PROSOCS: a platform for programming software agents in computational logic

Kostas Stathis, Wenjin Lu
Department of Computing,
City University,
London EC1V 0HB, UK.
email: {kostas,lue}@soi.city.ac.uk

Antonis Kakas, Neophytos Demetriou
Department of Computer Science,
University of Cyprus,
Nicosia CY-1678, Cyprus.
email: {antonis,nkd}@cs.ucy.cy

Ulle Endriss
Department of Computing,
Imperial College,
London SW7 2AZ, UK.
email: ue@doc.ic.ac.uk

Andrea Bracciali
Dipartimento di Informatica,
Universita di Pisa,
Pisa 56127, Italy.
email: braccia@di.unipi.it

Abstract

We present the design and implementation of PROSOCS, a platform supporting the programming of software agents that have a *mind* and a *body*. The mind reasons autonomously and logically via a collection of logic theories with generic functionality, developed using various extensions of logic programming, and controls the overall behaviour of the agent via a *cycle theory* that specifies preferred patterns of operation. The body, on the other hand, provides sensors and effectors for the mind to be able to access and change the environment in which the agent is situated.

PROSOCS has been developed using Prolog - to program the functionality of the mind, Java - to program the functionality of the body, and the Peer-to-Peer system JXTA - to provide the functionality required for agent bodies to communicate and interact in an open distributed environment.

1 Introduction

PROSOCS (**Pro**gramming **So**cieties of **C**omputee**s**) is a platform that allows a developer to build software agents in global computing environments [GC, 2003]. The platform is being developed as part of SOCS (*Societies of Computies*), an EU research project investigating the computational and logical models of individual and collective behaviour of computational entities – referred to as *computees*. Computees are software agents; they are called computees to emphasise a strong computational logic component representing the cognitive and social capabilities of such agents.

The work for building PROSOCS is motivated by the observation that techniques for developing interactions in open computing environments result either in low-level implementations with no obvious logical characterisation, which are, therefore, not verifiable, or in abstract specifications possibly employing expressive logics with modalities, but which - as argued by [Rao, 1996] - have shed very little light in actual implementations of agent-based systems.

Another motivation of PROSOCS is based on the observation that although a number of multi-agent systems platforms and tools are available, for example see [Nwana *et al.*, 1999; Poslad *et al.*, 2000; Bellifemine *et al.*, 2001], the programmer is often left alone with the responsibility to develop the reasoning part of the agent from scratch. Alternatively, whenever a more sophisticated language is available for building the reasoning component [Bordini *et al.*, 2002], the programmer is often left alone again, this time to develop the communication and interaction of the agent in an open and distributed environment.

PROSOCS is a platform that offers to the programmer of an agent the reasoning and communication capabilities the agent needs to operate in an open environment for free. In this context, the programmer is only required to specify a set of logic programming theories describing the background knowledge necessary for the agent to operate within a specific environment.

In this work we focus on the development of the generic functionality offered by PROSOCS. We first outline the components of PROSOCS in section 2, including the logical model of agency in which the reasoning of PROSOCS agent is based. Then in section 3 we discuss the implementation of the logical model and its integration with the peer-to-peer platform JXTA. We compare our approach with existing work in section 4, and we conclude in section 5, where we also present our plans for future work.

2 Components of PROSOCS

2.1 The SOCS Reference Model

PROSOCS implements the reference model of SOCS shown in Fig. 1. This reference model assumes that an agent platform should contain a *discovery service* that allows agents to discover each other dynamically, a *communication module* to support inter-agent communication, a *society infrastructure* to provide verification of agent interactions and compliance to social rules, an *agent template* to facilitate the creation of agent instances, and an *agent management* module to help with the creation and management of agents in the platform.

