ELSEVIER

Artificial Intelligence 117 (2000) 277–296

Artificial Intelligence

www.elsevier.com/locate/artint

On agent-based software engineering ☆

Nicholas R. Jennings [1]

Department of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK

Received 21 September 1999

Abstract

Agent-based computing represents an exciting new synthesis both for Artificial Intelligence (AI) and, more generally, Computer Science. It has the potential to significantly improve the theory and the practice of modeling, designing, and implementing computer systems. Yet, to date, there has been little systematic analysis of what makes the agent-based approach such an appealing and powerful computational model. Moreover, even less effort has been devoted to discussing the inherent disadvantages that stem from adopting an agent-oriented view. Here both sets of issues are explored. The standpoint of this analysis is the role of agent-based software in solving complex, real-world problems. In particular, it will be argued that the development of robust and scalable software systems requires autonomous agents that can complete their objectives while situated in a dynamic and uncertain environment, that can engage in rich, high-level social interactions, and that can operate within flexible organisational structures.

Keywords: Agent-based computing; Software engineering; Multi-agent systems; Agent interactions; Social level

1. Introduction

An increasing number of computer systems are being viewed in terms of autonomous agents. Agents are being espoused as a new theoretical model of computation that more closely reflects current computing reality than Turing Machines [58]. Agents are being advocated as a next generation model for engineering complex, distributed systems [36,59]. Agents are also being used as an overarching framework for bringing together the component AI subdisciplines that are necessary to design and build intelligent entities [41,49]. Yet despite this intense interest, a number of fundamental questions about the nature and the use of the agent-oriented approach remain unanswered. In particular:

☆ This article is based on the Computers and Thought Award lecture that was delivered at the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) in Stockholm.

[1] Email: nrj@ecs.soton.ac.uk.

0004-3702/00/$ – see front matter
PII: S0004-3702(99)00107-1

- what are the essential concepts and notions of agent-based computing?
- what makes the agent-based approach an appealing and powerful computational model?
- what are the drawbacks of adopting an agent-oriented approach?
- what are the wider implications for AI and computer science of agent-based computing?

These questions can be tackled from many different perspectives, ranging from the philosophical to the pragmatic. This paper proceeds from the standpoint of using agent-based software to address real-world problems. However in the course of this analysis, a number of broader points are made about general direction and emphasis of future AI research.

Building high quality software for real-world applications is difficult. Indeed, it has been argued that such developments are one of the most difficult construction tasks humans undertake (both in terms of the number and the flexibility of the constituent components and in terms of their interconnections). Moreover, this statement is true no matter what models and techniques are applied: it is a consequence of the "essential complexity of software" [4]. Such complexity manifests itself in the fact that the software has a large number of parts that have many interactions [53]. [2] Given this state of affairs, the role of software engineering is to provide models and techniques that make it easier to handle this complexity [46,54]. To this end, a wide range of software engineering paradigms have recently been devised (e.g., object-orientation [2,42], component-ware [55], design patterns [18] and software architectures [6]). Each successive development either claims to make the engineering process easier or to extend the complexity of applications that can feasibly be built. Although evidence is emerging to support these claims, researchers continue to strive for more efficient and powerful techniques, especially as solutions for ever more demanding applications are sought.

In this article, it is argued that although contemporary methods are a step in the right direction, when it comes to developing complex, distributed systems they fall short in two main ways:

(i) the interactions between the various computational entities are too rigidly defined; and

(ii) there are insufficient mechanisms available for representing the system's inherent organisational structure (see Section 4 for more details of these arguments).

Against this background, the two central arguments of this paper can be expressed:

The Adequacy Hypothesis. *Agent-oriented approaches can significantly enhance our ability to model, design and build complex, distributed software systems.*

The Establishment Hypothesis. *As well as being suitable for designing and building complex systems, the agent-oriented approach will succeed as a mainstream software engineering paradigm.*

[2] In this context, the term "complexity" is used in a general manner (as in [11,15,57]); *not* in the specific technical sense of algorithmic or computational complexity.

In seeking to argue for these hypotheses, it is clear that this work differs in flavour from the majority of scientific papers. It presents no new theorems, has no experimental results, and does not describe a novel application. Rather, it represents a (qualitative) analysis of an important and fast growing area of computer science. The aim of this analysis is to provide the intellectual justification of precisely why agent-based systems are well suited to engineering complex software systems. The analysis contained herein is based on more than a decade of experience in using agent-based techniques to construct large-scale, real-world applications in a wide variety of industrial and commercial domains (see [17,25,30, 33,34]). Despite these caveats, this paper *does* make a number of important contributions to the state of the art. Firstly, despite agent-based systems being touted as an approach that will have a major impact on future generation software ("pervasive in every market by the year 2000" [27] and "the new revolution in software" [21]), there has been no systematic evaluation of *why* this may be the case. Thus, although there are an increasing number of deployed agent applications (see [37,44] for a review), nobody has systematically analysed precisely what makes the paradigm effective. This is clearly a major gap in knowledge that this paper seeks to address. Secondly, there has been comparatively little work on viewing agent-based computing as a serious software engineering paradigm that can significantly enhance developments in a wide range of applications. This shortcoming is rectified by recasting the essential components of agent systems into more traditional software engineering concepts. From here, it can be shown that the agent-based approach is a both a natural and a logical evolution of a range of contemporary approaches to software engineering.

The remainder of the paper is structured as follows. Section 2 discusses the essence of agent-based computing. Section 3 makes the case as to why an agent-oriented approach is well suited to engineering complex, distributed systems. Section 4 argues why agent-based techniques are likely to succeed and make it into the mainstream of software engineering. Section 5 highlights the potential disadvantages of adopting an agent-oriented approach. Section 6 advocates a new perspective on modeling computer systems (the *social level* [32]) as a promising means of remedying the identified shortcomings. Finally, Section 7 places the work in a broader AI and computer science context.

2. The essence of agent-based computing

The first step in arguing for an agent-oriented approach to software engineering is to precisely identify and define the key notions and concepts of agent-based computing. Defining and classifying phenomena is always a task fraught with difficulty—there will always be objections to basic definitions, arguments that important points have been overlooked, or claims that it is really nothing new anyway. Such observations are especially pertinent if the entity to be defined is both intangible and a relatively new phenomenon. Nevertheless, such definitions are precisely what are needed in order to argue for agent-oriented software engineering. Given this necessity, the approach taken here is to offer a definition that is sufficiently encompassing to cover a broad range of phenomena that can reasonably go under the heading of agent-based systems, yet sufficiently tight that it can rule out systems that are clearly not agent-based. Around the edges there will always

be debate. Moreover, the definitions offered here concentrate on necessary, rather than sufficient, conditions so they can always be extended.

