Information-seeking agent dialogs with permissions and arguments

Sylvie Doutre
Université Toulouse 1 – IRIT
Manufacture des Tabacs
21 allées de Brienne
31042 Toulouse cedex France
doutre@irit.fr

Peter McBurney, Michael Wooldridge, and William Barden
Department of Computer Science
University of Liverpool
Liverpool L69 7ZF UK
{p.j.mcburney,m.j.wooldridge,cs3wb}@csc.liv.ac.uk

ABSTRACT

Many distributed information systems require agents to have appropriate authorization to obtain access to information. For some applications, such as those in medicine, authorization may be granted in exceptional cases even to agents without the necessary permissions, provided they can provide an appropriate justification for having access. We may consider these agent interactions to be information-seeking dialogs with the agent seeking the information and the agent controlling access to it engaged in argument together over data access. We present a denotational semantics for such dialogs, drawing on Tuple Centers (programmable Tuple Spaces). We illustrate our approach with an example in medicine, and describe an implementation we have created using *TuCSoN*.

ACM CATEGORIES AND SUBJECT DESCRIPTORS:
I.2.11 **[Artificial Intelligence]** Distributed Artificial Intelligence: *Coherence and co-ordination; Languages and Structures; Multiagent systems.*
GENERAL TERMS: Design, Languages, Theory.
KEYWORDS: Argumentation, Agent Communications Languages, Information-seeking dialogs, Semantics, Tuple Spaces.

1. INTRODUCTION

Consider a common medical situation where an emergency-room doctor requires access to the records of a newly-arrived, but unconscious patient, perhaps the victim of a traffic accident. Because the doctor is not the patient's usual medical practitioner, the doctor will require permission to access the patient's records from the relevant medical organization. Because the patient is unconscious, he or she cannot give this permission. The emergency-room doctor will need to persuade the patient's own doctor or medical center personnel that access should, exceptionally, be granted. It is possible to view this interaction between emergency-room doctor and the patient's own doctor as an information-seeking dialog, where access to the information requires appropriate permissions, and where arguments may be presented and considered to change the current level of these permissions. As services move online, we can imagine this interaction taking place between software agents acting for human principals, rather than between the principals themselves. To enable this electronic interaction, we present a formal syntax and semantics for such information-seeking dialogs involving permissions and arguments, and report on an implementation.

First, however, we make some general remarks regarding agent interaction protocols and their semantics. Agent researchers and developers have devoted considerable attention recently to the design and study of protocols for agent communication using dialog games taken from philosophy, e.g., [13]. Much of this attention has focused on the syntax of protocols, with perhaps less attention paid to their semantics. There are several different functions that a semantics for an agent communications language or dialog protocol may be required to serve:

- To provide a shared understanding to participants in a communicative interaction of the meaning of individual utterances, of sequences of utterances, and of dialogues.
- To provide a shared understanding to designers of agent protocols and to the designers (who may be different) of agents using those protocols of the meaning of individual utterances, of sequences of utterances, and of dialogues.
- To provide a means by which the properties of languages and protocols may be studied formally and with rigor, either alone or in comparison with other languages or protocols.
- To provide a means by which languages and protocols may be readily implemented.

In this paper, our focus is on semantics for agent protocols which meet this last objective. Drawing on research in programming language semantics, Rogier van Eijk [8] identified three generic types of semantics of agent communications languages. An **axiomatic** semantics defines each locution of a communications language or protocol in terms of the pre-conditions which must exist before the locution can be uttered, and possibly also the post-conditions which apply following its utterance, in a STRIPS-like fashion. For example, the semantic language, *SL*, for the locutions of the FIPA Agent Communications Language, is an axiomatic semantics of the speech acts of the language, defined in terms of the beliefs, desires and intentions of participating agents [9]. Similarly, the semantics defined for many dialog game protocols for agent interaction, e.g., [15], are also axiomatic semantics.

A second type of semantics, an **operational semantics**, considers the dialog locutions as instructions which operate successively on the states of some abstract machine. Here, the semantics defines the locutions in terms of the transitions they effect on the states of this machine. Operational semantics have recently been defined for some agent dialog protocols, e.g., [14]. Third, in **denotational semantics**, each element of the language syntax is assigned a relationship to an abstract mathematical entity, its denotation. Perhaps the first example of a denotational semantics for a dialog protocol was the possible-worlds semantics for question-response interactions defined by Charles Hamblin in 1957 [11]. Although possible-worlds and other denotational semantics have a long subsequent history in mathematical linguistics, only recently have denotational semantics been defined for agent dialog protocols. For instance, [16] presents a category-theoretic semantics for a broad class of deliberation dialog protocols, and uses this semantics to prove properties of dialogs conducted under these protocols.

Of these different types of semantics, axiomatic semantics would seem to be of most value to software engineers tasked with building a system to allow multi-agent communications. However, if the pre-conditions and post-conditions of an axiomatic semantics are defined in terms of the private mental states of the agents communicating, then problems of semantic verifiability arise [23]. Since these mental states are inherently *private*, there can be no way in general to verify their properties. Insofar as the agent's internal states are assumed part of the larger virtual machine, as in [14], the same problem arises with an operational semantics.