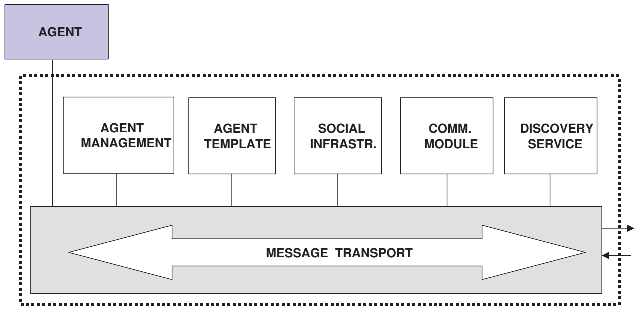

Figure 1: The SOCS Reference Model

Multiple instances of agents can be created from the agent template. These agents can then communicate with each other either privately or via the social infrastructure that will test for the compliance of their interactions to the social rules and protocols available in it. The social infrastructure currently used in PROSOCS is discussed separately in a companion paper [Alberti *et al.*, 2004].

2.2 The Agent Template

Creation of PROSOCS agents is based on the *agent template* whose design [Stathis *et al.*, 2002] builds upon previous work in multi–agent systems, in particular, the *head/body* metaphor described by [Steiner *et al.*, 1991] and [Haugeneder *et al.*, 1994], and the *mind/body* architecture introduced by [Bell, 1995] and more recently used by [Huang *et al.*, 2001].

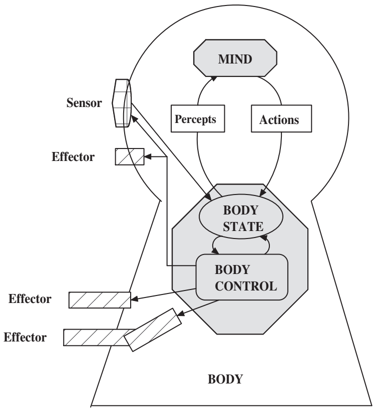

Figure 2: The architecture of a single agent in the special case where the agent's body has three effectors and one sensor.

As depicted in Fig. 2, the body senses what is external to it by using *sensors* that can access the current state of the external world. Information from the sensors is then stored in the *body state*, which is a data structure containing all the necessary information that enables the body to act. The body state is accessed by the *mind* which is effectively treated as a process that produces actions for execution, to enable the body to choose what to do next. More importantly, the mind and the body can function as co-routines, thus allowing the reasoning processes of the mind to be performed concurrently with body's action execution and sensing of the environment.

2.3 A Logical Model of the Mind

In SOCS we have developed a logical model of agency for building the mind of an agent that is called the KGP (**K**nowledge, **G**oals and **P**lan) model. As shown

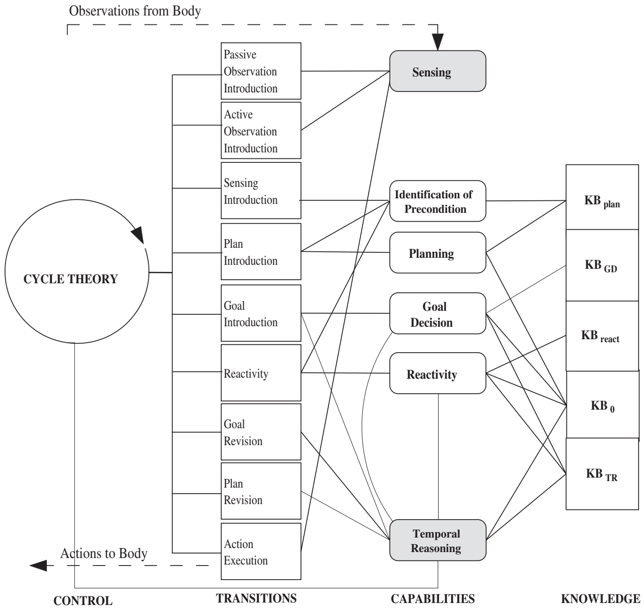

Figure 3: The KGP Model of Agency

in Fig. 3, a KGP agent is based on a set of *reasoning capabilities* that allow that agent to perform *planning, temporal reasoning, identification of preconditions of actions, reactivity* and *goal decision*, together with a *sensing capability* for the agent to perceive the environment in which it is situated. An agent has an *internal (or mental) state* on which its various capabilities operate. The capabilities are used by a set of *transition rules* describing how the internal agent changes, given input from the environment. A *cycle theory* is finally used to describe how transitions are sequenced for the agent to behave in a particular way.

