Implementing Multi-Agent Systems: Languages, Frameworks, and Standards [Extended Abstract]*

K. Decker
U. Delaware
decker@cis.udel.edu

T. Finin
U. Maryland
Baltimore County
finin@umbc.edu

C. Manning
MIT
caroma@ai.mit.edu

M. Singh
North Carolina
State University
mpsingh@eos.ncsu.edu

J. Treur
Vrije Universiteit
Amsterdam
treur@cs.vu.nl

As multi-agent systems research nears the end of its second decade, researchers have moved beyond stand-alone, one-off systems and have begun to create the software infrastructure for quickly creating new, highly interoperable systems. However, the desire for seamless interoperability ("open" systems) brings with it the push to standardize on agent communication languages and related agent service facilities. Although a host of theoretical and practical controversies surround the specification of such standards, some shared, guiding principles have emerged.

Many researchers focus their work on the internal structures of agents (architecture), while others focus more on the activities and structures between agents (organization). The first principle is the realization that most principles can be recast at both the intra- and inter- (architectural and organizational) levels. The second principle is a drive toward the reuse of models and particular behaviors both conceptually, in reusable implementations, and in the creation of generic infrastructure components. The third principle revolves around how to structure individual or multi-agent knowledge and behaviors that includes techniques for composition, layering, and abstraction. The final guiding principle is the centrality of the interoperability question. After all, what would a multi-agent system be unless there are in fact multiple agents interoperating with one another?!

Methodologically, research in the area of agent frameworks and languages has been pursued in both a theory-driven and application-driven manner. Often there can be significant interaction between theory and practice, as in the development of BDI-based systems (Bratman 1987; Rao & Georgeff 1995). Although there are as yet no complete solutions, multi-agent system development methodologies will be important for the software engineering of commercial multi-agent systems (Brazier et al. 1997). Methodological techniques share certain methods. For instance, most researchers place their work in the context of a shared design ontology that defines what are the important questions to consider and answer in the design of an individual agent (e.g. planning and scheduling subcomponents, goal specification mechanisms, mappings from desires to intentions, etc.) or an agent organization (e.g. authority relations, cooperative vs. self-interested stances, joint intentions (Cohen & Levesque 1990; Grosz & Sidner 1990), social commitments (Castelfranchi 1993)). Even if the answers to such questions are very different, the set of issues indicated by these questions can represent different conceptual approaches. A second methodological issue important to many researchers is the controversial issue of designing appropriate agent communication languages. No matter where a researcher stands on the issue of agent communication language standardization, the centrality of the interoperability issue causes all serious multi-agent systems researchers to at least consider the agent communication language issue. KQML (Knowledge, Query, and Manipulation Language) (Finin et al. 1994) is widely used at a syntactical and very broad semantic level, but the detailed semantics vary wildly between research groups. A third shared feature of methodological approaches is a preoccupation with how to specify agent knowledge and behaviors in a semantically meaningful and practically reusable way (Decker & Sycara 1997). Fourth, methods for verifying and validating multi-agent systems appear in many lines of work as common methodological components. Finally, for each of these varied concerns, graphical representations have been developed and have proven to be helpful.

In current practice, many research systems, drawing from both theoretical and application-oriented perspectives, have been implemented. However, few of these systems see day-to-day use even in non-industrial settings. Most such systems are implemented using conventional programming languages (Java, C++, Lisp, and even Perl5)

---

*This extended abstract is a short summary of discussions held by the Working Group on Implementing Multi-Agent Systems held at the International Workshop on Multi-Agent Systems, MIT Endicott House, in October of 1997. All errors are the fault of K. Decker.

rather than in languages developed exclusively for multi-agent systems programming. Partial reuse of agent subcomponents, and even whole agents (especially general service-oriented agents such as agent name servers, matchmakers, facilitators, and brokers) has been demonstrated many times (though mostly *within* particular research groups). Agent communication languages are common, but usage is fairly idiosyncratic within groups (e.g. many variations on KQML). Clearly building good systems for simple applications is feasible and is no longer an interesting research question. Similarly clearly, there are frameworks and tools that are being used over extended periods of time for multiple projects. However, not all available theoretical results have been exploited by application or general framework designers, and there exist some applications that are in need of supporting theory.

Several research issues in the general area of agent languages and frameworks are being actively pursued. The first is the question of just how similar agent communication languages need to be to human languages. Most of the current popular approaches are based on speech act theory (Searle 1969), but of course there is no necessity for artificial languages to be limited by human prototypes. A second important issue is how agents should some to decide how much trust to place in others, and whom to believe. This general issue encompasses varied ideas ranging from deciding between cooperativity or self-interest to tracking the reliability of certain information and highly practical research into agent authentication methods. Another issue is how best to facilitate industrial acceptance of agents, which is made more complex by the use of the term "agent" by some commercial software developers in contexts only very tenuously related to any sense of the term as used in the multi-agent systems field. As infrastructure matures, an issue arises in how the developing multi-agent infrastructure is integrated with existing networking technology. An example of this is how the use of UDP or other network transport mechanisms (as opposed to TCP) would impact popular agent communication languages such as KQML. A fifth issue is that of agent communication language *evolution*—even if agents could interoperate initially with some standard, how could the agents themselves automatically extend communication and content languages appropriately. Besides the agent communication language issue, interoperability of internal agent architectural components is an active issue. With the necessity in many applications of agents that plan, schedule, reason about beliefs, form coalitions, negotiate, etc. comes the desire to share and reuse such reasoning components. Finally, multi-agent systems language designers face the need to overcome the impression of industrial programmers that Java is sufficient for all agent development needs.

For the language and framework researcher, it is not so much the rise of a "killer multi-agent app" that is desired as the rise of a true hit toolkit to spread use and attract interest.

