Multi-Agent Systems – Theory, Approaches and NASA Applications

MIKE HINCHEY and EMIL VASSEV
Lero— the Irish Software Engineering Research Centre
mike.hinchey@lero.ie, emil.vassev@lero.ie

Abstract. This chapter introduces the multi-agent paradigm and presents concepts and approaches related to multi-agent systems and paradigm-inspired popular research topics such as grid computing, intelligent swarms, sensor networks, service-oriented computing, cloud computing and autonomic computing. Additionally, the chapter reflects the authors' experience and presents a review of the NASA multi-agent applications of ground control and space-exploration systems. The Multi-Agent Systems approach helps NASA systems deal efficiently with huge amounts of data and to perform unmanned tasks in deep space where single monolith spacecraft are impractical.

Keywords. multi-agent systems, ground control systems, space-exploration systems, NASA.

1. Introduction

The concept of multi-agent systems arose to help researchers in artificial intelligence (AI) develop theories, techniques and systems that overcome the constraints of a single cognitive entity and tackle more complex, realistic and large-scale problems. Multi-Agent Systems (MAS) deal with problems which are beyond the capabilities of an individual agent. Because the capacity of an individual agent is limited by its knowledge, its computing resources and its perspective, the goal of *multi-agent systems research* is to find methods that help us build complex systems composed of autonomous agents which operate on both local and distributed knowledge and overcome their individual limitations through cooperative work and shared goals. The basic idea behind a multi-agent system is that of a society of autonomous software agents acting independently, but interacting to achieve user and system-wide goals. By giving each agent the ability to act independently of the others, many of the common disadvantages associated with large-scale distributed computing such as synchronization, single points of failure, and the difficulty of modifying the system at runtime can be mitigated. Therefore, research in MASs is mainly concerned with the study, behavior and construction of a *society of autonomous agents* that interact with each other and their environment. Note that to study and develop a MAS properly, we must go beyond the individual intelligence of a single agent and pay attention to the problem-solving capabilities of the entire system, i.e., problem-solving that has social aspects [1]. As Durfee and Lesser point out in [2], a MAS can be defined as a loosely

coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver.

This chapter discusses the multi-agent paradigm, elaborates the MAS concept and briefly presents some of the most popular research initiatives inspired by and derived from MASs. Moreover, to show the practical application of the multi-agent paradigm, we give an overview of some of the MAS applications developed at NASA. The rest of the chapter is organized as follows. Section 2 elaborates the definition and types of agent, presents the multi-agent paradigm, and gives an overview of contemporary MAS research initiatives. Section 3 pays special attention to autonomic computing as one of the most significant research initiatives built on top of the multi-agent paradigm. Section 4 presents MAS applications developed at NASA and finally Section 5 concludes with a short summary.

2. Agent and Multi-agent Systems

Definitions of "agent" have been the subject of much debate and, as a result, the computer science community has come up with various such definitions [3 – 7]. Although all the definitions referenced above are different, they all share a basic set of concepts: the notion of an agent, its environment, and autonomy. For example, according to Wooldridge's definition [7], an agent is "a software (or hardware) entity that is situated in some environment and is able to autonomously react to changes in that environment". Going into more detail [8], an agent can be viewed as an entity that perceives its environment through special *sensors* and acts upon that environment through *actuators* (or also called effectors). As shown in Figure 1 illustrating this definition, an agent interacts with the surrounding environment by using sensors in order to observe certain aspects of its environment and actuators to perform actions and eventually change the environment.

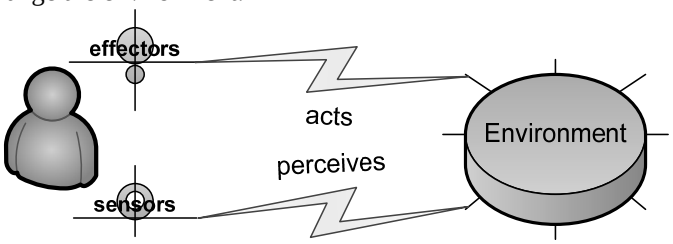

Figure 1. An agent perceives and acts in its environment.

Ideally, the environment is everything external to the agent. The environment may be physical (e.g., the computer hardware) or it may be the computing environment (e.g., data sources, computing resources, and other agents). Moreover, an agent is not only situated in its environment, but it may also act *autonomously* in response to environmental changes [7]. The concept of *autonomy* means that an agent exercises control over its own actions, thus eventually resulting in scheduling certain actions for execution. Thus, a common agent function f is to map the perceptions to actions performed in the environment, i.e., an agent program runs to produce f:

$$[f: P^* \rightarrow A] \quad (1)$$

Note that the clear borderline between the two notions – *agent* and *environment*, leads to the conclusion that agents are inherently distributable.

2.1. Core Attributes of Agents

Despite the diversity of the agent definitions, there are some essential attributes typical for all the agents called the "notions of agency" [9]. These attributes are divided into three categories: *weak*, *stronger* and *optional*. The *weak notions*, listed below, are considered essential:

- *Autonomy*. Autonomous agents may respond unpredictably (or later, or not at all) to external influence, such as messages from other agents; agents can say "no." Thus, most practical agents include some kind of thread of control.
- *Reactivity*. Agents are typically situated in a rich and dynamic environment. Environmental changes are detected using sensors, and changes are affected using actuators (or effectors). Agents are "reactive," in that they will respond in a timely way to certain environmental conditions.
- *Proactivity.* An agent will select appropriate programmer-defined actions or plans, and execute them in order to achieve its goals. Actions and plans may fail, because the environment may change at any time. Thus, agents must be prepared to adapt.
- *Social ability.* MAS use social concepts, such as speech-like communication, cooperation, and competition, in order to organize their activities effectively.

The *stronger notion* of agency represents a much narrower definition: *the agent's internal state should be described or implemented in human or "mentalistic" terms*. So an agent may have *knowledge*, *belief*, *desires*, *intentions*, *capability*, *trust* and even *emotions*. The *optional notions* are:

- *Rationality*. An agent will always act to achieve its goals.
- *Veracity*. An agent will never deliberately mislead.
- *Benevolence.* An agent will never adopt directly conflicting goals.
- *Mobility*. An agent may migrate between hosts across a network.

2.2. Intelligent Agents

Intelligent agents are a special category of *autonomous agents* providing additional concepts that may help us realize intelligent systems, e.g., systems capable of self-management. This is the reason why a great deal of research effort is devoted to developing the intelligent agent technology, which has become a rapidly growing area of research and new application development. Although very popular, there is no commonly agreed definition of the term "intelligent agent". Probably the most popular definition is the one given by IBM research as:

> *"Intelligent agents are software entities that carry out some set of operations on behalf of a user with some autonomy and employ either knowledge or representation of the user's goals and desires."* [10]

However, there is consensus that *autonomy*, i.e., the ability to act without human intervention or such of other systems, is the key feature of an intelligent agent. Table 1

presents some common features of agents and those of intelligent agents as presented in the Agent Builder toolkit [11], a framework for constructing intelligent agents.

Table 1. Agent Characteristics

Agent	Autonomous.
	Able to communicate with other agents and user.
	Monitors the state of its execution environment.
Intelligent Agent	Able to use symbols and abstractions.
	Able to exploit significant amounts of domain knowledge.
	Capable of adaptive goal-oriented behavior.
	Capable of operation in real-time.
	Able to learn from the environment.
	Able to communicate with the user.