Here the key definitional problem relates to the term "agent". At present, there is much debate [16], and little consensus, about exactly what constitutes agenthood. However, an increasing number of researchers find the following characterisation useful [59]:

> *An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.*

There are a number of points about this definition that require further explanation. Agents are:

(i) clearly identifiable problem solving entities with well-defined boundaries and interfaces;
(ii) situated (embedded) in a particular environment—they receive inputs related to the state of their environment through sensors and they act on the environment through effectors;
(iii) designed to fulfill a specific purpose—they have particular objectives (goals) to achieve;
(iv) autonomous—they have control both over their internal state and over their own behaviour; [3]
(v) capable of exhibiting flexible problem solving behaviour in pursuit of their design objectives—they need to be both reactive (able to respond in a timely fashion to changes that occur in their environment) and reactive (able to act in anticipation of future goals) [60].

When adopting an agent-oriented view of the world, it soon becomes apparent that most problems require or involve multiple agents; to represent the decentralised nature of the problem, the multiple loci of control, the multiple perspectives or the competing interests [3]. Moreover, the agents will need to interact with one another, either to achieve their individual objectives or to manage the dependencies that ensue from being situated in a common environment [9,29]. These interactions can vary from simple information interchanges, to requests for particular actions to be performed and on to cooperation, coordination and negotiation in order to arrange interdependent activities. In all of these cases, however, there are two points that qualitatively differentiate agent interactions from those that occur in other computational models. Firstly, agent-oriented interactions are conceptualised as taking place at the *knowledge level* [40]. That is, they are conceived in terms of which goals should be followed, at what time, and by whom (cf. method invocation or function calls that operate at a purely syntactic level). Secondly, as agents are flexible problem solvers, operating in an environment over which they have only partial control and observability, interactions need to be handled in a similarly flexible

[3] Having control over their own behaviour is one of the characteristics that distinguishes agents from objects [59]. Although objects encapsulate state and behaviour (more accurately behaviour realisation) [2], they fail to encapsulate behaviour activation or action choice. Thus, any object can invoke any publicly accessible method on any other object at any time. Once the method is invoked, the corresponding actions are performed. In this sense, objects are totally obedient to one another and do not have autonomy over their choice of action.

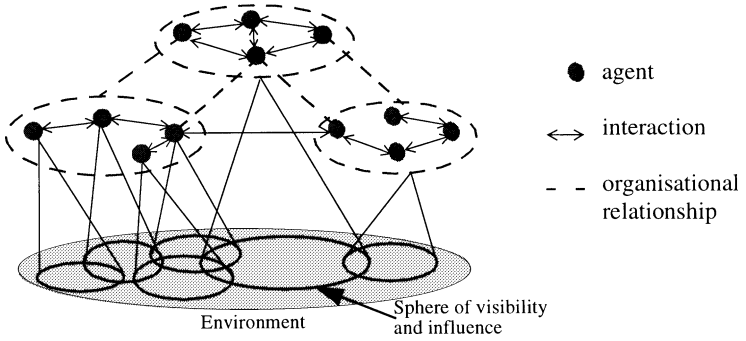

Fig. 1. Canonical view of an agent-based system.

manner. Thus, agents need the computational apparatus to make *run-time* decisions about the nature and scope of their interactions and to initiate (and respond to) interactions that were not foreseen at design time (cf. the hard-wired engineering of such interactions in extant approaches).

In most cases, agents act to achieve objectives either on behalf of individuals/companies or as part of some wider problem solving initiative. Thus, when agents interact there is typically some underpinning organisational context between them [14,19]. This context defines the nature of the relationship between the agents (e.g., they may be peers working together in a team or one may be the manager of the other agents) and consequently influences their behaviour. Since agents make decisions about the nature and scope of interactions at run time, it is imperative that this key shaping factor is taken into account. Thus organisational relationships need to be represented explicitly. In many cases, these relationships are subject to ongoing change: social interaction means existing relationships evolve and new relations are created. This means the temporal extent of relationships can also vary significantly, from just long enough to deliver a particular service once, to a permanent bond. To cope with this variety and dynamic, agent researchers have: devised protocols that enable organisational groupings to be formed and disbanded; specified mechanisms to ensure groupings act together in a coherent fashion; and developed structures to characterise the macro behaviour of collectives (see [37,60] for an overview).

Drawing these points together (Fig. 1), the essential concepts of agent-based computing can be seen to be: agents, high-level interactions and organisational relationships (see [14,19,23] for broadly similar characterisations).

3. The case for an agent-based approach to software engineering

Probably the most compelling argument that could be made for adopting an agent-oriented approach to software development is to have a set of quantitative data that showed, on a standard set of software metrics, the superiority of the agent-based approach (in terms of productivity, software reliability, system maintainability, etc.) over a range of other techniques. However such data simply does not exist (as it doesn't for other contemporary

methods in software engineering).[4] Hence the arguments for agent-oriented software engineering must be qualitative in nature.

The structure of the argument that will be used here is based on the suitability of agent-based techniques for tackling complex, real-world problems and it has the following broad form. On the one hand, there are a number of well-known techniques for tackling complexity in software. Also the key characteristics of complex software systems are (reasonably) well understood. On the other hand, the essential concepts and notions of the agent-based paradigm have been elucidated (Section 2). Thus, an argument and an evaluation can be made by examining the degree of match between these two perspectives; a high degree of match would be indicative of the suitability of the agent-based approach, whereas a poor degree would be indicative of its unsuitability.

Before this match process can commence, however, the techniques for tackling complexity in software need to be introduced. Booch [2] identifies three such tools:

- *Decomposition:* The most basic technique for tackling large problems is to divide them into smaller, more manageable chunks each of which can then be dealt with in relative isolation. This helps tackle complexity because it limits the designer's scope; at any given instant only a portion of the problem needs to be considered.
- *Abstraction:* The process of defining a simplified model of the system that emphasises some of the details or properties, while suppressing others. Again, this technique works because it limits the designer's scope of interest at a given time. Attention can be focused on the salient aspects of the problem, at the expense of the less relevant details.
- *Organisation:*[5] The process of identifying and managing the interrelationships between the various problem solving components. The ability to specify and enact organisational relationships helps designers tackle complexity in two ways. Firstly, by enabling a number of basic components to be grouped together and treated as a higher-level unit of analysis (e.g., the individual components of a subsystem can be treated as a single coherent unit by the parent system). Secondly, by providing a means of describing the high-level relationships between various units (e.g., a number of components may work together (cooperate) to provide a particular functionality).

