# The Third International Workshop on Massively Multi-Agent Systems: Models, Methods and Tools (MMAS'09)

The Third International Workshop on Massively Multi-Agent Systems: Models, Methods and Tools (MMAS'09).

URL $http://indus.usask.ca/MMAS2009/$

New information systems and recent applications (ubiquitous computing, networking, transport...) are often distributed, large scale, open, heterogeneous and characterized by a dynamic environment. Millions of software entities or electronic devices with computing facilities are connected with each other, and are required to behave coherently often in pursuit of extremely complex and distributed goals in dynamic environments. Challenges in modelling, implementing, deploying and controlling these systems are significant.

Massively multi-agent systems provide a suitable design paradigm and an implementation method for these systems. As infrastructure of massively multi-agent systems, technologies such as grid computing together with semantic annotation can be combined with agent technologies. A new system design approach – society-centred design – may be realized by embedding participatory technologies in human society. Applications include large-scale navigation, scientific or social simulations, e-science, e-homes, e-offices and e-cities.

MMAS 2009 will build on the success of its predecessors: CCMMS 2007, held in conjunction with AAMAS 2007 in Honolulu, Hawaii; MMAS 2006, held jointly with LSMAS 2006, in conjunction with AAMAS 2006; and MMAS 2004,which was held at Kyoto Research Park, Kyoto, Japan, December 10 - 11, 2004. The aim of CCMMS 2008 is to encourage existing activity in the field; to bring together computer science, information science and social science experts concerned with coordination and Control in MMAS, and applications of MMAS; and to share their perceptions and explore future research challenges.

## Committees

Program Co-Chairs:

- Zahia Guessoum, LIP6, University of Paris 6, France
- Nadeem Jamali, University of Saskatchewan, Canada
- Toshiharu Sugawara, Waseda University, Japan

Program Committee members:

- Myriam Abramson, Naval Research Laboratory, USA
- Gul Agha, University of Illinois at Urbana-Champaign, USA
- Dan Corkill, University of Massachusetts, USA
- Raj Dasgupta, University of Nebraska, USA
- Keith Decker, University of Delaware, USA
- Alexis Drogoul, IRD, France
- Nora Faci, LIRIS, University of Lyon, France
- Satoru Fujita, Hosei University, Japan
- Hiromitsu Hattori, Kyoto University, Japan
- Toru Ishida, Department of Social Informatics, Kyoto University, Japan
- Nadia Kabachi, LIRIS, University of Lyon, France
- WooYoung Kim, Intel, USA
- Yasuhiko Kitamura, Kwansei University, Japan
- Satoshi Kurihara, Osaka University, Japan
- Victor R. Lesser, University of Massachusetts, USA
- Jiming Liu, Hong Kong Baptist University, China
- Roger Mailler, University of Tulsa, USA
- René Mandiau, Université de Valenciennes et du Hainaut Cambrésis, France
- Ryusuke Masuoka, Fujitsu Laboratories of America, Inc., USA
- Michael J. North, Argonne National Laboratory, USA

- Hideyuki Nakashima, Future University, Japan
- Akihiko Ohsuga, University of Electro-Communications, TOSHIBA Corporation, Japan
- Charlie Ortiz, Artificial Intelligence Center, USA
- Ei-ichi Osawa, Future University, Japan
- Ichiro Satoh, National Institute of Informatics, Japan
- Paul Scerri, Robotics Institute, Carnegie Mellon University, USA
- Olivier Simonin, Université Henri Poincaré, France
- Carlos Varela, Rensselaer Polytechnic Institute, USA
- Walt Truszkowski, NASA Goddard Space Flight Center, USA
- Gaku Yamamoto, IBM Software Group, Japan
- Jung-Jin Yang, The Catholic University of Korea, Korea
- Franco Zambonelli, Universita' di Modena e Reggio Emilia, Italy

Contents

1 **Engineering Strategies for Market-based Scheduling**
Nikolay Borissov, University of Karlsruhe

19 **A Nature-inspired Approach for Large-Scale Pervasive Service Ecosystems**
Cynthia Villalba and Franco Zambonelli,University of Modena and ReggioEmilia

