T-LAIMA: Answer Set Programming for Modelling Agents with Trust

Marina De Vos[a] Owen Cliffe[a] Richard Watson[b] Tom Crick[a]
Julian Padget[a] Jonathan Needham[a] Martin Brain[a]

[a] *Department of Computer Science*
University of Bath
Bath BA2 7AY, UK
Telephone: +44 1225 38 5053
Fax:+ 44 1225 38 3493
Email: {mdv,occ,tc,jap,jmn20,mjb}@cs.bath.ac.uk
[b]*Department of Computer Science*
Texas Tech University
Lubbock Texas, 79409, US
Telephone: +1 806 742 3527
Fax: +1 806 742 3519
Email: rwatson@cs.ttu.edu

Abstract

In a time where multi-agent systems (MAS) become increasingly more popular, they come in many forms and shapes depending on the requirements of the agents that need to populate them. Amongst the more demanding properties with respect to the design and implementation is how these agents may individually reason and communicate about their knowledge and beliefs, with a view to cooperation and collaboration. With information coming from various sources, it becomes vital for agents to have an idea how reliable these information providers are, especially if they start to contradict each other. In this paper we present a hybrid multi-agent platform, called T-LAIMA, using an extension of answer set programming (ASP). We show that our framework is capable of dealing with the specification and implementation of the system's architecture, communication and the individual reasoning capacities of the agents. We discuss both the theoretical framework which models a single, fixed encounter between a number of agents and the implemenation that sets ups these encounters in a open multi-agent domain.

1 Introduction

The world of multi-agent systems is commenly said to be populated with four sorts of agents [25]: deductive reasoning, practical, reactive and hybrid agents. In all of these, the problem of translating and representing the real world in an accurate and adequate symbolic description are still largely unsolved (although there has been some encouraging work in the area, for example, the OpenCyc project[21]).

Answer set programming (ASP) [1] is a formal, logical language designed for declarative problem solving, knowledge representation and real world reasoning. It represents a modern approach to logic programming based on analysis of which constructs are needed in reasoning applications rather than implementing subsets of classical or other abstract logical systems. ASP has a formal background based on first order and epistemic logic, yet does not work entirely as a classical theorem prover. Its semantics is based on models to describe the solutions of the encoded problem, being it a planning, decision or constraint problem. These models are based on a reactive mechanism of satifying rules in the problem.

In this paper, we present a formalism for hybrid multi-agent systems in which the agents use answer set programming for representing their reasoning capabilities. Such systems are useful for modelling decisionproblems; not just the solutions of the problem at hand but also the evolution of the beliefs of and the interactions between the agents can be characterised.

A system of logic programming agents consists of a set of agents connected by means of uni-directional communication channels. To model a single agent's reasoning, we use Ordered Choice Logic Programs [3], an extension of answer set programming that provides for explicit representation of preferences between rules and dynamic choice between alternatives in a decision. Agents use information received from other agents as a guidance for their reasoning. Each agent assigns a trust level to each of agent it could receive information from. Information from more trusted agents will be preferred if conflict arises. It is possible that an agent receiving information might prefer the incoming information over that she holds herself. This allows to model a whole range of relationships between the various agents: e.g. student-teacher.

The knowledge an agent possesses is captured by its answer set semantics. Part of this knowledge is shared with agents listening on the outgoing channels. This is regulated by filters that for each connecting agent decide what sort of information this agent is allowed to receive. The semantics of the whole system corresponds to a stable situation where no agent, with stable input, needs to change its output.

Note that, within this paper, each example should be viewed as a single interaction between agents. Although we don't discuss it here, between such interactions the agents may update trust values, make or break connections with other agents, etc. We view this a single step within an "agent loop". Within such a loop the agent may also choose goals to try and satisfy, plan to achieve such goals, make observations, perform diagnosis, execute actions, etc. There have been a number of papers detailing such an approach for the modeling of agents using answer set programming. A general overview can be found in [14].

Our implementation uses the JADE platform[15] to provide basic communications and agent services, an ontology developed in Protégé[22] and OCT [3, 6] to model, maintain and reason about the agent's beliefs and knowledge.

2 Why Answer Set Programming?

One component of agents 'intelligent' behaviour is the ability to reason: perform logical inference, handle combinatorial constraints and being able to handle complex, logical queries over a large search domain. These actions are simple to express and implement in declarative logic programming formalisms. By using these tools, the developer of an agent system can focus on the reasoning and 'thought' processes of the agents rather than being bogged down in the detail how to implement them. Using existing logical formalisms rather than an ad-hoc systems also brings a greater degree of robustness and certainty to the agent's reasoning, i.e. because it is possible or easier to prove and verify the behaviour of participating agents. Finally, the availability of a number of powerful and mature implementations contributes to reduced development time.

The question then becomes one of "Which logical formalism?". Due to its power, expressiveness, and the availability of fast inference mechanisms, we use ASP. A detailed discussion of the benefits of ASP over other formalisms can be found in Section 1.1 of [1].

