Preface

Intelligent software agents are a unique generation of information society tools that independently perform various tasks on behalf of human user(s) or other software agents. The new possibility of the information society requires the development of new, more intelligent methods, tools, and theories for the modeling and engineering of agent-based systems and technologies. This directly involves a need for consideration, understanding, and analysis of human factors, e.g., people's knowledge and skill, learning, and performance capabilities, and compatibilities in certain software development environments. This is because software developers utilize their experience and represent their mental models via development and engineering of intelligent agent(s). Therefore, the study of interrelated factors such as people's and agents' capabilities, constitutes an important and critical area in intelligent agent software engineering. This should eventually lead to more robust, intelligent, interactive, learning, and adaptive agents.

From the theoretical and practical viewpoints, the application of intelligent software agents is a topic of major interest. There has been a growing interest not only in new methodologies for development of intelligent software agents but also in the way in which these methodologies can be supported by theories and practice. Engineering intelligence in a software system is a young area, and there are many issues that need new and further research.

Many definitions and types of agents have been introduced in research literature in order to define a common understanding of this phenomenon in modern virtual environments. In fact, these definitions reflect a variety of viewpoints, problems, and applications of the agents, and therefore, they outline the major directions in this important research and practical field. It is recognized that general characteristics of intelligent agents are an agent's character, reasoning, and learning capabilities, autonomy, reactivity, proactivity, goal orientation, mobility, communication, and cooperation. The intelligent agents can be self-organized and can be integrated into a multiagent system that performs the tasks and solves the problems.

It is important to take into account the specific character of the cognitive tasks. In many cases:

- They are dependent on the agents' knowledge and learning capabilities.
- They are managed by the agent (self-managing work), rather than being "*administratively"* managed.
- They cannot be subdivided into *smaller subtasks*, i.e., cognitive tasks cannot be defined as nested or fine-grained tasks.
- They can be performed in *parallel*.

In addition, agents can be available and capable of overseeing cognitively driven tasks, but factors such as knowledge and technical, may be *incompatible*. *Agent resource capability and compatibility* have become the focus of agent-based software engineering. These factors have considerable impact on the formation of a multiagent system (i.e., a team of intelligent agents) for cognitive tasks, i.e., capability and compatibility factors define an *integration* problem *in engineering the task of intelligence in software systems*. Because agent availability issues do not contribute much to the success of performance, there is a risk of selecting agents to do work because they are available, not because they have a particular combination of key capability and compatibility factors. In addition, (single) agents can be available and capable of overseeing the individual cognitive tasks, but factors such as the knowledge of agents and their implementation may *not* be *compatible* with the goals and constraints of a multiagent system's application. Hence, managerial aspects such as intelligent capability and compatibility evaluation have to be an important focus of intelligent agent software engineering and improvement of agent-based processes.

However, for cognitive tasks involving agents' *capabilities and compatibilities* as critical variables, existing approaches to the definition of resource capability and compatibility do not sufficiently examine how effectively agent resources fit the needs for their implementation. The same cognitive tasks may be performed by different (single) agents or by a multiagent system using different capabilities, which correspond to different performance constraints (e.g., costs, risk, duration, productivity). Moreover, for completeness, data regarding the intelligent capabilities of agents and performance capabilities should be provided in connection with the learning methods utilized by agents to perform their cognitive tasks.

This book consists of work where intelligent agent software engineering is considered as the application of the integration of formal methods and heuristic approaches to ensure support for the evaluation, comparison, analysis, and evolution of agent behavior.

MISSION OF THE BOOK

This book does not attempt to teach any software/information technology engineering techniques. It does not claim to be comprehensive. There are many books on the market that present the classical approaches to software engineering and process modeling. Instead, it is designed to be used to support solutions to problems in developing and engineering the agent-based software/information technology and in engineering the intelligence in software systems.

Available books on intelligent systems and software/information technology engineering focus too much on the technical problems in software engineering. They provide limited coverage of the critical factors in engineering the intelligence in software systems, such as agent intelligent capability and compatibility, motivation, and agent's capability integration problems. In addition, little attention has been placed on the problems of engineering the intelligence in a multiagent system. Furthermore, we consider intelligent agent software engineering on a multiagent system's level, not only on the (single) agent level, and define the task of intelligent agent software engineering not only as a software engineering task, but also as a task of engineering the intelligence in a multiagent system. This book is an attempt to fill these gaps.

