Access-as-you-need: A computational logic framework for flexible resource access in artificial societies

Francesca Toni[1] and Kostas Stathis[2]

[1] Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK,
URL: `http://doc.ic.ac.uk/~ft`

[2] Department of Computing, School of Informatics, City University London,
Northampton Square, London EC1V 0HB, UK,
URL: `http://soi.city.ac.uk/~kostas`

Abstract. We investigate the application of abductive logic programming, an existing framework for knowledge representation and reasoning, for specifying the knowledge and behaviour of software agents that need to access resources in a global computing environment. The framework allows agents that need resources to join artificial societies where those resources are available. We show how to endow agents with the capability of becoming and ceasing to be members of societies, for different categories of artificial agent societies, and of requesting and being given or denied resources within societies. The strength of our formulation lies in combining the modelling and the computational properties of abductive logic programming for dealing with the issues arising in resource access within artificial agent societies.

1 Introduction

Global computing [5] is a new vision for information and communication technologies whereby most physical objects that people use will be transformed into electronic devices with computing processors and embedded software designed to perform (or control) a multitude of tasks in people's everyday activities. Many of these devices will also be able to communicate with each other and to interact with the environment in carrying out the tasks they are designed for. The expectation is that the design of computational and information processing systems will rely entirely on the available infrastructure and processing power around us. The challenge this vision poses is huge in that it requires the definition and exploitation of dynamically configured applications of software entities interacting in novel ways with other entities to achieve or control their computational tasks.

Global computing environments are intended to be highly *open and dynamic*: physical devices will be mobile or portable, connectivity and bandwidth will not be constant, computational processes and data would need to migrate, and applications will come and go in the environment the same way people who use them will. In such unpredictable and open environments one of the main issues is how to control the availability of the resources that are active in an application at any given time.

From our involvement in the SOCS project[15], we are motivated by a *travelling businessman scenario* presented in [16]. This scenario can be summarised as follows. A businessman travels for work purposes to a foreign country and, in order to make his trip easier, carries a personal communicator, namely a device that is a hybrid between a mobile phone and a PDA. The application running on this personal communicator provides the environment for a personal service agent [8], namely a piece of software that augments the direct manipulation interface of the device with implicit management. By implicit management we mean that the agent provides proactive information management within the device [19] and flexible connectivity to smart services available in the global computing environment the businessman travels within. We interpret connectivity to smart services as *access to resources*.

We assume that resources are available within and managed by *artificial agent societies*[9] in the global computing environment. Examples of such societies in the travelling businessman scenario are (the society of agents supporting) an airport, a hotel, a train station.

In SOCS [15], we use the term agent society simply to refer to a possibly complex organisation of agents that supports and complements the activities of a physical, human society, in a global computing environment. In particular, artificial societies complement physical societies by being composed of agents that act on behalf of people (whether individuals or groups) in order to access, manage, control and authorise the use of resources required by global computing applications. Artificial societies can be understood as systems being composed of autonomous computational entities where

– activity is not centrally controlled, either because global control is impossible or impractical, or because the entities are created or controlled by different owners.
– the configuration of the system varies over time; for instance, the system is open to the introduction of new computational entities and the departure of already existing ones.

We do not regard artificial societies merely as locations where resources can be found. Although artificial societies are accessible from physical locations, they are further characterised by sets of rules and conventions that govern the accessibility, management, control and authorisation to use resources, as we will see later on.

We are particularly interested in investigating the situation whereby an agent, who is in pursuit of its goals in order to satisfy the needs of a person (or group) in a human society, becomes a *member* of one or more artificial societies to obtain resources that are required by the person (or group). We follow the categorisation of artificial societies given in [1], whereby societies may be *open*, with no eligibility requirements for members-to-be, *closed*, with no agent not already part of these societies being allowed to enter, *semi-open*, with agents allowed to become members if they satisfy some eligibility criteria, and *semi-closed*, with agents not allowed to become members under any circumstances, but allowed, if eligible, to have a representative created or nominated within the society, to act on their behalf.

In this paper, we provide a knowledge representation and reasoning framework based on *abductive logic programming* [6] for specifying the knowledge and behaviour of agents joining and leaving societies according to the resources they need in order to satisfy their goals. We rely upon the computational properties of abductive proof procedures that abductive logic programming is equipped with, to reason with the knowledge of agents and allow agents to exhibit the required behaviour. The execution model of an agent uses the proof procedure within an observe-reason-act cycle that interleaves the execution of the proof procedure with observation and action execution [7, 12–14].

One of the advantages of our approach is that the specifications that we develop are executable, so that our approach could be used to actually build artificial societies. At the same time, our approach does not exclude the proposed specifications to be implemented in a different framework, as long as the operational semantics of the proof procedure is maintained. Thus, our framework can be seen as a logical model of the required agent behaviour. In this sense, societies in our framework are open to agents built in different frameworks, as soon as they conform to the specified behaviour, in the same way agents are required to conform to communication protocols.

Another advantage of our approach is that it lends itself to a formal analysis (such as the one in [13, 14]) of the multi-agent systems and artificial societies that are designed or built according to our approach.