Here, the central idea of agents is delegation of tasks that an agent can autonomously perform on behalf of the user who delegates those tasks. Thus, in order to be operational, an agent must be able to communicate with the user to receive instructions and provide results back to the same.

Based on the level of intelligence, we distinguish four basic types of intelligent agents (increasing intelligence) [8]:

- simple reflex agents;
- model-based reflex agents;
- goal-based agents;
- utility-based agents;
- learning agents.

A simple reflex agent (see Figure 2) implements a mechanism based on condition-action rules that help the agent to "make the connection from percept (a single perception) to action.

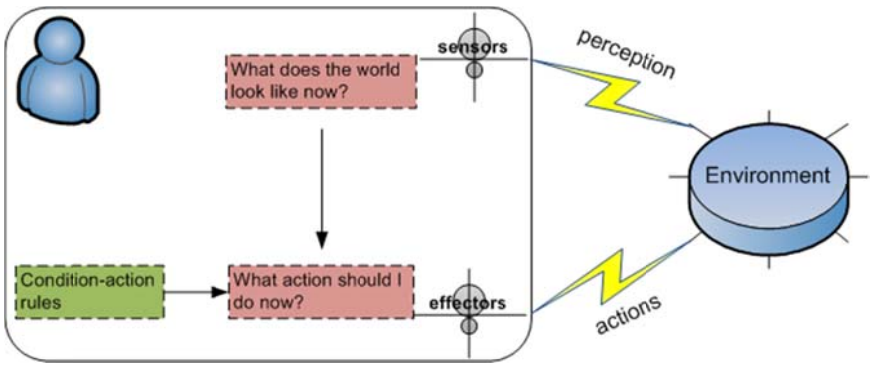

Figure 2. Structure of a single reflex agent.

A single reflex agent performs correctly only if the single perception of the environment leads to a correct decision. Thus, as shown in Figure 2, such an agent does not keep any track of the environment changes and how performed actions change that environment. This approach does not allow an agent to make decisions based on previous decisions. A more intelligent variant of the single reflex agent is the model-based agent, which has 1) the ability to track how the environment evolves independently of the agent; and 2) information about how the agent's actions affect the environment. Figure 3 shows the structure of such an agent where the current percept is combined with the old internal state to generate an updated perception of the current *world state* (model).

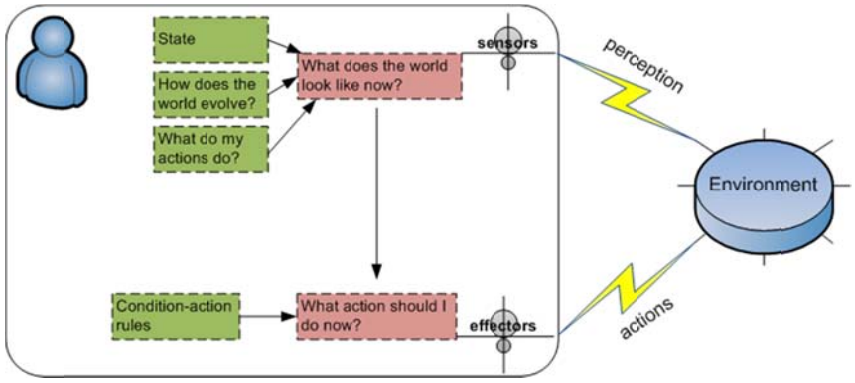

Figure 3. Structure of a model-based reflex agent.

Note that a reflex agent would need to be reprogrammed each time the goal changes, as it has no concept of a goal. A *goal-based agent* is more flexible than the reflex agents in that it uses goals to change its behavior (see Figure 4). In addition to the current state description, a goal-based agent incorporates some sort of goal information that further describes what situations are desirable. Thus, having a goal helps the agent determine what the correct action to take next is.

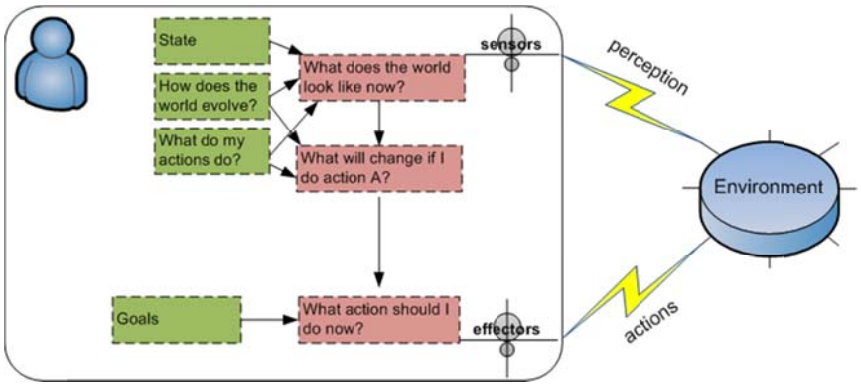

Figure 4. Structure of a goal-based agent.

Goals alone are not enough in generating a sophisticated behavior in most environments. Goals alone just describe a crude binary distinction between *happy* and *unhappy* states, but offer no information about how, or at what cost, the agent can get to that state [8]. To overcome this problem, the utility-based agents implement a special *utility function* that maps a state, or a sequence of states, onto a real number, which describes the associated level of happiness. The utility function is responsible for adjusting compromise between two conflicting goals (speeding vs speeding-tickets). It is also responsible for weighing up the likelihood of success against the importance of the goal and selecting what goal to aim for next, in situations where success is probabilistic and not guaranteed. This uncertainty is inherent in the so-*called partially observable environments* (see the environment types below).

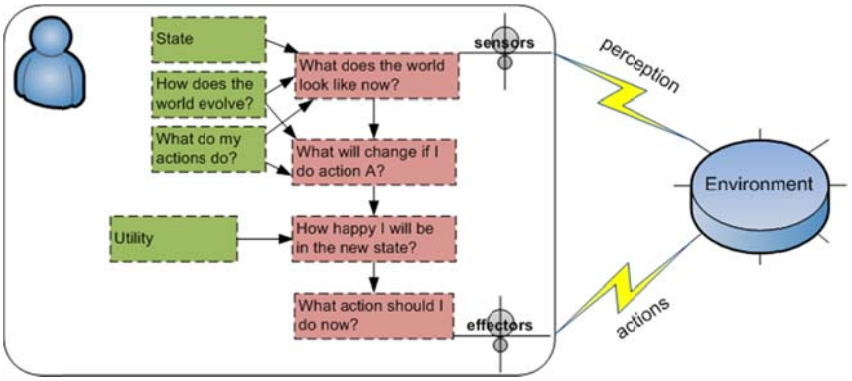

Figure 5. Structure of a utility-based agent.

With the advancement of new MAS initiatives such as *autonomic computing* (see Section 3) emerged a new class of intelligent agents, which we call *learning agents* (see Figure 6). Such agents (also called Autonomic Elements - see Section 3.1) incorporate a special *control loop* that helps them self-check the world's states against their goals expressed as service-level objectives (SLO). In Section 3.1, this approach is presented in more details.