An important aspect of ASP is its treatment of negation. The semantics of ASP naturally gives rise to two different methods of calculating negation, negation as failure and constraint based negation. Negation as failure, (i.e. we cannot prove p to be true) is characterised as epistemic negation (i.e. we do not know p to be true). For a full discussion of such issues, see [1]. Constraint-based negation introduces constraints that prevent certain combinations of atoms from being simultaneously true in any answer set. This is characterised as classical negation as it is possible to prevent a and $\neg a$ both being simultaneously true, a requisite condition for capturing classical negation. This is a significant advantage in some reasoning tasks as it allows reasoning about incomplete information. More importantly, the "closed world assumption" (inherent in Prolog) is not present in ASP.

In the extension of ASP that we propose to use for modelling agents, we only allow implicit negation coming from the modelling decisions (i.e. you have decide upon exactly one alternative when forced to make a choice). However, as described in [8], it is perfectly possible to embed any form of negation (classical negation or negation as failure) using the order mechanism. In other words, the ordering of rules replaces use of negation, making OCLP a suitable tool to model the exact type of negation one wishes to use. In terms of complexity, having both concepts (negations and order) is no different to having just one.

The semantics of ASP also clearly give rise to multiple possible world views - the exact formal declarative meaning of answer sets is still under debate[9] - in which the program is consistent. The number and composition of these varies with the program. The benefit of this will become apparent in the examples given later in the paper. Attempting to model the same ideas in Prolog can lead to confusion as the multiple possible views may manifest themselves differently, depending on the query being asked. In ASP terms, Prolog would answer a query on a as true if there is at least one answer set in which a is true. However, there is no notion of 'in which answer set is this true'. Thus, a subsequent query on b might also return true, but without another query it would not be possible to infer if a and b could be simultaneously true.

As was briefly mentioned in introduction, ASP has been used to model a number of agent tasks (such as planning, diagnosis, and learning). One of the benefits of the ASP approach was that the agents model of its domain as well as the modules for planning, diagnosis, etc, were all written in a single language, ASP. Moreover, the domain model is general purpose. Planning is done by adding a small, domain independent, planning module to the domain model. For diagnosis one simply replaces the planning module with one for diagnosis. Many other approaches use a different language to model the domain than the one they use for the desired task, such as planning. It is also uncommon for the domain model to be general purpose - a domain model used for planning could not also be used for diagnosis. ASP gives us a flexible language that can be used for all of an agent's tasks.

3 Ordered Choice Logic Programming

OCLP ([3]) was developed as an extension of ASP to reason about decisions and preferences. This formalism allows programmers to explicitly express decisions (in the form of exclusive choices between multiple alternatives) and situation dependent preferences.
We explain the basics of OCLP by means of a simplified grid-computing situation. Full details can be found in [3].

***Example* 1** *Suppose a grid-computing agent capable of performing two tasks which are mutually exclusive because of available resources. Your agent has permissions to use three clusters ($cluster_1$,$cluster_2$, $cluster_3$). Because of prices, computing power and reachability, the agent prefers $cluster_3$ over $cluster_2$ and $cluster_2$ over $cluster_1$. However, in order to perform task two, she needs access to both a database and a mainframe. The former can only be provided by $cluster_1$, while the latter is only available from $cluster_2$. The above information leads to two correct behaviours of our agent:*

- *performing task one, the agent uses $cluster_3$; and*
- *executing task two, the agent requires access to clusters 1 and 2 in order to use the required database and mainframe.*

To model such behaviour, we represent the agent as an answer set program capable of representing preference-based choices between various alternatives. The preference is established between components, groups of rules. Components are linked to each other by means of strict order denoting the preference relation between them. Information flows from less specific components to the more preferred ones until a conflict among alternatives arises, in which case the most specific one will be favoured. Alternatives are modelled by means of choice rules (rules that imply exclusive disjunction). In other words, an OCLP P is a pair $\langle \mathcal{C}, \prec \rangle$ where $\mathcal{C}$ is a collection components, containing a finite number of (choice) rules, and strict order relation $\prec$ on $\mathcal{C}$. We will use $\preccurlyeq$ to denote the reflexive closure of $\prec$. For the examples, we represent an OCLP as a directed acyclic graph (DAG), in which the nodes are the components and the arcs represent the relation "$\prec$". For two components C_1 and C_2, $C_1 \prec C_2$ is represented by an arrow from going from C_1 to C_2, indicating that information from C_1 takes precedence over C_2.

***Example* 2** *The agent mentioned in Example 1 can be easily modelled using an OCLP, as shown in Figure 1. The rules in components P_1, P_2 and P_3 express the preferences between the clusters when only computing power is required. The rules in P_4 indicate the grid computing problem and the required choice between the possible alternatives. In component P_5, the first rule states the goals of the agent in terms of the tasks. The third and the fourth rules specify the resources, apart from computing power, needed for a certain tasks. The last two rules express the availability of the extra resources in a cluster.*

The semantics of an OCLP is determined by means of *interpretations*, i.e. sets of atoms that are assumed to be true (with atoms not in the interpretation being unknown).

Given an interpretation, we call a rule $A \leftarrow B$ *applicable* when the precondition B, the *body*, is true (all the elements in the body are part of the interpretation). A rule is *applied* when it is applicable and when the consequence A, called the *head*, contains exactly one true atom. The latter condition is reasonable as rules with more than one element in their head represent decisions where only one alternative may be chosen.