For this reason, we are driven to consider denotational semantics for agent communications protocols. However, the translation of dialog utterances and protocols into abstract mathematical entities, such as the collections of categories of [16], may not necessarily assist a software developer in implementing the protocol. Accordingly, we propose a denotational semantics for an agent interaction protocol which translates utterances under the protocol into commands in a version of the *Linda* distributed computing language, and into actions on associated tuple spaces. Because of their adoption by SUN (under the name *Javaspaces*)[1] and by IBM (under the name *T-Spaces*)[2], tuple spaces are a popular technology for implementation of distributed computing applications. In the next section, we introduce tuple space theory, and present our ideas regarding their use as a semantics for agent dialog protocols.

To illustrate the application of these ideas, we present, in Section 3, an extended example of a multi-agent dialog in a medico-legal domain. The example is that of the opening paragraph of this paper, of an Information-seeking dialog between two agents, where the requesting agent may need to provide the responding agent with an argument or justification in order to obtain access to the information or objects requested. Without the argument component, such dialogs may be viewed as consisting of *Ask* and *Tell* locutions, with the additional requirement that the responding agent may only utter *Tell* if the requesting agent has permission to receive the requested information or object. The protocol for Information-seeking dialogs which we present enables not only these *Ask* and *Tell* locutions, but also allows the participants to justify their requests or responses through argument. The requesting agent may present these arguments with the intention of changing the level of authorization it has to specific information; the responding agent may present arguments to explain its decisions to *Tell* or not to *Tell* the requested information. Section 3 presents our protocol for such interactions, along with its denotational semantics.

Section 4 reports on an implementation of this protocol undertaken using *TuCSoN*, a software platform for tuple centres, and the associated programming language, *ReSpecT*. The paper ends with a discussion of related and future work in Section 5.

[1]See: `http://java.sun.com/developer/products/jini/`.

[2]See: `http://www.alphaworks.ibm.com/tech/tspaces/`.

2. DIALOG SYSTEMS

In this section, we will first recall the general architecture of an agent dialog system, and summarize relevant aspects of Tuple Space theory. We then show how a version of tuple spaces may be understood as a denotational semantics for dialog systems.

2.1 Dialog systems

The common elements of dialog systems are: a *dialog goal*; a set of *participants* who may have their own dialog goals, knowledge base, mental states, and reasoning capabilities; a *topic language*; a *context* built on the topic language, which contains the knowledge that is presupposed and must be respected during a dialogue; a *communication language* defining the set of dialogues, and whose syntax contains the *locutions* which can be uttered in a dialogue; a *protocol* (the heart of the dialog system) which specifies the utterances permitted at each point in a dialogue; a set of *effect rules*, specifying the effects of utterances on the participants' commitments and possibly also, on their knowledge base; a set of *outcome rules* defining the outcome of a dialogue. A set of public *commitments* may also be associated to each participant as indicated in [13].

A typology of human dialogs was articulated by Walton and Krabbe [22], based upon the overall goal of the dialogue, the participants' individual dialog goals, and the information they have at the commencement of the dialog (the topic language and the context). For example, *Information-seeking dialogues* are those where one participant seeks the answer to some question(s) from another participant, who is believed by the first to know the answer(s). *Persuasion dialogues* involve one participant seeking to persuade another participant to endorse a statement he or she does not currently endorse. Dialogue-game protocols, some using argumentation, for these types of dialog have been proposed, e.g., [13]. But we know of no protocols for secure information-seeking in which argumentation-based persuasion is enabled.

2.2 Tuple spaces

David Gelernter's theory of *tuple spaces* [10, 3] was proposed as a model of communication between distributed computational entities. *Linda* is the associated programming language. The essential idea is that computational agents connected together may create named object stores, called *tuples*, which persist, even beyond the lifetimes of their creators, until explicitly deleted. In their Javaspaces manifestation, tuples may contain data, data structures, programs, objects or devices. They are stored in tuple-spaces, which are blackboard-like shared data stores, and are normally accessed by other agents by associative pattern matching. The use of shared stores means that communication between multiple agents can be spatially and temporally decoupled. There are three basic operations on tuple spaces:

`out` with which an agent creates a tuple with the specified contents and name in a shared space accessible to all agents in the system.

`rd` with which an agent makes a copy of the contents of the specified tuple from the shared space to some private store.

`in` with which an agent makes a copy of the contents of the specified tuple from the shared space to some private store, and then deletes it from the shared space.

Tuple spaces are public-write, public-read spaces: any entity in the system may create a new tuple, and any entity may delete an existing one. A refinement of Linda, *Law-Governed Linda (LGL)* [18], established an administrative layer which authorizes all attempts to execute `out`, `in` and `rd` commands according to pre-defined security and privacy policies. This administrative layer (the *Law* of the model) also describes the effect of the invocation of these operations on the model.

More precisely, an LGL model is a 5-tuple $< \mathcal{C}, \mathcal{P}, \mathcal{CS}, \mathcal{L}, \mathcal{E} >$ where: $\mathcal{C}$ is a tuple space; $\mathcal{P}$ is a set of *processes* that interact with each other via $\mathcal{C}$; $\mathcal{CS}$ is a set of *control states*, one being associated with each process; $\mathcal{L}$ is the *global law* of the system, which governs the interactions of the various processes in $\mathcal{P}$ with the tuple space $\mathcal{C}$, and thus, with each other; $\mathcal{E}$ is a mechanism that enforces the law.

The law $\mathcal{L}$ prescribes the consequences required of events, such as the invocation of an `out`, `rd` of `in` operation, that occur at the boundary between a process (called the *home process* of the event) and the tuple space. These effects have the form of a sequence of operations which must be carried out in response to the event. The effects may concern the tuple space or the control state of a process. An example of an event is an `out(t)` operation invoked by a process `p`: a first effect may be to actually store the tuple `t` in the tuple space, and a second one may be to update the control state of `p`.