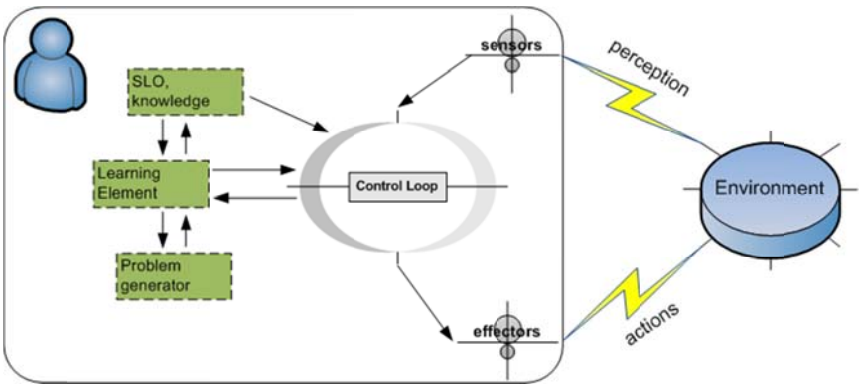

Figure 6. Structure of a learning agent.

An agent's behavior can be based on both its own experience and the built-in knowledge used in constructing the agent for the particular environment in which it operates. Therefore, environment knowledge plays a crucial role in an agent's behavior. There are a few important properties that must be considered to model properly environment knowledge. Environment can be [8]:

- *Fully observable* (vs. partially observable). An agent's sensors give it access to the complete state of the environment at each point in time.
- *Deterministic* (vs. stochastic). The next state of the environment is completely determined by the current state and the action executed by the agent.
- *Episodic* (vs. sequential). The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.
- *Static* (vs. dynamic). The environment is unchanged while an agent is deliberating. The environment is semi-dynamic if the environment itself does not change with the passage of time but the agent's performance score does.
- *Discrete* (vs. continuous). A limited number of distinct and clearly defined percepts and actions.
- *Single agent* (vs. multi-agent). An agent operating by itself in an environment.

2.3. Multi-Agent Systems

A MAS can be perceived as a society of *autonomous agents* sharing global system goals, but acting independently to achieve those goals. Nowadays MAS technology is being used for a wide range of control applications including scheduling and planning [12, 13], diagnostics [14], condition monitoring [15 – 17], distributed control [16, 18], hybrid control [19], congestion control [20], [21], system restoration [22], market simulation [23, 24], network control [24, 25], and automation [26]. In addition, FIPA (the Foundation for Intelligent Physical Agents) [38], an official standards committee of the IEEE Computer Society, imposes some standards for building MASs (e.g., standard for inter-agent communication).

Practice has shown that MASs are not always the best approach to software engineering. However, there are a few important contexts where MASs are most suitable:

- The environment is complex, dynamic, and unpredictable. Thus, modelling based on the agent paradigm assists in raising levels of abstraction in the design process.
- Data and expertise are geographically distributed. Agents may be particularly suitable where communication between the nodes is intermittent or of variable bandwidth.
- The solution requires interoperation between several existing systems. Agents are especially useful if those systems cannot be modified, for example, because they are operated by different organizations.
- The intentional stance seems particularly suitable at the design stage; for example, if the solution requires human-like behaviors such as negotiation or competition.

Combining multiple agents in a single system can lead to the so-called *emergent behavior*, which is a system-level behavior due to the aggregation of multiple agents, and not directly attributable to any particular agent properties. In short, the whole is greater than sum of the parts. The prevailing (traditional) view has been that emergence is undesirable in the software engineering context, although it has much to offer in newer classes of applications. Note that artificially emergent systems are notoriously difficult to test and modify.

2.4. Popular Research Venues

To date, many research and development initiatives have arisen for developing technologies and frameworks for multi-agent systems. In this section, we describe some of the most popular research venues in the field of multi-agent systems.

2.4.1. Grid Computing

Grid computing relies on self-managing features to achieve two distinct but related goals: 1) to provide remote access to IT infrastructures; and 2) to aggregate computational resources. The general idea behind grid computing [27] is to make efficient use of computational resources with the power of multi-agent system paradigm. A computing grid is conceptually like an electrical grid, but composed of computers that form a MAS. All the computers within a computing grid form a computational structure by coupling wide-area distributed resources.

These resources are considerably scalable and can be *databases*, *storage servers*, *high-speed networks*, *supercomputers*, etc. The key concept of the grid computing is the transparent access to the computational resources in the grid irrespective of their physical source. Thus, in order to ensure resource transparency, grid computing introduces a special self-managing middleware that is used to coordinate disparate IT resources across a grid network. As a result, these resources are transparently exposed to the final user as a virtual whole.

2.4.2. Intelligent Swarms

Artificial Intelligence (AI) emerged over 50 years ago when Alan Turing introduced his prototype for intelligent machines. For over five decades, AI research has gone over tremendous evolution conceiving research fields such as natural language processing (including speech recognition and speech generation), data mining, machine learning, automated reasoning, neural networks, genetic algorithms, fuzzy logic, etc. Intelligent agents provide both concepts and technologies that are products of this very evolution. Today, intelligent agents are considered one of the key concepts that may help us realize self-regulating systems. This is the reason why a great deal of research effort is devoted to developing *intelligent agent technology*, which has become a rapidly growing area of research and new application development. An extension of the "intelligent agent" paradigm (see Section 2.2) is the so-called *intelligent swarm systems* [36], where intelligent agents are considered to be autonomous entities that interact either cooperatively or non-cooperatively (on a selfish base). Intelligent swarm systems are MASs based on biologically-inspired concepts (see Section 3.2). By their nature, such systems are complex multi-agent systems where the individual members of the swarm imply independent intelligence. Traditionally, a swarm-based system offers many advantages compared with the single-agent system, such as greater redundancy,

reduced costs and risks, and the ability to distribute the overall work among the swarm members, which may result in greater efficiency and performance.

A good example of an intelligent swarm system is the NASA ANTS (Autonomous Nano-Technology Swarm) mission, a swarm-based exploration mission representing a new class of concept exploration missions based on the cooperative nature of hive cultures [37]. A mission of this class necessitates special autonomous mobile agents exhibiting both self-organizing and self-adapting features. Note that a swarm-based space-exploration system has the ability to explore regions of space where a single large spacecraft would be impractical.

2.4.3. Sensor Networks

A *sensor network* (SN) [28] is a network consisting of interconnected autonomous intelligent devices equipped with sensors to provide cooperative monitoring. Although equipped with intelligent software, each individual network node of a SN is inherently resource constrained in terms of limited processing speed, storage capacity, and communication bandwidth. As a result, the network nodes have substantial joint processing capability and not so-significant individual computational power. Sensor networks operate over sensors collecting and processing data in diverse domains such as air quality control, weather forecast, traffic control, security and surveillance applications, etc. In most SN applications, a network is intended for a long-term operation and the SN nodes are wireless and must provide autonomous features to some extent. For example, a SN node must operate in an optimal energy-saving mode, i.e., it must monitor its energy resources and find the optimal tradeoff between performance and long-term operation. Here, to minimize energy consumption, most of the node's hardware components, such as radio, are likely to be turned off most of the time. Note that these factors make the networking protocol highly complex, which necessitates special capabilities to tackle self-organizing and self-healing problems.

2.4.4. Web Services

Web services (WSs) share some principles with MASs characterized by the concept of program-to-program interactions [29]. By its nature, a web service is an abstraction of a collection of operations that are network-accessible. Each web service (ideally an intelligent agent) is presented in the network of WSs with a special service description. This description provides all the details necessary to make the interaction with a service possible, such as *message formats*, *transport protocols*, *service location*, etc. WSs imply the so-called service-oriented architecture (SOA). Initially, SOA sets forth three service roles and three service operations:

- roles - service provider, service requester, and service registry;
- operations - publish, find, and bind.

