MAS as Complex Systems: A View on the Role of Declarative Approaches

Andrea Omicini[1] and Franco Zambonelli[2]

[1] DEIS, Università degli Studi di Bologna a Cesena
via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it
[2] DISMI, Università degli Studi di Modena e Reggio Emilia
via Allegri 13, 42100 Reggio Emilia, Italy
franco.zambonelli@unimore.it

Abstract. The ever growing complexity of software systems calls for new forms of understanding and conceptual tools. It has been argued that some "Laws of Complexity" exist, which govern the behaviour of complex systems of any sort, from natural to artificial ones.
Along this line, in this paper we draw from the most recent findings of evolutionary biology to develop an original view over Multiagent Systems (MAS). A schema for a "layered", hierarchical view of MAS is introduced, aimed at providing computer scientists and engineers with a powerful conceptual framework for MAS observation / modelling / construction. We first introduce the three levels of the hierarchy in general, and then show how they impact on current proposals for methodologies of agent-oriented analysis and design. Finally, we exploit the hierarchy to provide an overall organised view over declarative approaches to MAS, by using as a reference those presented in the other contributions in this book.
On the one hand, a hierarchical view allows the many different approaches to be distinguished, classified and possibly compared. On the other hand, it makes it possible to reveal the richness and diversity of declarative models and technologies for MAS, as well as to show the multiplicity of ways in which they impact on MAS modelling and engineering.

1 Introduction

In the context of computer science, complexity is everyday more claiming for its own space among the main issues to address. Today computer-based systems are undeniably complex to design, complex to develop, and complex to maintain. The very promise of agent-oriented computing, indeed, is in fact to help computer scientists and engineers to envision and build complex artificial systems that they can manage. It is not commonly agreed, to some extent, whether agent abstractions and technologies are either another source of complexity – with respect to current mainstream object technologies, they surely are –, or rather the conceptual and practical tools required to harness the intrinsic complexity of today artificial systems. Indeed, they are likely to be both, and maybe

J. Leite et al. (Eds.): DALT 2003, LNAI 2990, pp. 1–16, 2004.

this is also a condition that they share with any newer and more sophisticated theory or technology making its appearance in any field, not only in computer science. For our purposes here, however, it suffices to agree on the fact that Multiagent Systems (MAS henceforth) can be generally seen as a class of complex software systems, wide and meaningful enough to legitimate in principle some sort of conceptual association between complex system in general, and MAS in particular.

Even though a satisfactory and shared definition of the very notion of complexity is not available, yet there is a common understanding that complexity raises the same sorts of problems within many diverse and heterogeneous areas of human knowledge. From this perspective, it has been argued [1] that some "Laws of complexity" exist, that cross the strict boundaries of traditional separations between distinct scientific disciplines. As a result, rather than useful or merely inspiring, exploiting inter- and trans-disciplinary approaches in order to understand complex system comes to be almost mandatory.

Accordingly, if some Laws of Complexity exist, they should apply to artificial systems, and to MAS in particular. On the one hand, this is likely to be why so many different techniques or theories coming from many different disciplines (biology, social sciences, organisational sciences, economics, politics, philosophy) are currently brought to agent-based models and systems [2,3,4,5]. On the other hand, this apparently legitimates extrapolation and adaptation of general results about complexity from heterogeneous research fields to computer science: or, at least, it makes it worth a try.

Under this perspective, evolutionary biology is surely one of the most intriguing related fields. There, in fact, the interest of the scientists lays not only in understanding the behaviour of complex biological systems, but also (and mainly) in modelling and motivating their evolution over time, their history – how they came to reach such a level of complexity from the very simple systems that marked the appearance of life on the Earth. In some sense, software systems are undergoing their own form of "evolution" too, moving from quite simple (algorithm-based) systems to intricate (interaction-based) systems [6] – e.g., from computer programs to MAS. Even though here we do not need to give too much meaning to such a transition, this fact indeed makes the parallel seem even more suggestive.

According to the *theory of hierarchies* [7], biological systems are amenable to be represented as organised on different layers: from genes and cells up to organisms, species and clades. On the one hand, each layer exhibits its own characteristic behaviour and dynamics, and it is governed by its own set of independent laws and rules. On the other hand, each layer is inextricably blended with the others, and strictly depends on both upper and lower layers. According to this view, every attempt to explain any non-trivial behaviour of biological systems – such as, of course, their evolution over time – has to be rooted in at least one of the layers, as well as in its interaction with the others along the whole hierarchy.

In this article, we start (Section 2) by taking as our reference the three levels devised out by Eldredge [8] (microevolutionary, intermediate, and macroevolutionary – see Subsection 2.1), and elaborate on how and to which extent this view can apply to software systems. So, in Subsection 2.2 we first essay the construction of a hierarchical view over complex software systems, by using MASs as our case study. Then, in order to demonstrate the potential of such a view, we critically discuss some key features of current agent-oriented analysis and design methodologies (Section 3), by showing how a hierarchical view is indeed necessary toward an effective approach to MAS modelling, engineering, and development. Finally, as the main aim of this article, we exploit the hierarchical view of MAS (Section 4) to recast the many different declarative approaches to MAS – as they emerge from this volume in particular – , so as to reveal the multiplicity of the levels at which declarative models and technologies can impact on the modelling and engineering of agents and MAS.