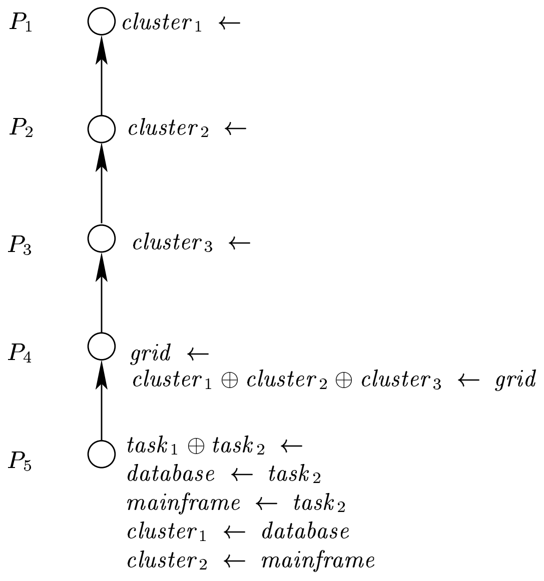

P_1
$cluster_1 \leftarrow$
P_2
$cluster_2 \leftarrow$
P_3
$cluster_3 \leftarrow$
P_4
$grid \leftarrow$
$cluster_1 \oplus cluster_2 \oplus cluster_3 \leftarrow grid$
P_5
$task_1 \oplus task_2 \leftarrow$
$database \leftarrow task_2$
$mainframe \leftarrow task_2$
$cluster_1 \leftarrow database$
$cluster_2 \leftarrow mainframe$

Figure 1: The GridComputing Agent from Example 2

Using interpretations we can reason about the different alternatives. Two atoms are considered to be alternatives with respect to an interpretation if and only if a choice between them is forced, i.e. there exists a more specific and applicable choice rule with a head containing (at least) the two atoms. So given an atom a in a component C, we can define the *alternatives* of a in that component C with respect to an interpretation I, written as $\Omega^I_C(a)$, as those atoms that appear together with a in the head of a more specific applicable choice rule.

***Example* 3** *Reconsider Example 2. Let I and J be the following interpretations:*

- $I = \{cluster_2, task_1\}$ *and* $J = \{grid, cluster_1, task_1\}$.

The alternatives for $cluster_2$ in P_2 w.r.t. J are $\Omega^J_{P_2}(cluster_2) = \{cluster_1, cluster_2\}$. W.r.t. I, we obtain $\Omega^I_{P_2}(cluster_2) = \emptyset$, since the choice rule in P_4 is not applicable. When we take P_5 instead of P_2, we obtain w.r.t. J: $\Omega^J_{P_5}(cluster_2) = \emptyset$, since the applicable choice rule is in a less preferred component which makes it irrelevant to the decision process in the current component.

Unfortunately, interpretations are too general to convey the intended meaning of a program, as they do not take the information in the rules into account. Therefore, models are introduced. The model semantics for choice logic programs, the language used in the components [7], and for ASP is fairly simple: an interpretation is a model if and only if every rule is either not applicable (i.e. the body is false) or applied (i.e. the body is true and the head contains exactly one head atom). For OCLP, taking individual rules into account is not sufficient: the semantics must also consider the preference relation. In cases where two or more alternatives of a decision are triggered, the semantics requires a mechanism to deal with it. When the considered alternatives are all more specific than the decision itself, the decision should be ignored. When this is not the case, the most specific one should be decided upon. If this is impossible, because they are unrelated or equally specific, an arbitrary choice is justified. This selection mechanism is referred to as defeating.

***Example* 4** *Reconsider Example 2. The rule $cluster_1 \leftarrow$ is defeated w.r.t. interpretation $I = \{cluster_2, grid\}$ because of the applicable rule $cluster_2 \leftarrow$.*
W.r.t. interpretation $J = \{cluster_1, cluster_2, grid, database, mainframe\}$ we have that the rule $cluster_1 \oplus cluster_2 \oplus cluster_3 \leftarrow grid$ is defeated by the combination of the rules $cluster_1 \leftarrow database$ and $cluster_2 \leftarrow mainframe$.

So we can define a *model* for an OCLP as an interpretation that leaves every rule either not applicable, applied or defeated. Unfortunately, in order to obtain the true meaning of a program, the model semantics tends to be too crude.

For traditional ASP, the Gelfond-Lifschitz transformation or reduct [13] was introduced to remove models containing unsupported assumptions. Interpretations that are the minimal model of their Geldond-Lifschitz transformation are called answer sets, as they represent the true meaning of the program or alternatively the answer to the problem encoded by the program. Therefore, algorithms and their implementations for obtaining the answer sets of a program are often referred to as answer set solvers.

For OCLP, a reduct transformation to obtain the answer sets of our programs is also required. The transformed logic program can be obtained by adding together all components of the OCLP. Then, all defeated rules are taken away, together with all false head atoms of real choice rule (i.e. more than one head atom). The remaining rules with multiple head atoms are transformed into constraints, assuring that only one of them can become true whenever the body is satisfied. The relationship between those two definitions of reduct clearly convey that order and negation are interchangeable and explains why negation can be easily embedded in OCLP [8].

