ATAL'98 Panel statements

Original set of questions by John-Jules Meyer:

- is there a need for special agent-oriented languages anyway? (Or can every 'agent-oriented' thing be done as easily with familiar paradigms: OO, LP, ...)
- current state of agent-oriented languages: Agent0, PLACA, AgentSpeak, ...; is there a need for new agent-oriented languages?
- what makes a language agent-oriented? Should there be an explicit mention (by means of language primities or so) of BDI-like features in the language?
- the need for formal semantics and formal methods for specification & verification of agent programs?
- what are the expectations for the future of agent languages? (In 10 years as popular as OO languages nowadays?)

Statement by Jean-Pierre Briot:

Regarding the whole set of questions asked, I would like to discuss the four following points.

- First, I think that when one mentions about agent languages, there are different kinds of language concerns, at least:
  - Programming languages for expressing the algorithmic part of the agent.
  - Knowledge representation languages for expressing the knowledge part (acquaintances, mental state...) - in case of cognitive agents.
  - Communication/coordination language between agents - message passing (objects, actors) is too unconstrained to be called a communication language, but proposals like KQML or FIPA ACL get closer.

  I do not think we need (and could find) one agent language for all of these different language concerns. I think we should rather build upon existing high level programming languages (specially object/actor-oriented as they provide a good basic structuring model - see for instance the study by Les Gasser and myself published in the book "DAI: Theory and Praxis", Kluwer, 1992). Also please refer to current special series on Actors and Agents in IEEE Concurrency Journal.

We may then use different components for different concerns: knowledge base, perception, reasoning (different kinds, e.g. planning, constraints, case-based reasoning...), communication, coordination... I think agent languages like Agent0 or BDI are somehow both too general (for a given application) and too specific (for various applications). I do not mean rejecting works like Agent0. Its idea of a control loop tackles well what I believe is one of the fundamental machinery of an agent, i.e., the balance between reaction (to communication and perception) and pro-action (based on mental states for cognitive agents) - in other words selection and adaptation of actions. But I do not want to necessarily express everything in terms of Agent0 or BDI.

Then, having different components within a single agent means being able to compose, schedule and coordinate them. For some architectural proposal in that direction, see for instance work by Zahia Guessoum on the DIMA modular agent architecture and its meta-control through a state/transition model (published in the proceedings of MAAMAW'96).

- Second, I think that probably the most fundamental difference between cooperative agents (multi-agent) and conventional programming (distributed objects) is more at the level of design than at the level of programming language. Compared to conventional (object-oriented) design methodologies, identification of agents ("finding the agents") depends heavily on the explicit identification of relational and organizational behaviors, in complement to basic domain behaviors. See for instance the Cassiopee methodology proposed by Anne Collinot and Alexis Drogoul (published in the first issue of the new Journal on Autonomous Agents and Multi-Agent Systems, Kluwer).

  Needless to say, such methodologies should complement conventional object-oriented methodologies (with an organizational dimension).

- Third, one thing agent-oriented programming may learn from the object-oriented community is the notion of "design pattern". - Actually, Robert Tolksdorf gave a presentation at ATAL'98 where he also advocated the use of design patterns for agents. It is interesting to note that we both reached, although completely independently, similar conclusions. -

  The idea is in capitalizing on design expertise of the community, that is the mapping of design solutions to patterns of problems. The key aspect is to be able to well document for various classes of problems what patterns may apply, or do not apply, and may collaborate with other known patterns. For instance it would be valuable to well document known usages of well identified coordination protocols (e.g., variations of contract net protocol). And also bring to the surface redundant patterns of interaction which are buried deep into applications code.

Catalogs of patterns should be linked to the methodology and its environment. That is, from a given analysis (see my second point) identifying requirements for coordination between some agents, help at finding known patterns and protocols of coordination for actual implementation.

- Last, for me one of the main motivations of multi-agent systems is when control must be distributed. I mean, because of logical, physical, or/and temporal distribution. On large scale distributed architectures, fault tolerance becomes crucial as one cannot exclude the possibility of failures (processor or/and network).

  I believe that in order to be able to run multi-agent systems on a large scale and not just on single (or only a few) machine(s), we need to use distributed protocols for ensuring maximal reliability (e.g., by replicating critical agents and communications). Note that, in the case of reactive agents, some degree of redundancy is often built-in within organizations and interaction protocols, e.g. in the case of ant nests. But cognitive agents usually completely lack such properties. One example of application field where fault tolerance is critical is crisis management and decision systems.

  What I see as a promising direction is the possibility for the agents themselves to dynamically adapt the protocols to the ongoing situation (perception by agents of the environment and the possibly changing requirements/priorities from the users, as well as representation by agents of the problem solving process). The reason is that distributed protocols for fault tolerance may be costly to support and adapting them dynamically may improve the reliability and the use of resources in a large scale agent system.