. This section formalises: how to define protocols that are semantically annotated; how to obtain a context model from the execution of these protocols using data mining techniques; and how to build the training data set for these techniques.

3.1 Defining a protocol and its context

In brief, our framework is based on defining a *protocol model* as a graph $G = (V, E)$. In this graph, 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. Besides, each edge is labelled with a (conjunctive) list of (say, n) constraints $c(e) = \{c_1(t_1, \ldots, t_k), \ldots, c_n(t_1, \ldots t_{k_n})\}$. 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 logical language $\mathcal{L}$). All variables that occur in such constraints are implicitly universally quantified. The framework also assumes that all outgoing edges of a node result in messages with distinct performatives, i.e. for all $(v, v'), (v, v'') \in E$, $(m(v') = q(\ldots) \wedge m(v'') = q(\ldots)) \Rightarrow v' = v''$. Therefore, each observed message sequence corresponds to (at most) one path in G by virtue of its performatives. Figure 1 shows an example protocol model in this generic format for illustration purposes.

The *semantics* of a protocol model G can be defined by looking at the pairs $\langle \pi, \theta \rangle$ which specify the path and variable substitution that any message sequence $\mathbf{m}$ corresponds to in protocol model G. With this, we can define the *context* of $\mathbf{m}$ as $c(G, \langle m_1, \ldots, m_n \rangle) = \bigwedge_{i=1}^{n-1} c(e_i)\theta$ where $G(\mathbf{m}) = \langle \pi, \theta \rangle$. Crucial to our view of qualitative interaction analysis is the assumption that for any observed message sequence $\mathbf{m}$, the conjunction of edge constraints described by the context $c(G, \langle m_1, \ldots, m_n \rangle)$ was logically true at the time of the interaction.

3.2 Obtaining a context model by data mining

The basic method for applying data mining methods to protocol interactions 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 $G(\mathbf{m}) = \langle \pi, \theta \rangle$ as defined above. This approach assumes that only sequences allowed by G occur (if necessary, G can be modified on the fly to accommodate unexpected messages by adding constraint-free edges and message nodes). Assuming that a set of such substition-annotated paths are 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].

This paper proposes 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

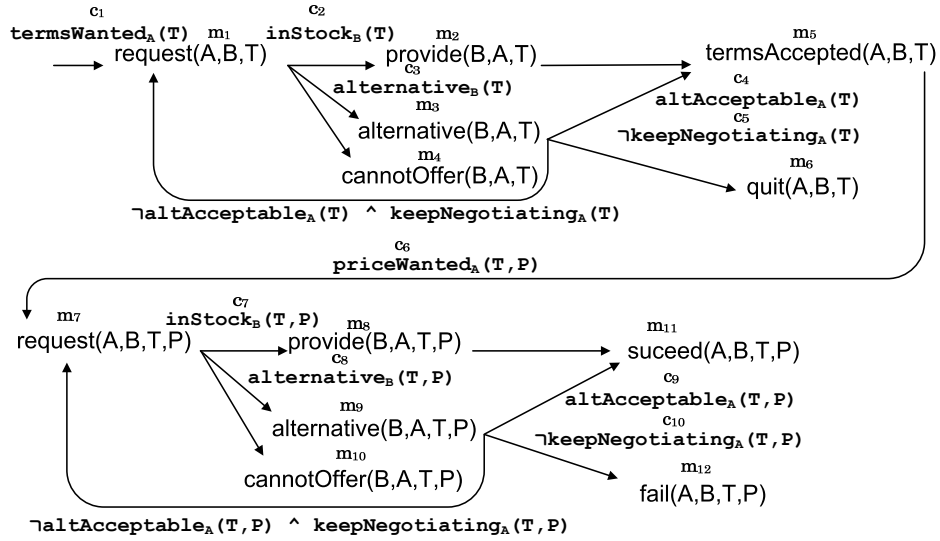

Figure 1: A simple negotiation protocol model: A decides the terms of a desired product, and requests T from B. The initial response from B depends on availability: if terms T cannot be satisfied, A and B go through an iterative process of negotiating new terms for the item, depending on the *keepNegotiating*, *altAcceptable*, and *alternative* predicates. In case of acceptance, the process of negotiation is repeated to decide the price P for the product. Edge constraints are annotated with the variable representing the agent that has to validate them (subscript A or B). Additional (redundant) shorthand notation c_i/m_j is introduced. Different out-edges represent XOR if constraints are mutually exclusive, and OR else.

can be directly inferred using the logical constraints provided by the definition of G. The model obtained with the learning algorithm using the context as training data is what we call a *context model*.