***Example* 5** *Reconsider the OCLP from Example 2. This program has two answer sets:*
- $M_1 = \{grid, cluster_1, task_2\ database, mainframe, cluster_2\}$ *and*
- $M_2 = \{cluster_3, grid, task_1\}$,

which matches our agent's intended behaviour.

In [3] it was shown that a bi-directional polynomial mapping exists between ordered choice logic programs and extended logic programs with respect to their answer set semantics. The mapping from OCLPs to normal logic programs is possible by introducing two new atoms for each rule in order to indicate that a rule is applied and defeated. For each rule in the OCLP, we generate a set of rules (the number of rules is equal to the number of head atoms) in the logic program that become applicable when the original rule is made applied. A rule is also created for each possible situation in which the original rule could become defeated. Finally add one rule that should be made applicable when the original rule is applied and not defeated. Constraints to make sure that the head elements cannot be true at the same time is are not necessary.

The reverse mapping is established by creating an OCLP with three components placed in a linear formation. The least preferred one establishes negation as failure. The middle component contains all the rules from the original program. The most preferred makes sure that for each pair $a, \mathrm{not}\ a$ only one can be true at any time, without given a a chance to be true without reason in the middle component

These polynomial mappings demonstrate that the complexity of both systems is identical (more information on the complexity aspects can be found in [1]). Having a polynomial mapping to a traditional logic program makes it possible implement a front-end to one of the existing answer set solvers like Smodels ([19]) or DLV ([11]). OCT[3] is such a front-end. This will be used in our agents to compute the knowledge of individual agents and the beliefs they hold about the knowledge of the others.

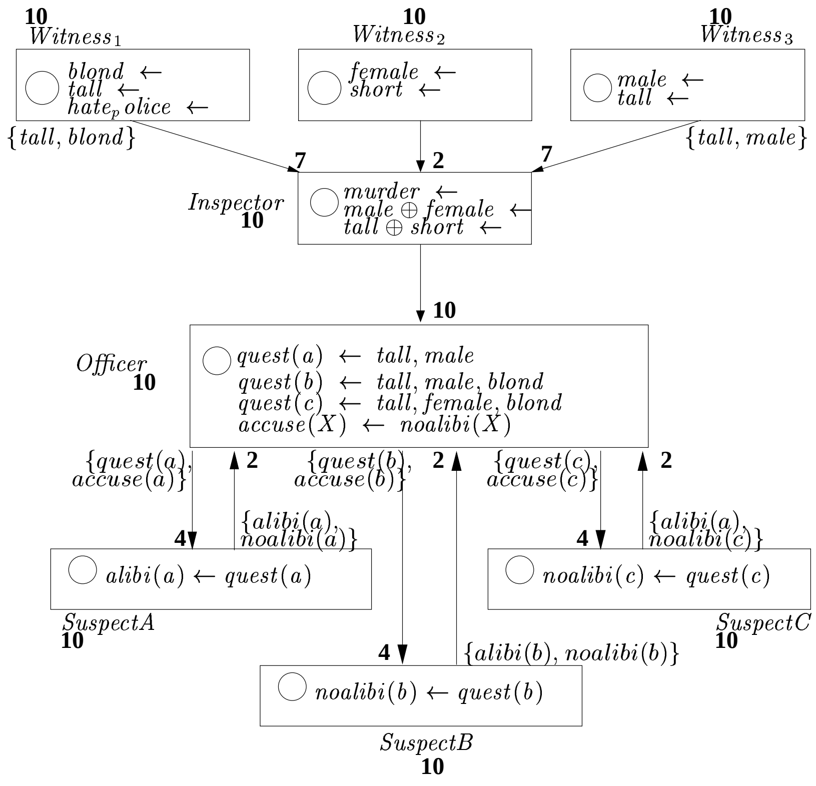
4 The T-LAIMA System

We assume that agents are fully aware of the agents they can communicate with, i.e. the communication structure is fixed, and that they can communicate by means of passing sets of atoms over uni-directional channels.

A *T-LAIMA* system is a triple $\langle \mathcal{A}, \mathcal{C}, \mathcal{L} \rangle$ containing a set of $\mathcal{A}$ of agents, a set $\mathcal{L}$ of atoms representing the language of the system and a relation $\mathcal{C}$ as a subset of $\mathcal{A} \times \mathcal{A} \times \mathcal{L} \times \mathbb{N}$ representing the communication channels between the agents, the filter they use when passing information and the trust the second agent gives to the first. The filter tells the listening agent which sort of information they can expect, if any. Since this is public information, we have opted for mentioning the information that could be passed in favour of the information that is kept secret. Furthermore, with each agent a we associate an OCLP F_a and the level of trust T_a it places on itself.

In examples we use the more intuitive representation of a graph. The set S is formed by all the atoms appearing in the OCLP associated with the agents. The filter is shown next to the arrow when leaving the transmitting agent. In order not to clutter the image, we assume that if no filter is present the agent could potentially transmit every atom of S. The trust levels are near the arrow for trust places on other agents and next to the name for the personal one.