Th internal state of a KGP agent is a triple $\langle KB, Goals, Plan\rangle$, where:

- **KB** describes what the agent knows of itself and the environment and consists of separate modules supporting the different reasoning capabilities. For example, KB_{plan} supports Planning,

KB_{gd} Goal Decision, etc. One part of the KB, called KB_0 reflects changes in the environment and is typically updated by the agent when it observes the environment through its sensing capability. We will assume that KB_0 is the only part of the knowledge of the agent that changes over time.

- **Goals** is a set of properties that the agent has decided that it wants to achieve by a certain time possibly constrained via some temporal constraints. *Goals* are split into two types: *mental goals* that can be planned for by the agent using its *Planning* reasoning capability and *sensing goals* that can not be planned for but only sensed, via the *Sensing* capability, to find out from the environment whether they hold or not. Goals are initially introduced via the Goal Introduction transition and the Goal Decision capability used within this.
- **Plan** is a set of "concrete" actions, partially time–ordered, of the agent by means of which it plans (intends) to satisfy its goals, and that the agent can execute in the right circumstances. Each action in *Plan* relies upon preconditions for its successful execution. These preconditions might be checked before the actions are executed. Preconditions are associated to actions via the reasoning capability for the *Identification of Preconditions*. Actions are added to the tree structure via the *Planning* reasoning capability and via the *Reactivity* reasoning capability.

The operation of an agent is then given by the application of transitions in sequences, which produce progressive changes over the state of the agent. In *KGP* these sequences are not determined by a one-size-fits-all cycle of behaviour, as in BDI architectures [Rao and Georgeff, 1995], but rather by reasoning with a cycle theory. We will see later that such a theory defines declaratively preference policies over the order of application of transitions. These policies are sensitive to changes in the environment and the internal state of an agent, and provide a means of declarative and intelligent control. Such declarative control is a highly novel feature of PROSOCS agents in that it allows the developer to build (behaviourally) *heterogeneous* agents based on different cycle theories.

3 The Implementation of PROSOCS

To realise the SOCS reference model we have chosen to implement agents as intelligent peers on top of the Peer-to-Peer JXTA Project [JXTA, 2004]. JXTA is suitable for our reference model in that it supports, amongst other, peer discovery protocols to implement facilities of the discovery service, facilities for message transport and structuring via a a pipe binding and resolver protocols, as well as facilities to name peers for agent management.

3.1 Implementation of the Body

When an agent is created in PROSOCS, the platform creates a generic body which uses a JXTA API to import sensors (called listen and see) and effectors (called speak and do). The body then creates a mind (whose details will be described shortly) and starts a new thread of control whose execution is based on the pseudo-code below:

```
void bodyControl() {
do {
  nextAction = askActionFromMind();
  if (nextAction != null)
    switch (isActionType(nextAction)){
     case SENSING: doSensors(nextAction);
     case COMMUNICATIVE: doSpeak(nextAction);
     case PHYSICAL: doEffectors(nextAction);
    }
  nextPercepts = sensors.passiveObservation();
  if (nextPercepts != null)
    tellMind(nextPercepts);
} while (!stopped);}
```

The body asks the mind for an action to be executed. This call activates the cycle theory to execute the next transition, resulting in a change of the mind's state. The body then checks if the transition has generated a new action and, if it has, the body carries it out. At this point, the body will also passively observe to see if there are any new events that are happening in the environment. If there are any, the body will inform the mind through as input to mind's percepts. This process goes on forever, until the agent is stopped.