Work on standards proceeds along several different fronts. One level of standards is for standard agent roles and capabilities. For example, how do agents find one another? Such high level infrastructure can be built now, relying on well-understood public services, such as agent name servers, matchmaker/yellow-page services, service brokering, mediators, facilitators, translation services, and so on. Another level of standards is at the transmission protocol level, one easier to agree on than that of core agent communication languages. Most proposed agent communication languages have at least some notion of a separate, extensible (or retargetable) "content" language where domain-specific information is communicated. These can be standardized on, even informally, within common domains (for example, the agent services ontology implied above of broker, matchmakers, mediators, etc.). Finally, there have been some proposals for tagging data on the web with appropriate meta-data tags so that web-searching agents would have an easier time of drawing out the appropriate information from web pages formatted more for human readability than for consumption by autonomous software agents.

Finally, many controversies still range in the agent language and framework area. The semantics of agent communication languages is a very controversial area. Arguments range from those who want languages simple enough to be easily implemented and validated, to those who desire the capabilities of much more complex agent communication languages that can be extended on the fly. Another question is how separate the semantics of an agent communication language can be from the internal architecture of the agents. Another controversy revolves around the usefulness of testbeds for agent implementation research. Now that non-simulated networks are so commonplace and easy to achieve, there is a question of how much trust to put into limited testbeds. Even a well-defined testbed problem can sometimes introduce regularities (or irregularities) that are not present in the real problem domain, and steps must be taken that this does not effect the resulting research results. On the other hand, testbeds allow a remarkable amount of comparability in results between different groups and approaches, and also sometimes the ability to do careful, well-instrumented, paired-response studies.

Another controversy in the implementation area is the impact of so-called "mobile" agents—in their purest form, software agents that can suspend execution on one platform, transfer their code, and resume execution on a remote platform. While the technology to do this already exists, the question

remains as to what it is useful *for*. Tremendous numbers of test systems have been built both in research and commercially, but few compelling applications have appeared. The two most common involve hand-held computing devices (PDAs) that are only intermittently connected to the network. A mobile agent can clearly transfer off such a device to do work, ad transfer back later. The second involves running complex data-dependent code (say, involving a large database) without having to transfer the data over the network—instead, the agent goes to the data. The non-compelling nature of these examples appears when we consider how these things actually get done: typically, the agent must be running "on" some software platform that is providing some security and otherwise protecting the underlying machine from rogue agents; the agent then must transfer to another such platform. These platforms can be considered as nothing more than *NON*-mobile agents, and the "mobile" agents are nothing more than fairly complex messages that contain not only a description of a task to do, but also specific code to carry out that task.

Another perennial controversy involves the structure of the agent's internal architecture: how many layers? Should it be application dependent? Other arguments ensue over cooperatively designed multi-agent systems, totally self-interested individual agents, and useful ways at arriving at socially constructed mutual beliefs. Yet another architectural question aims at scalable models: can an "organization" (made up of many agents) be thought of itself as a kind of "agent"? Clearly organizations (take human organizations as an example) have some agent-like properties, for example the ability to make commitments. An organizational commitment to deliver a particular product is clearly not a commitment of any single individual in the organization. On the other hand, it appears that some of the simple models that we typically use to understand the actions of individual agents (such as BDI models) may break down somewhat when trying to cope with the observed actions of large organizations operating mostly via standardized operating procedures and responses. An "organizational goal" or desire is clearly not a simple function of the goals of the organizations individuals, nor are the goals of the individuals a simple function of the organizational goals.

Finally, agent language work brings a set of controversies related to the purposes of such languages. Some are true "programming" languages, while others might be more properly characterized as "modeling" languages. Should work in the field be more organized towards providing toolkits built in existing languages (e.g. Java), or in *extending* existing traditional programming languages, or in fact creating totally new languages? An orthogonal controversy revolves around the need for different languages at different levels: building individual behaviors of an agent, building the agent's internal architecture, building inter-agent communication languages and coherent coordination protocols, and building multi-agent organizations.

The current prospects and outlook for this segment of the field is good. Clearly much larger and more complex systems are being constructed as time goes on. These bigger and better systems are being applied to new and different application areas, moving from Internet information retrieval to more complex information gathering and integration applications. Areas such as manufacturing are drawing fresh attention in the multi-agent field. New software tools, especially graphical specification interfaces, are being built and will be very important for spreading the use of multi-agent techniques and enlisting new allies, especially among industrial users. New methodologies for multi-agent system software engineering promise quicker and more consistent design results. Finally, while standards are still emerging, the concentration on a few competing possibilities has and will continue to enhance reusability and interoperability.

References

Bratman, M. 1987. *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard U. Press.

Brazier, F.; Keplicz, B.; Jennings, N.; and Treur, J. 1997. Formal specification of multi-agent systems: a real-world case. In *Proceedings of the 1st Intl. Conf. on Autonomous Agents*, 25–32.

Castelfranchi, C. 1993. Commitments:from individual intentions to groups and organizations. In Prietula, M., ed., *AI and theories of groups & organizations: Conceptual and Empirical Research*. AAAI Workshop. Working Notes.

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artificial Intelligence* 42(3):213–261.

Decker, K. S., and Sycara, K. 1997. Intelligent adaptive information agents. *Journal of Intelligent Information Systems* 9(3):239–260.

Finin, T.; Fritzson, R.; McKay, D.; and McEntire, R. 1994. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management CIKM'94*. ACM Press.

Grosz, B., and Sidner, C. 1990. Plans for discourse. In Cohen, P.; Morgan, J.; and Pollack, M., eds., *Intentions in Communication*. MIT Press.

Rao, A., and Georgeff, M. 1995. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, 312–319. San Francisco: AAAI Press.

Searle, J. R. 1969. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge U. Press.