***Example* 6** *The system in Figure 2 displays a multi-agent system where eight agents "cooperate" to solve a murder case. Three witnesses, agents $Witness_1$ to $Witness_3$ provide information to the $Inspector$ agent is called to a scene to establish a murder took place. The $Inspector$ agent has a lot less faith in $Witness_2$*



10
$Witness_1$
$blond \leftarrow$
$tall \leftarrow$
$hate_p olice \leftarrow$
$\{tall, blond\}$
10
$Witness_2$
$female \leftarrow$
$short \leftarrow$
10
$Witness_3$
$male \leftarrow$
$tall \leftarrow$
$\{tall, male\}$
7
2
7
$Inspector$
10
$murder \leftarrow$
$male \oplus female \leftarrow$
$tall \oplus short \leftarrow$
10
$Officer$
10
$quest(a) \leftarrow tall, male$
$quest(b) \leftarrow tall, male, blond$
$quest(c) \leftarrow tall, female, blond$
$accuse(X) \leftarrow noalibi(X)$
$\{quest(a), accuse(a)\}$
2
$\{quest(b), accuse(b)\}$
2
$\{quest(c), accuse(c)\}$
2
$\{alibi(a), noalibi(a)\}$
$\{alibi(a), noalibi(c)\}$
4
4
$alibi(a) \leftarrow quest(a)$
$noalibi(c) \leftarrow quest(c)$
$SuspectA$
10
$SuspectC$
10
4
$\{alibi(b), noalibi(b)\}$
$noalibi(b) \leftarrow quest(b)$
$SuspectB$
10

Figure 2: The Murder T-LAIMA System of Example 6

than in the two others. All witnesses will only pass on information relevant to the case. Information from the witnesses is passed to the $Officer$ agent for further processing. Depending on information provided she can decide to question the three suspect agents. If questioned and when no alibi can be provided, the Officer can accuse the suspect. Of course, the Officer will only pass on selective information about the case to the Suspects. The suspects from the side have no intention to tell more than if they have an alibi or not. In this context, all trust levels are based on a scale of 10 and each agent is more confident about her knowledge/beliefs than those provide by other agents.

For OCLP we defined the notion of interpretation to give a meaning or truth value to all the atoms in our program. For our T-LAIMA systems we have a number of agents that can hold sets of beliefs which do not necessarily have to be the same. It is perfectly acceptable for two agents (e.g. humans) to agree to disagree. To allow this in our system we need *interpretations* to be functions taking an agent as input.

***Example* 7** *Consider the Murder T-LAIMA system of Example 6. Then, the function I with:*

- $I(Witness_1) = \{hate\_police, blond, tall\}$,
- $I(Witness_2) = \{short, female\}$,
- $I(Witness_3) = \{male, tall\}$,
- $I(Inspector) = \{murder, male, blond, tall\}$,
- $I(Officer) = \{murder, blond, tall, male, quest(a), quest(b), accuse(b), alibi(a), noalibi(b)\}$
- $I(SuspectA) = \{quest(a), alibi(a)\}$
- $I(SuspectB) = \{quest(b), noalibi(b)\}$
- $I(SuspectC) = \{\}$

is an interpretation for this system.

In the current setting, an agent's output not only depends on the agent's current beliefs, defined by an interpretation, but also on the recipient of this information. The information sent is determined by the intersection of the agent's belief and the filter used for communication the agent he is sending information to, i.e. $Out_I^b(a) = I(a) \cap F$ with $(a, b, F, n) \in C$ with $F \subset L$, and $n \in \mathbb{N}$ On the other hand, an agent receives as input the output of all agents connected to its incoming channels, i.e. $In_I(b) = cons(\cup_{(a,b)\in C} Out_I^b(a))$.

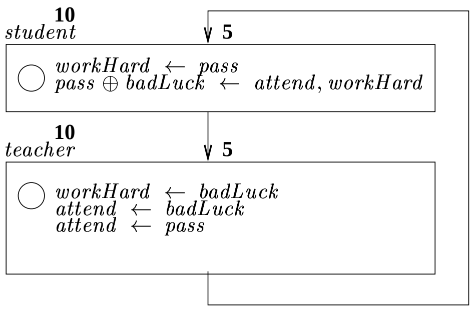


Figure 3: The Exam T-LAIMA system of Example 8

An agent reasons on the basis of positive information received from other agents (its input) and its own program that may be used to draw further conclusions, possibly contradicting incoming information. When agents receive information, they need to update their own knowledge base with this new information. This update version is a new OCLP, created from the old one by adding component based on the trust levels. Information from more trusted sources will be placed in more preferred components.
When modelling rational agents, we assume that agents would only forward information of which they can be absolutely sure. Therefore, it is only sensible to request from agents being modelled as OCLP, to communicate in terms of answer sets.

When an interpretation produces an answer set for each agent's updated version, we call it a *model* for the T-LAIMA system. Given that one agent can create multiple answer sets of its updated version, T-LAIMA system generates an extra source of non-determinism.
The model semantics provides an intuitive semantics for systems without cycles. Having cycles allow assumptions made by one agent to be enforced by another agent.