The control state of a process is a bag of terms (called *attributes*) which have the form `f(arguments)`, where `f` is a literal symbol and its zero or more arguments are terms. An attribute that is present in each control state is `self(i)`, where `i` is a unique identifier of the process. Two operations on control states are `+a`, which adds the term `a` to the control state of the home object, and `-a` which removes from the control state of the home object the term `a`.

The law is global, in the sense that all processes have to obey the same law. However, the application of the law for a given event might depend on the control state of the process in which the event happened. The law is enforced by means of the distributed enforcement mechanism $\mathcal{E}$, which usually, in LGL, consists in associating a local *controller* with every process in the system. This local controller deals with the application of the law according to the control state of the process it governs, and maintains this control state. For instance, an `out(t)` operation might be possible only if the control state contains a special attribute which authorizes this operation. The combination of a process and its controller is referred to as an *object*.

LGL may be seen as a special case of a more general artifact, a *Tuple Center* [19]. This is a tuple space that is able to react to specific events, such as the insertion or deletion of a tuple by particular processes, and so is dynamic, not static. A logic-based language for programming tuple centers, called *ReSpecT*, has been developed [6] and combined with a software platform for creating tuple centers, called *TuCSoN* [5]. The expressive power of LGL and *ReSpecT* are essentially the same [19], with the primary difference being the location of tuple center behavioral rules. As described above, the LGL controllers are typically stored locally, within communicating processes, while under *ReSpecT*, control rules are vested in the tuple center. In Section 4, we report an implementation of the protocol presented in Section 3 using the *TuCSoN* platform.

2.3 LGL as a semantics for dialog systems

We have recalled the general syntax of dialog systems in Section 2.1. We are now going to show how LGL can be used as a denotational semantics for these systems, by associating elements of an LGL 5-tuple $< \mathcal{C}, \mathcal{P}, \mathcal{CS}, \mathcal{L}, \mathcal{E} >$ to the elements of the dialog system. Note that the dialog goal and the outcome rules have no associated elements in LGL.

Participants

To each participant is associated an object (i.e. a process of $\mathcal{P}$ and its controller from $\mathcal{E}$) and a control state of $\mathcal{CS}$. The control state may contain attributes related to the knowledge base of the participant.

Topic language

The elements of the topic language are associated elements of LGL.

Context

The dialog takes place in the tuple space $\mathcal{C}$.

Communication language

To each locution is associated a tuple. An utterance of a locution then calls one or more events which may effect the tuple space and/or the control state of the home object associated to the participant making the utterance.

Protocol

Rules of $\mathcal{L}$ are associated with the rules of the dialog protocol.

Effect rules

With each effect rule of the dialog system is associated one or more operations on the knowledge base of the participants.

3. SECURE INFO-SEEK DIALOGUE

We present the syntax and an LGL semantics for a dialog system for an information-seeking dialogue, involving permissions and arguments. We illustrate this dialog system on an example of a multi-agent dialog in a medico-legal domain, extending the emergency-room situation described in Section 1.

3.1 Motivating example

Our example involves dialogs between agents in different legal jurisdictions. The agents are representatives of principals who are legal entities, namely the medical practice in the home-country of a travelling patient and the medical practice treating the patient in a foreign country where he has fallen ill. Either agent (and indeed, either principal) may claim to be acting in the best interests of the patient. The provision of information by an agent is constrained by the patient's prior instructions, the practices and preferences of the agent's legal principal, and the laws of the country in which the principal resides. An agent seeking information then requires permission to access the information, and an agent that would not have the permission to access some information would have to persuade the agent having control of the the information to give it this permission.

Robert is a British businessman visiting Brussels for a meeting. During his visit he becomes ill and is taken unconscious into hospital. The staff of the hospital suspect Robert has had a heart attack and seek to prescribe appropriate drugs for his condition. Unfortunately the safe choice of drugs depends upon various factors, including prior medical conditions that Robert might have and other drugs he may be taking. The hospital's agent is given the goal of finding out the required information about Robert, from the agent representing his London doctor.

Agent of Brussels Hospital: I would like to dialog with the agent of Robert's British doctor.

Agent of Robert's London Doctor: Yes, I agree to dialog with you.

Brussels agent: I request Robert's health record.

London agent: I can provide you with non-sensitive information (Robert's history of diabetes) but, because Robert has only given his British doctor limited consent to pass on his personal information, I cannot provide you sensitive information (Robert's psychiatric history).

Brussels agent: The missing information could possibly include information that could affect the treatment of Robert's heart failure. I request the sensitive information, Robert's life may be at stake!

London agent: I cannot divulge the sensitive information, because British law prohibits passing on information without the consent of the provider of the information.

Brussels agent: EC law takes precedence over British law when it would be in the interests of the owner to divulge the information. You should allow me to access the sensitive information.

London agent: Only Robert could decide what would be in his interests.

Brussels agent: Robert's doctor owes a duty of care to Robert and, should he die, the doctor might be sued by his family, or the Brussels hospital, or both.

London agent: I yield to this argument. You can access the sensitive information. I will provide it to you.

Brussels agent: Thank you.

3.2 Protocol syntax

This is the syntax of a dialog system for information-seeking which requires permission to access the information. In this system, an argument must be provided by an agent to justify it having permission to access some information. If access to information for agent x is refused by an agent y, then agent x may try to persuade agent y that it should be allowed permission. This persuasion is made using arguments. If agent y yields to agent x's arguments, then y provides x the information requested.