35 **Dross into Diamonds: Efficient Use of Untrustworthy Agents in a Large Scale Environment**
Chris L. D. Jones and K. Suzanne Barber, The University of Texas at Austin

51 **Towards a Transparent Middleware for SelfOrganizing MultiAgent Systems on Clusters**
Paulo Motta, Maíra Athanázio de Cerqueira Gatti and Carlos José Pereira de Lucena, PUC Rio

63 **Resource Management and Adaptive Replication for Fault-Tolerant MAS**
Sylvain Ductor, Zahia Guessoum and Mikal Ziane, Université Pierre et Marie Curie

# Engineering Strategies for Market-based Scheduling

Nikolay Borissov

University of Karlsruhe, Information Management and Systems, Englerstr. 14, 76131 Karlsruhe, {borissov}@iism.uni-karlsruhe.de

**Abstract.** The application of market mechanisms for the allocation of Grid-based services is a demanding task, which requires bridging economic and associated technical challenges. Even if the market-based approach promises an efficient allocation of computing services, the wide heterogeneity of consumer requirements and the diversity of computational services on the provider side are challenging the processes of finding, allocating, and using an appropriate service in an autonomous way. In this paper we firstly present a scenario and derive requirements to automate the bidding processes. Secondly, we evaluate existing agent frameworks, market mechanisms and bidding strategies, which can be adopted to achieve an market-based provisioning and usage of Grid-based services. Thirdly we propose a methodology for engineering bidding strategies. Fourthly, based on the decision processes of the methodology, we present a framework for implementing budding agents. Finally, we present a technical analysis of the implemented framework and the methodology and conclude the paper.

**Keywords:** Bidding Processes, Engineering Bidding Strategies, Bidding Framework

## 1 Introduction

Currently we are observing a rising number of Grid and Cloud-based service offerings on demand. Prominent service providers like Amazon, Google, Sun, IBM, Oracle and Salesforce are offering their computing infrastructures, providing top-level services for virtualising computing instances and storage, called also Infrastructure-as-a-Service as well as for offering databases and deployment technologies for web applications, called also Platform-as-a-Service. The companies frequently offer pay-per-use or subscription pricing models for static service configurations of computing instances. The lack of common interfaces and standards for the provisioning and usage of computing services hamper the acceptance of the promising technologies in the sense that a consumer has to select a particular provider and adapt its application for the computing services and interfaces of the selected provider. In the case of changing conditions, i.e. changing consumer requirements, failures of provider services or fluctuating demand and supply,

the migration of a consumer application from one provider service to another is currently not flexible and associated with high efforts, e.g. the redesign of the application to use the services and interfaces of the other provider. Furthermore static pricing models of service configurations can lead to inefficient utilization, reward and usability, as they does not reflect the dynamics of the market supply and demand.

Efficient provisioning and usage of Grid-based services as well as pricing in environments like Grid is not manually manageable. Such processes should be automated with no or minimal human interaction. On one side, based on predefined business polices for resource utilization, consumer classification and requested QoS, provider agents will automatically sell and bye Grid-based services on demand. On the other side, consumer application agents, will automatically scale by allocating additional computing instances on demand. Hence, standardization of interfaces and methodologies, the elaboration of suitable market mechanism and the analysis of strategic behavior play an important role for the design of the environment.

The first contribution of this paper is to survey and bring together results for existing agent frameworks and mechanisms for market-based allocation of computing services. The second contribution is to present a methodology and framework for designing and implementing bidding strategies. The third contribution is to perform a technical analysis of the introduced framework and methodology as well as to discus further evaluation steps.

Following the structure of the paper, section 2 presents a scenario for market-based scheduling. Based on the scenario, in section 3 we derive requirements for automated bidding. Section 4 discusses existing agent frameworks, market mechanisms and bidding strategies. Section 5 presents a methodology for engineering bidding strategies. Thus, section 6 introduces a framework and its components for implementing budding agents. Technical analysis of the implemented framework and methodology are depicted in section 7. The paper concludes with section 8.

## 2 Scenario for Market-based Scheduling of Computing Resources

In general, scheduling policies can be distinguish as "global scheduling policies", where consumer jobs are allocated to providers (or provider site e.g. computing center) and "local scheduling policies", where on provider side, the allocated jobs are scheduled to a target machine.