***Example* 8** *Consider the T-LAIMA system in Figure 3 where a student and a teacher discuss issues about the relationship between attending the lecture, working hard, passing the unit or having bad luck. This systems has three models:*

- $M(teacher) = M(student) = \emptyset$
- $N(teacher) = N(student) = \{attend, workHard, pass\}$
- $S(teacher) = S(student) = \{attend, workHard, badLuck\}$

Nothing in this system suggest that an exam has taken place or that the agents are not just hypothesising. Therefore, the only sensible situation is represented by interpretation M.

To avoid such self-sustaining propagation of assumptions, we require that the semantics is the result of a fixpoint procedure which mimics the evolution of the belief set of the agents over time. Initially, all agents start off with empty input from the other agents (agents do not have any dynamic beliefs regarding the other agents). At any stage agents receive the input generated by the previous cycle to update their current belief set. This evolution stops when a fixpoint (i.e. no agent changes the beliefs they had from the previous cycle) is reached. This final interpretation is then called a *global answer set*.

Thus, in an evolution, the agents evolve as more information becomes available: at each phase of the evolution, an agent updates its program to reflect input from the last phase and computes a new set of beliefs. An evolution thus corresponds to the way decision-makers try to get a feeling about the other participants. The process of reaching a fixpoint boils down to trying to get an answer to the question "if I do this, how would the other agents react", while trying to establish a stable compromise. Note that the notion of evolution is nondeterministic since an agent may have several local models. For a fixpoint, it suffices that each agent can maintain the same set of beliefs as in the previous stage.

***Example* 9** *The T-LAIMA system of Example 8 has exactly one global answer set, namely M, just as we hoped. At iteration 1, both agents will receive no input resulting in producing both empty answer sets as output. This makes that the input in iteration 2 is also empty for both, obviously resulting the same output. Since both iterations are exactly the same, we have reached a fixpoint.*

The T-LAIMA system of Example 6 has also one answer set: the model mentioned in Example 7. The evolution producing this answer set is slightly more interesting. We leave it to reader to construct the evolution in detail. In the first iteration, the witnesses produce their facts which are received by the inspector *in the second iteration. In second iteration both witnesses and inspector complete their belief set which is reported to the Officer for use in the third iteration. In this iteration, the Officer will decide which suspects to question. This is done in the fourth iteration. In the fifth iteration, the Officer will solve the crime. SuspectB will be notified of being accused during the sixth iteration.*

Even though agents agreed on a global answer set, this does not mean that they will all have the same knowledge (i.e. local answer sets). Even if agents would pass on unfiltered answer sets to each other, we would still have situations where agents agree to disagree. Furthermore, it is perfectly possible for a T-LAIMA system to have no global answer sets at all. Consider for example is you want to model two childeren arguing about about two toys ans who will have which one: once they have one they will want the other one.

5 JOF: The T-LAIMA Implementation

The theoretical multi-agent architecture T-LAIMA described in the previous section has been implemented as the JADE OCLP Framework (JOF). Although OCLP gives an interesting way of representing the believes, desires and intention of one agent in a multi-agent framework and can be reasoned about using an answer set solver, we need to embed our theoretical model into a multi-agent platform that runs the agents and supports the communication between the various agents. We have opted for using JADE for the MAS architecture, Protégé [22] for the ontology that supports the communication protocol and OCT [3] as a OCLP front-end to Smodels.

The choice of working within the JADE framework[15] and the use of the relevant FIPA[12] specifications allows for an architecture based on behaviours. The agents can be implemented at a high-level by defining their behaviours and reactions to events.

An ontology is required for the grounding of the concepts and relationships within JOF and to smooth out the communication between the various agents in the system and to allow easy expansion to different types agents.

JOF is based on a simple client-server model with the most of the computational work performed by the clients. The implementation has two main agents or roles: the Coordinator and JOF agents. Using the GAIA methodology of defining roles by four attributes (responsibilities, permissions, activities and protocols), we have defined the collection of behaviours for our two main agents. The Coordinator contains the graphical user interface for the application and is designed to be the interface between the human user and the MAS. When initialised, the Coordinator agent will simply start up the user interface, awaiting user input. The Coordinator is capable of ordering agents to perform a cycle, update and retrieve their information. It also a maintains list of agents that can be edited by the user, along with other administrative actions. The JOF agents are designed to also run autonomously. This distinction is important, as it allows the option of running JOF without the need for a human supervisor. Agents are then able to subscribe to the Coordinator or to each other, and receive periodic information (i.e. the answer sets).

It is possible to run JOF in two different modes: retained and runtime. Retained mode allows the user to add, remove or modify any of the agents in the JOF environment. All agents running in retained mode are controlled by the Coordinator agent. Runtime mode provides the autonomous functionality and events to fire upon relevant stages in the OCLP agent life-cycle but also gives means for a developer to alter all of the relevant information of the local agents.

The JOF ontology gives a basis for the representation of different objects and functionality in the JOF environment. For example, the ontology provides for the fundamental of notions like ‘rule’ or ‘OCLP program’. There are three main sections that we can identify in the ontology: OCLP programs, message passing and the knowledge base. The ontology itself is declared hierarchically using the BeanGenerator within Protégé and also encapsulates a number of message classes, such as the answer set broadcast and the Coordinator’s information request and retrieval messages.