Participants

There are two participants, a *Client* (requesting information), and a *Server* (controlling access to some information, which it may or may not provide).

Dialog goal

The Client may have one or two of the following goals prior to the start of the interaction: (a) To engage in a communicative interaction with the Server; (b) To obtain from the Server all the information it needs, using persuasion if necessary. The Server may have one or two of the following goals prior to the start of the interaction: (a) To engage in a communicative interaction with the Client; (b) To provide information to the Client according to the laws he is subject to and to the level of access permission the Client has.

Context

Client and Server may have disjoint knowledge bases. The knowledge base of the Server includes information about the the access permissions which each Client has, which may differ by the information concerned. One task of the Client may be to convince the Server to add some permissions to his knowledge base.

Topic language

We assume the information requested is represented by lower case Greek letters (ϕ, ψ...). This information may be any of: a data record (e.g., one patient's record); a database (e.g., records of many patients); an executable program (e.g., an algorithm for determining the appropriate dosage of a particular treatment); the output of an executed program (e.g., the appropriate dosage of a particular treatment); or even the protocol for another dialog (e.g., the client may first request the Server to enter into a second dialogue, which requires authorization to engage in). The actual content corresponding to information ϕ is denoted by $<\ \mathrm{content}\ \phi\ >$. A participant is represented by a lower case Roman letter (x, y, ...). The permission a participant x has to access the content of information ϕ is denoted by $\mathrm{perm}(x, \phi)$. An argument is denoted by upper case Roman letters (A, B, ...). We leave the structure and the origin of the arguments unspecified.

Communication language

The minimum locutions needed for a dialog between Client and Server are:

OpenDialogue(x,y) Participant x indicates to participant y that it wants to enter into a dialog with y.

Ask(x,y,ϕ) Client x asks Server y to provide it with some information ϕ.

Tell(x,y,ϕ) Server x tells Client y that it can provide y with information ϕ.

DontTell(x,y,ϕ) Server x indicates to Client y that it cannot provide y with information ϕ.

Provide(x,y,$<\ \mathrm{content}\ \phi\ >$) Server x provides Client y the actual content of information ϕ.

Argue($\mathrm{perm}(y, \phi)$, mode, A) A participant gives an argument A about the permission that a participant y has to access information ϕ: this permission may be true, false, or to be added. This is indicated by mode, which has the value:

- YES, if A indicates that participant y has the permission to access ϕ.
- NO, if A indicates that participant y does not have the permission to access ϕ.
- ADD(x), if A indicates that Server x should add to his permission base that participant y has the permission to access ϕ.

Accept(x,A,$\mathrm{perm}(y, \phi)$) Server x says that he believes that argument A referring to permission $\mathrm{perm}(y, \phi)$ is acceptable.

EndDialogue(x,y) Participant x indicates to participant y that it wants to leave the dialogue.

Protocol

The protocol will specify which locutions may be uttered at different points in a dialogue, and so define the rules governing the use of the locutions given above. One such protocol would comprise the following eleven rules:

R1: A participant x may initiate a new dialog with a participant y by uttering **OpenDialogue(**x**,**y**)**. The dialog commences when the other participant also utters this locution.

R2: A participant x may put an end to a dialog engaged with a participant y at any time, by uttering **EndDialogue(**x**,**y**)**. Upon such an utterance, the dialog between x and y terminates.

R3: Client x may utter **Ask(**x**,**y**,**ϕ**)** at any time after the commencement of a dialog between x and y.

R4: Server x may utter **Tell(**x**,**y**,**ϕ**)** at any time after the commencement of a dialog between x and y, but only if the permission $\mathrm{perm}(y, \phi)$ is in x's knowledge base.

R5: Server x must utter **Provide(**x**,**y**,**$<$ content $\phi >$**)** if **Tell(**x**,**y**,**ϕ**)** has been previously uttered.

R6: Server x utters **Argue(**$\mathrm{perm}(y, \phi)$**,** YES**,** A**)**, where A is an argument justifying that y has the permission to access ϕ, if and only if x has previously uttered **Tell(**x**,**y**,**ϕ**)**.

R7: Server x may utter **DontTell(**x**,**y**,**ϕ**)** at any time after the commencement of a dialog between x and y, only if the permission $\mathrm{perm}(y, \phi)$ is not in x's knowledge base.

R8: If Server x has uttered **DontTell(**x**,**y**,**ϕ**)**, then **Argue(**$\mathrm{perm}(y, \phi)$**,** NO**,** A**)** must be uttered (A is an argument indicating that y does not have the permission to access ϕ).

R9: Client y may only utter **Argue(**$\mathrm{perm}(y, \phi)$**,** $\mathrm{ADD}(x)$**,** A**)** (A is an argument supporting the fact that y should have the permission to access ϕ) if Server x has previously uttered **Argue(**$\mathrm{perm}(y, \phi)$**,** NO**,** B**)** (B is an argument justifying why y does not have the permission to access ϕ).

R10: **Argue(**$\mathrm{perm}(y, \phi)$**,** NO**,** A**)** (A is an argument justifying why y does not have the permission to access ϕ) may be uttered if **Argue(**$\mathrm{perm}(y, \phi)$**,** $\mathrm{ADD}(x)$**,** B**)** (B is an argument supporting the fact that y should have the permission to access ϕ) has been previously uttered.

R11: Server x may utter the locution **Accept(**x**,**A**,**$\mathrm{perm}(y, \phi)$**)** only if **Argue(**$\mathrm{perm}(y, \phi)$**,** $\mathrm{ADD}(x)$**,** A**)** has been previously uttered.