The structure of the paper is as follows. In section 2 we present the framework of abductive logic programming for modelling agents' beliefs and behaviour. In section 3 we address the problem of how resources are requested and how requests are processed. In section 4 we specify how agents might join or leave societies to obtain resources, and how societies admit/reject new members or expel existing ones. We concentrate on open, semi-open and semi-closed societies (we disregard closed societies as they do not allow for dynamic entry and are thus uninteresting for our purposes). In section 5 we exemplify the computational behaviour of agents in our framework, by means of a simple instance of the travelling businessman scenario outlined above. Section 6 concludes.

2 Abductive logic programming

Abductive logic programming is a general-purpose knowledge representation and reasoning framework that can be used for a number of applications [6, 7, 3, 11, 2, 12–14, 20]. It relies upon a symbolic, logic-based representation of the beliefs of agents, for any given domain of application, via abductive logic programs, and the execution of logic-based reasoning engines, called abductive proof procedures, for reasoning with such representations. *Abductive logic programs* consist of

– A *logic program*, P, namely a set of if-rules of the form $Head \leftarrow Body$, where $Head$ is an atom, $Body$ is either $true$ (in which case we write the if-rule simply as $Head$) or a conjunction of (negations of) atoms. All variables in $Head$ and $Body$ are implicitly universally quantified from the outside. P is understood as a (possibly incomplete) set of beliefs by the agent.

- A set of *integrity constraints*, I, namely if-then-rules of the form $Condition \Rightarrow Conclusion$, where $Condition$ is either $true$ (in which case we write the if-then-rule simply as $Conclusion$) or a conjunction of (negations of) atoms, $Conclusion$ is $false$ or a (possibly existentially quantified) conjunction of atoms, and all variables in $Condition$ are implicitly universally quantified from the outside. The integrity constraints are understood as properties that must be "satisfied" by any extension of the logic program by means of atoms of abducible predicates, in the same way integrity constraints in databases are understood as properties that must be "satisfied" in every database state.
- a set of *abducible predicates*, Ab, namely predicates occurring in the logic program and the integrity constraints but not in the $Head$ of any if-rule.

In this paper, for convenience of presentation, we allow for $findall$ and $forall$ conjuncts, understood as in PROLOG, to occur in the $Body$ of if-rules in P. Also, the set of abducible predicates Ab of an agent is structured as follows: $Ab = Actions \cup Observables$, such that $Actions \cap Observables = \emptyset$, where $Actions$ represent actions that the agent can execute and $Observables$ represent phenomena that the agent can observe. In general, the $Actions$ can be either physical or communicative, namely utterances that the agent can perform. In this paper, all abducibles of all agents we consider are communicative. The abducibles for sections 3-5 are:

join($Agent, Soc, Time$), **leave**($Agent, Soc, Time$),
request($Agent, Soc, Resource, Time$),
admit($Auth, Agent, Soc, Time$), **deny**($Auth, Agent, Soc, Time$),
represent($Auth, Agent, Proxy, Soc, Time$),
assign($Auth, Proxy, Agent, Soc, Time$), **expell**($Auth, Agent, Soc, Time$),
broadcast_joined($Auth, Soc, Agent, Time$),
broadcast_left($Auth, Soc, Agent, Time$),
accept($Controller, Agent, Soc, Resource, Time$),
refuse($Controller, Agent, Soc, Resource, Time$),

The meaning of these utterances will become clear later. Here, note that the first (second) argument of every such utterance is the utterer (receiver, resp.). Utterances with the agent as utterer (receiver) are actions (observables, resp.).

Given an abductive logic program (P, I, Ab) and an *observation* or *goal* (i.e. a possibly empty conjunction of literals) Q, the task of an *abductive proof procedure* is to compute a so-called *explanation* for Q, namely a pair (E, θ) consisting of a (possibly empty) set E of atoms in abducible predicates in Ab and a (possibly empty) variable substitution θ for the variables in Q, such that $P \cup E$ *entails* $Q\theta$ and $P \cup E$ *satisfies* I. Various notions of entailment and satisfaction can be adopted, for example entailment and satisfaction could be entailment and consistency, respectively, in first-order, classical logic. However, the provision of such notions is beyond the scope of this paper, see [6] for a detailed discussion.

Examples of abductive proof procedures can be found in [6, 4, 11]. Typically, abductive proof procedures interleave *backward reasoning* (also called *unfolding*) with if-rules in the logic program (reduction of $Head$ to $Body$) and *forward* *reasoning* (also called *propagation*) with if-then-rules in the integrity constraints (addition of $Conclusion$ to $Condition$). Backward reasoning with the if-rules allows to generate the kernel of candidate explanations, whereas forward reasoning with the if-then-rules allows to rule out unwanted candidate explanations and expand the kernel of promising ones. Indeed, some kernel explanations might allow to explain the observation but might fail to take into account the integrity constraints, whereas others might need to be augmented in order to do so. The interleaving terminates when no more backward or forward reasoning can be performed in a branch in the tree of candidate explanations: the explanation can then be extracted from the branch.

While modelling agents via abductive logic programming, an abductive proof procedure is executed within an *observe–reason–act* cycle that allows the agent to be alert to the environment and react to it as well as reason and devise plans [7, 14]. This cycle interleaves the execution of the abductive proof procedure (*reason*) for the computation of explanations for observations and goals, the perception of inputs from the environment, to be added dynamically to the observation or goal to be explained (*observe*) and the selection of actions (i.e. abducibles in explanations) to be performed in the environment (*act*). Note that, in order for the agents to be reactive to the environment in which they are situated, and thus to the inputs from the agents that reside in the environment, agents might need to select actions and perform them before the computation of explanations is finalised. In this paper we will refer to the interleaved execution of an abductive proof procedure as illustrated above, as the *execution model* of the agent. We will assume that the observation of observables (utterances from other agents) is left to the execution model, and that when executing an utterance, its time argument is instantiated to the current time.