2 A Theory of Hierarchies for MAS

Indeed, evolutionary biology deals with one of the most difficult and articulated issues in the wide scope of human knowledge: not only how biological systems (possibly the most complex systems we have direct experience of) work, but also (and mainly) how they did come to work like they do during and after millions of years of evolution. As a result, once recognised that complexity has its own patterns that traverse the boundaries of traditional disciplines, drawing from the findings of evolutionary biology seems a quite natural way to try better understanding the ever growing complexity of software systems – and of MAS in particular – both in terms of its current state, and of its possibly otherwise unforeseeable directions.

2.1 Hierarchies for Complex Biological Systems

Broadly speaking, complex systems do not allow for simple explanations. A recasting of this apparently trivial observation in terms of the most recent results in evolutionary biology, is that complex systems call for "layered", hierarchical explanations. A first fundamental result is the so-called *theory of the hierarchies* [7]: in order to understand biological systems, and their evolution over time as well, several different "layers" has to be accounted for – from genes, to cells, up to organisms, species and higher taxa. While each layer is in some sense autonomous, and exhibit its own independent laws and dynamics, at the same time layers are organised in a hierarchy, each one strictly connected with the upper and lower levels, each parts of bigger totalities, each also wholeness composed of smaller parts. When observing / understanding / explaining a biological system, then, many different levels of abstraction can be adopted – which in the case of biological systems may correspond, for instance, to the gene, cell, organism, or species levels – and provide different but equally meaningful views over the systems.

Along this line, Eldredge [8] interprets evolution of biological systems as occurring at (at least) three levels: the microevolutionary (genes), the macroevolutionary (demes/species/clades), and the intermediate (organisms) one. In accordance with the theory of hierarchies, each level is essential to the general understanding of the system's wholeness, it is autonomous with its own laws, patterns and behaviours, but it can not be understood in isolation independently of all the other levels. For instance, while Dawkins [9] seems to ascribe the whole dynamics of biological evolution to the dynamics of (selfish) gene reproduction[1], Eldredge [8] and Gould [11] clearly show how the dynamics of individual organisms, demes, species, etc., as well as their interactions with the ever changing environment where they live and reproduce, irreducibly affect the process of biological evolution, which cannot be simplistically imputed to the genetic mechanisms alone.

Even though further works [11] argues that a higher number of different levels may befit a hierarchical view over biological systems, we do not proceed along this path any further: the three levels individuated by Eldredge already seem detailed enough to formulate a first hierarchical view of complex software systems, and of MAS in particular.

2.2 Hierarchies for Software Systems & MAS

Correspondingly, there are at least three levels of a potential hierarchy that we can use to observe / interpret / build complex software systems in general, and MAS in particular.

At the lower level, corresponding to the microevolutionary level of genes in biological systems, software systems exhibit what we can call the *mechanism* level, where the basic mechanisms for building system components are in place, and play the most relevant role. Things like programming languages, basic network protocols, and the corresponding mechanisms and abstractions have a deep influence on the sorts of systems that programmers and engineers can put together. Mechanisms and abstractions like parameter passing in C, closure in Lisp or ML, sockets in Java, anchors in HTML pages, or records in a DB system, provide the basic expressive power that is required for building any software system today, including MAS. It is quite obvious that no complex system could be either represented or built by focusing solely on the mechanism level – in the same way as no history of biological evolution could be fully understood at the microevolutionary level alone, that is, by adopting the genetic mechanisms as the only forces for evolution. Analogously, then, further upper levels are required other than the mechanism one for complex software systems: however, this level is essential indeed, since it provides the basic bricks for complex system observation and construction.

[1] Even though some recent work seems to minimise the "reductionist attitude" of Dawkins [10], his work is taken here as a frequently cited example of the widespread attitude of scientists to disregard complexity as a sort of implicit threat to the role and power of human science – and of human scientists, as well.

At this very first level, also, some of the main differences between biological and computer-based systems come out clearly – which are basically reducible to the differences between natural and artificial systems. First of all, the microevolutionary level and the corresponding genetic mechanisms are given once and for all (to some extent), and the work of biologists is to understand them fully, and devise out their influence on biological systems and their evolution as well. Instead, the understanding of the basic mechanisms and laws that regulate software systems at the mechanism level typically leads as a secondary effect to the selection of some mechanisms and to the definition of new ones. For instance, the long history of programming languages and of their specific linguistic mechanisms, shows how tools at the mechanism level are invented, then spread and used widely, and then accompanied / surpassed by new ones, meant to address either the limitations of the existing ones or newly emerged issues. Secondly, given that software systems are built by humans, differently from biological systems, they come along with the tools that are used to build them. Very simple tools such as code debuggers, profiling tools, or JavaDoc, provide computer scientists and engineers with a sort of embedded view over software systems that allow them to check, monitor and verify the system behaviour at the mechanism level. The same embedded view is obviously not available in the context of biological system – which is likely to be one reason why a lot of the genetic research is currently devoted to observe and represent exhaustively biological systems at the microevolutionary level.