Effect rules

An effect rule concerns the knowledge base of a Server: if Server x has uttered **Accept(**x**,**A**,**$\mathrm{perm}(y, \phi)$**)**, then $\mathrm{perm}(y, \phi)$ is added to the knowledge base of x.

Outcome rules

Dialogs under this protocol may terminate or may not. In the latter case, neither participant leaves the dialogue, for example if they argue indefinitely. Termination occurs whenever one participant leaves the dialogue. Whether termination is considered successful or not depends on the goals of the respective participants, which may be different. From the perspective of the Client, *successful termination* of the dialog would occur if, prior to termination, the information requested by the Client has been provided by the Server. From the perspective of the Server, *successful termination* would occur if, prior to termination, the Client has received only that information from the Server for which it has the appropriate permissions.

3.3 LGL semantics

We give an LGL semantics to the dialog system described in the previous section.

Participants

One object is associated to the Client, another to the Server. The control state of the server contains attributes related to the permissions the Client has to access information. These attributes are of the form `perm(x,`ϕ`)` where `x` is the identifier of the process associated to the Client, and ϕ is an information the Client can access.

Context

The dialog takes place mediated through a tuple space. This tuple space is partitioned into three families of subspaces: the *requests-answers space* (subspace `ra`), dedicated to the information requests and answers; the *argumentation space* (subspace `arg`), dedicated to the arguments about permissions; and the *contents space* (subspace `co`), dedicated to information contents. One set of three subspaces is dedicated to each Client-Server pair. This division of the tuple space enables different participants to have differential access to the utterances in the dialogue. For instance, a government authority regulating medical practices could be given access to the requests and responses made by a Client-Server pair, and to the arguments justifying responses, without also having access to the actual contents of the information by the Server. The confidential exchange of the information itself may thus be secured.

Communication language

To each locution is associated a tuple. The first field of the tuple is the subspace in which the utterance of the locution is relevant (denoted by the term `sp(x)`, where `x` is either `ra`, or `arg`, or `co`). Among the other fields of the tuple may be the term `fr(x)` (resp. `to(x)`), which denotes that the locution is made by (resp. is directed to) the participant whose associated process has identifier `x`. The other fields of the tuple are close to the meaning of the locution to which they refer, so we leave their meaning implicit. Following each locution, we now state the associated tuple:

OpenDialogue(x**,**y**)**
`[sp(ra),fr(x),to(y),open]`
Ask(x**,**y**,**ϕ**)**
`[sp(ra),fr(x),to(y),ask(phi)]`
Tell(x**,**y**,**ϕ**)**
`[sp(ra),fr(x),to(y),tell(phi)]`
DontTell(x**,**y**,**ϕ**)**
`[sp(ra),fr(x),to(y),notell(phi)]`
EndDialogue(x**,**y**)**
`[sp(ra),fr(x),to(y),end]`
Argue($\mathrm{perm}(y, \phi)$**,** mode**,** A**)**
`[sp(arg),perm(y,phi),mode,a]`
Accept(x**,**A**,**$\mathrm{perm}(y, \phi)$**)**
`[sp(arg),perm(y,phi),accept(x,a)]`
Provide(x**,**y**,**$<$ content $\phi >$**)**
`[sp(co),fr(x),to(y),content(phi)]`

When a participant makes an utterance, a tuple corresponding to the locution uttered is inserted in the tuple space, and if the utterance is directed to a particular participant, then at least this participant reads the utterance. Note that when **Provide**(x,y,$<$ content$\phi >$ is uttered, then the participant y reads and then deletes the corresponding tuple of the tuple space (y makes an `in` operation). Note also that when **Tell**(x,y,ϕ) is uttered by a participant x, then if x had previously uttered **DontTell**(x,y,ϕ), x deletes the tuples corresponding to this last utterance before inserting the tuple corresponding to **Tell**(x,y,ϕ); this ensures that contradictory messages are not kept in the requests-answers space. Finally, the utterance **Accept**(x,A,perm(y, ϕ)) has the effect of inserting the permission `perm(y,`ϕ`)` into the control state of the participant x which has uttered it. Following each utterance, we now state its associated effects:

OpenDialogue(x,y)
```
Creates 3 subspaces dedicated to (x,y)
x:  out([sp(ra),fr(x),to(y),open])
y:  rd([sp(ra),fr(x),to(y),open])
```
Ask(x,y,ϕ)
```
x:  out([sp(ra),fr(x),to(y),ask(phi)])
y:  rd([sp(ra),fr(x),to(y),ask(phi)])
```
Tell(x,y,ϕ)
```
x:  in([sp(ra),fr(x),to(y),notell(phi)])
x:  out([sp(ra),fr(x),to(y),tell(phi)])
y:  rd([sp(ra),fr(x),to(y),tell(phi)])
```
DontTell(x,y,ϕ)
```
x:  out([sp(ra),fr(x),to(y),notell(phi)])
y:  rd([sp(ra),fr(x),to(y),notell(phi)])
```
Argue(perm(y, ϕ)**,** mode**,** A**)**
```
x:  out([sp(arg),perm(y,phi),mode,a])
y:  rd([sp(arg),perm(y,phi),mode,a])
```
Accept(x,A,perm(y, ϕ))
```
x:  out([sp(arg),perm(y,phi),accept(x,a)])
x:  +perm(y,phi)
y:  rd([sp(arg),perm(y,phi),accept(x,a)])
```
Provide(x,y,$<$ content $\phi >$)
```
x:  out([sp(co),fr(x),to(y),content(phi)])
y:  in([sp(co),fr(x),to(y),content(phi)])
```
EndDialogue(x,y)
```
x:  out([sp(ra),fr(x),to(y),end])
y:  rd([sp(ra),fr(x),to(y),end])
```

Protocol

The law contains the rules of the dialog protocol.