Determining the most suitable learning algorithms for a particular context mining task is beyond the scope of this paper, our method does not depend on the use of a specific algorithm. 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. In this case, a decision tree algorithm has been employed to learn a specific constraint in the protocol.

3.3 Preparing the training data set

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

```
persons = 2: F (158)
persons = 4: F (158)
persons = more
|   lug_boot = small
|   |   doors = 2: F (8)
|   |   doors = 3: F (7)
|   |   doors = 4: F (8)
|   |   doors = 5-more: T (105)
|   lug_boot = med
|   |   doors = 2: F (13)
|   |   doors = 3: F (8)
|   |   doors = 4: F (13)
|   |   doors = 5-more: T (120)
|   lug_boot = big: T (402)
```

Figure 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.

3.3.1 Dealing with different agents

In defining the datasets to be used for protocol mining, one important design decision is how to deal with the presence of multiple agents. If all messages and contexts that occur in observed interactions were treated as features of learning samples this would amount to an attempt to derive globally valid interaction patterns. This learning strategy would imply 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, the purpose of interaction mining is 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 groups of agents. Assume an assignment $\sigma : Var \to Ag$ 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 $a \in Ag$, $V_\sigma(a)$ 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 $A \subseteq Ag$ by taking the union over the respective sets for agents.

As an example, consider the protocol model depicted by the graph in figure 1. Assuming a set of agents $Ag = \{a_1, a_2\}$, a role assignment $\sigma = \{[A/a_1], [B/a_2]\}$, and the `request-provide-termsAccepted` path, $V_\sigma(a_1)$ would contain the `request` and `termsAccepted` messages and $E_\sigma(a_1) = \{termsWanted(T)\}$ as the only constraint of the incoming edges to utterances performed by a_1.

The most cautious form of data filtering in this setting would be to reduce the path π of every sample to those nodes and edges that pertain to the learn-

ing agent a_i. Only contextual information $c(G_{a_i}, \mathbf{m})$ from the restricted graph $G_{a_i} = \langle V_\sigma(a_i), E_\sigma(a_i)\rangle)$ would be considered because a_i can safely verify "own" constraints along π. With this strategy, 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. $G(\mathbf{m}) = \langle \pi, \theta \rangle$.

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. This strategy assumes that the definitions of constraints are common to all agents *and* that every agent verifies the constraints reliably and honestly.

3.3.2 Dealing with paths, loops, and variables

Standard data mining algorithms assume a fixed number of attributes (features) and values. Because of this, in our approach to qualitative analysis a number of issues arise that require certain further design decisions to be made.

Firstly, when collecting different paths for inclusion in the training dataset, their labels (messages/constraints) and the set of variables contained in these labels may differ. This is not a problem in principle, as data samples can be "padded" with "unknown" values for all messages and context constraints that do not occur in them, but this can be computationally wasteful. In many practical cases, it will be more appropriate to create a different data set for each observed path π. This is because any model learnt over such path-specific training data captures better the precise circumstances under which it occurs.

Moreover, 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. Different messages along the path can even be ignored introducing a single path label (or path group label) for each path to predict interaction outcomes. For example, an artificial boolean label *success* can be attached to a number of paths, effectively classifying different paths into two categories (where paths with *success* = *true* belong to the category of successful interactions, and all other paths are deemed unsuccessful).