At the highest level, by focusing now on MAS as our software systems for the sake of simplicity, we find notions like agent societies, organisations, teams, groups, and the very notion of MAS itself: we call this level the *social* one. This clearly matches the macroevolutionary level of biological systems, where demes, species, clades, and possibly phyla come into play. The first and main point here is that entities at the social level are to be interpreted as individual first-class entities, despite their apparent "plurality". In the same way as single populations or species exhibit their own dynamics, and either survive or die depending on their success as a whole in interacting with their environment (and with other populations or species, in particular), a MAS displays at the social level a global behaviour, whose conformance to the application requirements, for instance, determines its success as a whole. Social abstractions such as coordination media [12] or e-institutions [13] work as the basic bricks at this level – which is also the one where extra-agent notions, such as the agent infrastructure, become relevant, if not prevalent.

At the intermediate level, software components (agents, in MAS) obviously play the role of the individual entities of the system – basically, the same as the organisms in the biological setting. Focusing on MAS, we can easily name *agent level* such an intermediate level. What we conceptually see at this level of abstraction are things like agent architecture and inner dynamics, agent communication language and individual protocols, and the corresponding abstractions and mechanisms. BDI architecture, FIPA speech acts, agent semantics specifications – all come into play at the agent level, along with the troublesome (and

boring, in the end) issue of "what an agent really is". The dynamics of this level of the hierarchy is quite obvious in some sense: a good deal of the agent literature struggles at this level, actually. However, what we learn from the theory of hierarchy is that each level is autonomous and independent in the laws that rule its behaviours, but that no satisfactory view of the whole can be given at one level alone. This comes to say that any view of complex software systems, and of MAS in particular, should account for all the three levels – the mechanism, the social, and the component/agent levels – and that failure to do so may result in a global failure of any attempt to observe / model / build such systems. As a paradigmatic example, in the next section we will focus on Agent-oriented Software Engineering (AOSE) techniques, tools and methods, to show how the different views over a complex software system like a MAS affect the way in which such system is conceived, designed and engineered.

3 A Hierarchical View of AOSE

In the context of AOSE research, it is seemingly easy to devise out the three levels of the MAS hierarchy depicted above. So, even though typically they are not distinguished, nor are they explicitly revealed in current AOSE practice as well as in recently proposed agent-oriented analysis and design methodologies, nevertheless such three levels are to be identified, so as to clearly point out how they affect the way in which complex MASs are modelled and designed.

3.1 The Mechanism Level

What are the basic "ingredients" to build an agent? That is, the basic bricks required to build a computational organism capable of autonomous actions and decision making? Although such a question seems to be of paramount importance, as also outlined in the discussion of the previous section, current AOSE practice seems to mostly disregard it.

In the area of agent-oriented methodologies, the basic assumption underlying most proposal (sometimes implicit, as in TROPOS [14], or explicit, as in [15]) is that the mechanism level does not influence at all the way a MAS is modelled and designed. The mechanism level is assumed to influence the implementation phase only, and the modelling and design can proceed independently of it.

Other specific proposals for agent-oriented analysis and design methodology assume a totally different, and possibly more radical position, by stating that the basic underlying mechanisms will be that of, say, specific programming environments or paradigms. This is the case, for instance, of most mobile agent methodologies [16], which always consider Java (and related technologies and middleware) as the target development technology.

Both endeavours have advantages and drawbacks.

On the one hand, assuming a technology-neutral perspective enables designers to adopt a more modular approach to system design, and at the same time frees them from having to deal with low level implementation details. However, this may also lead to troubles during the following implementation phases,

where the required commitment to a specific technology could force developers to adopt tricky and ineffective solutions to match the design requirements against the technology constraints. In other words, with reference to what we stated in Subsection 2.2, such an endeavour misses in explicitly considering the reciprocal influences of the mechanisms, agent, and social levels.

On the other hand, assuming the presence of a specific technology allows for very efficient designs, that can be subsequently implemented without any mismatch. However, this makes it very hard to produce a general understanding of the characteristics of a complex MAS and to produce re-usable and modular model and design.

In our personal perspectives, some sort of intermediate approach should be enforced, so as to preserve the clear separation between the different layers of the hierarchy, but at the same time to take into account the influence of the underlying mechanisms over both the agent and the social level.