Effect rules

The effect rule of the protocol has been integrated in the effects on the tuple space of the utterance **Accept**.

3.4 Illustration

We now present the example dialog of Section 3.1 in terms of the dialog protocol syntax of Section 3.2 and its associated semantics given in Section 3.3.

The London agent is the Server (denoted by L in utterances, by `l` in Linda commands), the Brussels agent is the Client (denoted by B in utterances, by `b` in Linda commands). The Brussels agent requests a piece of information (denoted by info in utterances, by `i` in Linda commands) concerning the health of the patient Robert. This piece of information is a data record containing a sensitive part and a non-sensitive part; the non-sensitive part of the information is denoted by info.nonsens in utterances, by `i.ns` in Linda commands, its sensitive part is denoted by info.sens in utterances, by `i.se` in Linda commands. The control state of the London agent contains the permission that the Brussels agent has to access the non-sensitive part of the information (`perm(b,i.ns)`).

We rewrite the example in order to highlight the utterances (in bold font) and their effect on the tuple space (in Courier font). The agent who makes an utterance or operation is indicated just before the utterance or operation. The rule according to which the utterance is legal appears into bold brackets.

Brussels agent:

I would like to dialog with the the agent of Robert's London Doctor.
B: **OpenDialogue**(B, L) **(R1)**
```
Creates 3 subspaces dedicated to (b,h)
b:  out([sp(ra),fr(b),to(h),open])
h:  rd([sp(ra),fr(b),to(h),open])
```

London agent

Yes, I accept to dialog with you.
L: **OpenDialogue**(L,B) **(R1)**
```
h:  out([sp(ra),fr(h),to(b),open])
b:  rd([sp(ra),fr(h),to(b),open])
```

Brussels agent

I request Robert's health record.
B: **Ask**(B,L,info) **(R3)**
```
b:  out([sp(ra),fr(b),to(h),ask(i)])
h:  rd([sp(ra),fr(b),to(h),ask(i)])
```

London agent

I can provide you non-sensitive information because you are allowed to access it (argument `a1`), but, because Robert has only given to his London doctor limited consent to passing on his personal information (argument `a2`), I cannot provide you sensitive information.
L: **Tell**(L,B,info.nonsens) **(R4)**
```
h:  out([sp(ra),fr(h),to(b),tell(i.ns)])
b:  rd([sp(ra),fr(h),to(b),tell(i.ns)])
```
L: **Argue**(perm(B,info.nonsens)**,** YES**,** $a1$**)** **(R6)**
```
h:  out([sp(arg),perm(b,i.ns),yes,a1])
b:  rd([sp(arg),perm(b,i.ns),yes,a1])
```
L: **Provide**(L,B,<content info.nonsens>) **(R5)**
```
h:  out([sp(co),fr(x),to(y),content(i.ns)])
b:  in([sp(co),fr(x),to(y),content(i.ns)])
```
L: **DontTell**(L,B,info.sens) **(R7)**
```
h:  out([sp(ra),fr(h),to(b),notell(i.se)])
b:  rd([sp(ra),fr(h),to(b),notell(i.se)])
```
L: **Argue**(perm(B,info.sens)**,** NO**,** $a2$**)** **(R8)**
```
h:  out([sp(arg),perm(b,i.se),no,a2])
b:  rd([sp(arg),perm(b,i.se),no,a2])
```

Brussels agent

The missing information could possibly include information that could affect the treatment of Robert's heart failure. I request the sensitive information, Robert's life may be at stake (argument `a3`)!
B: **Ask**(B,L,info.sens) **(R3)**
```
b:  out([sp(ra),fr(b),to(h),ask(i.se)])
h:  rd([sp(ra),fr(b),to(h),ask(i.se)])
```
B: **Argue**(perm(B,info.sens)**,** ADD(L)**,** $a3$**)** **(R9)**
```
b:  out([sp(arg),perm(b,i.se),add(h),a3])
h:  rd([sp(arg),perm(b,i.se),add(h),a3])
```

London agent

I cannot divulge the sensitive information, because British law prohibits passing on information without the owner's consent (argument a4).

L: **DontTell**(L,B,info.sens) **(R7)**

```
h:  out([sp(ra),fr(h),to(b),notell(i.se)])
b:  rd([sp(ra),fr(h),to(b),notell(i.se)])
```

L: **Argue**(perm(B,info.sens), NO, *a*4) **(R8)**

```
h:  out([sp(arg),perm(b,i.se),no,a4])
b:  rd([sp(arg),perm(b,i.se),no,a4])
```

Brussels agent

EC law takes precedence over member state law when it would be in the interests of the owner to divulge the information (argument a5). You should allow me to access the sensitive information.

B: **Argue**(perm(B,info.sens), ADD(L), *a*5) **(R9)**

```
b:  out([sp(arg),perm(b,i.se),add(h),a5])
h:  rd([sp(arg),perm(b,i.se),add(h),a5])
```

London agent

Only Robert could decide what would be in his interests (argument a6).