3.2 Implementation of the Mind

The language in which the mind of a PROSOCS agent is specified comprises of the language of *Logic Programming with Priorities* (LPP) [Dimopoulos and Kakas, 1995] and the language of *Abductive Logic Programming* (ALP) [Kakas *et al.*, 1993] combined with the *Abductive Event Calculus* [Shanahan, 1989]. We build the mind using SICStus Prolog [SICStus, 2000] and the bidirectional Java-Prolog interface Jasper it provides; Jasper is used by the body to exchange information with the mind.

The State

We represent KB_0 as assertions of the form:

- `executed(Action, Time)` – records that `Action` has been executed at `Time`.
- `observed(Property, Time)` – represents that `Property` has been observed to hold at `Time`.
- `observed(Agent, Action, Time)` – states that an `Agent` has been observed to execute an `Action` at a `Time`.

Goals and actions are organised as a tree with goals and actions as nodes. Goals are represented by assertions of the form:

`goal(Goal, Parent, TempCs).`

`Goal` and `Parent` are both of the form `(Fluent, Time)`, indicating that the agent is trying to achieve the goal specified in the `Fluent` at a `Time` (`Time` is a non-negative integer constrained by a the list of temporal constraints `TempCs`). Similarly, actions are represented by assertions of the form:

```
action(Action, Parent, Preconds, TempCs).
```

`Action` is a pair `(Operator, Time)`, whose `Operator` must be executed at a `Time` (constrained as before by a list of temporal constraints `TempCs`) provided that the list of preconditions `Preconds` is satisfied in the current state of the agent.

Proof-Procedures, Capabilities, & Transitions

To reason with priorities in PROSOCS we use Gorgias [Gorgias, 2003], an interpreter that implements a formal proof for argumentation with context-sensitive attacking relation which is sound and complete with respect to the LPwNF semantics [Kakas *et al.*, 1994]. Gorgias is used to develop the Goal Decision capability and the cycle theory of the system. Similarly, for reasoning abductively, PROSOCS uses C-IFF [C-IFF, 2003], an interpreter that implements both an extension and a refinement of the IFF proof-procedure for abductive logic programming proposed by [Fung and Kowalski, 1997]. C-IFF is used to implement the capabilities of Planning, Reactivity, Temporal Reasoning, as well as a Constraint Solver.

On top of the proof procedures for preference and abductive reasoning we then build the various capabilities of the mind. For example, to implement the Goal Decision capability, we use a program of the form:

```
goal_decision(Goals):-
      findall(G, gorgias-solve(G, _), Gs),
      list_to_ordered_set(Gs, Goals).
```

This eventually calls Gorgias to select the list of goals to be considered next. Similarly, the state of the mind is then changed by transitions rules (which are formally specified [Kakas *et al.*, 2003]) based on the various capabilities and additional functionality for changing the agent's internal state.

Declarative Control

The cycle theory part of the mind, implemented in LPP, regulates the operation of the agent, by using preference reasoning to choose which transition to be executed next. For example, we may choose to build an agent that gives preference to responding to communicative acts received by another agent as a result of the Passive Observation (PO) transition. What we need to specify here is that after a PO transition, preference should be given to the Goal Introduction (GI) transition. We specify this as a priority logic rule of the form:

$$R^{PO}_{GI|*} : r_{PO|GI}(S,O) > r_{PO|*}(S,O) \leftarrow comm\_msg(O)$$

which in Gorgias syntax is written as:

```
ct_rule(prefer(step('GI',_), step(_,_)),
    prefer(step('GI',_), step(_,_)), []) :-
        last_transition('PO', O),
        comm_msg(O).
```

The Goal Decision capability then is used to decide the response to the observation of the communicative action held in `O`.