In the next two sections, we give a concrete abductive logic program modelling the knowledge of agents that need to access resources and might decide to join societies in order to gain access, and, in section 5, we use this modelling to illustrate the behaviour of the abductive proof procedure of [11] for an example within the travelling businessman scenario. Here, we illustrate by means of a simple example the use of abductive logic programming for handling requests between agents, in order to convey some intuition. We have two agents a and b. The beliefs of agent a are represented by the abductive logic program with

P: $get(R,T) \leftarrow request(a,b,R,T') \wedge accept(b,a,R,T'') \wedge T'' \geq T' \geq T$

I: $request(a,b,R,T) \wedge refuse(b,a,R,T') \Rightarrow false$

Ab: $request(a, Agent, Res, Time)$ (action), $accept(Agent, a, Res, Time)$ (observable).
Namely, in order to get a resource, a needs to request it from b and having the request accepted (P). Moreover, having a request refused amounts to failing, represented by $false$ (I). Note that, here and in the sequel, we assume, for simplicity, that accepting a request amounts to getting possession of it.

The goal of a is $get(r, 0)$, for some given resource r. From this goal, the abductive proof procedure generates, by two steps of backward reasoning,

$\exists T', T''[request(a,b,r,T') \wedge accept(b,a,r,T'') \wedge T'' \geq T' \geq 0]$.
The execution model of a selects the abducible $request(a, b, r, T')$ and executes

(utters) it, instantiating T' to, say, 5, and causing b to observe $request(a, b, r, 5)$. The beliefs of b are given by the abductive logic program with

P: $have(r)$

I: $request(Y, b, R, T) \wedge have(R) \Rightarrow \exists T'[accept(b, Y, R, T') \wedge T' \geq T]$
$request(Y, b, R, T) \wedge \neg have(R) \Rightarrow \exists T'[refuse(b, Y, R, T') \wedge T' \geq T]$

Ab: $accept(b, Agent, Resource)$ (action), $request(Agent, b, Resource)$ (observable).

Namely, b would grant any agent any requested resource it has, and would refuse any agent any requested resource it does not have.

From the observation $request(a, b, r, 5)$, the abductive proof procedure generates, by forward reasoning, $have(r) \Rightarrow \exists T'''[accept(b, a, r, T''') \wedge T''' \geq 5]$, and, by backward reasoning, $true \Rightarrow \exists T'''[accept(b, a, r, T') \wedge T''' \geq 5]$, which simplifies to $\exists T'''[accept(b, a, r, T') \wedge T''' \geq 5]$. The execution model of b selects the abducible and executes (utters) it, instantiating T''' to, say, 10, and causing a to observe $accept(b, a, r, 10)$, instantiating T'' to 10 and thus allowing a to achieve its original goal $get(r, 0)$. The set $E = \{request(a, b, r, 5), accept(b, a, r, 10)\}$ is an explanation for both the goal $get(r, 0)$ of a and the observation $request(a, b, r, 5)$ of b, for the abductive logic programs of the respective agents. As far as a is concerned, both abducible atoms are in E in order to entail the goal. As far as b is concerned, $request(a, b, r, 5)$ is in E in order to (trivially) entail the observation and $accept(b, a, r, 10)$ is in E to satisfy the integrity constraint of b.

3 Dealing with requests of resources

Until section 5 we will model, via an abductive logic program, the knowledge of some agent a in the system. This abductive logic program contains agent-independent if-rules and if-then-rules, that a shares with the other agents in the system, as well as agent-dependent if-rules, that are specific to a. Until section 5 we will concentrate on the agent-independent part of the the knowledge base of a, that can be adopted by other agents too, simply by replacing a with the name of the agents whenever required.

Figure 1 summarises the knowledge of agent a that a uses to obtain resources. By the if-rules P1-P3, a identifies all resources it needs and goes about getting them all, one by one. Instances of *need* are defined by agent-dependent if-rules, e.g as in section 5. The arguments T_0 and T indicates the time the agent starts and finishes (respectively) its search for the resources. A possible definition of *choose* is given by P15, assuming that sets are represented as lists.

By P4-P5 and P7-P8, a first attempts to get each resource within the societies of which a is a member and in which a believes the resource is available, by repeatedly choosing one such society until one of its requests for that resource is accepted. By P6 and P9-P12, if one resource cannot be obtained within the societies to which a belongs, a attempts to join other societies, one at a time, until it succeeds in joining one such society and one of its requests for the required resource is accepted within the newly joined society. By I1, if there are no more societies worth joining for the purposes of getting a resource, as all potentially useful societies have denied a to join or have admitted a but have refused to accept the request for the given resource, then a will be unable to obtain that resource and thus will fail, represented by $false$.