One of the key issues for a broad range of WSs, such as admission control, server selection, scheduling, pricing, and specifying service-level agreements, is the quality of service (QoS). The problem is that the QoS delivered to a customer is highly affected by various factors, e.g., performance of the web service itself, performance of the service provider (hosting platform), and performance of the underlying network. Thus, in order to achieve a high level of QoS, the software behind WS must incorporate autonomous features that can deal with the issues of service publishing, service discovery, and especially with the issue of service selection. A solution to the problem

of autonomous WSs is the Web Service Level Agreement (WSLA) framework developed at IBM [30]. WSLA is a framework for specifying, creating, and monitoring special service level agreements for web services. Note that WSLA complements various other WS-related software approaches addressing issues on proactive management of a web service environment, e.g., provisioning resources, workload management, etc.

2.4.5. Cloud Computing

Cloud computing is the commercial evolution of *grid computing* [31] relying on the inherited features of distributed computing and the MAS paradigm to provide users with a set of abstracted, virtualized and dynamically-scalable storage, platforms, and services delivered on demand over the Internet. In general, a cloud represents a "large pool of computing and/or storage resources, which can be accessed via standard protocols via an abstract interface". Services provided by cloud computing can be broadly grouped into three major categories:

- Software-as-a-Service (SaaS) – emphasizes end-user applications delivered as services. Such services are special-purpose software that is remotely accessible by consumers through the Internet with a usage-based pricing model.
- Platform-as-a-Service (PaaS) – provides an independent platform (or middleware) as a service that can be used by developers to build and deploy service applications. Common solutions provided in this category range from APIs and tools to databases, business-process management systems and security integration that help developers build applications and run them as services on a cloud platform.
- Infrastructure-as-a-Service (IaaS) – provisions the hardware and technology for computing power, storage, operating systems or any other equipment required to deliver software application environments with on-demand services rather than as on-side resources.

The main objective of cloud computing is to provide software, services and computing infrastructures carried out independently by the network. This concept is based on the development of dynamic, distributed and scalable software. To build cloud computing environments, developers rely on existing Service-Oriented Architectures (SOA) and agent frameworks, which provide tools for developing distributed systems and multi-agent systems that can be used for the establishment of a cloud of services.

3. Autonomic Computing

Autonomic computing (AC) [32] emerged as the new multi-agent paradigm, where the agents are cooperative (i.e., share common objectives) but also autonomous and capable of self-management [33]. In general, AC is a concept to describe computing systems that are intrinsically intended to reduce complexity through automation. Today, many researchers agree on the fact that AC provides the set of capabilities required to make computing systems self-managing via inherent self-adaptation. Typically, adaptive systems are contingent upon varying environments where they can reason

about the environment and adapt to changing situations [34]. In that context, AC emphasizes adaptive features and other self-managing ones to provide solutions to one or more real world problems in one or more domains spanning intra or inter-domain applications. The "Vision of Autonomic Computing" [33] defines the concept of self-management as comprising four basic policies – *self-configuring*, *self-healing*, *self-optimizing* and *self-protecting*. In addition, in order to achieve these self-managing objectives, an autonomic system (AS) must constitute the following self-* properties:

1) *self-awareness* – aware of its internal state;
2) *self-situation* (context awareness) – environment awareness, situation and context awareness;
3) *self-monitoring* – able to monitor its internal components;
4) *self-adjusting* (self-adaptiveness) – able to adapt to the changes that may occur.

Thus, despite their differences in terms of application domain and functionality, all ASs are capable of self-management and are driven by one or more self-management objectives. Note that this requirement automatically involves 1) *self-diagnosis* (to analyze a problem situation and to determine a diagnosis), and 2) *self-adaptation* (to repair the discovered faults). The ability to perform adequate self-diagnosis depends largely on the quality and quantity of system knowledge. Moreover, AC recognizes two modes of self-healing: *reactive* and *proactive*. In reactive mode, an AS must effectively recover when a fault occurs, identify that fault, and when possible repair the same. In proactive mode, an AS monitors vital signs to predict and avoid health problems or reach undesirable risk levels.

3.1. Autonomic Element – The New Concept of Intelligent Agent

A key concept in the AC paradigm is the so-called autonomic element (AE), which is a powerful *learning agent* (see Section 2.2). A widely accepted architecture for autonomic systems considers AEs as the system's building blocks [32]. By its nature, an AE extends programming elements (i.e., objects, components, services) to define a self-contained piece of software with specified interfaces and explicit context dependencies. Essentially, an AE encapsulates rules, constraints and mechanisms for self-management, and can dynamically interact with other AEs of the AS in question to provide or consume computational services. As stated in [32], the basic structure of an AE is a special *control loop*. The latter is described as a set of functionally related units: *monitor*, *analyzer*, *planner* and *executor*, where all share knowledge. By sharing knowledge, those units form an intelligent control loop that forms the self-managing behavior of the AE in question.

A closer look at the generic structure of an AE is presented by Figure 7. As depicted, an AE operates over a special managed resource. The latter is a generic presentation of software that can be managed by the control loop in order to leverage its functionality to a self-managing level. Here, through its control loop, an AE monitors the managed resource details, analyzes those details, plans adjustments, and executes the planned adjustments. It is important to mention that for these activities, an AE can use information from humans (administrators) as well as rules and policies defined (by humans) and learned by the AS [32]. Thus, a control loop helps an AE make decisions and controls the managed resource through monitoring and intensive interaction. Note that the managed resource is highly scalable [32], i.e., it can be any

software system such as a file, a server, a database, a network, a middleware software, a standalone software application, a business object, etc.

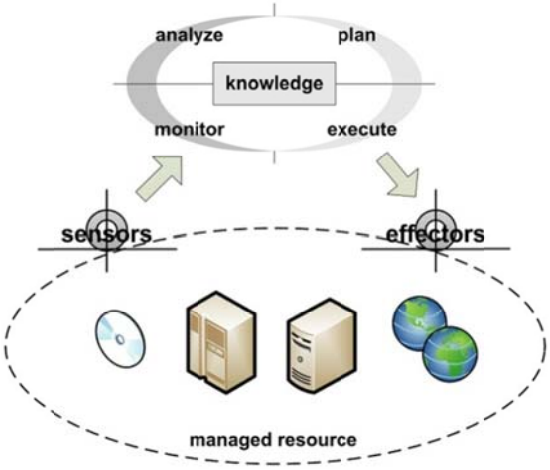

Figure 7. AE Architecture

A key factor in the success of AE self-management is the shared solution knowledge. Note that the lack of solution knowledge is a drawback for self-managing. For example, today there are a number of maintenance mechanisms for installation, configuration, and maintenance. The vast diversity in system administration tools and distribution packaging formats creates unexpected problems in the administration of complex systems. In addition, the same problems are leveraged by the use of dynamic environments where application functionality can be added and removed dynamically. Here, common solution knowledge removes the complexity introduced by diversity in formats and installation tools. Moreover, when acquired in a consistent way, this very knowledge is used by the AE's control loop in contexts different than configuration and maintenance, such as problem formation and optimization.