3.2 The Agent Level

Mechanisms are used to build agents and societies where agents live. As far as agents are concerned, the discussion about what an agent really is and how it should be modelled has been around for several years, and a considerable amount of research work still struggles about such an issue. Such a discussion appears dramatically important in the modelling and design of complex MASs, in that it implies defining the essential features of the organisms, that is, of the individuals, that will populate such MASs. Nevertheless, the discussion could become more constructive by simply recognising the fact that defining exactly what an agent is is indeed a matter of modelling and design choices, rather than an absolute truth about the agent-oriented paradigm.

It is clearly possible to identify different classes of entities, all exhibiting different degrees of situatedness, autonomy and decision-making capabilities, that we could call agents with some good reasons. For instance, according to the taxonomy introduced by Maja Mataric [17], one can identify the following classes:

- reactive agents, which are basically capable of perceiving events and, based on an internal state, perform some sort of reactive autonomous action in direct reaction to events or to inter-agent communications;
- deliberative agents, which are capable of internal planning and internal decision-making aimed at achieving specific goals, and which can dynamically change their plan based on the perceived events or the occurred inter-agent communications;
- behavioural agents, which locates somewhere in between reactive and deliberative agents. As deliberative agents, behavioural agents are capable of autonomous goal-oriented decisions making. However, as in reactive agents, their overall activity (i.e., the goals they have to achieve) is strictly related to their interaction / communication history.

The first class typically includes most mobile agent systems and embedded agents [16], the second one includes BDI agents [18], while the third class is typical of situated robots [19].

In the area of agent-oriented methodologies, agents are definitely the main focus of the design activity. That is, unlike the mechanism level, the agent level is always explicitly taken into account. Still, most methodologies fail in specifying what an agent is to them, and why the defined methodology suits that specific class of agent. In other words, most methodologies propose guidelines for building complex societies of agents without actually saying what types of agents they are considering. This is the case, for instance, of Gaia [15] and MASE [20], where all that is said is that an agent is an entity capable of autonomous actions, of communicating with other agents, and of sensing events in the environment, three features that all the three classes of agents introduced have to exhibit, and that makes such a characterisation of an agent non-informative at all.

The consequence of this missing specifications at the agent level is that several important aspects are left unspecified, that are likely to somewhat influence both the mechanism and the societal level. As a result, first of all such missing specifications may be the sources of further mismatches between design and implementation (besides those identified in the previous subsection) when it comes to implementing agents. Even more, specific assumptions about agent capabilities (thus belonging to the agent level) may somewhat affect the global design of a MAS (that is, the social level). For example, the capability of agents of sensing events and re-forging their goals on the basis of such events, is an individual agent feature that clearly influences the overall behaviour of the system.

In some cases, the specification of the reference agent type is indirectly suggested by the methodology itself. For instance, most methodologies for mobile agent systems assume a reactive agent model [21]. Only in a very few cases, such a specification is made explicit. One of these cases is that of the Prometheus methodology [22], which is explicitly conceived for MASs made of of BDI (that is, deliberative) agents.

Summarising, the construction of a complex system, as a MAS is, and the full modelling of its structure and dynamic behaviour, requires a precise and unambiguous definition of the features considered as essential at the agent level.

3.3 The Social Level

The ultimate goal of the process of an agent-oriented methodology analysis and design is that of building a MAS as a whole, capable of achieving a given application goal in an efficient and reliable way within a target operational environment. The resulting MAS will of course include a (possibly varying) number of agents and, thus, agents are the primary necessary components of it.

However, agents alone do not represent the full picture in MASs. On the one hand, a MAS may exhibit collective behaviours that cannot easily be modelled in terms of the behaviour of its individual components. On the other hand, a MAS may involve more components than agents and may require the explicit modelling of such additional components. As a simple example, consider the case

of an agent-based Web information system, or of an agent-mediated marketplace. In both cases, in addition to agents, the overall MASs include a computational environment made up of digital information and digital goods, and may include some sort of security infrastructure influencing the actions of the agents by, e.g., inhibiting malicious opportunistic behaviours. In general terms, the proper modelling and engineering of a MAS requires an explicit modelling of the societal aspects, i.e., of the environment in which the MAS situates and of the social laws to which the agents of the MAS must obey.

Most methodologies for agent-oriented analysis and design proposed so far plainly disregard the social level, and consider the specification of a MAS as complete for the only fact of having accurately specified the set of agents that are part of it. This is the case, for example, of the first version of the Gaia methodology [23] and of the MASE methodology [20], where neither the MAS environment nor any type of social laws are taken into any account.

The authors of this paper have been pioneering the adoption of explicit social abstractions in MAS analysis and design since several years ago [24], and have proposed and implemented several effective mechanisms (e.g., tuple-based social interaction spaces [25]) to support the development and execution of socially enriched MASs, i.e., of MAS explicitly dived in an environment and explicitly obeying a set of social laws. Also, they have contributed at enriching the Gaia methodology with the necessary set of environmental and social abstractions [26,15].

