Foreword

The increasing cooperation and convergence of various kinds of computing entities: computers, but also cell phones, personal digital assistants, house appliances..., is deeply changing the way we were traditionally considering computers and software. The size, the increasing complexity and the high evolutivity of future applications (e.g., community-support, collaborative work, supervision...) make nearly impossible a centralized and explicit control by the programmer. It is thus natural to delegate more autonomy and initiative to the various software modules and to provide them with cooperation abilities. Multi-agent systems have been proposed as a conceptual framework to help at designing and constructing such large scale autonomous and cooperative computing systems.

As a programming paradigm, we can analyze multi-agent systems through the history and evolution of programming. We may for instance observe three dimensions of progress: (1) higher levels of abstraction for entities processed or/and exchanged - from bits, and then objects and messages, up to agents, intentions and plans ; (2) later binding time, i.e., deferring the decision for what actual code is to be executed - from procedure call, and then method invocation, up to action selection by an autonomous agent ; (3) more flexible coupling between software modules - from procedures, and then objects, components and events, up to knowledge-based organizations of agents. Multi-agent systems may thus be seen as an important evolution of programming.

There is currently growing experience on how to construct multi-agent architectures and platforms, based on more conventional technology, objects, Java and components. Interoperability has recently been an important concern, notably, through the FIPA Agent Communication Language standard, which builds up on object interoperability standards (such as CORBA), and extends their levels of abstraction (e.g., expliciting interaction patterns and protocols, as well as ontologies of knowledge, within a communication). We also see now various attempts and studies for using novel approaches and technologies such as Model Driven Architecture (MDA), and Aspect-Oriented Programming (AOP) to the design and construction of multi-agent systems. Last, the inherent decentralization and autonomy of multi-agent systems also raise new questions about dependability and security.

On the software engineering perspective, we believe the challenges are perhaps even bigger. The decentralized, autonomous and adaptive nature of agents, combined with the large scale, mobility and general dynamicity of their environment support (e.g., ad-hoc networks, with associated security and robustness concerns), make it difficult to rely on traditional assumptions of software predictability. In other words, the traditional "defensive/pessimistic" approach - statically safeguarding as much as possible the behaviour of the program (through specifications, types, assertions...) -, reaches its limits and should be complemented (and *not replaced !*) by a "proactive/optimistic" approach - providing the agents with abilities to adapt to unexpected individual and collective behaviors. Another concern is also to include the users, as agents, from the initial stages of the design. Indeed, our ultimate goal is to provide a symbiotic collaboration between artificial agents and human agents, as opposed to confining users, either as supervisers with explicit control, or as end users with little initiative.

For designing practical methodologies, there is still a debate in the community if agent-oriented methodologies should be filiated to current (object-oriented) methodologies, or should be deeply restated (for instance by focusing analysis on social concepts such as roles and organization, rather than on individual objects or agents). In any case, future methodologies will still need steps (such as analysis, design, modeling, measurements. . .) as well as related techniques (such as requirements analysis, meta-modeling, notations, metrics. . .). As a consequence, there is currently much activity in studying how such steps or techniques may be partly reused from current technology, adapted, or completely rethinked.

As a conclusion, in order to achieve these challenges, we need to organize a research community at the crossing of software engineering, programming and multi-agent systems, with a concern for scalability of solutions. This book, the third volume of the very good series on "Software Engineering for Multi-Agent Systems", includes several important studies and proposals along the various perspectives we just sketched, and thus represents a very good contribution to that research agenda.

Jean-Pierre Briot
Paris, November 2004