Secondly, the results of the constraints functions in the context (but not the attributes in the arguments of these constraints) should be removed when the learning algorithm tries to predict the "outcome" value. This information is explicit in the definition of the protocol and is reflected in the path models it provides. Moreover, including these results may hinder learning techniques from relating the details of constraint argument values (note that the definition of the constraints in the protocol is still necessary) to the overall outcome of the protocol.

Thirdly, many common interaction protocols (e.g. negotiation protocols like auction and bargaining protocols) involve iterations of sub-sequences that can be repeated an arbitrary or number of times. The existence of a loop in a protocol means that variables occurring in a logical constraint or messages 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. Moreover, n may vary depending on the number of iterations occurred in each run. Different strate-

```
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  ) or (refuse()  => a(initiator, A))
```

Figure 3: LCC implementation of the "participant" role in the FIPA Request Interaction Protocol

gies can be employed to obtain fixed-length samples for use in the training dataset. (1) If N is the maximum number of iterations observed in a protocol across various runs, N "copies" of each variable can be kept (with copies of messages and context predicates that contain it as attributes in the learning samples). This, however, introduces a lot of redundancy in paths with fewer than N iterations of the loop which will have to use a value of "unknown". (2) Considering only the first/last ground term g_1/g_n for a specific variable V in a loop. In many cases (e.g. negotiations) keeping two copies of each variable, one for the first and one for the last value will suffice as intermediate steps are less important for the outcome of the interaction.

4 ProtocolMiner

ProtocolMiner is a prototypical tool that provides comprehensive functionality for qualitative protocol mining. 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 emerging knowledge extracted from past interaction.

We first briefly introduce the definition of semantically annotated protocols that ProtocolMiner is designed to operate on. This is followed by details about how ProtocolMiner implements the formal approach described in Section 3.

4.1 Defining protocols in ProtocolMiner

The ProtocolMiner tool is a plugin for the OpenKnowledge platform [13], a protocol specification and execution platform designed for large-scale heterogeneous multiagent systems. ProtocolMiner automates the construction of context models. This includes the registration and recovery of the training data for any protocol implemented and executed in OpenKnowledge. OpenKnowledge uses a protocol definition language called the *Lightweight Coordination Calculus* or LCC [11], 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 [2] in LCC is 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 terms of the message sequences allowed for that role. Incoming/outgoing messages are denoted by double arrows `<=` and

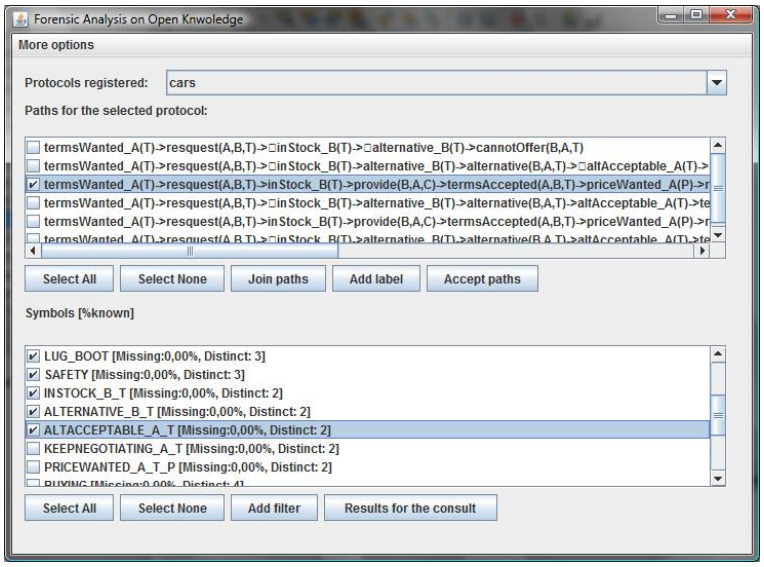

Figure 4: ProtocolMiner user interface

`=>` 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 `->`, these are 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 to use LCC based protocol specifications, other interaction platforms may be used as data sources as long as their specification mechanism can be translated to the protocol-model graph structure defined in section 3.1.