Grid middleware like *gLite* perform first global scheduling of jobs to provider sites with a so called *Resource Broker* (or *MetaScheduler*). The global scheduling is a based on the technical description of the offered resources on each provider site and the technical requirements of the jobs. When scheduled on the provider site, the job execution is managed by the provider's local site scheduler. An overview of various mechanisms for technical scheduling in Grids is presented by [1]. Grid middleware and target scheduling mechanisms are designed to bundle

worldwide computing resources in order to execute complex jobs (applications) more efficiently as well as to utilize available computing power more efficiently. This efficiency led that researcher worldwide can utilize idle resources on demand without buying new hardware and spending efforts and money for maintaining it.

Businesses like e.g. Amazon, Google, IBM and Salesforce recognized this trend and opportunities to commercialize the Grid-technologies and redefined the Grid paradigm to be useful for business consumers in so called Cloud Computing [2]. To map the aims of Grid-based resource provisioning and usage into Cloud needs to move from technical meta scheduling into market-based scheduling, where a technical description and price will do the allocation of jobs to provider machines.

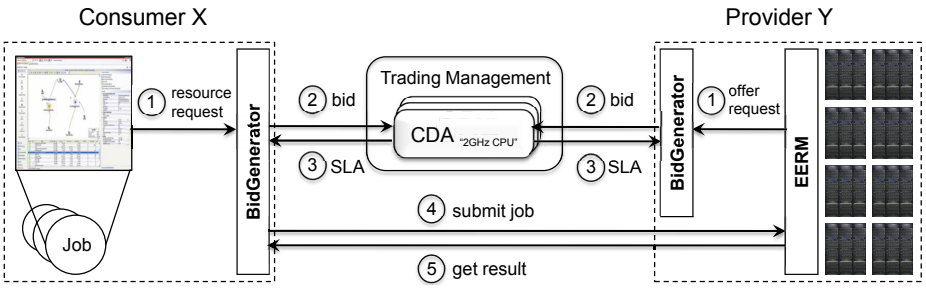


**Fig. 1.** Scenario for Market-based Scheduling

In a market-based scenario, the technical meta scheduler is replaced by the Trading Manager which executes the market mechanism(s), where consumer bids are allocated to provider offers. The basic scenario for market-based scheduling of Cloud services is illustrated in figure 1). Here the market mechanisms is a *Continuous Double Auction* for selling computing instances, which run on 2GHz-CPUs. As illustrated, on the one side a consumer wants to execute a job (application), which requires computing resource with specified technical and economic preferences. Example jobs can be such for complex video data analysis and aggregation of financial data. On the other side, a provider, such as Network.com and Amazon EC2, has computing resources, which he wants to offer on the market. The negotiation i.e. the generation of bids and offers is enforced by the so called BidGenerator component (see section 6). The BidGenerator received the consumer and provider technical (technical resource description e.g. JSDL, see section 6.2) and economic preferences (consumer valuation or provider reserve price as well as the preferred bidding strategy) in a form of a predefined message format (see section 6.2) and based on these initiates the bidding agents, which automatically execute the consumer or provider bidding processes (see section 5). When the market clears with a match, both the consumer and provider bidding agent are informed and the job is submitted for execution on to the target provider's

EERM (Economically Enhanced Resource Manager) [3], which manages the local provider resources and the execution of the allocated job. How to handle the result of a job (application) execution is specified by the consumer in the JSDL part of the message protocol.

## 3 Requirements for Automated Bidding

The scenario in figure 1 depicts the basic interactions between the involved components. In this paper we will look at the BidGenerator, which implements the strategic behavior of consumers and providers and less the design and implementation of the Trading Management [4] component. The aim of the Trading Management component is to provide a platform for implementing various market protocols, where consumer and provider agents can interact with each other by submitting bids and offers and exchanging market information.