One of the key implementation details is the interaction with OCT to provide the answer sets. In order to interact correctly with OCT, JOF is required to produce valid OCT input, parse OCT result output and deal with the OCT error messages when they occur. Therefore, valid input would be an OCLP program as a string, satisfying OCT syntax. Output would be an answer set broadcast. It is possible to either synchronously run OCT and wait for its output, or as a thread which fires an event when finished.

JOF agents need a way of knowing when it is time to process their inputs, which is the motivation for the Call For Answer Sets (CFAS) protocol. This controls the life-cycle of the JOF agents and is sent with a given deadline, after which the agents that not receive input will carry on using the empty set as input. Once all responses have been received, the solution to the updated OCLP is computed immediately.

6 Conclusions and Directions for Future Research

The deductive multi-agent architecture described in this paper uses OCLP to represent the knowledge and reasoning capacities of the agents involved. However, the architecture and its behaviour are designed to be able to deal with any extension of traditional answer set programs. [1] gives an overview of some the language extensions currently available.

There is an emerging view [8, 10, 20, 14, 18] that ASP, with or without preferences or other language constructs, is a very promising technology for building MAS. It satisfies the perspective that agent programming should be about describing *what* the goals are, not *how* they should be achieved— i.e. "post-declarative" programming [24]. Furthermore, having the same language for specification and implementation, makes verification trivial. Thanks to its formal grounding of all its language constructs, it becomes possible to verify the soundness and completeness of the language and implementation.

The Dali project [5] is a complete multi-agent platform entirely written in Prolog. The EU-funded research project SOCS ([17, 23]) has constructed a multi-agent architecture based solely on logic programming components, each responsible for a part of the reasoning of the agent. Using ASP, it would be possible to use the same language for all components, avoiding duplication of domain description and knowledge. This technique is used in [20] for diagnosis and planning for the propulsion system of the NASA Space Shuttle. Another multi-agent system is the Minerva architecture [18]. They build their agents out of subagents that work on a common knowledge base written as a MDLP (Multidimensional Logic Program) which is an extension of Dynamic Logic Programming. It can be shown that MDLP can easily be translated into OCLP such that their stable models match our answer sets. The advantage of using OCLP is that you have a far more flexible defeating structure. One is not restricted to decision comprising only two alternatives. and the decisions are dynamic. On the other hand, the Minerva does not restrict itself to modelling the beliefs of agents, but allows for full BDI-agents that can plan towards a certain goal. It would be interesting to see what results we obtain when OCLP approach was incorporated into Minerva and what results we would obtain by incorporating Minerva into LAIMAS.

The flexibility on the specification side of ASP is counter-balanced by the technology of current (grounding) answer set solvers, which mean that a major part of the solution space is instantiated and analysed before the any computation of a solution begins. The advantage of this is that the computation can be staged [16] into a relatively expensive part that can be done off-line and a relatively cheap part that is executed on-line in response to queries. For example, [20] reports that the space shuttle engine control program is 21 pages, of which 18 are declarations and 3 are rules describing the actual reasoning. Grounding takes about 80% of the time, while the 20% is used for the actual computation of the answer set and answering queries. There is a parallel between the way answer set solvers compute the whole solution space and model-checking.

As was mentioned in the introduction, each T-LAIMA system in the above examples models a single interaction between agents. A sequence of interactions would therefore be modeled by a sequence of T-LAIMA systems. The agents involved, and their level of trust of each other, may change from interaction to interaction.

One way to update trust levels would be to have each agent go through an introspective process between interactions. During the introspective process the agent could, for example, compare their final set of beliefs at the end of the interaction with the information given to them by other agents. Trust levels could then be modified based on whether the information agreed or disagreed with the agent's final beliefs.

It is interesting to note that computing the new trust values can also be done using ASP. One would simply need to combine the set of the agent's beliefs, the facts about what information was passed to the agent by other agents, the old trust values, and a set of ASP rules which expressed logically how new trust values should be obtained from the old. The set of facts and rules could then be passed to the same answer set solver used by the T-LAIMA system. The resulting model would contain the new trust values. As there could be many different possible strategies for updating trust, deciding which one to use (and, therefore, what rules are needed to compute the new trust values) is itself an interesting research question which is left for future research.

In our framework, an agent is given an initial set of beliefs and a set of reasoning capacities. Each time the agent receives information from the outside world, it will update its knowledge/beliefs base and computes the answer sets that go with it. This means that for even the smallest change the whole program has to be recomputed (including the expensive grounding process). To provide some additional flexibility, a recent development [2] describes a technique for *incremental* answer set solving. With an incremental solver that permits the assertion of new rules at run-time we are in a position to move from purely reactive agents to deliberative agents that can support belief-desire-intention style models of mentality. We aim to replace the fixed solver (OCT) we have built into JADE and described here with the incremental one (IDEAS) shortly.