AIMS OF THE BOOK

The primary aims of this book are as follows:

- To introduce the readers to the concept of intelligent agent software engineering so that they will be able to develop their own agents and implement concepts in practice, i.e., in unique conditions, constraints, and environments (The book covers theoretical and practical aspects.)
- To present current research in the area of engineering the intelligence in software systems—in this book, we consider *engineering the intelligence in software systems* as the application of mathematical techniques and thorough engineering methods to the development of intelligent agents
- To offer methodologies that provide guidelines and procedures for the development of intelligent agents by applying various theories and techniques, e.g., machine learning methods, cognitive science, and the object-oriented paradigm
- To describe current problems in engineering the intelligence in software systems, e.g., evaluation and application of contemporary machine learning technology, application of object-oriented methodology to the development of intelligent agents, etc.
- To define future work and directions in the area of intelligent agent software engineering

THE AUDIENCE FOR THE BOOK

We envisage a number of distinct audiences for this book that make it useful and important. This book is for readers who have interests in developing intelligent software agents. It is suitable for practitioners, who want to use state-of-the-art techniques in the management and engineering of intelligence in software systems and the production of high-quality software. And, it is directed to the academics, advanced undergraduate or postgraduate students interested in new perspectives and directions in engineering the software development processes and intelligent agents and systems. Academics may also find this book useful as a course book for specific courses on advanced software (IT) engineering and intelligent systems. This book is useful as an advanced text for final-year undergraduate students of computer science, intelligent systems, software engineering and information technology, or as part of postgraduate masters and doctoral programs.

OUTLINE OF THE BOOK

The book is organized into ten chapters as follows.

Chapter 1 (Daniel Kudenko, Dimitar Kazakov, and Eduardo Alonso) discusses the differences between pure Machine Learning and the one performed by (single) Learning Agents. The authors describe critical aspects in machine learning technology and discuss the relationship between learning and communication, learning to collaborate and compete, the learning of roles, evolution and natural selection, and distributed inductive learning, These discussions are followed by a number of general recommendations for learning agent designers and some practical applications of machine learning to agents and multiagent systems. Two case studies being developed at the University of York are also presented.

Chapter 2 (Darryl N. Davis) introduces research into the nature of drives and motivations in computational agents from a perspective drawing on artificial life and cognitive science. The background to this research is summarized in terms of the possibility of developing artificial minds. A particular cognitive architecture is described in terms of control states. Steps toward producing an implementation of this architecture are described by means of experimentation into the nature of specific control states. The design of a simple A-life architecture for a predator–prey scenario is described using a transition state diagram. This architecture is then used as a platform with which to develop an agent with drives. This second architecture is used to develop an agent with explicit motivations. The discussion of these (and other) experiments shows how these simple architectures can help to provide some answers to difficult research questions in cognitive science.

Chapter 3 (L. Moreau, N. Zaini, D. Cruickshank, and D. De Roure) presents a versatile multiagent framework (named SoFAR, the Southampton Framework for Agent Research) designed for Distributed Information Management tasks. SoFAR embraces the notion of proactivity as the opportunistic reuse of the services provided by other agents and provides the means to enable agents to locate suitable service providers. The contribution of SoFAR is to combine ideas from the distributed computing community with the performative-based communications used in other agent systems: communications in SoFAR are based on the start point/end point paradigm, a powerful abstraction that can be mapped onto multiple communication layers. SoFAR also adopts an XML-based declarative approach for specifying ontologies and agents, providing a clear separation with their implementation.

Chapter 4 (P. Kefalas, M. Holcombe, G. Eleftherakis, and M. Gheorghe) introduces a detailed and comprehensive account of the ways in which some modern software engineering research can be applied to the construction of effective and reliable agent-based software systems. More specifically, the authors show how simple agents motivated from biology can be modeled as X-machines. Such modeling facilitates verification and testing of an agent model, because appropriate strategies for model checking and testing are already developed around the X-machine method. In addition, modular construction of agent models is feasible, because X-machines are provided with communicating features that allow simple models to interact.