4 Related Work

At a first glance, the KGP model of agency used in PROSOCS and the classical BDI model [Bratman *et al.*, 1988] appear to be similar. However, the models are quite different in detail. One major difference is that KGP is not based on a modal-logic approach to represent an agent's beliefs but instead it is based on a non-monotonic computational logic that supports defeasible reasoning for the knowledge of agents. As we saw earlier, KGP does not have a fixed cycle and like BDI allows plans to be generated dynamically (as part of the reasoning process) and statically through the use of plan libraries. For lack of space, we refer the interested reader to [Kakas *et al.*, 2003], for an up-to-date review of KGP with the most important models of agency available in the literature, including BDI.

We examine next an important effort in the development of agent platforms, namely, the standardisation work of FIPA. This proposes a reference model that is based on a Directory Facilitator (DF) providing "yellow pages" services to other agents. A number of FIPA compliant platforms, most notably JADE [Bellifemine *et al.*, 2001], FIPAOS [Poslad *et al.*, 2000], ZEUS [Nwana *et al.*, 1999], and 3APL [3APL, 2004], rely on a DF of this kind. PROSOCS differs from these platforms in that the PROSOCS reference model relies on a discovery service, so that when an agent is started is being discovered by JXTA automatically. Our use of JXTA pushes the handling of an agent's presence at the lower-level implementation of the platform, so that the agent developer does not need to think about registering with a yellow pages service at the application level (as a notion we find DF being closer – at least conceptually – to the social organisation of a MAS application and not a component of the lower-level platform).

In PROSOCS an agent is treated as a first-class object in the sense that the developer can start an agent and inherit a set of tools supporting the development of both a reasoning component and the interaction with the environment for free. Most of the FIPA compliant platforms referred to in the previous paragraph support only the interaction of the agent with the environment, while the programmer has to develop from scratch the reasoning component. To the best of our knowledge, only the 3APL platform [Hindriks *et al.*, 1999; 3APL, 2004] provides a set of tools that support the reasoning capabilities of the agent (for the mind), including a deliberation cycle. However, like all FIPA compliant platforms, the 3APL platform does not have a society infrastructure [Alberti *et al.*, 2004] to test for conformance of protocols as we have in PROSOCS.

The IMPACT platform [Arisha *et al.*, 1999; Subrahmanian *et al.*, 2000] too treats agents as first-class objects (in the sense discussed earlier), further facilitating the creation, deployment, interaction, and collaborative aspects of applications in a heterogeneous, distributed environment. As with FIPA compliant platforms, IMPACT relies on a *yellow pages server* for keeping information about agents. However, in IMPACT, when an agent is deployed the registration is done automatically by the runtime environment of the platform, and thus an agent does not have to register explicitly by executing a communicative act. The platform also provides an Agent Development Environment for creating, testing, and deploying agents. Unlike PROSOCS, where agents are built from scratch by assuming a logic programming approach, an IMPACT agent may be built on top of an arbitrary piece of software, defined in any programming language. IMPACT uses deontic concepts to represent action and action policies which in KGP correspond to ALP integrity constraints, while the system relies on an interval–based logic for temporal reasoning rather than the Event Calculus approach of PROSOCS. However, IMPACT does not have a flexible cycle theory and thus cannot support (behaviourally) heterogeneous agents, as we can in PROSOCS.

IndiGolog [Giacomo *et al.*, 2001] is a high-level programming language for robots and intelligent agents that supports on-line planning and plan execution in dynamic and incompletely known environments. Programs may perform sensing actions that acquire information at runtime and react to exogenous actions. IndiGolog is a member of the Golog family of languages [Levesque *et al.*, 1997] that use a Situation Calculus theory of action to perform the reasoning required in executing the program. Instead in PROSOCS we rely on abductive logic programming and logic programming with priorities combined with an Event Calculus approach to program an agent. Moreover, goals cannot be decided dynamically in IndiGolog, instead in PROSOCS change according to the patterns specified in a goal decision theory.