Next, the characteristics of complex systems need to be enumerated [53]:

- Complexity frequently takes the form of a hierarchy. That is, a system that is composed of interrelated subsystems, each of which is in turn hierarchic in structure, until the lowest level of elementary subsystem is reached. The precise nature of these organisational relationships varies between subsystems, however some generic forms

[4] Software paradigms generally go through three main phases. Firstly, early pioneers identify a new way of doing things (based on intuition and insight). Secondly, individuals and organisations that are early adopters of leading-edge technologies recognise the potential (based on qualitative arguments) and start to build systems using the new concepts. Thirdly, the advocated concepts, and knowledge of their advantages (sometimes backed up by quantitative data), become more widespread and enter the mainstream of software engineering. At this time, agent-oriented techniques are firmly in phase two, but one of the aims of this paper is to start the movement towards phase three.

[5] Booch [2] actually uses the term "hierarchy" for this final point. However, the more neutral term "organisation" is preferred here.

(such as client-server, peer, team, etc.) can be identified. These relationships are not static: they often vary over time.

- It is possible to distinguish between the interactions *among* subsystems and the interactions *within* subsystems. The latter are both more frequent (typically at least an order of magnitude more) and more predictable than the former. This gives rise to the view that complex systems are *nearly decomposable*: subsystems can be treated almost as if they are independent of one another, but not quite, since there are some interactions between them. Moreover, although many of these interactions can be predicted at design time, some simply cannot.
- The choice of which components in the system are primitive is relatively arbitrary and is defined by the observer's aims and objectives.
- Hierarchic systems evolve more quickly than non-hierarchic ones of comparable size. In other words, complex systems will evolve from simple systems more rapidly if there are *stable intermediate forms*, than if there are not.

With these two characterisations in place, the precise form of the match process argument in favour of agent-based software engineering can now be expressed:

- show that agent-oriented decomposition is an effective way of partitioning the problem space of a complex system (Section 3.1);
- show that the key abstractions of the agent-oriented mindset are a natural means of modeling complex systems (Section 3.2);
- show that the agent-oriented philosophy for dealing with organisational relationships is appropriate for complex systems (Section 3.3).

3.1. The merits of agent-oriented decomposition

The agent-oriented approach advocates decomposing problems in terms of autonomous agents that can engage in flexible, high-level interactions. Considering the autonomous nature of the problem solving entities first. Autonomy, in this context, means that the problem solvers have their own persistent thread of control (i.e., they are active) and that they decide for themselves which actions they should perform at what time. Decomposing a problem in such a way aids the process of engineering complex systems in two main ways. Firstly, it is simply a natural representation for complex systems that are invariably distributed ("all real systems are distributed" [22]) and that invariably have multiple loci of control ("real systems have no top" [42, p. 47]). [6] This decentralisation, in turn, reduces the system's control complexity and results in a lower degree of coupling between components. The fact that agents are active means they know for themselves when they should be acting and when they should update their state (cf. passive objects that need to be invoked by some external entity to do either). Such self-awareness reduces control complexity since the system's control know-how is taken from a centralised repository and localised inside each individual problem solving component. Secondly, since decisions about what actions should be performed are devolved to autonomous entities, selection can be based on the

[6] Indeed the view that decompositions based upon functions/actions/processes are more intuitive and easier to produce than those based upon data/objects is even acknowledged within the object-oriented community [42, p. 44].

local situation of the problem solver. This enables selection to be responsive to the agent's actual state of affairs, rather than some external entity's perception of this state,[7] and means that the agent can attempt to achieve its individual objectives without being forced to perform potentially distracting actions simply because they are requested by some external entity.

Moving onto the flexible nature of interactions. The fact that agents make decisions about the nature and scope of interactions at run-time makes the engineering of complex systems easier for two main reasons. Firstly, the system's inherent complexity means it is impossible to know *a priori* about all potential links: interactions will occur at unpredictable times, for unpredictable reasons, between unpredictable components. For this reason, it is futile to try and predict or analyse all the possibilities at design-time. Rather, it is more realistic to endow the components with the ability to make decisions about the nature and scope of their interactions at run-time. From this, it follows that components need the ability to initiate (and respond to) interactions in a flexible manner (see Section 5 for a discussion of the downside of this flexibility). Thus agents are specifically designed to deal with unanticipated requests and they can spontaneously generate requests for assistance whenever appropriate. Secondly, the problem of managing control relationships between the software components is significantly reduced (see above discussion). All agents are continuously active and any coordination that is required is handled bottom-up through inter-agent interaction. Thus, the ordering of the system's top-level goals is no longer something that has to be rigidly prescribed at design time. Rather, it becomes something that is handled in a context-sensitive manner at run-time.

To illustrate how an agent-oriented stance affects the manner in which a problem is decomposed, consider the domain of flexible manufacturing control and, in particular, the task of producing individually tailored goods (such as cars built according to a customer's specification) (Fig. 2). The manufacturing process involves a number of basic parts (A, B and C) that have various operations (O_1 to O_9) performed upon them by various machines (M_1 to M_9). Operations may be performed on a single component (e.g., O_1 by M_1 on part A's) or they may involve the joining of multiple parts to form a new composite (e.g., O_5 by M_5 joins parts of type A and B). Some operations may fail (e.g., O_5 and O_2) and consequently will need to be redone. The end products (P_1 to P_n) are composed of the constituent components with various sequences of operations performed upon them.

The industry standard approach to this problem is to devise a global schedule, typically covering one day, for the entire manufacturing process. This indicates when the various parts should be released from their stores, which machines they should be routed through, and what operations should be performed at the various machines. The problem with this centralised and pre-planned approach, however, is that the schedule is rarely adhered to in practice: machines and operations fail and operations take longer than expected. When such disturbances occur, the plant controller either has to initiate a costly rescheduling exercise or use the out-of-date schedule as an approximate guide.

[7] Recognising the importance of allowing decisions about action execution to be based on local state, object-oriented languages such as Eiffel allow the server to assert, and subsequently test, preconditions that need to be established before one of its routines can be invoked [42].