L: **Argue**(perm(B,info.sens), NO, *a*6) **(R10)**

```
h:  out([sp(arg),perm(b,i.se),no,a6])
b:  rd([sp(arg),perm(b,i.se),no,a6])
```

Brussels agent

Robert's doctor owes a duty of care to Robert and, should he die, the doctor might be sued by his family, or the Brussels hospital, or both (argument a7).

B: **Argue**(perm(B), ADD(L), *a*7) **(R9)**

```
b:  out([sp(arg),perm(b,i.se),add(h),a7])
h:  rd([sp(arg),perm(b,i.se),add(h),a7])
```

London agent

I yield to this argument. Now you can access the sensitive information (argument a8), I provide it to you.

L: **Accept**(L,*a*7,perm(B,info.sens)) **(R11)**

```
x:  out([sp(arg),perm(b,i.se),accept(h,a7)])
x:  +perm(b,i.se)
y:  rd([sp(arg),perm(b,i.se),accept(h,a7)])
```

L: **Tell**(L,B,info.sens) **(R4)**

```
h:  in([sp(ra),fr(h),to(b),notell(i.se)])
h:  out([sp(ra),fr(h),to(b),tell(i.se)])
b:  rd([sp(ra),fr(h),to(b),tell(i.se)])
```

L: **Argue**(perm(B,info.sens), YES, *a*8) **(R6)**

```
h:  out([sp(arg),perm(b,i.se),yes,a8])
b:  rd([sp(arg),perm(b,i.se),yes,a8])
```

L: **Provide**(L,B,<content info.sens>) **(R5)**

```
h:  out([sp(co),fr(h),to(b),content(i.se)])
b:  in([sp(co),fr(h),to(b),content(i.se)])
```

Brussels agent

Thank you.

B: **EndDialogue**(B,L) **(R2)**

```
b:  out([sp(ra),fr(b),to(h),end])
h:  rd([sp(ra),fr(b),to(h),end])
```

The dialog then terminates.

4. IMPLEMENTATION

In Section 1, we stated that our primary objective was the development of a semantics for these Information-seeking dialogs which facilitated implementation of the protocol. In order to assess whether the protocol and semantics of Section 3 met this objective, we undertook an implementation. For this purpose we used the *TuCSoN* software platform for tuple center applications [5], developed at the Alma Mater Studiorum – Università di Bologna, in Cesena, Italy.[3] As mentioned in Section 2.2, the associated programming language, *ReSpecT*, is equivalent to LGL in expressive power. Because the implementation was not intended for production use, only the protocol itself (and the supporting tuple space semantic framework) was implemented, and we did not create agents capable of interacting via the protocol. The selection of legally-possible locutions at each step in a dialog and the creation of content for these locutions, was left to human participants. The implementation was undertaken on a standard desktop PC running linux, with simulation of the client (requesting access to some information) and server (controlling access to that information) enabled through *TuCSoN*.

The key outcome of this exercise was how readily the protocol was successfully implemented.[4] Our prior experience with developing a multi-agent co-ordination application using *TuCSoN* meant that no learning of the platform was required. The ease of implementation compares very favorably with another recent experience of implementation of dialog protocols. Atkinson and colleagues [1] implemented a persuasion dialog protocol in Java from an axiomatic semantics and required significantly more effort. This could have been due to the greater complexity of Atkinson's Persuasion protocol compared with our Information-seeking protocol, and/or it may have been due to the absence of a tuple spaces architecture and associated platform in her implementation. Atkinson also reported that her implementation led to the identification of gaps and errors in the specification of the protocol, something we did not experience.

One issue that did arise in the implementation concerned the partitioned nature of the tuple space. The semantic framework described in Section 3.3 partitions the tuple space for a dialog into three subspaces for each client-server pair, with potentially differential access for different clients to these subspaces. During the implementation, it was found that the current version of *TuCSoN* does not permit a space to be partitoned in this way explicitly, and so we developed a virtual, on-the-fly, partitioning of the tuple center. This was achieved through the use of client identifiers in the names of output tuples created in response to successful requests for information. In this way, clients without the appropriate identifier would not be able to read the particular tuples, thus maintaining information security.

5. RELATED WORK AND CONCLUSIONS

As mentioned, Hamblin presented a possible-worlds semantics for question-response interactions in [11], and his subsequent publications on this topic, especially [12], have been influential in linguistics. Within computer science, de Boer and colleagues [4] present operational and failure semantics for agent dialogs involving *ask* and *tell* locutions, where these utterances are understood as exchanging constraints on variable values, rather than simply variable values themselves. The failure semantics uses failure sets, defined in terms of information irrelevant to the question or answer of an agent, to provide a semantic account of deadlock behaviors in a request-response interaction. This work does not consider permissions explicitly, nor arguments over the permissions. In contrast, Bench-Capon [2] provides an axiomatic semantics for requests and responses with permissions, but does not consider argument.

[3] Available from: `http://lia.deis.unibo.it/research/tucson/`

[4] The implementation is available from: `www.csc.liv.ac.uk/research/techreports/techreports.html`

Several authors have considered or utilized tuple space models as semantic frameworks for agent dialog protocols. For example, McGinnis and colleagues [17] briefly mention an implementation of an agent protocol using *Linda*. McBurney and Parsons [15] compare unpartitioned tuple space models to partitioned commitment stores as semantic frameworks for negotiation dialogs. In the most extensive treatment, Vasconcelos and colleagues [20] created a simulation platform for multi-agent electronic institutions using a tuple spaces architecture, with agents communicating with one another via the shared tuple space. The rules of the e-institution and any communications protocols are enforced by specific administrative agents, the effect of which is similar to LGL. Although the e-institution approach is quite general, the example studied is a negotiation, not an information-seeking dialog. In addition, the tuple space is not partitioned.