Recently, several other proposals have started recognising the need for an explicitly modelling of the social level. For instance, it is now acknowledged that the design of open MASs where multitudes of self-interested agents interact in an opportunistic way, requires modelling the "social laws" that rule agent interactions. This has led to the general concept of "agent institutions" [13], intended as societies of agents interacting in accord to an explicitly modelled set of "institutionalised" rules.

The explicit modelling of the social level in the engineering of a MAS is very important, and may influence both the agent level and the mechanism level. At the agent level, it is important to recognise that the autonomous decision-making capabilities of an agent are likely to be somewhat limited by the social level (e.g., by the super-imposition of social laws), and to model agents and their capabilities accordingly. At the mechanism level, it has to be acknowledged that the super-imposition of social laws may require specific mechanism as, e.g., programmable interaction media to mediate and rule interactions [27]. Therefore, the appropriate modelling of the social level cannot abstract from the societal mechanisms available, and vice versa. In addition, as a general consideration to be accounted for at every level, it is important to outline that the environment in which a MAS situates (specifically, the dynamics of such an environment) may affect the global behaviour of a MAS in unpredictable ways, and that the actual effectiveness of a MAS strictly depends on its capabilities of preserving an acceptable behaviour independently of environmental factors.

4 A Hierarchical View of Declarative Approaches to MAS

The old, silly question "Why should I use Prolog to program?" has become "Why should I use Prolog for my agents?" in the agent age – and it is possibly even sillier than it was then. On the one hand, the range of the declarative and logic-based models and technologies developed in the last ten years is so wide that talking of Prolog only is simply non-sense. On the other hand, moving from simple programs and algorithms to the elaborate structure of MASs makes the space for logic-based approaches even larger than it was before. Along this line, the multi-layer, hierarchical view over agent systems introduced in this paper is exactly meant to provide a well-defined conceptual framework where to classify and evaluate the potential contributions of declarative and logic-based approaches to the modelling and engineering of MAS.

In order to avoid excessive spreading of this paper scope, and also to promote a unitary view of this book, we will draw our cases and examples from the papers that follow this one, discussing and framing them accordingly. So, in the rest of this section, we will exploit the hierarchy for MASs devised out in the previous sections in order to illustrate the many levels of contributions that may come from declarative approaches to MAS, by taking as our main reference the remaining contributions of this book.

4.1 The Mechanism Level

The shift from current mainstream object/component-based technology to agent technology could not be more evident when taking into account programming languages at the mechanism level. Take, for instance, Java as the mainstream reference technology for the mechanism level, and consider on the one hand the intermediate (component) level in component-based approaches, on the other hand the intermediate (agent) level in agent-based approaches. Intuitively, the distance between the mechanism level and the component level in component-based approaches is all but large. Think, for instance, of the small conceptual distance between the Java language, at the mechanism level, and a JavaBeans component, at the component level: few lines of properly written yet simple Java code make a working JavaBeans. On the other hand, the incredible amount of the efforts to build up a suitable support for agents upon Java-based technology in the last years (see JADE as a fundamental example of this [28]) witnesses the much larger distance between the mechanism and the agent levels in agent-based approaches. Whichever non-trivial acceptation of an agent one may adopt, a Java agent will easily not be a simple chunk of Java code: think, for instance, of the mere implementation of agent social abilities in term of properly handling a fully-fledged agent communication language.

Such a distance is the main reason why declarative technologies and logic-based languages come so strongly into play in the agent arena: abstractions and metaphors at the agent level require mechanisms that directly support them – and embedding an agent knowledge base or a BDI architecture upon, say,

Prolog, is clearly much easier than doing the same upon, say, Java. It is not by chance, then, that most declarative approaches to MAS place themselves to the mechanism level, but in such a way to clearly head toward the agent level, by either implicitly or explicitly incorporating agent-level abstractions and models. Picking up the examples from this book, ϕ-log and GOLOG [29] are first of all logic-based languages, but the whole ϕ-log project (that by the way is concerned with evolutionary biology) promotes an agent architecture that clearly belongs to the agent level. Analogously, [30] basically introduces the multi-paradigm programming language Go!, which incorporates a series of (quite heterogeneous) linguistic mechanisms, but also provides a blackboard-based agent architecture that should be more properly placed at the agent level. Logic-based mechanisms are introduced by [31] (the mechanisms of Restricted Entailment) and [32] (a non-standard multi-modal logic) to support abstractions and models at the agent level – such as agent reasoning about its own ignorance. Finally, [33] defines an agent-oriented logic language, clearly belonging to the mechanism level, that takes into account the agent-level (embodying the agent level with the BDI) and looks at the social level, through Speech-Act Communications.

In the end, applying the hierarchical view to declarative approaches to MAS makes a couple of relevant things clear at the mechanism level. Firstly, current object-oriented mainstream languages (such as Java and C++) implicitly define a mechanism level that is too poor for MASs: a simple logic language like Prolog intrinsically provides a more expressive support for agent-based models and abstractions. Secondly, Prolog itself does not directly endorse any agent level metaphor: this explains why most of the research at this level (as demonstrated by the contributions in this book, too) concerns the definition of a technological platform at the mechanism level to adequately support and promote suitably expressive declarative models at the agent level.