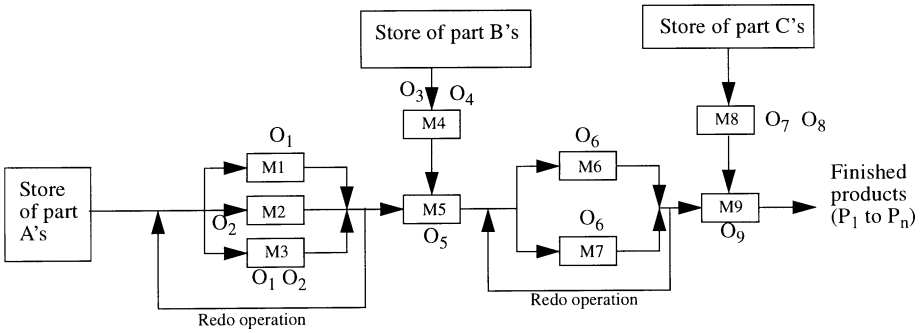

Fig. 2. Exemplar flexible manufacturing system.

As a consequence of these difficulties, several organisations have deployed an agent-oriented approach (see [7,44] for details of specific systems). In such systems, each manufactured part is represented by an autonomous agent that has the objective of getting itself to the end of the manufacturing line, having had a specified set of operations performed upon it. Each machine is also represented by an agent. Such agents have the objective of maximising their throughput and they do this by deciding what components will be accepted in what order and what operations will be performed at what time. Thus, for a given part to have an operation performed upon it, its agent must negotiate with a machine agent capable of performing that operation. Component agents representing the constituent parts of a composite item also need to coordinate their actions so they arrive at joining machines at the same time. When components are joined in this manner, a new organisational structure representing the composite is formed.

The success of such agent-oriented systems, both in terms of increased throughput and greater robustness to failure, can be attributed to a number of points. Firstly, representing the components and the machines as agents means the decision making is much more localised. It can, therefore, be more responsive to prevailing circumstances. If unexpected events occur, agents have the autonomy and proactiveness to try alternatives. Secondly, because the schedules are built up dynamically through flexible interactions, they can readily be altered in the event of delays or unexpected contingencies. For example, if one of the constituent parts of a composite item is delayed *en route* to a synchronisation point, it can inform the remaining team members. Together they can then re-arrange the meeting time and adapt their individual behaviour accordingly. Thirdly, the explicitly defined relationships between the constituent parts of a composite item identify those agents that need to coordinate their actions. Moreover, a composite item team can be treated as a single conceptual entity by machines further on down the manufacturing line. This, in turn, eases the scheduling task by reducing the number of items that need to be considered during decision making.

3.2. The suitability of the agent-oriented abstractions

A significant part of any design process is finding the right models for viewing the problem. In general, there will be multiple candidates and the difficult task is picking the

most appropriate one. When it comes to designing software, the most powerful abstractions are those that minimise the semantic distance between the units of analysis that are intuitively used to conceptualise the problem and the constructs present in the solution paradigm.

In the case of complex systems, the problem to be characterised consists of subsystems, subsystem components, interactions and organisational relationships. Taking each in turn:

- There is a clear and strong degree of correspondence between the notions of subsystems and agent organisations. They both involve a number of constituent components that act and interact according to their role within the larger enterprise.
- The suitability of viewing subsystem components as agents has already been made (Section 3.1).
- The interplay between the subsystems and between their constituent components is most naturally viewed in terms of high level social interactions. For instance, Booch [2] begins his analysis of complex systems from the following standpoint: "at any given level of abstraction, we find meaningful collections of entities that collaborate to achieve some higher level view" [2, p. 34]. This view and level of abstraction accords precisely with the treatment of interaction afforded by the agent-oriented approach. Agent systems are invariably described in terms of "cooperating to achieve common objectives", "coordinating their actions" or "negotiating to resolve conflicts".
- Complex systems involve changing webs of relationships between their various components. They also require collections of components to be treated as a single conceptual unit when viewed from a different level of abstraction. On both levels, the agent-oriented mindset again provides suitable abstractions. A rich set of structures is typically available for explicitly representing and managing organisational relationships (e.g., roles [38], norms [10] and social laws [52]). Interaction protocols exist for forming new groupings and disbanding unwanted ones (e.g., [50,51]). Finally, structures are available for modeling collectives (e.g., joint intentions [30] and teams [56]). The latter point is especially useful in relation to representing subsystems since they are nothing more than a team of components working together to achieve a collective goal.

3.3. The need for flexible management of changing organisational structures

Organisational constructs are first-class entities in agent systems. Thus explicit representations are made of organisational relationships and structures. Moreover, agent-based systems have the concomitant computational mechanisms for flexibly forming, maintaining and disbanding organisations. In the flexible manufacturing scenario, for example, individual part agents form themselves into ever more complex structures as they move through the assembly process. In this case, the part agents explicitly represent the other components to which they will eventually be joined. This organisational collective then negotiates, as a single conceptual entity, with subsequent machine agents that need to perform operations upon it. Similarly, if some part of the team is delayed *en route* to a synchroni-

sation rendezvous, then the explicit organisational model identifies those agents that need to re-coordinate their activities. Finally, if a constituent component agent is destroyed or ruined in the manufacturing process (e.g., by a faulty machining operation), then the remaining team members will enter a negotiation process in order to find a replacement. This organisational updating is typical of the dynamic nature of groupings in complex systems.

This representational power enables agent-oriented systems to exploit two facets of the nature of complex systems. Firstly, the notion of a primitive component can be varied according to the needs of the observer. Thus at one level, entire subsystems can be viewed as singletons, alternatively, teams or collections of agents can be viewed as primitive components, and so on until the system eventually bottoms out. Secondly, such structures provide a variety of stable intermediate forms, that, as already indicated, are essential for the rapid development of complex systems. Their availability means individual agents or organisational groupings can be developed in relative isolation and then added into the system in an incremental manner. This, in turn, ensures there is a smooth growth in functionality.

4. Towards the software engineering mainstream

Having made the case that an agent-oriented approach is well suited to designing and building complex systems (Section 3), the next step is to determine whether it will succeed as a mainstream software engineering paradigm. This question is important because the history of computing is littered with apparently promising technologies that were never widely adopted. Fortunately, however, there are two compelling reasons for believing that agent-based techniques will become widely adopted. Firstly, the agent-based approach can be viewed as a natural next step in the evolution of a whole range of approaches to software engineering.[8] Secondly, agent-based techniques are the ideal computational model for developing software for open, networked systems (such as the Internet). Each of these issues will now be dealt with in turn.