Our contribution in this paper is a novel semantics for information-seeking agent communications protocols involving permissions and arguments, in which utterances under the protocol are translated into commands in *Law-Governed Linda* and, through them, into actions on certain associated tuple spaces. Although created with a specific protocol in mind, the semantics could readily be applied to other agent protocols. Translation of agent dialog utterances into the commands of a programming language may be seen as merely mapping one syntax into another. However, the LGL programming language commands are understood in terms of their effects on actual shared memory stores (tuple spaces), and so this translation may be viewed as a semantic mapping — from agent utterances under a dialog protocol to actions in the real world of memory stores. As mentioned in the Introduction, we believe that the popularity and simplicity of tuple space and tuple center models, such as *Javaspaces* and *TuCSoN*, means that our approach to semantics will facilitate implementation of agent dialog protocols; our own implementation has demonstrated the ease with which a protocol may be implemented. The semantic framework we have presented here therefore provides a connecting bridge between the formal specification of a protocol, usually involving an axiomatic semantics, and its software implementation. In future work, we intend to consider the connections with recent research abstracting from tuple centers to co-ordination artifacts in general [21].[5]

6. REFERENCES

[1] K. Atkinson, T. Bench-Capon, and P. McBurney. Implementation of a dialogue game for persuasion over action. Technical Report ULCS-04-005, Department of Computer Science, University of Liverpool, UK, 2004.

[2] T. J. M. Bench-Capon. Specifying the interaction between information sources. In G. Quirchmayr et al., editors, *DEXA 1998*, LNCS 1460, pages 425–434, Berlin, 1998. Springer.

[3] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[4] F. S. de Boer, R. M. van Eijk, W. van der Hoek, and J.-J. C. Meyer. A fully abstract model for the exchange of information in multi-agent systems. *Theoretical Computer Science*, 290(3):1753–1773, 2003.

[5] DEIS. *TuCSoN Guide: TuCSoN Version 1.4.0*. DEIS, Alma Mater Studiorum – Università di Bologna, Italy, revision 003 edition, 2002. Last changes 16.12.2004.

[6] E. Denti, A. Natali, and A. Omicini. On the expressive power of a language for programming coordination media. In *1998 ACM Symposium on Applied Computing (SAC'98)*, pages 169–177, Atlanta, GA, USA, 1998. ACM.

[7] S. Doutre, P. McBurney, and M. Wooldridge. Law-governed linda as a semantics for agent interaction protocols. In F. Dignum et al., editors, *Proc. AAMAS 2005*, pages 1257–1258, New York, USA, 2005. ACM Press.

[8] R. Eijk. *Programming Languages for Agent Communications*. PhD thesis, Department of Computer Science, Utrecht University, The Netherlands, 2000.

[9] FIPA. Communicative Act Library Specification. Standard SC00037J, Foundation for Intelligent Physical Agents, 2002.

[10] D. Gelernter. Generative communication in Linda. *ACM Trans. Programming Lang. & Systems*, 7(1):80–112, 1985.

[11] C. L. Hamblin. *Language and the Theory of Information*. Ph.D. thesis, University of London, UK, 1957.

[12] C. L. Hamblin. Questions in Montague English. *Foundations of Language*, 10:41–53, 1973.

[13] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game-based protocols: new trends in agent communications languages. *Knowledge Engineering Review*, 17(2):157–179, 2002.

[14] P. McBurney, R. Eijk, S. Parsons, and L. Amgoud. A dialogue-game protocol for agent purchase negotiations. *J. Auton. Agents & Multi-Agent Systems*, 7(3):235–273, 2003.

[15] P. McBurney and S. Parsons. Posit spaces: a performative theory of e-commerce. In J. S. Rosenschein et al., editors, *AAMAS 2003*, pages 624–631, New York, 2003. ACM Press.

[16] P. McBurney and S. Parsons. A denotational semantics for deliberation dialogues. In N. R. Jennings et al., editors, *AAMAS 2004*, pages 86–93, New York, 2004. ACM Press.

[17] J. P. McGinnis, D. Robertson, and C. Walton. Using distributed protocols as an implementation of dialogue games. In M. d'Inverno et al., editors, *EUMAS 2003*, 2003.

[18] N. H. Minsky and J. Leichter. Law-governed Linda as a coordination model. In P. Ciancarini et al., editors, *Object-based Models and Languages for Concurrent Systems*, LNCS 924, pages 125–146. Springer, Berlin, 1995.

[19] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, 2001.

[20] W. Vasconcelos, D. Robertson, C. Sierra, M. Esteva, J. Sabater, and M. Wooldridge. Rapid prototyping of large multi-agent systems through logic programming. *Annals of Mathematics and AI*, 41(2–4):135–169, 2004.

[21] M. Viroli and A. Ricci. Instructions-based semantics of agent-mediated interaction. In N. R. Jennings et al., editors, *AAMAS 2004*, pages 102–109, New York, 2004. ACM Press.

[22] D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press, Albany, NY, USA, 1995.

[23] M. J. Wooldridge. Semantic issues in the verification of agent communication languages. *J. Auton. Agents and Multi-Agent Systems*, 3(1):9–31, 2000.

[5]The first three authors are grateful for financial support received from the EC, through project ASPIC (IST-FP6-002307). We are also grateful to technical assistance received from the *TuCSoN* team at the Alma Mater Studiorum – Università di Bologna. A preliminary version of this work was presented in [7].