An extension of Golog called ConGolog [Giacomo *et al.*, 2000] has also been introduced, adding support for concurrent processes with possibly different priorities, interrupts, and exogenous events. However, like earlier planning-based systems, Golog and ConGolog, assume an off-line search model. To overcome the limitation of off-line search, IndiGolog uses the notion of exogenous actions assuming that there is a concurrent process executing these actions outside the control of the agent. This assumption implies that it is up to the programmer of an agent to interface IndiGolog to the rest of the agents of an application domain.

The logic programming language Go! [Clark and McCabe, 2003] is descendant of the multi-threaded symbolic programming language April [McCabe and Clark, 1995], with influences from IC-Prolog II [Chu. and Clark, 1993] and L&O [McCabe, 1992]. Go! has many features in common with Prolog, particularly multi-threaded Prologs, and provides types, higher–level programming constructs, such as single solution calls, iff rules, and the ability to define 'functional' relations as functions. However, as with IndiGolog, Go! does not directly rely on any specific agent architecture or agent programming methodology, hence the system does not treat agents as first class objects. As a result, facilities such as planning, temporal reasoning, and preference reasoning are not available but the programmer has to build them from scratch; the system though provides library modules, so that components can be reused once they are developed.

5 Conclusions and Future Work

We have presented PROSOCS, a platform supporting the programming of software agents in open computing environments. The contribution of the platform is that it integrates novel computational logic techniques to build a mind for an agent, and advanced peer-to-peer computing techniques to allow an embodied mind to interact and communicate with an open and distributed environment. A modular approach allows an agent to be built in terms of components, such as the mind and the body, which in turn consist of additional components, such as the mind's cycle theory and knowledge, or the body's control, sensors, and effectors.

For future work we plan to experiment with the platform on global computing applications and test the PROSOCS agents and social infrastructure with complex problems. We also plan to extend PROSOCS to support interactions of agents with the environment, other than the current inter-agent communication support.

Acknowledgments

We want to thank the anonymous referees for their comments to a previous version of this paper. This research has been funded by the European Union as part of the SOCS project (Societies Of ComputeeS), IST-2001-32530.

References

[3APL, 2004] 3APL User Guide (visited 19/01/2004). http://www.cd.uu.nl/3apl/download.html, 2004.

[Alberti *et al.*, 2004] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Compliance verification of agent interaction: a logic-based tool. In J. Müller and P. Petta, editors, *Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation" (AT2AI-4 – EMCSR'2004 Session M)*, Vienna, Austria, April 13-16 2004.

[Arisha *et al.*, 1999] K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, March/April 1999.

[Bell, 1995] J. Bell. A Planning Theory of Practical Rationality. In *Proceedings of AAAI'95 Fall Symposium on Rational Agency*, pages 1–4. AAAI Press, 1995.

[Bellifemine *et al.*, 2001] F. Bellifemine, A. Poggi, and G. Rimassa. JADE: a FIPA2000 compliant agent development environment. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 216–217. ACM Press, May 2001.

[Bordini *et al.*, 2002] R.H. Bordini, A. L. C. Bazzan, R. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III*, pages 1294 – 1302, Bologna, Italy, July 15–19 2002. ACM Press.

[Bratman *et al.*, 1988] M.E. Bratman, D.J. Israel, and M.E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4, 1988.

[C-IFF, 2003] C-iff Web Site, (visited 20/12/2003). http://www.doc.ic.ac.uk/ ue/ciff/, 2003.

[Chu. and Clark, 1993] D. Chu. and K. L. Clark. IC-Prolog II: A Multi-threaded Prolog System. In *Proceedings of the ICLP-93 Post-conference Workshop on Concurrent, Distributed and Parallel Implementations of Logic Programming*, 1993.

[Clark and McCabe, 2003] K. Clark and F. McCabe. Go! for multi-threaded deliberative agents. to appear in Annals of Mathematics and AI, also available http://www.doc.ic.ac.uk/ klc/gowp.html, 2003.