A number of trends become evident when examining the evolution of programming models from assembly languages, to procedural and structured programming, to object-based and declarative programming, onto component-ware, design patterns, and software architectures (see, for example, [1]). Firstly, there has been an inexorable move from languages that have their conceptual basis determined by the underlying machine architecture, to languages that have their key abstractions rooted in the problem domain. Here the agent-oriented world view is perhaps the most natural way of characterising many types of problem. Just as the real-world is populated with (passive) objects that have operations performed on them, so it is equally (if not more) full of active, purposeful

[8] It is *not* envisaged that agent-based approaches will supplant techniques such as object-orientation, design patterns or component-ware. Rather, agent-based computing should be seen as providing a higher level of computational abstraction and this may, in turn, be realised through object-based systems or in a component-based fashion.

agents that interact to achieve their objectives.[9] Indeed, many object-oriented analyses start from precisely this perspective: "we view the world as a set of autonomous agents that collaborate to perform some higher level function" [2, p. 17]. Secondly, the basic building blocks of the programming models exhibit increasing degrees of localisation and encapsulation [44]. Agents follow this trend by localising purpose inside each agent, by giving each agent its own thread of control, and by encapsulating action selection. Thirdly, ever richer mechanisms for promoting re-use are being provided. Here, the agent view also reaches new heights. Rather than stopping at re-use of subsystem components (design patterns and component-ware) and rigidly preordained interactions (application frameworks), agents enable whole subsystems and flexible interactions to be re-used. In the former case, agent designs and implementations are re-used within and between applications. Consider, for example, the class of agent models that has beliefs (what the agent knows), desires (what the agent wants) and intentions (what the agent is doing) at its core. Such Belief-Desire-Intention (BDI) architectures have been used in a wide variety of applications including air traffic control [39], process control [30], simulation [47], fault diagnosis [26], transportation [5], and scientific data interpretation [17]. In the latter case, flexible patterns of interaction such as the Contract Net Protocol [12] (an agent with a task to complete advertises this fact to others that it believes are capable of performing it, these agents may submit a bid to perform the task if they are interested, and the originator then delegates the task to the agent that makes the best bid) and various forms of resource-allocation auction (e.g., English, Dutch, Vickrey) have been re-used in significant numbers of applications (see [8], for example). The third notion of re-use is that agents enable legacy (non-agent) software to be incorporated in a relatively straightforward manner [35]. The technique used is to place wrapping software around the legacy code. The wrapper presents an agent interface to the other software components and thus from the outside it looks like any other agent. On the inside, the wrapper performs a two-way translation function: taking external requests from other agents and mapping them into calls in the legacy code, and taking the legacy code's external requests and mapping them into the appropriate set of agent communication commands. This ability to wrap legacy systems means agents may initially be used as an integration technology. However, as new requirements are uncovered, so bespoke agents may be developed and added. This feature enables a complex

[9] Although there are certainly similarities between object- and agent-oriented approaches (e.g., both adhere to the principle of information hiding and recognise the importance of interactions), there are also a number of important differences. Firstly, objects are generally passive in nature: they need to be sent a message before they become active. Secondly, although objects encapsulate state and behaviour realisation they do not encapsulate action choice (Section 2). Thirdly, object-orientation fails to provide an adequate set of concepts and mechanisms for modeling complex systems: for such systems "we find that objects, classes and modules provide an essential yet insufficient means of abstraction" [2, p. 34] Individual objects represent too fine a granularity of behaviour and method invocation is too primitive a mechanism for describing the types of interactions that take place. As has already been argued, agents with their coarser level of granularity and higher-level view of interaction are eminently more suitable. Finally, object-oriented approaches provide only minimal support for specifying and managing organisational relationships (basically relationships are defined by static inheritance hierarchies). In recognition of this fact, Hewitt and Inman [24] introduced the notion of ORGs into the basic Actor model. This provided a number of inbuilt organisational structures that designers could exploit during their developments. Gasser and Briot [20] also note similar limitations of object-based concurrent programming for modeling complex social relationships.

system to grow in an evolutionary fashion (based on stable intermediate forms), while adhering to the important principle that there should always be a working version of the system available. In summary, agent-oriented techniques represent a natural progression of current software engineering thinking and, for this reason, the main concepts and tenets of the approach should be readily acceptable to mainstream practitioners.

Turning now to the question of software models for open, networked systems. Such systems are characterised by the fact that there is no single controlling organisation, by the fact that the software represents the interests of a diverse range of stakeholders, and by the fact that there is constant change [19,23]. In such environments, the dominant software model needs to be based on synthesis or construction, rather than decomposition or reduction. Thus the "system" is simply the collection of independently developed software entities that are interacting with one another at any instant in time. From this perspective, a number of requirements can be placed upon the computational model:

(i) the individual problem solving entities need to be able to act to achieve specified objectives (i.e., they must be active and autonomous);
(ii) these entities must do so in a flexible manner in order to cope with the inherent uncertainty they face (i.e., they need to be reactive and proactive);
(iii) the computational entities need to be capable of interacting with entities that were not foreseen at design time and in a manner that is appropriate to their current situation (i.e., they must be able to engage in flexible interactions); and
(iv) any organisational relationships that do exist between the stakeholders must be reflected in the behaviour and actions of the problem solvers (i.e., the organisational relationships need to be explicitly represented and taken into account during the action selection process).

In short, the desired computational model needs to be agent-based.

5. The downside of an agent-based approach to software engineering

Having highlighted the potential benefits of agent-based software engineering (Sections 3 and 4), this section seeks to pinpoint some of the concomitant drawbacks. Here the aim is to identify and isolate those aspects of complex system developments that are made more difficult by adopting an agent-based approach. Thus, it does not address those difficulties that arise from engineering large systems *per se* (e.g., issues of performance engineering and security), nor with those problems that are caused by the fact that agent-based systems are both distributed and concurrent, nor with the issues that arise as a result of software having to maintain an ongoing interaction with a dynamic and unpredictable environment [45]. Finally, the aim is to concentrate on issues that are intrinsic to the agent-based philosophy (cf. the many social and pragmatic problems often associated with developing systems using any new technology [61]). Against this background, there are two major drawbacks associated with the very essence of an agent-based approach:

- the patterns and the outcomes of the interactions are inherently unpredictable;
- predicting the behaviour of the overall system based on its constituent components is extremely difficult (sometimes impossible) because of the strong possibility of emergent behaviour.