The aim of the BidGenerator is to offer a platform and methodology for implementing strategic behavior i.e. bidding strategies. Through its simple design and clear interfaces it has to reduce the complexity in a way that participants are empowered to make use of the Cloud without having to worry too much about how it works. In essence, the BidGenerator framework must support the automation of the bidding process, taking into account the current demand and supply situation. In order to achieve this facility, its design and methodology has to meet following requirements:

- *R1 – communication facility:* In order to communicate with the environment, the BidGenerator has to adopt a common and well-defined communication protocol ([5]).
- *R2 – market information:* A BidGenerator has to be able to collect and interpret market information like available auctions, auction rules, transaction cost, current and past (bid) prices [6].
- *R3 – strategic behavior:* The BidGenerator has to offer a methodology for implementing bidding strategies. The bidding strategies implement strategic decisions, based on information they have about the environment, which may be incomplete or incorrect. Strategic decisions incorporate the goals to be achieved (maximize profit, minimize completion time), actions (generate bid) to be performed and evaluate the effects (payoffs) of the actions.
- *R4 – automated bidding* Bidding strategies has to support and automate the bidding processes i.e also adapt to market dynamics ([7]) like price fluctuations, changing conditions and QoS of provided resources. Learning mechanisms can be adopted to aggregate information of past decisions, actions and available market information in order to "tune" the bid generation processes.
- *R5 – simple and adoptable* The BidGenerator has to be designed to be simple, extensible and reusable by providing clear and well-defined interfaces, using open tools, technologies and standards (W3C, FIPA).

In this section we derived requirements for the BidGenerator, supporting providers and consumers by the market-based provisioning and usage of Cloud-

based services. Next section discusses state of the art agent frameworks, market mechanisms and bidding strategies.

## 4 Related Work

### 4.1 Agent Frameworks

The research of AI and autonomous system applications has produced many agent frameworks. Some of them are designed to be more general other to conduce specific scenarios. Bodies like FIPA [5] and MASIF [8] make efforts to standardize agent frameworks, methodologies, interfaces and communication protocols in order to reduce complexity by the creation of multi agent systems.

*FIPA-OS*[1] and *JADE*[2] are reference implementations of *FIPA*. For agents running on mobile devices, the LEAP ([9]), *SHUFFLE* ([10]) and *CRUMPET* ([11]) projects offer agent frameworks based on a light-weight version of the *FIPA* specification and its communication protocol.

Agent frameworks like *JADE*, *Grasshopper*, *Cougaar* and *Aglets* ([12]) offer interfaces for implementing agents and interaction protocols that can be reused in different scenarios. Such frameworks will easy the modeling of the basic interactions on the agent side, but they do not support the design and implementation of bidding strategies. Furthermore, they do not offer methodologies and interfaces for implementing market mechanisms.

*JACK agents* ([13]) offer interfaces, methodologies and communication protocols for implementing generic agents. The JACK framework has been used in applications for decision support and modeling human behavior. The *MAGMA* ([14]) framework describes an overall infrastructure for building agent-based virtual marketplaces. It introduces a framework for implementing trading agents and provides interaction protocols for negotiations between agents as well as a general protocol for interactions in auctions.

Projects like *AuctionBot* ([15]), *e-Game* ([16]) and *JASA*[3] – the current agent framework of the *Trading Agent Competition* – offer frameworks for implementing auction mechanisms. *AuctionBot* and *e-Game* specify how to describe markets and specify rules on them. Although it do not offers methodology for designing bidding strategies, such rules and market description can be considered by the development of bidding strategies. *JASA* offers a methodology and interfaces for implementing market mechanisms and bidding strategies. The main design principle of these frameworks is to evaluate market mechanisms and agent strategies in a tournament environment in which worldwide agents can connect, interact and influence the tournament by submitting bids for resource allocation.

[1] `http://sourceforge.net/projects/fipa-os/`
[2] `http://jade.tilab.com`
[3] `http://jasa.sourceforge.net`

### 4.2 Market-based Scheduling

Economic models for resource scheduling are widely explored in the literature [17, 18]. According to the coordination modes, scheduling mechanisms can be categorized into "centralized mechanisms", where the allocation decision of bids and offers is taken by a central unit, and "decentralized mechanisms", where the allocation decision is decentralized, i.e. taken by the requesters based on all of the responses they receive from the environment. According to the allocation modes, scheduling mechanisms can be divided into mechanisms that execute periodically, called also "off-line mechanisms", and mechanisms that execute continuously, called also "on-line mechanisms". Examples of off-line and centralized mechanisms are the *Call Double Auction* [19] and combinatorial mechanisms like those proposed in [20] and [21]. Examples of centralized on-line mechanisms are the *Continuous Double Auction* and *Tycoon* [22]. Decentralized market mechanisms for machine scheduling have been explored to a lesser extent in the current literature. A prominent example of on-line machine scheduling is the *Decentralized Local Greedy Mechanism* [23].