The predicates *member*, *try_to_join* and *joined* are defined in figure 3 and will be discussed in section 4. Instances of *available* are defined by agent-dependent if-rules, e.g as in section 5.

$$
\begin{array}{l}
\text{(P1) } get\_all\_needed\_resources(T_0, T) \leftarrow need(Rs) \wedge get\_all(Rs, T_0, T) \\
\text{(P2) } get\_all(Rs, T, T) \leftarrow Rs = \emptyset \\
\text{(P3) } get\_all(Rs, T_0, T) \leftarrow Rs \neq \emptyset \wedge choose(R, Rs, Rs') \wedge \\
\qquad\qquad get(R, T_0, T') \wedge get\_all(Rs', T', T) \\
\text{(P4) } get(R, T_0, T) \leftarrow member\_socs(a, R, T_0, Ss) \wedge get\_from(R, T_0, T, Ss) \\
\text{(P5) } get\_from(R, T_0, T, Ss) \leftarrow Ss \neq \emptyset \wedge choose(S, Ss, Ss') \wedge \\
\qquad\qquad \mathbf{request}(a, S, R, T') \wedge T' \geq T_0 \wedge \\
\qquad\qquad process(S, R, T', T, Ss') \\
\text{(P6) } get\_from(R, T_0, T, Ss) \leftarrow Ss = \emptyset \wedge non\_member\_socs(a, R, T, Ss') \wedge \\
\qquad\qquad get\_join(R, T_0, T, Ss') \\
\text{(P7) } process(S, R, T_0, T, Ss) \leftarrow \mathbf{accept}(A, a, S, R, T) \\
\text{(P8) } process(S, R, T_0, T, Ss) \leftarrow \mathbf{refuse}(A, a, S, R, T') \wedge get\_from(R, T', T, Ss) \\
\text{(P9) } get\_join(R, T_0, T, Ss) \leftarrow Ss \neq \emptyset \wedge choose(S, Ss, Ss') \wedge \\
\qquad\qquad try\_to\_join(S, T_0, T') \wedge process\_join(S, R, T', T, Ss') \\
\text{(P10) } process\_join(S, R, T_0, T, Ss) \leftarrow joined(a, S, T') \wedge \\
\qquad\qquad \mathbf{request}(a, S, R, T'') \wedge T'' \geq T' \wedge \\
\qquad\qquad \mathbf{accept}(A, a, S, R, T) \\
\text{(P11) } process\_join(S, R, T_0, T, Ss) \leftarrow joined(a, S, T') \wedge \\
\qquad\qquad \mathbf{request}(a, S, R, T'') \wedge T'' \geq T' \wedge \\
\qquad\qquad \mathbf{refuse}(A, a, S, R, T''') \wedge \\
\qquad\qquad get\_join(R, T''', T, Ss) \\
\text{(P12) } process\_join(S, R, T_0, T, Ss) \leftarrow \mathbf{deny}(A, a, S, T') \wedge get\_join(R, T', T, Ss) \\
\text{(P13) } member\_socs(X, R, T, Ss) \leftarrow findall(S, (member(X, S, T) \wedge \\
\qquad\qquad available(R, S)), Ss) \\
\text{(P14) } non\_member\_socs(X, R, T, Ss) \leftarrow findall(S, (\neg member(X, S, T) \wedge \\
\qquad\qquad available(R, S)), Ss) \\
\text{(P15) } choose(X, Y, Z) \leftarrow Y = [X|Z] \\
\text{(I1) } get\_join(R, T_0, T, Ss) \wedge Ss = \emptyset \Rightarrow false
\end{array}
$$

Fig. 1. Obtaining resources: knowledge base of agent a.

Figure 2 summarises the knowledge of agent a for processing requests from other agents. This knowledge is only used if a controls some resource in some society. If the request comes from an agent who is a member of the society to which the request is made then, and if the resource is indeed available in that society, then the request is accepted (I1), whereas, if the request is not available in that society, the request is refused (I2). The request is also refused if it comes from an agent who is not a member of the society (I3). Note that a would only handle requests for resources it controls. Instances of *control* are defined by agent-dependent if-rules, e.g as in section 5.

Note that, since resources are assumed to be either always available or always unavailable (due to the predicate *available* not being time-stamped), requests by members are always accepted by agents controlling the requested resources, and no concept of the resource being free at the time of the request is needed. Also, resources are considered to be reusable and can be used by multiple users simultaneously.

$$
\begin{array}{l}
\text{(I1) } \mathbf{request}(X,S,R,T) \wedge control(a,R,S) \wedge member(X,S,T) \wedge available(R,S) \Rightarrow \\
\hfill \exists T'[\mathbf{accept}(a,X,S,R,T') \wedge T < T'] \\
\text{(I2) } \mathbf{request}(X,S,R,T) \wedge control(a,R,S) \wedge \neg member(X,S,T) \Rightarrow \\
\hfill \exists T'[\mathbf{refuse}(a,X,R,S,T') \wedge T < T'] \\
\text{(I3) } \mathbf{request}(X,S,R,T) \wedge control(a,R,S) \wedge \neg available(R,S) \Rightarrow \\
\hfill \exists T'[\mathbf{refuse}(a,X,R,S,T') \wedge T < T']
\end{array}
$$

Fig. 2. Processing requests: knowledge base of agent a

4 Joining/leaving societies and admitting/rejecting new members

Figure 3 summarises the knowledge of agent a that a uses to join a society, on the basis of its need for resources. Trying to join a society amounts to simply joining if the society is open (P1) and joining after having performed a self-imposed eligibility check, if the society is semi-open or semi-closed (P2 or P3, respectively). Note that the actual eligibility criteria of the society might be different from the ones the agent believes they are. In particular, the society might perform additional checks, e.g. concerning the trustworthiness of the agent, that the agent might not perform itself.