Although the flexibility of agent interactions has many advantages when it comes to engineering complex systems, the downside is that it leads to unpredictability in the run-time system. As agents are autonomous, the patterns and the effects of their interactions are uncertain. Firstly, agents decide at run-time which of their objectives require interaction in a given context, which acquaintances they will interact with in order to realise these objectives, and when these interactions will occur. Hence decisions about the number, pattern and timing of interactions depend on a complex interplay of the agent's internal state, the agent's perception of the environment (perhaps including the state of its acquaintances), and the organisational context that exists when the decision is made. Combining these multifarious factors means that it is difficult to make predictions about the system's interactions. Secondly, there is a de-coupling, and potentially a considerable degree of variability, between what one agent first requests through an interaction and the outcome that eventually ensues. Since agents have autonomy over their own choices: the request may be immediately honoured as it is, it may be refused completely, or it may be modified through some form of social interchange. In short, in the general case, both the nature (a simple request versus a protracted negotiation) and the outcome of an interaction cannot be determined at the onset.

The second source of unpredictability in agent-oriented systems relates to the notion of emergent behaviour. It has long been recognised that interactive composition—collections of processes (agents) acting side-by-side and interacting in whatever way they have been designed to interact [43]—results in behavioural phenomena that cannot be deconstructed solely in terms of the behaviour of the individual components. That is, the whole is often greater than the sum of the parts. Such emergent behaviour is a consequence of the interaction between components and given their sophistication and flexibility in agent systems, it is clear that the scope for unexpected collective behaviour is considerable. In certain situations (e.g., social simulations and market systems) emergence is not necessarily a bad thing since the ensuing behaviour is a more accurate model of the problem being addressed. However, when predictability is a desirable system property, then the aim is to minimise its occurrence and impact.

Both of the aforementioned drawbacks apply to the general case of using an agent-based approach. However in specific systems and applications, designers are able to circumvent these difficulties by using interaction protocols whose properties can be formally analysed (sometimes borrowing techniques such as mechanism design from game theory [48]), by adopting rigid and preset organisational structures, and/or by limiting the nature and the scope of the agent interplay. In all of these cases, the aim is to reduce the system's unpredictability. However these restrictions also limit the power of the agent-based approach; thus, in order to realise its full potential some longer term solutions are required. In particular, a better understanding is needed of the impact of sociality and organisational context on an individual's behaviour and of the symbiotic link between the behaviour of the individual agents and that of the overall system.

One means of tackling these fundamental issues is to follow an approach that proved successful in elucidating the foundational principles and structures of individual (asocial) agents. Newell's [40] knowledge level analysis provided the seminal characterisation of intelligent agents—it stripped away implementation and application specific details to reveal the core of asocial problem solvers. Since the aim here is to do the same for social agents,

Newell's basic approach appears an obvious point of departure. Thus a new computer level needs to be defined (see [28] for details of the main arguments). This level can be called the *social level* [32]. It should sit immediately above the knowledge level and should provide the social principles and foundations for agent-based systems. The primary benefit of developing a social level description is that it enables the overall system's behaviour and key conceptual structures to be studied without the need to delve into the implementation details of the individual agents or the specifics of particular interaction protocols [28,32]. Thus prediction of the behaviour of the social agents and of the overall system can be made more easily. To this end, the next section presents a preliminary vision of the social level.

6. A social level view

This section presents the outline of a proposal for a social level characterisation of agent-based systems (Table 1). This characterisation follows Newell's basic nomenclature for specifying computer system levels.

The *system* is the entity to be described at that computer level. For the knowledge level it is an (asocial) agent. For the social level it is an agent organisation; that is, a collection (or grouping) of agents that are arranged in various relationships to one another.

The *components* are the primitive elements from which the system is built up. For the knowledge level, an agent is conceived of in terms of the goals it has to achieve and the actions that it can perform in their pursuit. For the social level, an agent organisation consists of four main components that together represent the objective basis upon which the organisation functions. Firstly, there are the agents that go together to constitute the organisation. Secondly, there are the various channels through which these agents can communicate and interact with one another. These encompass both the underlying

Table 1
Summary of the knowledge and social levels

Dimension	Description	Knowledge level	Social level
System	Entity to be described	(asocial) Agent	Agent organisation
Components	The system's primitive elements	Goals, Actions	Agents, Interaction channels, Dependencies, Organisational relationships
Compositional law	How the components are assembled	Various	Roles, Organisation's rules
Behaviour law	How the system's behaviour depends upon its composition & components	Principle of rationality	Principle of organisational rationality
Medium	The elements to be processed to obtain the desired behaviour	Knowledge	Organisation and social obligations, Means of influencing others, Means of changing organisational structures

mechanism (e.g., message passing structures, blackboard systems or the environment) as well as the content (e.g., agent communication languages and the associated domain ontologies). The third component is the dependencies that exist between the agents.

Such dependencies can be between the objectives that the agents wish to achieve or through the environment's shared resources. In either case, it is the concept of dependence that drives the agents to interact with one another [9,29]. Finally, there are the various organisational relationships that exist between the agents. For example, the agents may be peers, competitors or situated in a variety of authority relationships.

The *compositional laws* define how the components are assembled to form the system. For the knowledge level, Newell simply states that an agent's goals and actions can be arranged in multifarious ways. For the social level, the primitive components are assembled according to their roles within the system and the organisation's rules. Roles can be undertaken by individual or multiple agents and their purpose is to define the achievable objectives, to indicate the ensuing organisational relationships between the participants, to set the channels through which interaction should occur, and to dictate the patterns of interchange that are appropriate [19,23,38]. Accompanying the role definitions are the organisation's rules that define the concomitant procedures or the emergent norms in which role enactment takes place. Thus, the rules specify, among other things, which agents can adopt which roles and under what terms and conditions, what should happen if roles are updated/modified and how conflicts between roles should be handled.

The *behaviour law* specifies how the system's behaviour depends upon its composition and its components. For the knowledge level, the behaviour law is the principle of rationality which simply states that if an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action [40]. For the social level, the behaviour of the organisation depends upon the ways in which the roles are enacted and the degree to which the organisation's rules are adhered to. Thus, this organisational rationality indicates how the collective will actual behave in practice. For example, the agents may well decide to follow their designated/assigned role in the organisation and also to adhere to the organisation's rules. However, there may equally be situations in which these constraints are deliberately violated. Thus the notion of organisational rationality indicates to what degree and under what circumstances the agents will follow their organisational obligations. Since social interchange is an integral part of a role's specification, organisational rationality also covers social obligations between the participating agents. Thus it defines the situations in which agents may make social commitments, when they can violate them, and what compensating actions should be performed in such circumstances.