### 4.3 Defining Bidding Strategies

Autonomous bidding agents has been developed and elaborated in many fields. Agents are elaborated in stock markets [24] and supply chain management [25]. [26] give an overview of the various agents and their strategies taken place in the trading agent competition (TAC). In general, bidding strategies can be classified into *non-adaptive*, where the generated bid price do not depend on past market information like Truth-Telling and Zero-Intelligence (ZI) as well as *adaptive* strategies, where the generated bid price depends on current and past bids like Zero-Intelligence-Plus (ZIP), Gjerstad-Dickhaut (GD) [27] and Adaptive-Aggressiveness (AA) [7] strategies.

Current research focuses more on intelligent agents, which can adapt and learn from the environment. [28] propose co-evolution algorithms to learn the strategy space for autonomous bidding by the allocation of resources in market mechanisms. The authors of AA strategy [7] propose a bidding strategy, which implements short and long-term learning, considering and responding also to dynamic market fluctuations. In [29], we prosed a novel bidding strategy for Clouds, Q-Strategy, which explores the environment by offering and utilizing resources with varying prices and requirements and use the learned information to maximize the consumers or providers utility in different market mechanisms.

Bidding strategies incorporate decisions processes which can get very complex, when utilized in dynamic environments like Clouds and adapting many parameters like technical and economic preferences. Thus methodologies for designing and implementing bidding strategies become important. Methodologies like in [30–32] discuss general processes for designing agent systems and bidding strategies [32]. Methodologies for including learning techniques are investigated by[33].

In general and in the field of Cloud-computing there is a lack of methodologies for designing bidding strategies. In following sections we introduce a methodology for developing bidding strategies for Cloud-based service provisioning and usage.

## 5 Methodology for Engineering Bidding Strategies

In open markets, consumers and providers will always handle to maximize their own utilities. The design of market mechanisms and bidding strategies aims to provide incentives for both, consumers and providers, to use the mechanisms as well as to reduce the space of strategic behavior in order to make the mechanisms more predictive and tractable. However, [34] showed that there is no double-side auction mechanism, which is incentive compatibility, budget-balance, individual rationality and allocation efficient.

The design of bidding strategies for providing and purchasing Cloud-based services can be a very complex tasks, which depends on many decision factors. The decision factors incorporate the available information from the environment like consumer and provider preferences, available market mechanisms and their behavior and efficiency, available bidding strategies and their behavior and efficiency as well as available market information about current and past prices. Such information can be outdated, incorrect or incomplete. Thus consumers and providers has to deal with this uncertainty. Common methodologies for handling uncertainty are the *Evidence Theory*, *Probability Theory* and *Fuzzy theory* [30, 35]

Based on the discussed scenario in section 2, the evaluation of existing market mechanisms, bidding strategies and methodologies, we developed a case-based decision model for designing bidding strategies for market-based scheduling (Figure 2). Figure 2 illustrates the decision steps of consumers and providers when purchasing or offering a computational service.

The decision problem of a bidding agent consists of the generation of a bid, based on a set of decision factors like consumer or provider preferences, selected market mechanism and selected bidding strategy. Thus the decision model of consumers and providers goes trough following steps.

**Consumer Step 1 – Preference Elicitation**

We assume that consumers of Cloud-based services e.g. researcher and application providers, will have well-known or estimated [36] minimum technical requirements of the required computing resources. A common problem in economics is the estimation of the maximum willingness to pay for a given resource or more specifically, what is the value of a given job or application. The answer of this question will require a analysis of the current market situation like current and past prices, what types of resource configurations are offered on the market and what QoS and resources are required in order to meet the consumer goals. The result of this analysis are the specification of the economic preferences e.g. consumer's valuation for the job, required duration, desired start and finish times.