The predicates *open*, *semi_open*, *semi_closed* and *eligible* are defined by agent-dependent if-rules, e.g as in section 5.

An agent can be thought of as having actually joined a society immediately as it joins, if the society is open (P6), or after having been accepted, either by being admitted directly, if the society is semi-open (P7), or by being assigned a representative, if the society is semi-closed (P8). Note that the if-rules P6-P8 can be used by a to reason about itself, with $X = a$, to determine whether a has actually joined a society, or about some agent different from a, to determine whether such agent is member of some society when, e.g., this agent has put forward a request and a decision needs to be made by a on whether to accept the resource or not (see section 3).

An agent can be also thought as having joined a society, and thus being one of its members, from the moment that this information has been broadcast (P9).

If an agent (either a or not) has actually joined a society, then, by the if-rule P5, the agent is a member of that society until it leaves it. Moreover, an agent is to be thought of as a member of a society if it was a member of it at the initial time 0 and until it has left it. The predicate $left$ is defined in figure 4.

Figure 4 summarises the knowledge of agent a relative to leaving societies. By I1, a decides to leave a society when it realises that being in that society

$$
\begin{array}{l}
\text{(P1) } try\_to\_join(S,T1,T2) \leftarrow open(S) \wedge \mathbf{join}(a,S,T2) \wedge T1 < T2 \\
\text{(P2) } try\_to\_join(S,T1,T2) \leftarrow semi\_open(S) \wedge eligible(a,S,T1) \wedge \\
\qquad\qquad\qquad\qquad\qquad\quad \mathbf{join}(a,S,T2) \wedge T1 < T2 \\
\text{(P3) } try\_to\_join(S,T1,T2) \leftarrow semi\_closed(S) \wedge eligible(a,S,T1) \wedge \\
\qquad\qquad\qquad\qquad\qquad\quad \mathbf{join}(a,S,T2) \wedge T1 < T2 \\
\text{(P4) } member(X,S,T) \leftarrow member(X,S,0) \wedge 0 < T \wedge \neg left(X,S,0,T) \\
\text{(P5) } member(X,S,T) \leftarrow joined(X,S,T') \wedge T' < T \wedge \neg left(X,S,T',T) \\
\text{(P6) } joined(X,S,T) \leftarrow open(S) \wedge \mathbf{join}(X,S,T) \\
\text{(P7) } joined(X,S,T) \leftarrow semi\_open(S) \wedge \mathbf{admit}(Y,X,S,T) \\
\text{(P8) } joined(X,S,T) \leftarrow semi\_closed(S) \wedge \mathbf{represent}(Y,X,A,S,T) \\
\text{(P9) } joined(X,S,T) \leftarrow \mathbf{broadcast\_joined}(Y,S,X,T)
\end{array}
$$

Fig. 3. Joining societies: knowledge base of agent a.

is of no benefit, since it does not need any of the resources that are available in that society or, if it does need some resource, this has been refused to it (P1). Note that, by I1, a is not allowed to leave the society if either it is the authority in the a society or it controls some of the resources in that society. Here $authority(a, S)$ represents the fact that the agent a, whose knowledge we are defining, has authority in S to accept or reject new applicants.

The remaining if-then-rules in figure 4 only apply to a if a has the authority over some society. By I2, a will expel any un-trustworthy agent in a society over which it has authority. By I4, a will broadcast to all agents in that society that the given agent has been expelled. By I3, a will broadcast to all agents in a society over which it has authority that some agent has left.

The predicates *authority*, *control* and *untrustworthy* are defined by agent-dependent if-rules, e.g as in section 5.

The if-rules P3-P5 define the predicate $left$: an agent X (either a or another agent) is to be considered as having left a society if it has voluntarily left it (P3), if it has been expelled from it (P4), or if there has been a broadcast message within the society that the agent X has left.

Figure 5 summarises the knowledge of a relative to dealing with memberships applications. The if-then rules in this figure all apply only if a has the authority over some society. The if-then-rules I1-I2 express the conditions under which a new applicant is accepted, and deal with the case of semi-open (I1) and semi-closed societies (I2). In both case, a needs to perform an eligibility check on the applicant and broadcast to all agents in the society that a new member has been accepted. In the case of a semi-open society, the applicant is notified of admission, via the utterance in the **admit** predicate. In the case of semi-closed societies, a first needs to choose a proxy for the applicant, notify the proxy if its new task of representing the applicant within the society, and then notify the applicant that it has been accepted and that it is going to be represented by the chosen proxy. The atom $proxy(A, S, T)$ holds for those agents A in the society S that can act on behalf of other agents outside the society. The predicate *proxy* is defined by agent-dependent if-rules, e.g. as in section 5. The if-then rule I5 deals with the