3.2. Biological Inspiration

Inspiration from biology necessitated paradigms related to ASs. The mechanisms behind the principles of self-management and self-governance are inspired by the human body's Autonomic Nervous System (ANS). In fact, this is the focus of AC. The idea is that mechanisms that are autonomic, in-built, and requiring no conscious thought in the human body are used as inspiration for building mechanisms that will enable a computer system to become *self-managing* [32]. Hence, the general properties necessitated from ANS for an AS can be summarized by the four AC self-managing objectives: being self-configuring, self-healing, self-optimizing and self-protecting. Note that the self-managing objectives cannot be considered independently. Similar to the complex behavior observed in the human body, an AS adapts to changes by following encoded objectives. For example, if a malicious attack is successful, this will necessitate self-healing actions, and a mix of self-configuration and self-optimization actions, in the first instance to ensure dependability and continued operation of the system, and later to increase self-protection against similar future attacks [35]. Moreover, being computing software, an AS goes even further by ensuring that

autonomic features add minimal impact over the system performance, thus avoiding significant delays in processing.

Another biological inspiration that copes with the multi-agent nature of the ASs comes from the so-called social animals, e.g., the social insects such as ants, bees, etc. Colonies of ants or other insects, flocks of birds, swarm of bees, herds of animals, schools of fish, etc., also inspired approaches to building ASs, such as *evolutionary systems* or highly-efficient and *highly-complex distributed multi-agent systems* consisting from large numbers of largely homogeneous components (or agents). Insects have tremendous diversity in color, shape and size and their behavior is interesting to observe. For example, the so-called *emergent behavior* [36] inspired a special class of self-managing systems called "intelligent swarms" (see Section 2.4.2), e.g., the NASA's ANTS (Autonomous Nano-Technology Swarm) prospective mission [37]. This behavior helps social insect colonies to collectively solve complex problems without centralized control. Thus, colony (or swarm) behavior appears out of local interactions between individuals with simple rule sets and no global knowledge.

3.3. Developing Autonomic Systems with ASSL

Formal methods have proven to be a valuable approach to the development of ASs. The advantage is that AC formal methods provide developers with special formal notations (AC formalisms) that help developers build AS specifications that serve as a basis for further design, implementation, and verification of ASs. Formalisms dedicated to AC have been tackled by a variety of industrial and university projects. One of the most prominent AC-dedicated formal methods is the Autonomic System Specification Language (ASSL) [39]. ASSL is a declarative specification language and framework for ASs with well-defined semantics. It implements modern programming language concepts and constructs like inheritance, modularity, type system, and high abstract expressiveness.

ASSL [39] is based on a specification model exposed over hierarchically organized formalization tiers (see Table 2). This specification model provides both infrastructure elements and mechanisms needed by an AS (autonomic system). Each tier of the ASSL specification model is intended to describe different aspects of the AS in question, such as *service-level objectives, policies, interaction protocols, events, actions, autonomic elements*, etc. This helps to specify an AS at different levels of abstraction (imposed by the ASSL tiers) where the AS in question is composed of special autonomic elements (AEs) interacting over interaction protocols (IPs). As shown in Table 2, the ASSL specification model decomposes an AS in two directions: 1) into levels of functional abstraction; and 2) into functionally related *sub-tiers*. The first decomposition presents the system at three different tiers [39]:

1) *a general and global AS perspective* – we define the general system rules (providing autonomic behavior), architecture, and global actions, events, and metrics applied in these rules;
2) *an interaction protocol (IP) perspective* – we define the means of communication between AEs within an AS;
3) *a unit-level perspective* – we define interacting sets of individual computing elements (AEs) with their own autonomic behavior rules, actions, events, metrics, etc.

The second decomposition presents the major tiers AS, ASIP and as composed of functionally related sub-tiers, where new AS properties emerge at each sub-tier.

Table 2. ASSL multi-tier specification model.

AS	**AS Service-level Objectives**	
	AS Self-management Policies	
	AS Architecture	
	AS Actions	
	AS Events	
	AS Metrics	
ASIP	**AS Messages**	
	AS Channels	
	AS Functions	
AE	**AE Service-level Objectives**	
	AE Self-management Policies	
	AE Friends	
	AEIP	**AE Messages**
		AE Channels
		AE Functions
		AE Managed Elements
	AE Recovery Protocols	
	AE Behavior Models	
	AE Outcomes	
	AE Actions	
	AE Events	
	AE Metrics	

The AS Tier specifies an AS in terms of *service-level objectives* (AS SLOs), *self-management policies*, *architecture topology*, *actions*, *events*, and *metrics* (see Table 2). The AS SLOs are a high-level form of behavioral specification that helps developers establish system objectives such as performance. The *self-management policies* are driven by *events* and trigger the execution of *actions* driving an AS in critical situations. The metrics constitute a set of parameters and observables controllable by an AS. With the ASIP Tier, the ASSL framework helps developers specify an AS-level interaction protocol as a public communication interface expressed with special *communication channels*, *communication functions*, and *communication messages*. At the AE Tier, the ASSL formal model exposes specification constructs for the specification of the system's AEs. Note that AEs are considered to be analogous to software agents able to manage their own behavior and their relationships with other AEs. An AE may also specify a private *AE interaction protocol* (AEIP) shared with special AE considered as "friends" (AE Friends tier).

It is important to mention that the ASSL tiers are intended to specify different aspects of the AS in question, but it is not necessary to employ all of them in order to develop an AS. Conceptually, it is sufficient to specify self-management policies only, because those provide self-management behavior at the level of AS (the AS tier) and at the level of AE (AE tier). These policies are specified within the AS/AE Self-management Policies sub-tier (the ASSL construct is AS[AE]SELF_MANAGEMENT) with special ASSL constructs termed *fluents* and *mappings* [39]. A fluent is a state where an

AS enters with fluent-activating events and exits with fluent-terminating events. A mapping connects fluents with particular actions to be undertaken. Usually, an ASSL specification is built around one or more self-management policies, which make that specification AS-driven. Self-management policies are driven by events and actions determined deterministically. The following ASSL code presents a sample specification of a self-healing policy.

```
ASSELF_MANAGEMENT {
    SELF_HEALING {
        FLUENT inLosingSpacecraft {
            INITIATED_BY { EVENTS.spaceCraftLost }
            TERMINATED_BY { EVENTS.earthNotified } }
        MAPPING {
            CONDITIONS { inLosingSpacecraft  }
            DO_ACTIONS { ACTIONS.notifyEarth } }
    }
} // ASSELF_MANAGEMENT
```

As shown, fluents are expressed with *fluent-activating* and *fluent-terminating* events. In order to express mappings, conditions and actions are considered, where the former determine the latter in a deterministic manner. Once a specification is complete, it can be validated with the ASSL built-in verification mechanisms (e.g., consistency checking) and a functional application skeleton can be generated automatically. The application skeletons generated with the ASSL framework are fully-operational multithreaded event-driven applications with embedded messaging.

4. Multi-agent Systems at NASA

4.1. NASA Swarm Missions

The Autonomous Nano Technology Swarm (ANTS) concept sub-mission PAM (Prospecting Asteroids Mission) is a novel approach to asteroid belt resource exploration under development at NASA. By its virtue, ANTS provides extremely high autonomy, minimal communication requirements with Earth, and a set of very small explorers (intelligent agents) with few consumables [37]. These explorers, forming the swarm, are pico-class, low-power, and low-weight spacecraft units, yet capable of operating as fully autonomous and adaptable agents for multiple years in space. The agents in a swarm are able to interact with each other, thus helping them to self-organize based on the *emergent behavior* of the simple interactions. The swarm exhibits self-organization since there is no external force directing its behavior and no single agent has a global view of the intended macroscopic behavior. Such type of behavior is observed in insects and flocks of birds (see Section 3.2).