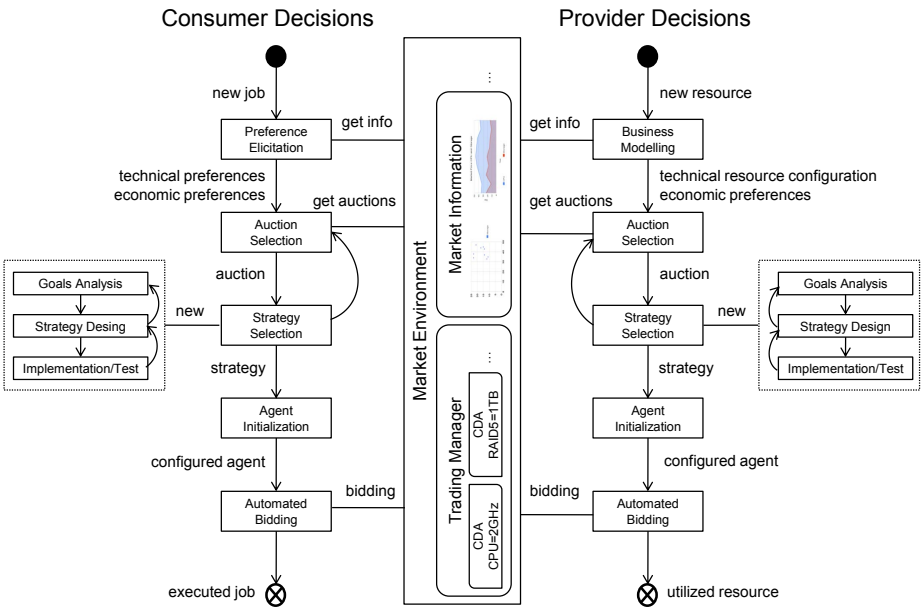


**Fig. 2.** Consumer and Provider Decision Processes

**Provider Step 1 – Business Modeling**

The opposite step of consumer's preference elicitation is provider's business modeling. We assume that providers of Cloud-based services will be computing centers or businesses, which maintain computing infrastructures with surplus capacities of computing resources for the most of their time. In order to be profitable and utilize their resources, providers has to be aware of the demand and supply of Cloud-based services in order to remain competitive in such markets. The decisions they have to met are regarding what resource configuration to offer, for what conditions and price. To answer this questions, they need to analyze the supply and demand of Cloud-based services, the evolution of prices and future resource trends. Moreover, provider need to define the rules for offering their free capacities by analyzing their interned usage and needs, average utilization in the time, appropriate scheduling policies and special conditions for business clients. The result of this analysis are technical description of the offered computing resources, economic preferences and the rules (policies) for offering them like selected scheduling policies for internal and external jobs, time frames, business policies for specific clients etc.

**Step 2: Auction Selection**

Knowing the technical and economic preferences and business models, providers and consumers have to make a decision and select a particular market mechanism to place their offers or bids. To do this, consumers and providers will request market information about what mechanisms do exist on the market and specific information about their behavior, bidding rules and prices [37]. The efficiency

value of a specific market mechanisms can be reported within the market information, but can be different from the consumers or providers point of view and relative to the applied metric. Thus, provider and consumers may select and evolutionary evaluate different market mechanisms.

**Step 3: Strategy Selection**

The selection of a bidding strategy can be tightly connected to the selection of the market mechanism e.g. a specific bidding strategy can perform well and a given market mechanism, but perform worse in other markets. Like in previous step, the selection of bidding strategies to market mechanisms can be done through an evolutionary process. In this context, consumers and providers has to explore the space of available bidding strategies and their efficiency in available market mechanisms. The efficiency of a strategy can be reported in the market information or be an individual metric, which is measured during the exploration precess of various strategies in the available market mechanisms. If the space of available strategies do not suits the private requirements, consumers and providers will design, implement and test their own strategies. To do this, they have to investigate and specify their goals into scoring and pricing policies, decide which and how to use available market information and put intelligence by an appropriate selection of prediction and learning techniques.