The current implementation of grounding in ASP limits its use to finite domains. Clearly, a fixed solution space is quite limiting in some domains, but reassuring in others. OCLP significantly simplifies the development of such systems by constructing a finite domain. To go beyond the grounding solver is currently one of the key challenges, together with the more practical aspects of ASP like updates, methodology, and debugging.

It seems likely that the next few years will see significant developments in answer set solver technology, deploying both incremental and ungrounded solvers. We foresee future improvements on the framework itself. Currently, all our agents take input only from other agents. Of course, one could argue that the environment can be seen as an agent. At present, agents can only communicate that they have an answer set. When an agent fails to produce information, due to contradictions in its knowledge base, the whole process halt, while failure on its own could also been seen as information.

The current system is tailored towards success as failure to produce an answer set is not reported. When an agent fails to produce an answer set for its updated version, communication will stop at this agent, without warning the others. From both a theoretical and implementation point of view this raises interesting possibilities.

In the near future, we aim to use T-LAIMA and its extensions for the development of larger systems. One of our goals is to try to incorporate the ALIAS[4] system, an agent architecture for legal reason based on abductive logic, into ours.

Acknowledgements This work was partially supported by the European Fifth Framework Programme under the grant IST-2001-37004 (WASP) and the Nuffield Foundation under the grant NAL/00689/A.

References

[1] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[2] M. J. Brain. Undergraduate dissertation: Incremental answer set programming. Technical Report 2004–05, University of Bath, U.K., Bath, May 2004.

[3] Martin Brain and Marina De Vos. Implementing OCLP as a front-end for Answer Set Solvers: From Theory to Practice. In *ASP03: Answer Set Programming: Advances in Theory and Implementation*. Ceur-WS, September 2003. online CEUR-WS.org/Vol-78/asp03-final-brain.ps.

[4] A. Ciampolini and P Torroni. Using abductive logic agents for modeling the judicial evaluation of crimimal evidence. *Applied Artificial Intelligence*, 18:251–275, 2004.

[5] Stefania Costantini and Andrea Tocchio. A Logic Programming Language for Multi-agent Systems. In *Logics in Artificial Intelligence, Proceedings of the 8th European Conference, Jelia 2002*, volume 2424 of *Lecture Notes in Artificial Intelligence*, Cosenza, Italy, September 2002. Springer-Verlag, Germany.

[6] Marina De Vos. Implementing Ordered Choice Logic Programming using Answer Set Solvers. In *Third International Symposium on Foundations of Information and Knowledge Systems (FoIKS'04)*, volume 2942, pages 59–77, Vienna, Austria, February 2004. Springer Verlag.

[7] Marina De Vos and Dirk Vermeir. On the Role of Negation in Choice Logic Programs. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Non-Monotonic Reasoning Conference (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 236–246, El Paso, Texas, USA, 1999. Springer Verslag.

[8] Marina De Vos and Dirk Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, 42(1–3):103–139, September 2004. Special Issue on Computational Logic in Multi-Agent Systems.

[9] Mark Denecker. What's in a Model? Epistemological Analysis of Logic Programming. Ceur-WS, September 2003. online CEUR-WS.org/Vol-78/.

[10] Juergen Dix, Thomas Eiter, Michael Fink, Axel Polleres, and Yingqian Zhang. Monitoring Agents using Declarative Planning. *Fundamenta Informaticae*, 57(2–4):345–370, 2003. Short version appeared in Gnther/Kruse/Neumann (Eds.), Proceedings of KI 03, LNAI 2821, 2003.

[11] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.

[12] FIPA. `http://www.fipa.org/`.

[13] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.

[14] Michael Gelfond. Answer set programming and the design of deliberative agents. In *Procs. of 20th International Conference on Logic Programming*, number 3132 in Lecture Notes in Artificial Intelligence (LNCS), pages 19–26, September 2004.

[15] Jade:. `http://jade.tilab.com/`.

[16] U. Jørring and W. Scherlis. Compilers and staging transformations. In *Proceedings of 13th ACM Symposium on Principles of Programming Languages*, pages 86–96, New York, 1986. ACM.

[17] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F Toni. Declarative agent control. In J Leite and P. Torroni, editors, *5th Workshop on Computational Logic in Multi-Agent Systems (CLIMA V)*, September 2004.

[18] J. A Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII*, number 2002 in LNAI, pages 141–157. Springer-Verlag, 2002.

[19] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.

[20] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. A A-Prolog Decision Support System for the Space Shuttle. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Represenation and Reasoning*. American Association for Artificial Intelligence Press, Stanford (Palo Alto), California, US, March 2001.

[21] Opencyc. `http://www.cyc.com/opencyc`.

[22] Protégé. `http://protege.stanford.edu/`.

[23] SOCS. `http://lia.deis.unibo.it/research/socs/`.

[24] M. J. Wooldridge and N. R Jennings. Agent theories, architectures and languages: a survey. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents; ECAI-94 Workshop on Agent Theories, Architectures, and Languages (Amsterdam 1994)*, volume 890 of *LNCS (LNAI)*, pages 1–39. Berlin: Springer, 1994.

[25] Mike Wooldridge. *An introduction to multiagent systems*. Wiley, 2002. ISBN: 0 47149691X.