4.2 The Agent Level

At the agent level, the same problems pointed out in Subsection 3.2 seem to be evident in the literature on declarative and logic-based approaches. That is, on the one hand, the virtual absence of a plain and unambiguous discussion about the notion of agent supported (and the motivations behind it, as well) affects many of the research works in this area, too. On the other hand, the power of declarative approaches as an expressive means to denote and model architectures and behaviours of systems is clearly fundamental for MASs at the agent level.

As a first example of this, [34] introduces a formal framework for agent specification, that promotes a layered description (in terms of multiple "tiers") of agent architecture and behaviour. Also, BDI-based approaches clearly belongs to the agent level, since they explicitly define the inner structure and dynamics of an agent. In this perspective, [35] generalises BDI logics to encompass the notions of observation / expectation for an agent, while Coo-BDI [36] extends the original BDI model with the notion of agent cooperativity – which builds a bridge toward the upper social level. Analogously, the social issue of cooperative

problem solving is one of the main target of [37]: there, Linear Logic is basically used to model agents as social entities.

More generally, it easy to see that, at the agent level, declarative and logic-based models work well both as formal tools for the description and definition of agent architectures and behaviours, and as sources of metaphors and abstractions to enrich and empower the individual agent model. So, in the same way as in the case of AOSE literature, the fact that a good deal of the literature on declarative approaches to MAS is concerned with the agent level, but anyway disregards the central point of the precise and exhaustive definition of what an agent is, does not prevent relevant results and promising directions to emerge.

For instance, as it comes out from some of the examples in this book, one of the most frequent "implicit definitions" for agents is the BDI one: in short, one agent is such if it is built around a BDI architecture [35,36]. On the one hand, the BDI model constitutes a conceptual bridge between the mechanism and the agent level, since it provides an architectural principle for agents that can be supported by suitably designed languages at the mechanism level. On the other hand, it is not by chance that BDI is essentially a logic-based model. In fact, whenever features like deliberative capabilities, intelligence, and sociality are taken as defining agent traits, declarative and logic-based approaches are typically the most suitable ones at the agent level, since they provide the adequate expressive power as well as the required level of abstraction – as obvious, for instance, in the case of knowledge-based agents. Less obvious, but manifest from works like [36], is that expressiveness of declarative models is such that not only it helps in bridging between the mechanism and the agent levels, but also enables the agent level to encapsulate and promote abstractions that directly refer to the upper, social level.

4.3 The Social Level

Intuitively, the social level is where the complexity of MAS is going to explode. Handling a MAS composed by possibly hundreds or thousands of agents, as an open system where both known and unknown agents coexist and interact in an unpredictable way, is obviously more than a challenge to MAS engineers. For this very reason, the social level promises to be the one where declarative models and technologies have the potential to provide the most relevant contribution: for instance, by allowing system properties to be assessed at design time, and then enforced at run time by suitable declarative technologies, independently of the MAS dynamics, and of the MAS environment as well.

This is exactly the case of the notion of social integrity constraints [38], which formalises within a logic-based framework social concepts such as violation, fulfilment, social expectation, that can be enforced at run-time through a suitably-defined logic-based infrastructure. The infrastructural concept of institution is then endorsed by other two papers of this book: the notion of Basic Institution is formally defined in [39], founded on the social interpretation of agent communicative acts, while Logic-based Electronic Institutions [40] are first-order logic tools aimed at the specification of open agent organisations. Finally, the inte-

gration of DALI and Lira [41] shows a very interesting example of an agent technology vertically spanning the whole MAS hierarchy. While DALI is a logic language (at the mechanism level) that implicitly embeds an agent model (at the agent level), Lira works at the social level – and their integration definitely aims at providing a complete declarative framework for MAS engineering.

Even though there are already several interesting applications of declarative models and technologies at the social level (such as the ones included in this book and just cited above), the potential of such approaches in taming the complexity of MAS is seemingly so high that in the near future the authors expect to see many other works pursuing this very promising direction. XML was maybe the first declarative technology to become mainstream: along this line, while artificial systems grow everyday more complex and intricate, many others of this sort are likely to follow.

5 Final Remarks

Drawing from the field of evolutionary biology to computer science does not imply that we endorse a view of software systems as biological systems – not so plainly, at least. Instead, it simply means that, at the current stage of technology and human knowledge, there are similarities and commonalities enough between the two sorts of systems that an interdisciplinary approach to complexity in software systems seems worth to explore.

Indeed, the undisciplined use of interdisciplinarity is generally dangerous: for its obvious fascination, as well as for the virtual absence of shared and well-defined "operating instructions". The risk of unfruitful speculations, led just for the seek of scientific curiosity, but basically unpurposefully, is surely high and never to be underestimated. However, apart from the obvious consideration that *a priori* undirected speculation is part of the "normal" scientific activity [42], there are several manifest good reasons to adopt the interdisciplinary path we followed along this paper.