4.1.1. ANTS Architecture

Figure 8 gives an overview of the ANTS concept mission. A transport spacecraft launched from Earth carries a laboratory that will assemble the tiny spacecraft. Once it reaches the L1 Lagrangian point, where gravity forces are balanced, the transport releases the assembled swarm, which will head for the asteroid belt. Each spacecraft is equipped with a solar sail, which means it relies primarily on power from the sun, using only tiny thrusters to navigate independently. Moreover, each spacecraft also has

onboard computation, artificial intelligence, and heuristics systems for control at the individual and team levels. Spacecraft use low bandwidth to communicate within the swarm and high bandwidth for data transfer back to Earth.

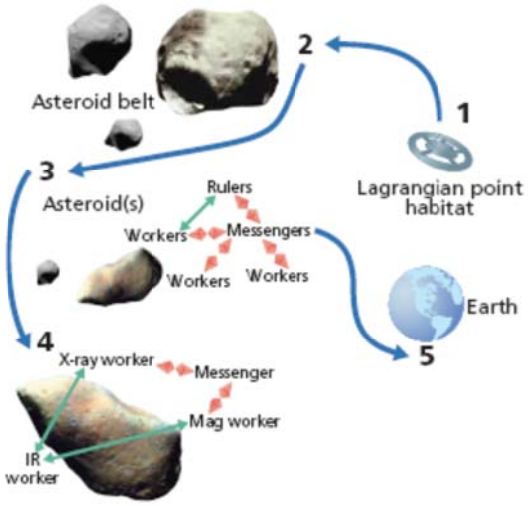

Figure 8. ANTS Mission Concept [37]

As Figure 8 shows, teams consist of spacecraft units from three classes of spacecraft, and members in each class combine in certain ways to form teams that explore particular asteroids:

- *Workers*, up to 80 percent of the swarm, bear the instruments and gather data. Instruments can include a magnetometer, *x*-ray, gamma-ray, visible/infrared, or neutral mass spectrometers. A worker gathers only its assigned data types.
- *Rulers* coordinate data gathering through the use of rules about what asteroid types and data are of interest.
- *Messengers* coordinate communications among the workers, rulers, and mission elements on Earth. Messengers, for example, can alert NASA to send replacement spacecraft from Earth or spacecraft with additional instruments.

The internal organization of a swarm depends on the global task to be performed and on the current environmental conditions. In general, a swarm consists of several sub-swarms, which are temporal groups organized to perform a particular task. Each swarm group has a group leader or *ruler*, a messenger, and a number of workers. A team comprises workers carrying a specialized instrument, a ruler playing the role of a team leader, and optionally one or more messengers. The latter connect a worker with other workers or with the ruler from the same team.

4.1.2. ANTS Autonomic Properties

For ANTS exploration, individual autonomy is not crucial, but the mission cannot succeed unless each *team* has all the autonomic properties of being. There are four such properties, which by their nature do not have clear boundaries.

- *Self-configuration.* ANTS must be able to adapt to changes in the system. Moreover, ANTS must be fully reconfigurable to support concurrent

exploration and examination of hundreds of asteroids. Reconfiguration may also be required because of failure or anomaly of some sort.

- *Self-optimizing.* ANTS must be able to improve their performance on the fly. Self-optimization is important to mission efficiency. Leaders can use the gained experience to self-optimize, thus improving their ability to identify asteroids. Messengers, strive to find the best position to improve the communication among the swarm units. Workers also self-optimize through learning and experience.
- *Self-healing*. ANTS must be able to recover from errors or damage. Any ANTS unit, teams, sub-swarms, and the entire swarm must be able to recover from both mistakes and failures, including those caused by damage due either to a solar storm or to a collision with an asteroid or another spacecraft.
- *Self-protecting.* ANTS must be to anticipate and cure intrusions. For example, ANTS must protect itself from solar storms, where charged particles can degrade sensors and electronic components, or destroy the solar sails.

4.1.3. Developing Autonomic Properties of ANTS with ASSL

ASSL (see Section 3.3) has been successfully used to develop prototype models for a variety of *autonomic features* of ANTS. In this section, we briefly present the ASSL approach to developing a self-healing property for ANTS. Ideally, this section is intended to show the roadmap to a successful autonomic MAS in terms of *formal specification, formal verification* and *implementation* (code generation). For more details, the interested reader is advised to consult [40].

In ANTS, self-healing is about recovering from failures, including those caused by damage due to a crash or any outside force. In our scenario, we assume that each *worker* sends, on a regular basis, *heartbeat messages* to the *ruler*. The latter can use those messages to determine when a *worker* is not able to continue its operation, due to a crash or malfunction in its communication device. Moreover, a *worker* sends a *notification message* to the *ruler* if its instrument is malfunctioning or it has been broken, due to a crash with an asteroid or another spacecraft. Thus, a ruler is notified in two ways for a worker loss:

- a heartbeat message from the worker has not been received;
- a message from the worker, notifying for a broken instrument, has been received.

Once the loss of an operational unit has been detected, the ruler checks if the number of workers is below the critical minimum, and if so, it requests a replacement from another ruler. If such a replacement is not possible it could notify the ground control on Earth of the situation and request a replacement or further instructions from there. An ASSL specification of the ANTS self-healing behavior requires a specification at the AS tier for the global ANTS behavior and at the AE tier for the self-healing behavior of every *worker* and *ruler* of the swarm.

In order to specify the self-healing autonomic property of a worker, we used a SELF_HEALING self-management policy specified at both AS tier and AE tiers (*worker* and *ruler*) (to consult the ASSL specification model, see Table 2). The SELF_HEALING policy is specified with a set of *fluents* and *mappings* (see Section 3.3) where the latter map the *fluents* to ASSL *actions*. In addition, we specify the necessary ASSL *actions*, *events* and *metrics* driving the spacecraft rulers and workers (recall that those are

agents). Moreover, three interaction protocols are specified to handle the communication between the spacecraft (inter-agent communication). Those protocols specify the messages to be exchanged between the *workers* and *rulers*, the *communication functions*, and two *communication channels.*

By using the ASSL framework, we specify an AS (or MAS) at an abstract formal level. Next, that formal model is translated into a programming model consisting of units and structures that inherit names and features from the ASSL specification. Recall that the framework is responsible for ensuring consistent specification before proceeding to the programming model. Thus, from the ASSL specification of ANTS self-healing, an operational Java application was generated [40]. In general, the architecture of the generated programming model (see Figure 9) conforms to the ASSL multi-tier specification model. Every AS is generated with [39]:

- a global AS autonomic manager (implements the AS tier specification) that takes care of the AS-level *self-management policies* and *SLO*;
- a communication mechanism (implements the specifications of both ASIP and AEIP tiers; see Section 3.3) that allows AEs to communicate with a set of ASSL messages via ASSL channels;
- a set of AEs (implement the AE tier's specification) where every AE takes care of its own *self-management policies* and *SLO*.

Note that both the AS autonomic manager and the AEs incorporate a distinct *control loop* (see Learning Agent in Section 2.2 and AE in Section 3.1) and orchestrate the self-management policies of the entire system. The AS autonomic manager is a sort of coordinator for the AEs (autonomic elements or learning agents). This coordination goes over AS-level *self-management policies*, *SLO*, *events*, *actions*, and *metrics*.