**Step 4: Agent Initialization**

When the above decisions are determined, the bidding agents can be initialized in order to start the bidding process. The agents are configured with a instance of the selected bidding strategy and the endpoint reference of the selected market mechanism. The technical and economic description of the offered or requested resources are described in a well-defined message protocol and are taken as input by the agent, which initiates the start of the bidding process.

**Step 5: Automated Bidding**

The final process of the consumer and provider decision-making processes is *Automated Bidding*. This process incorporates the generation of the bid, which contains the technical and economic preferences of the provider and consumers, the submission of the bid to the selected market mechanism and the handling of the market message which can be the successful allocation of provider offer to consumer bid or not allocation. Each bid and offer is assigned a "Time To Live" property, which is a part of the economic preferences. This property indicates the dwell time of a bid or offer in the market's order book. When the agent receives an allocation from the market, the consumer job is submitted to the target provider and the bidding process terminates. The absence of an allocation can be caused by many reasons e.g. missing demand or supply, higher provider prices, lower consumer prices etc.

The above mentioned decision processes can be executed manually or be automated by appropriate bidding frameworks as well as market platforms which execute the market mechanisms and provide market information about the mechanisms, bidding rules, prices and their efficiency [37]. In following section we will present the architecture and components of a framework, which adopts the presented methodology.

## 6 Framework for Automated Bidding

This section presents the architecture and components of the BidGenerator, implemented within the SORMA project (www.sorma-project.eu).

### 6.1 Architecture of the Bidding Framework

The main purpose of the BidGenerator is to generate bids, based on the consumer or provider preferences and support the bidding process, i.e. submit bids to the target market mechanism, handle market messages and report the matches back to the end-user (consumer or provider). Furthermore, the BidGenerator is designed to provide interfaces and a methodology for implementing bidding agents and strategies in order to automate the bidding processes and the interaction with the market. Currently, as proof-of-concept, it offers a pool of implemented state-of-the art and novel bidding strategies (section 4.3). The selection of an appropriate market mechanism and a preferred bidding strategy, both part of the preferences, can be done manually by the consumer and provider or be automated by the bidding agent. The automatic selection is based on past experience with available market mechanisms and applied bidding strategies from BidGenerator's strategy pool.

The proposed *BidGenerator Framework* in Figure 3 is composed of several light-weight APIs, which are structured in the packages *agent*, *strategy*, *learner*, *policy*, *bid* and *MarketExchangeConnection*.

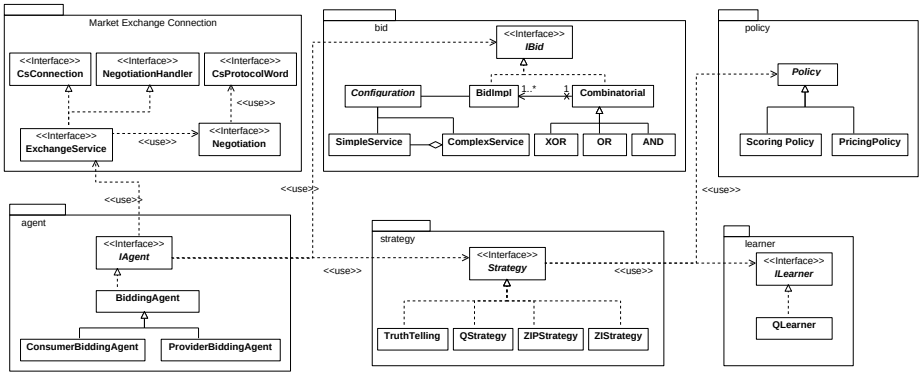


**Fig. 3.** Class Diagram of BidGenerator's Concepts

Package *agent* provides a light-weight *Agent API* with interfaces for implementing agents that apply the implemented bidding strategies, communicate with the market and react to market messages.

The *strategy* package provides interfaces for implementing bidding strategies. Bidding strategies can be composed by well-defined scoring (e.g. utility function)

or pricing policies and utilize a particular learning mechanism. *Scoring* (see example algorithm 1) and *pricing* are externalized into a rule engine in order to enable their configuration and modification outside of the binary code. In our framework we adopted *Jess* ([38]) as a prominent, widely applied, open to academia and Java-based rule engine.