4.2 ProtocolMiner features

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 using LCC language. ProtocolMiner also integrates the Weka [5] algorithms allowing the tool to use a large number of data mining algorithms to obtain the context models describes in section 3.2. The strategies to build a training data described in section 3.3 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. 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 procedure. The tool also allows a developer or agent to select the paths considered in the data set and to label them with an "output attribute" by means of the option *"add label"*. Moreover, several paths that only differ in the number of iterations of a loop can be selected and merged to a single path using the *"join paths"* option. Besides, the tool can be configured to (1) log all the values given to an attribute in a path (every value is stored in a new attribute), (2) to log only the first and last values along a path, or (3) to log only the last value given to an attribute at the end of a protocol execution.

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.

5.1 Description of the system under test

To make the discussion concrete, we apply our system to a well-known database for car evaluation [7]. 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 luggage boot and 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 $T = (doors, persons, lug\_boot, safety)$.

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 the two attributes refer to the cost of purchase and maintenance.

We define ten customer agents C_i, where $1 \leq i \leq 10$, with associated mental states, i.e. different preferences regarding T and P that determine what offers they will accept, where $C_i := MS_{i \bmod 5}$. Therefore, 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[1].

[1]Due to space limitations, this section does not detail the possible values of T, P, the mental states for the customers, and the decisions made by customers and sellers to follow the protocol described in figure 1. An extended version of this evaluation can be found at `http://ants.dif.um.es/staff/emilioserra/QA/EE.pdf`, and provides the necessary detail to reproduce our results.

5.2 Strategy to build the training data

In converting raw sequences of message exchanges to training data samples (see section 3.3), we make use of the simplest, most general data generation method that ProtocolMiner offers.

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.

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. Our strategy to deal with loops is to only record the last value of every variable occurring in multiple iterations. We introduce a variable $outcome \in \{S, F, N\}$ to denote Successful completion of the negotiation (path ended with m_{11}, see figure 1), Failure to unsuccessful (paths m_4 and m_6) and Neutral for a "neutral" outcome (the remaining paths).

Three open source implementations of data mining techniques are employed using their default parameters to illustrate the impact of using different algorithms in an exemplary way, and also to show that our method 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 neighbor like algorithm) and the *BayesNet* technique (a Bayesian network classifier) [5].

5.3 Learning 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.

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

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

This formula is logically equivalent to the mental state implemented by the customer, $MS_1(T)$, and, therefore, the constraint has been learned completely by the seller using context models even without having access to the customer's implementation.

5.4 Protocol outcome prediction by context models

To conduct a more exhaustive evaluation, a seller tries to learn a context model for the overall outcome of the protocol. Figure 5 shows the average model accuracy across 100 repeated experiments. The accuracy of the context models is evaluated using cross-validation across 100 experiments with 10000 negotiations each.

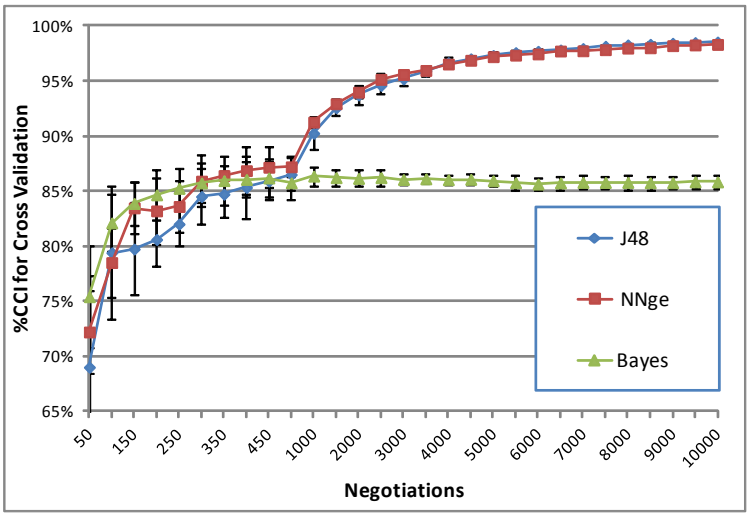

Figure 5: Average model accuracy (based on cross-validation) shown across total number of negotiation (100 experiments). Learning algorithms: J48, NNge and BayesNet. Error bars show standard deviation.