(I1) $member(a, S, T) \wedge no\_benefit(S, T) \wedge \neg authority(a, S) \wedge control\_nothing(a, S) \Rightarrow$
$\exists T'[\mathbf{leave}(a, S, T') \wedge T < T']$
(I2) $untrustworthy(X, T) \wedge member(X, S, T) \wedge authority(a, S) \Rightarrow$
$\exists T'[\mathbf{expell}(a, X, S, T') \wedge T < T']$
(I3) $\mathbf{leave}(X, S, T) \wedge member(X, S, T) \wedge authority(a, S) \Rightarrow$
$\exists T'[\mathbf{broadcast\_left}(a, S, X, T')]$
(I4) $\mathbf{expell}(a, X, S, T) \Rightarrow \exists T'[\mathbf{broadcast\_left}(a, S, X, T')]$
(P1) $no\_benefit(S, T) \leftarrow findall(R, available(R, S), Z) \wedge$
$forall(in(R, Z), no\_benefit\_resource(R, T))$
(P2) $control\_nothing(X, S) \leftarrow \neg \exists R[control(X, R, S)]$
(P3) $left(X, S, T1, T2) \leftarrow \mathbf{leave}(X, S, T) \wedge T1 < T \leq T2$
(P4) $left(X, S, T1, T2) \leftarrow \mathbf{expel}(A, X, S, T) \wedge T1 < T \leq T2$
(P5) $left(X, S, T1, T2) \leftarrow \mathbf{broadcast\_left}(Y, S, X, T) \wedge T1 < T \leq T2$
(P6) $no\_benefit\_resource(R, T) \leftarrow \neg need\_resource(R)$
(P7) $no\_benefit\_resource(R, T) \leftarrow \mathbf{accept}(X, a, R, S, T') \wedge T' < T$
(P8) $no\_benefit\_resource(R, T) \leftarrow \mathbf{refuse}(X, a, R, S, T') \wedge T' < T$
(P9) $need\_resource(R) \leftarrow need(Rs) \wedge R \in Rs$

Fig. 4. Leaving societies: knowledge base of agent a

case of open societies, where any applicant is accepted automatically, without notification, and all a needs to do is to broadcast about the new member. The if-then-rules I3-I4 express that non-eligible applicants are denied membership, both for semi-open (I3) and semi-closed societies (I4).

(I1) $\mathbf{join}(X, S, T) \wedge authority(a, S) \wedge semi\_open(S) \wedge eligible(X, S, T) \Rightarrow$
$\exists T1, T2[\mathbf{broadcast\_joined}(a, S, X, T1) \wedge \mathbf{admit}(a, X, S, T2) \wedge T < T1 < T2]$
(I2) $\mathbf{join}(X, S, T) \wedge authority(a, S) \wedge semi\_closed(S) \wedge eligible(X, S, T) \Rightarrow$
$\exists Y, T1, T2, T3[proxy(Y, S, T) \wedge \mathbf{broadcast\_joined}(a, S, X, T1) \wedge$
$\mathbf{assign}(a, Y, S, X, T2) \wedge \mathbf{represent}(a, X, S, Y, T3) \wedge \wedge T < T1 < T2 < T3]$
(I3) $\mathbf{join}(X, S, T) \wedge authority(a, S) \wedge semi\_open(S) \wedge \neg eligible(X, S, T) \Rightarrow$
$\exists T1, T2[\mathbf{deny}(a, X, S, T1) \wedge T < T1 < T2]$
(I4) $\mathbf{join}(X, S, T) \wedge authority(a, S) \wedge semi\_closed(S) \wedge \neg eligible(X, S, T) \Rightarrow$
$\exists T1, T2[\mathbf{deny}(a, X, S, T1) \wedge T < T1 < T2]$
(I5) $\mathbf{join}(X, S, T) \wedge authority(a, S) \wedge open(S) \Rightarrow$
$\exists T1[\mathbf{broadcast\_joined}(a, S, X, T1) \wedge T < T1]$

Fig. 5. Admitting/rejecting new members: knowledge base of agent a

5 Example

We consider the "travelling businessman scenario" outlined in the introduction, and the situation where, while the businessman is waiting at the airport gate to catch his plane, his agent a, knowing that the businessman will need information about trains at the time of arrival, registers the (device owned by) the businessman with a service providing up-to-date train information in the global computing infrastructure of the country that the businessman will be visiting.

We describe, step by step, the reasoning and behaviour of agent a, assuming that a holds the knowledge given in sections 3 and 4 as well as the following a-dependent if-rules

$member(a, s_1, 0)$	(a is a member of the mobile phone network s_1)
$need(a, \{r\})$	(a needs the train timetable r for some destination)
$available(r, s_2)$	(r is available from the service provider for train info s_2)
$semi\_open(s_2)$	(s_2 is a semi-open society)
$eligible(X, s_2, T)$	(a believes that anybody is eligible to s_2)
$member(d, s_2, 0)$	(a believes that d is a member of s_2)

The resource r is indeed available in the society s_2, which contains two members b and c (and possibly other members), of which b has authority over the society and d controls r. We assume that agents b, c hold the knowledge given in sections 3 and 4 for agent a, but with the symbol a replaced by the symbols b, c, respectively, whenever needed, as well as the following b, c-dependent if-rules:

b: $member(b, s_2, 0)$	c: $member(c, s_2, 0)$
$authority(b, s_2)$	$control(c, r, s_2)$
$eligible(X, s_2, T) \leftarrow member(X, s_1, T)$	$eligible(X, s_2, T) \leftarrow \neg untrustworthy(X, T)$
$untrustworthy(d, T)$	$available(r, s_2)$
$semi\_open(s_2)$	

Note that the agents have a partial knowledge of members of societies, e.g. b, c do not know/believe that a is a member of s_1, and neither b nor c knows/believes that the other is a member of s_2. Also, the agents may have inaccurate knowledge of societies, e.g. a believes that d is a member of s_2, which is not the case (because if it were the case, b, the authority in s_2, would know it). Moreover, the agents might hold conflicting beliefs, e.g. b and c have different beliefs about the eligibility criteria for s_2 (the only one that matters is the one held by the authority b). Further, the information about the type of societies and the availability of resources in societies is common to all relevant agents. Thus, this information can be thought of as global, rather than local to individual agents. In general, such information could be local, and possibly partial and inaccurate. Finally, for simplicity we assume that in each society there is only one agent with authority to deal with applicants and only one controller for each resource. The authority does not need to know who controls which resource in a society.