First of all, as discussed at the beginning of this article, complexity has its own ways, that prescind from the strict boundaries of individual scientific disciplines. This makes monodisciplinary approaches surely myopic and potentially sterile, and calls for a broader view over the fields of interest that should not be limited any longer to computer science and artificial intelligence, and possibly even trespass the boundaries of cognitive sciences.

Also, the ever growing intricacy of software systems, and of MASs in particular, ask computer scientists to provide their own vision of future problems and solutions, and their own view of the possible directions. Otherwise, the evolution of pervasive artificial systems like MASs is going to be driven mostly by the urgency of (possibly non consistent) market or societal pressures, rather than by a clear understanding of the system's potential benefits and dangers. Such a vision is unlikely to be found now within computer science – too young and immature is the field, today, from a scientific viewpoint, to provide such deep insights over future evolutions of computer-based systems. It is then natural to look for conceptual contributions toward more mature fields, where complexity

spreads its wide wings both over space and over time – evolutionary biology being obviously one of such fields, and one of the most intriguing and stimulating, by far: from there, some inspiring and unfathomed visions of the directions that artificial systems like MASs are going to take are more likely to come.

Within the field of computer science, agent models and technologies promise to provide the conceptual and practical tools to face the ever increasing complexity of computer-based systems. On the one hand, research on AOSE straightforwardly deals with the complexity issue. In fact, AOSE abstractions and technologies aim at enabling software engineers to build increasingly complex MAS (thus addressing a form of "spatial" complexity, related to MAS dimension), whereas AOSE processes and methodologies are meant to harness the intricacies of MAS engineering from analysis to deployment (thus addressing a form of complexity "over time", related to the MAS development history). On the other hand, declarative models and technologies are by their very nature made to face complexity. In fact, roughly speaking, declarativeness allows properties of systems to be denoted / imposed while transcending the system dynamics.

However, a consideration like the one above seems obviously too generic to actually promote declarative approaches in the MAS context: a more precise and articulated vision is required, of all the many benefits that declarative, logic-based models and technologies may bring to the agent field, and to AOSE in particular. Under this perspective, the approach based on the Theory of Hierarchies presented in this article, and applied first to AOSE in general, then to declarative models and technologies, may well represent a first, meaningful step along this direction.

Acknowledgements

The authors are grateful to the many people whose work and remarks have helped in conceiving and shaping the material that has been used in this paper. In particular, we are deeply indebted to Paolo Torroni, for his persistence in trying to convince us to write this article, as well as for his continuous support and enduring patience that have made this work possible. Also, we would like to thank all the authors that contributed to the DALT 2003 workshop, and to this book in particular, which provided us with such a vast amount of relevant material that we could use to build up this article.

This work has been partially supported by MIUR, Project COFIN 2003 (ex 40%) "Fiducia e diritto nella società dell'informazione", by MIPAF, Project SIPEAA "Strumenti Integrati per la Pianificazione Eco-compatibile dell'Azienda Agricola", and by the EC FP6 Coordination Action "AgentLink III".

References

1. Kauffman, S.A.: Investigations. Oxford University Press (2001)
2. Calmet, J., Daemi, A., Endsuleit, R., Mie, T.: A liberal approach to openess in societies of agents. [43]