[Dimopoulos and Kakas, 1995] Y. Dimopoulos and A. C. Kakas. Learning non-monotonic logic programs: Learning exceptions. In *Proceedings of the 8th European Conference on Machine Learning*, 1995.

[Fung and Kowalski, 1997] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.

[GC, 2003] Global Computing, Future and Emerging Technologies (visited 10/06/2003). http://www.cordis.lu/ist/fetgc.htm, 2003.

[Giacomo *et al.*, 2000] G. De Giacomo, Y. Lesperance, and H. J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[Giacomo *et al.*, 2001] G. De Giacomo, H. J. Levesque, and S. Sardia. Incremental execution of guarded theories. *ACM Transactions on Computational Logic*, 2(4):495–525, 2001.

[Gorgias, 2003] Gorgias User Guide (visited 20/12/2003), 2003. http://www.cs.ucy.ac.cy/ nkd/gorgias/.

[Haugeneder *et al.*, 1994] H. Haugeneder, D. Steiner, and F.G. McCabe. IMAGINE: A framework for building multi-agent systems. In S. M. Deen, editor, *Proceedings of the 1994 International Working Conference on Cooperating Knowledge Based Systems (CKBS-94)*, pages 31–64, DAKE Centre, University of Keele, UK, 1994.

[Hindriks *et al.*, 1999] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[Huang *et al.*, 2001] Z. Huang, A. Eliens, , and P. de Bra. An Architecture for Web Agents. In *Proceedings of EUROMEDIA'01*. SCS, 2001.

[JXTA, 2004] Project JXTA (visited 19/01/2004). http://www.jxta.org/, 2004.

[Kakas *et al.*, 1993] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[Kakas *et al.*, 1994] A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics for logic programs. In *Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy*, pages 504–519, 1994.

[Kakas *et al.*, 2003] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. A logic-based approach to model computees. Technical report, SOCS Consortium, 2003. Deliverable D4.

[Levesque *et al.*, 1997] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[McCabe and Clark, 1995] F.G. McCabe and K.L. Clark. April: Agent process interaction language. In Wooldridge M. J. and Jennings N. R., editors, *Intelligent Agents*, pages 324–340. LNCS, Vol. 890, Springer Verlang, 1995.

[McCabe, 1992] F. G. McCabe. *Logic and Objects*. Prentice Hall, 1992.

[Nwana *et al.*, 1999] Hyacinth S Nwana, Divine T. Ndumu, Lyndon C. Lee, and Jaron C. Collis. ZEUS: a toolkit and approach for building distributed multi-agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 360–361, Seattle, WA, USA, 1999. ACM Press.

[Poslad *et al.*, 2000] S. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS agent platform: Open Source for Open Sstandards. In *Proceedings of PAAM'00*, pages 355–368, 2000.

[Rao and Georgeff, 1995] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the 1st International Conference on Multiagent Systems, San Francisco, California*, pages 312–319, San Francisco, CA, June 1995.

[Rao, 1996] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer-Verlag, 1996.

[Shanahan, 1989] M.P. Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1055–1060, 1989.

[SICStus, 2000] SICStus prolog user manual, release 3.8.4, May 2000. http://www.sics.se/isl/sicstus/.

[Stathis *et al.*, 2002] K. Stathis, C. Child, W. Lu, and G. K. Lekeas. Agents and Environments. Technical Report Technical Report IST32530/CITY/005/DN/I/a1, 2002.

[Steiner *et al.*, 1991] D. E. Steiner, H. Haugeneder, and D.E. Mahling. Collaboration of knowledge bases via knowledge based collaboration. In S. M. Deen, editor, *CKBS-90 — Proceedings of the International Working Conference on Cooperating Knowledge Based Systems*, pages 113–133. Springer Verlang, 1991.

[Subrahmanian *et al.*, 2000] V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, and R. Ross. *Heterogenous Active Agents*. MIT-Press, 2000.