Let us now describe the behaviour of a, as given by executing an abductive proof procedure for a's goal $get\_all\_needed\_resources(0, T)$. First we obtain

$get(r, 0, T)$	(by repeatedly unfolding with P1-P3 in figure 3)
$get\_from(r, 0, T, \emptyset)$	(by unfolding with P4 and P13 in figure 3)
$get\_join(r, 0, T, \{s_2\})$	(by unfolding with P6 and P14 in figure 3)

Then, by unfolding with P9-P10 in figure 3, we obtain:

$$\exists T', T'', T'''[try\_to\_join(s_2, 0, T') \wedge joined(a, s_2, T'') \wedge$$
$$\mathbf{request}(a, S, r, T''') \wedge \mathbf{accept}(X, a, S, R, T) \wedge T''' > T'']$$

Now the atom in the predicate try_to_join can be unfolded, using the if-rules in figure 3, thus giving, since s_2 is semi-open and since a passes its own eligibility check for s_2

$\exists T', T'', T'''[\mathbf{join}(a, s_2, T') \wedge joined(a, s_2, T'') \wedge$
$\mathbf{request}(a, s_2, r, T''') \wedge \mathbf{accept}(X, a, S, R, T) \wedge T''' > T'']$

At this point, the execution model selects the communicative action (abducible) in the predicate **join** and executes it, instantiating its time parameter, e.g. to $T' = 10$. The execution of this action by agent a corresponds to the observation by all members of s_2 of $\mathbf{join}(a, s_2, 10)$. This observation will trigger the authority agent b in s_2 to propagate with the if-then-rule I1 in figure 4 and to execute actions $\mathbf{broadcast\_joined}(b, s_2, a, T2)$, with $10 < T2$, and $\mathbf{admit}(b, a, s_2, T3)$, with $T2 < T3$. Assume the execution models of b instantiates $T3$ to 20. Agent a will observe $\mathbf{admit}(b, a, s_2, 20)$, giving, after unfolding *joined* with P7 in figure 3,

$\exists T'''[\mathbf{join}(a, s_2, 10) \wedge \mathbf{admit}(b, a, s_2, 20) \wedge$
$\mathbf{request}(a, S, r, T''') \wedge \mathbf{accept}(X, a, S, R, T) \wedge T''' > 20]$

Similarly as before, the execution model selects the communicative action in the predicate **request** and executes it, instantiating its time parameter, e.g. to $T''' = 30$. The execution of the **request** action by agent a corresponds to the observation by all members of s_2 of $\mathbf{request}(a, s_2, r, 30)$. Repeating a similar process as the one outlined before, agent c grants the resource r to a, by triggering I1 in figure 2, since agent c controls r in s_2. After having obtained the resource, a will leave s_2, by reasoning as before but with I1 in figure 4, since there is no benefit anymore for a to stay in s_2.

6 Concluding remarks

We have applied abductive logic programming to specify the knowledge and behaviour of software agents that need to access resources in a global computing environment populated by artificial societies. Our investigation has focused on the situation where an agent that needs a resource has to join artificial societies that control the provision of that resource.

We have made a number of simplifying assumptions. We have not taken into account applications for temporary membership in a society. We have assumed that the definitions of *authority*, *control*, *open*, *semi_open*, *semi_closed* and *available* do not change over time. We have assumed that resources are needed until they have been obtained, and we have ignored the possibility that resources may be needed all the time, or for specific periods. Also, we have not allowed the generation of new resources within societies, the creation of new societies, the creation of new agents in societies, and the assignment of new roles (authority/control) to agents in societies. As a consequence, we disallow agents holding authority and control to leave societies. Furthermore, we give no justification of refusal of resources, denial of admittance in a society, or reasons for an agent to leave while broadcasting. Future work is needed to extend the framework to relinquish these assumptions. Finally, we have assumed that, whenever agents need to share knowledge, they do so through the explicit replication of this knowledge in their knowledge bases (abductive logic programs). As a result, when new members join or leave a society, the authorised agent need to notify all agents in the society. This of course may cause communication bottlenecks. In order to cut-down communication we could extend the framework to access – through observations – the objects available in the shared state managed by the environment (for example as in [17]). Such an object could be a *yellow pages* resource, maintaining information about available resources within different societies.

However, note that the in our approach agents do not need to share all their knowledge. In particular, agents might adopt different eligibility criteria. Indeed, the communication-based approach that we have taken allows encapsulation of the knowledge of agents and guarantees the privacy of agents in societies.