3. Tolksdorf, R., Menezes, R.: Using Swarm intelligence in Linda systems. [43]
4. Feltovich, P.J., Bradshaw, J.M., Jeffers, R., Suri, N., Uszok, A.: Social order and adaptability in animal and human cultures as analogues for agent communities: Toward a policy-based approach. [43]
5. McBurney, P., Parsons, S.: Engineering democracy in open agent systems. [43]
6. Wegner, P.: Why interaction is more powerful than computing. Communications of the ACM **40** (1997) 80–91
7. Grene, M.J.: Hierarchies in biology. American Scientist **75** (1987) 504–510
8. Eldredge, N.: Unfinished Synthesis: Biological Hierarchies and Modern Evolutionary Thought. Oxford University Press (1985)
9. Dawkins, R.: The Selfish Gene. Oxford University Press (1989)
10. Sterelny, K.: Dawkins vs. Gould. Survival of the Fittest. Revolutions in Science. Icon Books Ltd., Cambridge, UK (2001)
11. Gould, S.J.: The Structure of Evolutionary Theory. The Belknap Press of Harvard University Press (2002)
12. Omicini, A., Ossowski, S.: Objective versus subjective coordination in the engineering of agent systems. In Klusch, M., Bergamaschi, S., Edwards, P., Petta, P., eds.: Intelligent Information Agents: An AgentLink Perspective. Volume 2586 of LNAI: State-of-the-Art Survey. Springer-Verlag (2003) 179–202
13. Noriega, P., Sierra, C.: Electronic institutions: Future trends and challenges. In Klusch, M., Ossowski, S., Shehory, O., eds.: Cooperative Information Agents VI. Volume 2446 of Lecture Notes in Computer Science. Springer Verlag (2002) 6th International Workshop (CIA 2002), Madrid, Spain, September 18-20, 2002. Proceedings.
14. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: A knowledge level software engineering methodology for agent oriented programming. In: 5th International Conference on Autonomous Agents (Agents 2001). ACM Press, Montreal, Canada (2001) 648–655
15. Zambonelli, F., Jennings, N.R., Wooldridge, M.J.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology **12** (2003) 417–470
16. Karnik, N.M., Tripathi, A.R.: Design issues in mobile-agent programming systems. IEEE Concurrency **6** (1998) 52–61
17. Mataric, M.J.: Situated robotics. Encyclopedia of Cognitive Science (2002)
18. Kinny, D., Georgeff, M., Rao, A.: A methodology and modelling technique for systems of BDI agents. In Van de Velde, W., Perram, J.W., eds.: Modelling Autonomous Agents in a Multi-Agent World. Volume 1038 of LNAI. Springer-Verlag (1996) 56–71 7th International Workshop (MAAMAW'96), 22–25 January 1996, Eindhoven, The Netherlands.
19. Mataric, M.J.: Integration of representation into goal-driven behaviour-based robots. IEEE Transactions on Robotics and Automation **8** (1992) 59–69
20. Wood, M.F., DeLoach, S.A., Sparkman, C.H.: Multiagent system engineering. International Journal of Software Engineering and Knowledge Engineering **11** (2001) 231–258
21. Cabri, G., Leonardi, L., Zambonelli, F.: Engineering mobile agent applications via context-dependent coordination. IEEE Transactions on Software Engineering **28** (2002) 1034–1051
22. Padgham, L., Winikoff, M.: Prometheus: A methodology for developing intelligent agents. In: 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002). ACM Press, Bologna, Italy (2002)

23. Wooldridge, M.J., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems **3** (2000) 285–312
24. Ciancarini, P., Omicini, A., Zambonelli, F.: Multiagent system engineering: The coordination viewpoint. In Jennings, N.R., Lespérance, Y., eds.: Intelligent Agents VI. Agent Theories, Architectures, and Languages. Volume 1757 of LNAI., Springer-Verlag (2000) 250–259 6th International Workshop (ATAL'99), Orlando, FL, USA, 15–17 July 1999. Proceedings.
25. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems **2** (1999) 251–269 Special Issue: Coordination Mechanisms for Web Agents.
26. Zambonelli, F., Jennings, N.R., Omicini, A., Wooldridge, M.J.: Agent-oriented software engineering for Internet applications. In Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R., eds.: Coordination of Internet Agents: Models, Technologies, and Applications. Springer-Verlag (2001) 326–346
27. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Science of Computer Programming **41** (2001) 277–294
28. Bellifemine, F., Poggi, A., Rimassa, G.: JADE – a FIPA-compliant agent framework. In: 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'99). (1999) 97–108
29. Son, T.C., Pontelli, E., Ranjan, D., Milligan, B., Gupta, G.: An agent-based domain specific framework for rapid prototyping of applications in evolutionary biology. In this volume.
30. Clark, K.L., McCabe, F.G.: Go! for multi-threaded deliberative agents. In this volume.
31. Flax, L.: A proposal for reasoning in agents: Restricted entailment. In this volume.
32. van der Hoek, W., Lomuscio, A.: A logic for ignorance. In this volume.
33. Moreira, Á.F., Vieira, R., Bordini, R.H.: Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In this volume.
34. Bergenti, F., Rimassa, G., Viroli, M.: Operational semantics for agents by iterated refinement. In this volume.
35. Tran, B.V., Harland, J., Hamilton, M.: A combined logic of expectations and observation (a generalisation of BDI logics). In this volume.
36. Ancona, D., Mascardi, V.: Coo-BDI: Extending the BDI model with cooperativity. In this volume.
37. Küngas, P., Matskin, M.: Linear logic, partial deduction and cooperative problem solving. In this volume.
38. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Modeling interaction using *Social Integrity Constraints*: A resource sharing case study. In this volume.
39. Colombetti, M., Fornara, N., Verdicchio, M.: A social approach to communication in multiagent systems. In this volume.
40. Vasconcelos, W.W.: Logic-based electronic institutions. In this volume.
41. Castaldi, M., Costantini, S., Gentile, S., Tocchio, A.: A logic-based infrastructure for reconfiguring applications. In this volume.
42. Kuhn, T.S.: The Structure of Scientific Revolutions. The University of Chicago Press, Chicago (1962)
43. Omicini, A., Petta, P., Pitt, J., eds.: Engineering Societies in the Agents World IV. LNAI. Springer-Verlag (2004) 4th International Workshop (ESAW 2003), London, UK, 29–31 October 2003. Post-proceedings.