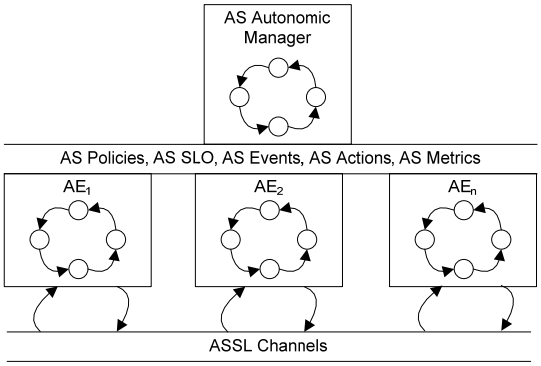

Figure 9. AS architecture for ASSL

4.2. The Livingstone Project

Livingstone [41] is a MAS developed at NASA Ames [42] as a *health monitoring system*. It uses a special symbolic, qualitative model of a physical system, such as a spacecraft, to infer its state and diagnose faults. Livingstone is one of the three parts of the Remote Agent (RA), an autonomous spacecraft controller developed by NASA Ames Research Center jointly with the Jet Propulsion Laboratory. The two other components are called Planner and Smart Executive. Whereas the former generates flexible sequences of tasks for achieving mission-level goals, the latter commands

spacecraft systems to achieve those tasks. Remote Agent was demonstrated in flight on the Deep Space One mission (DS-1) in May 1999, which made it the first case where the control of an operational spacecraft was overtaken by AI software [43]. Moreover, Livingstone is used in other control applications such as the control of a propellant production plant for Mars missions [44], the monitoring of a mobile robot [45], and intelligent vehicle health management (IVHM) for the X-37 experimental space transportation vehicle.

The structure and operation of Livingstone is illustrated in Figure 10 [41]. As shown, the *Mode Identification* module (MI) estimates the current state of the system by tracking the commands issued to the device. It then compares the predicted state of the device against observations received from the actual sensors. If a discrepancy is noticed, Livingstone performs a diagnosis by searching for the most likely configuration of component states that are consistent with the observations. Using this diagnosis, the *Mode Recovery* module (MR) can compute a path to recover to a given goal configuration.

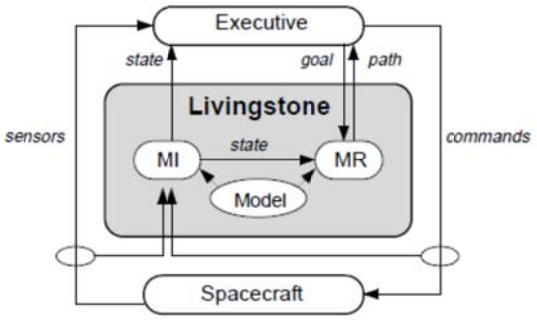

Figure 10. Livingstone mode identification (MI) and mode recovery (MR) [41]

4.3. NASA's OCA Mirroring System

Orbital Communications Adaptor (OCA) Flight Controllers, in NASA's International Space Station Mission Control Center, use different computer systems to *uplink*, *downlink*, *mirror*, *archive* and *deliver files* to and from the International Space Station (ISS) in real time. The OCA Mirroring System (OCAMS) [46] is a multi-agent software system (MAS) that is operational in NASA's Mission Control Center. From a MAS perspective, the OCAMS system is divided into three separate distributed agents running in a separate and special Brahms Hosting Environment [46]. Such an environment can run on any desired computer and network configuration, making the architecture easily adaptable to the computer architecture and network. The communication between the OCAMS components is established by using the so-called NASA Collaborative Infrastructure (CI). The CI is an infrastructure that provides an application programming interface and a set of services that allows components to:

- interact with structured messages (transport service),
- find one another (directory service),
- share data via a publish/subscribe service (data distribution service);
- be managed using a common interface (management service).

The components that make use of the CI are called agents or actors. The CI, in addition, provides process management tools to manage the startup, monitoring, and shutdown of actor hosting environments and their actors.

4.4. Lights out Ground Operations System (LOGOS)

The Lights out Ground Operations System (LOGOS) is a prototype MAS employed at NASA to automate the ground operations for satellites. LOGOS relies on a community of cooperative intelligent agents that perform the ground operations by replacing the human operators and using the traditional ground system software tools, such as orbit generators, schedulers and command sequence planners [47]. In general, LOGOS provides a comprehensive environment that supports the development, deployment, and evaluation of evolving agent-based software concepts in the context of lights-out operations. Moreover, LOGOS supports evaluation of information visualization concepts and cognitive studies. Additionally LOGOS helps researcher realize how agents being "tool users" can be integrated into an agent community responsible for running ground control operations.

5. Summary

This chapter has presented an overview of the *multi-agent systems* (MASs) paradigm. The main concept behind a MAS is a *society of autonomous software agents* acting independently, but interacting to achieve user and system-wide goals. In general, an agent is a software (or hardware) entity that is situated in some environment and is able to autonomously react to changes in that environment. An agent perceives its environment through special *sensors* and acts upon that environment through effectors. The bottom line is that the agents of a MAS *act independently* to achieve the shared system goals. This helps a MAS overcome many of the common disadvantages associated with large-scale distributed computing such as synchronization, single point of failure and difficulty of modifying the system at runtime.

Research in MASs is mainly concerned with the study, behavior and construction of a *society of autonomous agents* that interact with each other and their environment. In this chapter, we have elaborated on the agent concepts and classes of agents by emphasizing the so-called *intelligent agents*, their types and types of agent environments. Moreover, the chapter has covered the most modern trends of MAS research such as *grid computing*, *intelligent swarms*, *sensor networks*, *web services* and *cloud computing*. A distinct section has been dedicated to the *autonomic computing* initiative, which is considered as one of the most prominent approaches to intelligent MASs. As case studies of MASs, this chapter has also presented some of the most significant of NASA's multi-agent applications such as the NASA ANTS space-exploration prospective mission, the Livingston Project, the OCA (Orbital Communications Adaptor) Mirroring System and the Lights out Ground Operations System (LOGOS), which is a prototype system aiming at automation of the ground operations for satellites.

In conclusion, it should be noted that MASs have inspired great research effort, because they provide the IT community with powerful AI-inspired approaches to problems that otherwise are difficult or impossible to solve with a single agent or a monolithic system.

Acknowledgement

This work was supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre.