The *medium* represents the elements that are processed in order to obtain the desired behaviour. For the knowledge level, an agent processes knowledge in order to attain its goals. At the social level, organisations process three main types of elements. Firstly, the various organisational and social obligations that the agents enter into: either as a result of their organisational roles/relationships or as a consequence of the social interactions in which they engage. Secondly, the various mechanisms and structures that are available for the organisation's components to influence the behaviour of one another (enacted through the interaction channels). These include, for example, negotiation techniques, cooperation protocols, and coordination models. All of these interactions can be characterised as means

by which agents influence one another's behaviour (in order to manage their respective dependencies). The final element to be processed is the various means that are available for changing or modifying the organisational structure. That is, the elements that are processed in order to create new roles, to change organisational rules or to modify the prevailing organisational rationality.

At this stage, the aim of the social level characterisation is to provide a means of tackling the aforementioned fundamental drawbacks of the agent-based approach. While it is highly likely that this description will undergo modification and refinement in the light of experience, it nevertheless provides a line of attack to these issues. Moreover, the core concepts can be viewed as being relatively stable (see the evolution from [28] to [32] and onto the current proposal). To this end, the social level aids the science of agent-based computing by providing a comprehensive model for specifying and understanding behaviour in agent-based systems. This contrasts with the majority of the extant work in this area that typically concentrates on a small fragment of the overall picture. For example, the BDI models typically fail to incorporate the influence of organisational structures on an agent's behaviour and the organisational models tend to neglect the autonomy of the constituent agents. A social level perspective also aids the engineering aspects of agent-based systems. By identifying the key constituent components and their interrelationships, the social level provides a sound basis for developing tools that can support the development of agent-based systems. Moreover, social level models provide a basis for agent-oriented analysis and design. Indeed [62] follows precisely this approach; presenting a methodology in which agent-based systems are viewed as computational organisations that are defined in terms of roles, interactions and obligations.

7. Discussion

This paper has sought to justify the claim that agent-based computing has the potential to significantly improve our ability to model, design and build complex, distributed software systems. In making this claim, a series of qualitative arguments were developed to highlight the high degree of match between the requirements of complex system development paradigms on the one hand and the key concepts and notions of agent-based computing on the other. The second claim contained herein is that the agent-based approach will succeed as a mainstream software engineering paradigm. The basis for this belief is that agent-based computing is a logical evolution of a number of contemporary approaches to software engineering and also because it is ideally suited to developing software in truly open systems. Against this promise, the inherent unpredictability of agent interactions and the strong possibility of emergent behaviour were identified as inherent drawbacks. To help provide a long-term means of addressing these problems, a social level characterisation of agent-based systems was advocated as a promising point of departure.

Although this paper has concentrated predominantly on the perspective of developing complex systems, agent-based computing should not be viewed merely as a good solution technology. Rather, it should be seen in its broader context as a general-purpose model of computation that naturally encompasses the major trends in software. In particular, there is an inexorable move towards regarding distributed and concurrent systems as the norm

rather than the exception, towards placing greater onus on flexible interactions between (independently developed) software systems, and towards reflecting real-world relationships (i.e., organisational context) in computer systems. In short, the agent-based approach should be regarded as the foundation of the networked generation of computer systems.

Acknowledgements

This paper is a revised and extended version of [31]. The work contained herein has benefited considerably from discussions with Grady Booch, Jean-Pierre Briot, Ed Durfee, Mike Fisher, Carl Hewitt, Mike Huhns, Van Parunak and Mike Wooldridge. The flexible manufacturing example was inspired by discussions with Stefan Bussmann, Van Parunak and Mike Huhns.

References

[1] ACM-SIGPlan, History of programming languages conference, SIGPLAN Notices 13 (8) (1978).
[2] G. Booch, Object-Oriented Analysis and Design with Applications, Addison-Wesley, Reading, MA, 1994.
[3] A.H. Bond, L. Gasser (Eds.), Readings in Distributed Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, 1988.
[4] F.B. Brooks, The Mythical Man-Month, Addison-Wesley, Reading, MA, 1995.
[5] B. Burmeister, A. Haddadi, G. Matylis, Application of multi-agent systems in traffic and transportation, IEE Proceedings on Software Engineering 144 (1) (1997) 51–60.
[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stahl, A System of Patterns, Wiley, New York, 1998.
[7] B. Chaib-draa, Industrial applications of distributed AI, Comm. ACM 38 (11) (1995) 47–53.
[8] S.H. Clearwater, Market-Based Control, World Scientific, Singapore, 1996.
[9] C. Castefranchi, Modelling social action for AI agents, Artificial Intelligence 103 (1–2) (1998) 157–182.
[10] R. Conte, C. Castelfranchi, Norms as mental objects: From normative beliefs to normative goals, in: Proc. AAAI Spring Symposium on Reasoning about Mental States, Stanford, CA, 1993.
[11] P. Courtois, On time and space decomposition of complex structures, Comm. ACM 28 (6) (1985).
[12] R. Davis, R.G. Smith, Negotiation as metaphor for distributed problem solving, Artificial Intelligence 20 (1983) 63–100.
[13] P. Faratin, C. Sierra, N.R. Jennings, Negotiation decision functions for autonomous agents, Robotics and Autonomous Systems 24 (3–4) (1998) 159–182.
[14] J. Ferber, Multi-Agent Systems, Addison-Wesley, Reading, MA, 1999.
[15] R. Flood, E. Carson, Dealing with Complexity, Plenum Press, New York, 1988.
[16] S. Franklin, A. Graesser, Is it an agent, or just a program?, in: J.P. Mueller, M.J. Wooldridge, N.R. Jennings (Eds.), Intelligent Agents III, Springer, Berlin, 1997, pp. 21–36.
[17] R.J. Gallimore, N.R. Jennings, H.S. Lamba, C.L. Mason, B.J. Orenstein, Cooperating agents for 3D scientific data interpretation, IEEE Trans. Systems Man Cybernet., Part C 29 (1) (1999) 110–126.
[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, Reading, MA, 1995.
[19] L. Gasser, Social conceptions of knowledge and action: DAI foundations and open system semantics, Artificial Intelligence 47 (1–3) (1991) 107–138.
[20] L. Gasser, J.-P. Briot, Object-based concurrent programming and distributed artificial intelligence, in: N.M. Avouris, L. Gasser (Eds.), Distributed AI: Theory and Praxis, Kluwer, Dordrecht, 1992, pp. 81–107.
[21] C. Guilfoyle, E. Warner, Intelligent agents: The new revolution in software, Ovum Report, 1994.
[22] F. Hayes-Roth, Towards a framework for distributed AI, SIGART Newsletter (1980) 51–52.
[23] C. Hewitt, Open Information System Semantics for Distributed Artificial Intelligence, Artificial Intelligence 47 (1–3) (1991) 79–106.