Chapter 5 (P. Baillie, M. Toleman, and D. Lukose) examines the engineering of a mechanism that synthesizes and processes an artificial agent's internal emotional states: the Affective Space. Through use of the Affective Space, an agent can predict the effect that certain behaviors will have on its emotional state and, in turn, decide how to behave. Furthermore, an agent can use the emotions produced from its behavior to update its beliefs about particular entities and events. This chapter explores the psychological theory used to structure the Affective Space, the way in which the strength of emotional states can be diminished over time, how emotions influence an agent's perception, and the way in which an agent can migrate from one emotional state to another.

Chapter 6 (R.E. Smith and C. Bonacina) describes new implications of Evolutionary Computation theories and practices. Agents that interact according to these theories are no longer locked inside the laboratory conditions imposed by Evolutionary Computation researchers and users. Thus, it is important that authors consider the embodiment of Evolutionary Computation in "real" agents, that is, agents that involve the real restrictions of time and space within a multiagent system. This issue is addressed in two ways. First, authors have developed a platform independent agent system to assure that we work within the generic, but

realistic, constraints of agents. Second, authors have developed an agent-based system of Evolutionary Computation objects. The prime thrust of this research with these tools is to facilitate understanding of Evolutionary Computation within agents and understanding of more general agent interactions in the light of Evolutionary Computation theories. This chapter presents the platform independent agent system, along with the generic software framework for Evolutionary Computation in a multiagent system.

Chapter 7 (L. Brito, P. Novais, and J. Neves) discusses the use of agents in Electronic Commerce (EC) environments. A four-step method is introduced for developing EC-directed agents, which are able to take into account nonlinearites such as gratitude and agreement. Negotiations that take into account a multistep exchange of arguments provide extra information, at each step, for the intervening agents, enabling them to react accordingly. This argument-based negotiation among agents has much to gain from the use of Extended Logic Programming mechanisms. Incomplete information is common in EC scenarios; therefore, arguments must also take into account the presence of statements with an unknown valuation.

Chapter 8 (J. Debenham and B. Henderson-Sellers) expands on object-oriented (OO) methodologies to adequately support agent-oriented technology. It is recognized that OO methodologies are highly prescriptive and overly specified and are hard to extend when a new variant or a new paradigm appears. Authors consider OPEN (Object-oriented Process, Environment, and Notation) as a more flexible approach to building methodologies or processes. Using OPEN, process components are selected from a repository, and the actual methodology (or process) is constructed using identified construction and tailoring guidelines. Proposed extensions (new Tasks and Techniques for OPEN) are illustrated in two case studies of business processes. Goal orientation is readily accommodated in the existing and new tasks and techniques of the OPEN process, leading to a specification of the individual agents in the system.

Chapter 9 (V. Dignum and H. Weigand) proposes a framework that describes all the stages of development of a multiagent system, takes an organizational perspective on systems design, and specifies all the development steps for the design and development of an agent-based system for a particular domain. Specific agent-oriented methodologies can be used for the development and modeling of each of the development steps. Authors believe that such a generic framework, based on the organizational view, will contribute to the acceptance of multiagent technology by organizations. Authors define a social framework for agent communities based on organizational coordination models that "implements" the generic interaction, cooperation, and communication mechanisms that occur in the problem domain. The proposed methodology allows a generic coordination

model to be tailored to a given application and to determine its specific agent roles and interactions.

Chapter 10 (R. Aler, D. Camacho, and A. Moscardini) concludes this book with a multiagent system approach, with its purpose to build computer programs. Each agent in the multiagent system will be in charge of evolving a part of the program, which in this case, can be the main body of the program or one of its different subroutines. There are two kinds of agents: the manager agent and the genetic programming (GP) agents. The former is in charge of starting the system and returning the results to the user. The GP agents include skills for evolving computer programs, based on the genetic programming paradigm. There are two sorts of GP agents: some can evolve the main body of the program, and the others evolve its subroutines. Both kinds of agents cooperate by telling each other their best results found so far, so that the search for a good computer program is made more efficient. In this chapter, this multiagent approach is presented and tested empirically.

Valentina Plekhanova, PhD
University of Sunderland, UK