References

[1] K. Sycara, Multiagent systems, *AI Magazine* **19**, **2** (1998), 79–92.
[2] E. Durfee and V. Lesser, Negotiating task decomposition and allocation using partial global planning, *Distributed Artificial Intelligence*, Volume **2** (1989), 229–244, San Francisco, CA, Morgan Kaufmann.
[3] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.
[4] P. Maes, Artificial life meets entertainment: lifelike autonomous agents, Communications of the ACM, **38**, **11** (1995), 108–114.
[5] L. N. Foner, Entertaining agents: a sociological case study, *Proceedings of the 1st International Conference on Autonomous Agents* (1997), 122–129.
[6] B. Hayes-Roth, An architecture for adaptive intelligent systems, *Artificial Intelligence*, **72**, **1-2** (1995), 329–365.
[7] M. Wooldridge, Intelligent agents, *Multi-Agent Systems* (M. Wooldridge and G. Weiss, Eds.), 3–51, MIT Press, Cambridge, Mass, USA, 1999.
[8] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*, Prentice Hall, 2009.
[9] M.Wooldridge and N. R. Jennings, Intelligent agents: theory and practice, *The Knowledge Engineering Review*, **10**, **2** (1995), 115–152.
[10] D. Gilbert, M. Aparicio, B. Atkinson, S. Brady, J. Ciccarino, B. Grosof, P. O'Connor, D. Osisek, S. Pritko, R. Spagna and L. Wilson, *IBM Intelligent Agent Strategy*, White Paper, IBM Corporation, 1995.
[11] Acronymics Inc., *AgentBuilder: An integrated toolkit for constructing intelligent software agents - Reference manua*l, ver. 1.4, Acronymics Inc, (2004), url (www.agentbuilder.com/ Documentation/ReferenceManual-v1.4.pdf).
[12] A. Colombo, R. Schoop and R. Neubert, An agent-based intelligent control platform for industrial holonic manufacturing systems, *IEEE Transactions on Industrial Electronics*, **53**, **1** (2006), 322–337.
[13] D. Zhang, A. Anosike and M. K. Lim, Dynamically integrated manufacturing systems (DIMS) - a multi-agent approach, *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, **37**, **5**, (2007), 824–850.
[14] E. Davidson, S. McArthur, J. McDonald, T. Cumming and I. Watt, Applying multi-agent system technology in practice: automated management and analysis of SCADA and digital fault recorder data, *IEEE Transactions on Power Systems*, **21**, **2** (2006), 559–567.
[15] S. McArthur, S. Strachan and G. Jahn, The design of a multi-agent transformer condition monitoring system, *IEEE Transactions on Power Systems*, **19**, **4** (2004), 1845–1852.
[16] D. Buse and Q. H. Wu, Mobile agents for remote control of distributed systems, *IEEE Transactions on Industrial Electronics*, **51**, **6** (2004), 1142–1149.
[17] J. Zhou, G. Chen, H. Zhang, W. Yan and Q. Chen, Multiagent based distributed monitoring and control of hazard installations, *Proceedings of the 2nd IEEE Conference on Industrial Electronics and Applications (ICIEA '07)*, 2892–2895, 2007.
[18] J. Galdun, L. Takac, J. Ligus, J. Thirie and J. Sarnovsky, Distributed control systems reliability: consideration of multi-agent behavior, *Proceedings of the 6th International Symposium on Applied Machine Intelligence and Informatics (SAMI '08)*, 157–162, 2008.
[19] K. Fregene, D. Kennedy and D. Wang, Toward a systems- and control-oriented agent framework, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, **35**, **5** (2005), 999–1012.
[20] K. Hwang, S. Tan, M. Hsiao and C. Wu, Cooperative multi-agent congestion control for high-speed networks, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, **35**, **2** (2005), 255–268.
[21] D. Srinivasan and M. Choy, Cooperative multi-agent system for coordinated traffic signal control, *IEE Proceedings: Intelligent Transport Systems*, **153**, **1** (2006), 41–50.
[22] T. Nagata and H. Sasaki, A multi-agent approach to power system restoration, *IEEE Transactions on Power Systems*, **17**, **2** (2002), 457–462.
[23] S. Widergren, J. Roop, R. Guttromson and Z. Huang, Simulating the dynamic coupling of market and physical system operations, *Proceedings of IEEE Power Engineering Society General Meeting*, **1**, 748–753, 2004.

[24] D. Koesrindartoto, J. Sun and L. Tesfatsion, An agent-based computational laboratory for testing the economic reliability of wholesale power market designs, *Proceedings of IEEE Power Engineering Society General Meeting*, **3**, 2818–2823, 2005.
[25] A. Dimeas and N. Hatziargyriou, Operation of a multiagent system for microgrid control, *IEEE Transactions on Power Systems*, **20**, **3** (2005), 1447–1455.
[26] D. Buse, P. Sun, Q. Wu and J. Fitch, Agent-based substation automation, *IEEE Power and Energy Magazine*, **1**, **2** (2003), 50–55.
[27] F. Berman, G. Fox and A. Hey, *Grid Computing - Making the Global Infrastructure a Reality*, Wiley Press, New York, USA, 2003.
[28] D. Culler, D. Estrin and M. Srivastava, Overview of sensor networks, *IEEE Computer* **37**, **8** (2004), 41–49.
[29] K. Gottschalk, S. Graham, H. Kreger and J. Snell, Introduction to web services architecture, *IEEE Computer* **41**, **2** (2002), 170–177.
[30] IBM Corporation, Web Service Level Agreements (WSLA) Project - SLA Compliance Monitoring for e-Business on demand, url (http://www.research.ibm.com/wsla/).
[31] I. Foster, Y. Zhao, I. Raicu and S. Lu, Cloud Computing and Grid Computing: 360-degree compared, *CoRR*, abs/0901.0131 (2009).
[32] IBM Corporation, *An architectural blueprint for autonomic computing*. White paper, 4th ed., IBM Corporation, 2006.
[33] J. O. Kephart and D. M. Chess, The vision of Autonomic Computing. *IEEE Computer* **36**, **1** (2003), 41–50.
[34] D. Ghosh, R. Sharman, H. R. Rao and S. Upadhyaya, *Self-healing systems - survey and synthesis*, *Decision Support Systems* **42**, **4** (2007), 2164–2185.
[35] M. Hinchey and R. Sterritt, 99% (biological) inspiration .... *In Proceedings of the First IFIP International Symposium on Biologically-Inspired Cooperative Computing (BICC 2006)*, Springer (2006), 1–14.
[36] E. Bonabeau and G. Théraulax, Swarm smarts, *Scientific American* (2000), 72–79.
[37] W. Truszkowski, M. Hinchey, J. Rash and C. Rouff, NASA's swarm missions: The challenge of building autonomous software, *IT Professional* **6**, **5** (2004), 47–52.
[38] FIPA - Foundation for Intelligent Physical Agents, url (http://www.fipa.org/).
[39] E. Vassev, *ASSL: Autonomic System Specification Language - A Framework for Specification and Code Generation of Autonomic Systems*. LAP Lambert Academic Publishing, Germany, 2009.
[40] E. Vassev and M. Hinchey, ASSL specification and code generation of self-healing behavior for NASA swarm-based systems, *Proceedings of the 6th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'09)*, IEEE Computer Society, 77–86, 2009.
[41] C. Pecheur, R. Simmons and P. Engrand, Formal verification of autonomy models, *Agent Technology from a Formal Perspective*, Springer-Verlag, 2006.
[42] B. Williams and P. Nayak, A model-based approach to reactive self-configuring systems, *Proceedings of AAAI/IAAI*, **2** (1996), 971-978.
[43] N. Muscettola, P. Nayak, B. Pell and B. Williams, Remote agent: to boldly go where no AI system has gone before, *Artificial Intelligence* **103**, **1-2** (1998), 5-48.
[44] D. Clancy, W. Larson, C. Pecheur, P. Engrand and C. Goodrich, Autonomous control of an in-situ propellant production plant, *Proceedings of the Technology 2009 Conference*, Miami, USA, 1999.
[45] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan and M. Velosso, Experience with a layered robot architecture, *ACM SIGART Bulletin*, **8** (1997), 1-4.
[46] M. Sierhuis, W. Clancey, R. von Hoof, C. Seah, M. Scott, R. Nado, S. Blumenberg, M. Shafto, B. Anderson, A. Bruins, C. Buckley, T. Diegelman, T. Hall, D. Hood, F. Reynolds, J. Toschlog and T. Tucker, NASA's OCA Mirroring System: An application of multiagent systems in Mission Control, *Proceedings of Autonomous Agents and Multi Agent Conference (Industry Track)*, Budapest, Hungary, 2009.
[47] C. Rouff, J. Rash, M. Hinchey and W. Truszkowski, Formal methods at NASA Goddard Space Flight Center, *Agent Technology from a Formal Perspective*, Springer-Verlag, 2006.