[24] C. Hewitt, J. Inman, DAI betwixt and between: From intelligent agents to open systems science, IEEE Trans. Systems Man Cybernet. 21 (6) (1991) 1409–1419.
[25] J. Huang, N.R. Jennings, J. Fox, An agent-based approach to health care management, Internat. J. Appl. Artificial Intelligence 9 (4) (1995) 401–420.
[26] F.F. Ingrand, M.P. Georgeff, A.S. Rao, An architecture for real time reasoning and system control, IEEE Expert 7 (6) (1992).
[27] P.C. Janca, Pragmatic application of information agents, BIS Strategic Report, 1995.
[28] N.R. Jennings, Towards a cooperation knowledge level for collaborative problem solving, in: Proc. 10th European Conference on Artificial Intelligence, Vienna, Austria, 1992, pp. 224–228.
[29] N.R. Jennings, Commitments and conventions: The foundation of coordination in multi-agent systems, Knowledge Engineering Review 8 (3) (1993) 223–250.
[30] N.R. Jennings, Controlling cooperative problem solving in industrial multi-agent systems, Artificial Intelligence 75 (2) (1995) 195–240.
[31] N.R. Jennings, Agent-based computing: Promise and perils, in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 1429–1436.
[32] N.R. Jennings, J.R. Campos, Towards a social level characterisation of socially responsible agents, IEE Proc. Software Engineering 144 (1) (1997) 11–25.
[33] N.R. Jennings, P. Faratin, M.J. Johnson, T.J. Norman, P. O'Brien, M.E. Wiegand, Agent-based business process management, Internat. J. Cooperative Information Systems 5 (2–3) (1996) 105–130.
[34] N.R. Jennings, E.H. Mamdani, J.M. Corera, I. Laresgoiti, F. Perriolat, P. Skarek, L.Z. Varga, Using ARCHON to develop real-world DAI applications. Part 1, IEEE Expert 11 (6) (1996) 64–70.
[35] N.R. Jennings, L. Varga, R.P. Aarnts, J. Fuchs, P. Skarek, Transforming standalone expert systems into a community of cooperating agents, Internat. J. Engineering Appl. AI 6 (4) (1993) 317–331.
[36] N.R. Jennings, M. Wooldridge, Agent-oriented software engineering, in: J. Bradshaw (Ed.), Handbook of Agent Technology, AAAI/MIT Press, 2000, to appear.
[37] N.R. Jennings, M. Wooldridge (Eds.), Agent Technology: Foundations, Applications and Markets, Springer, Berlin, 1998.
[38] V.R. Lesser, D.D. Corkill, The distributed vehicle monitoring testbed, AI Magazine 4 (3) (1983) 63–109.
[39] M. Ljunngberg, A. Lucas, The OASIS air traffic management system, in: Proc. 2nd Pacific Rim Conference on AI, Seoul, South Korea, 1992.
[40] A. Newell, The knowledge level, Artificial Intelligence 18 (1982) 87–127.
[41] N.J. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann, San Mateo, CA, 1998.
[42] B. Meyer, Object-Oriented Software Construction, Prentice Hall, Englewood Cliffs, NJ, 1988.
[43] R. Milner, Elements of interaction, Comm. ACM 36 (1) (1993) 78–89.
[44] H.V.D. Parunak, Industrial and practical applications of DAI, in: G. Weiss (Ed.), Multi-Agent Systems, MIT Press, Cambridge, MA, 1999, pp. 377–421.
[45] A. Pnueli, Specification and development of reactive systems, in: Information Processing 86, Elsevier/North-Holland, Amsterdam, 1986.
[46] R.S. Pressman, Software Engineering: A Practitioner's Approach, McGraw Hill, New York, 1997.
[47] A.S. Rao, D. Morley, M. Sekvestrel, G. Murray, Representation, selection and execution of team tactics in air combat modeling, in: Proc. 5th Australian Joint Conference on AI, 1992, pp. 185–190.
[48] J.S. Rosenschein, G. Zlotkin, Rules of Encounter, MIT Press, Cambridge, MA, 1998.
[49] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ, 1995.
[50] T. Sandholm, Distributed rational decision making, in: G. Weiss (Ed.), Multi-Agent Systems, MIT Press, Cambridge, MA, 1999, pp. 201–258.
[51] O. Shehory, S. Kraus, Methods for task allocation via agent coalition formation, Artificial Intelligence 101 (1–2) (1998) 165–200.
[52] Y. Shoham, M. Tennenholtz, On the synthesis of useful social laws for artificial agent societies, in: Proc. AAAI-92, San Jose, CA, 1992, pp. 276–280.
[53] H.A. Simon, The Sciences of the Artificial, MIT Press, Cambridge, MA, 1996.
[54] I. Sommerville, Software Engineering, Addison-Wesley, Reading, MA, 1998.
[55] C. Szyperski, Component Software, Addison-Wesley, Reading, MA, 1998.
[56] M. Tambe, Towards flexible teamwork, J. Artificial Intelligence Res. 7 (1997) 83–124.

[57] M. Waldrop, Complexity: The Emerging Science at the Edge of Order and Chaos, Simon and Schuster, New York, 1992.
[58] P. Wegner, Why interaction is more powerful than algorithms, Comm. ACM 40 (5) (1997) 80–91.
[59] M. Wooldridge, Agent-based software engineering, IEE Proc. Software Engineering 144 (1) (1997) 26–37.
[60] M. Wooldridge, N.R. Jennings, Intelligent agents: Theory and practice, Knowledge Engineering Review 10 (2) (1995) 115–152.
[61] M. Wooldridge, N.R. Jennings, Software engineering with agents: Pitfall and pratfalls, IEEE Internet Computing 3 (3) (1999) 20–27.
[62] M. Wooldridge, N.R. Jennings, D. Kinny, The Gaia methodology for agent-oriented analysis and designs, Internat. J. Autonomous Agents and Multi-Agent Systems 3 (2000), to appear.