In our approach, artificial societies are not merely locations providing resources. Indeed, societies are characterised by a set of conventions, e.g. in deciding the admission of new members, which are held by the authority within a society, or in deciding when a request should be granted, which are held by the controller of that resource. Note that, for simplicity, we have assumed that all societies share the same such conventions, but different ones could be easily incorporated within the approach. For example, a controller could grant some resources only to members that have newly joined, or to members that have been part of the society for a specified period.

At a very general level, our work is influenced by the research programme outlined in [10], whereby artificial societies support and augment the activities of people in human societies. We share the idea of [10] of having electronic agents acting on behalf of people, and that these societies should be situated in the physical environment of real ones. However, we differ from that work in that we do not rely upon the *ownership* model that it advocates. Instead, we focus on the notion of *service*, viewing a service as a resource that is available in specific societies. Our approach is neutral as to which model of ownership is being adopted.

Our work also closely relates to [1] in that we use the classification scheme for artificial societies that it proposes. We have investigated how this classification scheme can be utilised by agents to reason about joining artificial societies, for the purposes of accessing resources in a flexible manner. We have also provided a computational framework that demonstrates the usefulness of this classification, by instantiating it with respect to a specific representation for agents.

Our computational approach extends existing work on the application of abductive logic programming to the resource re-allocation problem [12–14]. There, the same abductive logic programming approach that we have used is used to show how agents can negotiate the exchange of resources in a multi-agent system environment. Our work complements that approach by providing a framework where the accessing of resources is dependent on globally situated societies that control the resources. Future work is needed to provide a formal analysis, along the lines of the formal analysis in [13, 14], of the framework we have proposed. Some interesting properties to be analysed would be: *"An agent only belongs to a society if this is beneficial to the agent or the agent has responsibilities within that society"*, *"If a resource is available in some society and the agent is eligible to be a member of that society, the agent will obtain the resource in finite time"*.

Acknowledgements

This research was supported by the EU-funded project SOCS, [15]. The authors would like to thank the anonymous referees for helpful comments.

References

1. P. Davidsson, Categories of artificial societies, [9].
2. P. Dell'Acqua and L.M. Pereira, Preferring and updating in abductive multi-agent systems. [9].
3. P. Dell'Acqua, F. Sadri, F. Toni, Combining Introspection and Communication with Rationality and Reactivity in Agents, *Proc. JELIA'98*, U. Furbach, L. Farinas del Cerro eds., Springer Verlag LNAI 1489, 17-32, 1998.
4. T.H. Fung, R.A. Kowalski, The iff procedure for abductive logic programming. *Journal of Logic Programming* 33(2):151–165, Elsevier.
5. Global Computing, Future and Emerging Technologies Web-site, http://www.cordis.lu/ist/fetgc.htm (visited 10/06/2002).
6. A.C. Kakas, R.A. Kowalski, F. Toni, The role of abduction in logic programming. *Handbook of Logic in AI and Logic Programming* 5:235–324, OUP, 1998.
7. R.A. Kowalski, F. Sadri, From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, Baltzer Science Publishers, J. Dix, J. Lobo (Eds), volume 25(3,4):391–420, 1999.
8. A. Mamdani, J. Pitt, K. Stathis, Connected communities from the standpoint of multi-agent systems. *New Generation Computing Journal*, Special issue on New Challenges in Intelligent Systems, 17(4), T. Nishida (ed.), 1999.
9. A. Omicini, P. Petta, R. Tolksdorf (Eds.): *Proc. ESAW'01*, Springer Verlag LNAI 2203, 2001.
10. J. V. Pitt, A. Mamdani, and P. Charlton, The open agent society and its enemies: a position statement and research programme, [18], pp 67-87.
11. F. Sadri, F. Toni, Abduction with negation as failure for active and reactive rules. *Proc. AI*IA 99*, E. Lamma, P. Mello eds, Springer Verlag LNAI 1792, 49–60, 2000.
12. F. Sadri, F. Toni, P. Torroni, Logic Agents, Dialogue, Negotiation - an Abductive Approach, *Proc. AISB*, M. Schroeder, K. Stathis eds, 2001.
13. F. Sadri, F. Toni, P. Torroni, Dialogues for negotiation: agent varieties and dialogue sequences, *Proc. ATAL'01*, J.J. Maher ed., LNAI 2333, 405–421, 2001.
14. F. Sadri, F. Toni, P. Torroni, Resource reallocation via negotiation through abductive logic programming, Proc. *JELIA 2002*, to appear.
15. SOCS (Societies Of ComputeeS) Technical Annex, IST-2001-32530, 2001, `http://lia.deis.unibo.it/Research/Projects/SOCS/`.
16. K. Stathis, *Location-aware SOCS: The 'Leaving San Vincenzo' Scenario*, SOCS Technical Report, IST32530/CITY/002/IN/PP/a2, 2002.
17. K. Stathis, *A game-based architecture for developing interactive components in computational logic*, J. of Functional and Logic Programming, 2000 (1), MIT Press.
18. K. Stathis and P. Purcell (Eds.): Special issue on *LocalNets: Environments for Community-based Interactive Systems*, Journal of Telematics and Informatics, Elsevier, 18(1), 2001.
19. K. Stathis, O. deBruijn, and S. Macedo, Living memory: agent-based information managment for connected local communities, *Interacting with Computers*, to appear. (Also available in: http://dx.doi.org/10.1016/S0953-5438(02)00014-0).
20. F. Toni, Automated Information Management via Abductive Logic Agents ,[18], pp 89-104.