The experiments demonstrate that accurate context models can be built using contextual 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. Therefore, as the following section shows, an agent can build a context model after only a few negotiations, consider a specific context as input for the model, and predict the outcome of the protocol without actually interacting with another agent.

5.5 Agents predicting 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 build and use context models to choose a good seller for the concrete context (including the 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 different mental state models used by customers.

We compare the prediction accuracy of our system against two alternative analysis strategies: (1) *Random* and (2) *Quantitative*. For (1), 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). For (2), the seller is chosen using a distance function based on the number of past successes and failures with them in the customer's personal

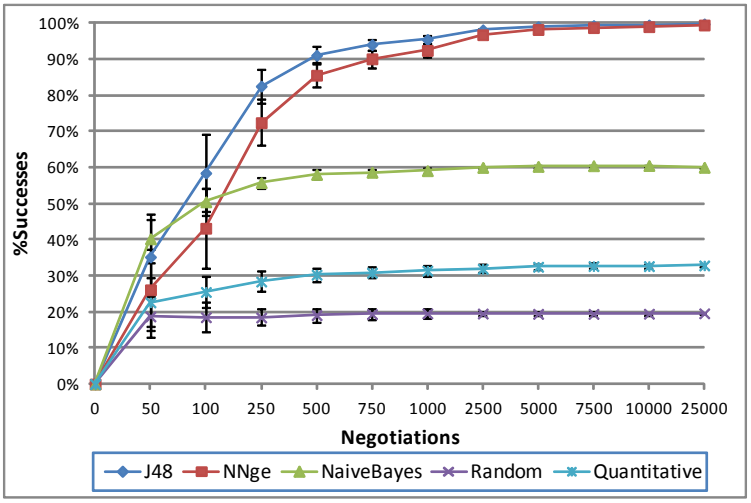

Figure 6: Average number of successful negotiations against number of total negotiations (100 experiments); error bars show standard deviation. Learning algorithms: J48, NNge and BayesNet.

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 a probability proportional to $D(s, f)$ [12].

As figure 6 shows, after 100 negotiations (10 negotiations per customer) the use of context models (independently of the data mining technique used) greatly outperforms the random and quantitative strategies. Besides, the use of decision trees converges faster to optimal performance than the other two learning techniques considered. In these experiments, using context models built with J48 or NNge, customers reach over 90% of successes after only 750 negotiations (75 negotiations per customer) and over 99% after 7500 conversations.

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 these protocols, messages are associated with logical constraints, which can be used as "semantic" annotations of communication in a natural way. This work is motivated by shortcomings in existing multiagent systems analysis methods which mostly ignore this rich source of *contextual information* when analysing run-time multiagent interactions. The main advantage of using contextual information is that data mining methods can be used to infer qualitative information from observed message exchanges.

A formal approach has been detailled with the aim of making interaction data available for qualitative data mining. In this approach, information about the shared protocol models has been used as background knowledge. As part of the

approach presented, this paper has discussed different alternatives for dealing with the specific nature of agent interaction protocols when converting interaction experiences to training data. This involves addressing issues such as the presence of multiple agents, variable-length execution paths, and loops that are commonly present in common multiagent interaction protocols. Subsequently, an implementation of our formal approach, ProtocolMiner, has been presented. Finally, a case study has been described (with an extended version available on-line) to hint at the potential of applying data mining in real-world 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.

References

[1] FIPA Contract Net Interaction Protocol Specification. SC00030, 2002. Foundation for Intelligent Physical Agents.

[2] FIPA Request Protocol Specification. SC00026, 2002. Foundation for Intelligent Physical Agents.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] D. N. Lam and K. S. Barber. Comprehending agent software. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 586–593. ACM, 2005.

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

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

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

[13] 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.

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

[15] 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.