**Algorithm 1** Scoring policy $p_s = w_j * C_{i,j} + \pi_{i,j}$ [23] in *Jess*

```
(deffunction DLGM(?weight ?completionTime ?payment)
(return (+ (* ?weight ?completionTime) ?payment))
```

**Definition 1.** Policy: *The* policy *concept defines atomic decision-making steps like a scoring or pricing function, supporting a given part of the bid generation process.*

**Definition 2.** Pricing Policy: *The* pricing policy *represents the set of price building functions for a given configuration and strategy state (exploration, exploitation). Pricing policies of an implemented bidding strategy may differ in time – day, evening, weekend, etc. – and price interval. An example of a pricing policy is the generation of a price in a given interval $b \in [s * v_j, v_j]$ where $v$ is the valuation and $s \in [0, 1]$ is a scalar.*

**Definition 3.** Scoring Policy: *A* scoring policy *is a utility function defined by the consumer or provider that is to be maximized during the bidding process. Examples of scoring policies are the minimization of the weighted completion time, $max(-w_j C_{i,j})$, or maximization of the profit i.e. $max(v_j - \pi_j)$.*

Preferences for a given application are often subject to change as technology evolves (e.g. a consumer's desired quality of streamed video might increase as her Internet bandwidth increases). It is thus desirable to dynamically adapt the service preferences to the changing market conditions and to automate the resource provisioning and purchasing processes. Thus learning mechanisms are adopted to predict requirements of applications with similar characteristics.

**Definition 4.** Learner: *The* learner *concept is the learning mechanism used to aggregate the past data of each decision taken, exploited to "fine tune" the bid generation process.*

A prominent example of a learning algorithm is *Q-Learning*, which is widely adopted in the praxis. Its explorative concept "learning through repeated play" is suitable for on-line decision making with no or incomplete market information.

The package *MarketExchangeConnection* consist of the interfaces and classes that constitute the connection to the message bus in order to exchange messages with *Trading Management*.

Finally, the package *bid* handles the consumer and provider message types section 6.2 and bid types, which can be simple or combinatorial. The message

types for communicating consumer and provider preferences are presented in the following section.

### 6.2 Message Protocol for Bids

In order to communicate with each other and automate the processes of bidding, job submission and execution, consumer and provider agents in a Cloud environment have to adopt common message protocols in order to express their economic and technical preferences.

Existing message protocols for technical resource description are the *Job Submission Description Language* (JSDL) [39], *GLUE*-Schema [40], *Common Information Model* (CIM) [41] and *WS-Agreement* [42]. *JSDL* and *GLUE*-Schema are standard resource description languages currently adopt in the most Grid middleware for describing provider resources and consumer technical preferences. *CIM* is a general description model aiming to describe computing systems and their services in more detail. *WS-Agreement* is a message protocol for establishing service level agreements between two parties, designed and applied mainly in SLA-Negotiation mechanisms than for auctions.

Thus there is a lack of existing message protocols for auctions or existing one does not consider standardized message protocols, in following we propose schema extensions for bids, on top of XML-based message protocols like *JSDL* and *WS-Agreement*.

Table 1 shows the economic extensions, which are structured in three message formats. The EJSDLPrivate message format contains private information like valuation and preferred bidding strategy, which is submitted between the end-user and BidGenerator (see also figure 1, step 1). The data format for the generated bids is EJSDLBid, which are submitted to the selected market mechanism (figure 1, step 2). When the market clears with a match, the created message format is in a form of EJSDLMarket, which is submitted back to the end-users over BidGenerator (figure 1, step 3).

## 7 Technical Analysis

The presented BidGenerator[4] framework in section 6 is implemented as a java library with an additional Web service interface for submitting bid requests in form of EJSDLPrivate (section 6.2) messages. Using the components of the framework we implemented state-of-the-art bidding strategy and a novel bidding strategy, Q-Strategy [29], which is introduced in our previous works. The Q-Strategy adopts the presented methodology in section 5.

Going through the delivered requirements in section 3, *R1* complies with the presented message protocol (section 6.2) of the framework. The integration interface with Trading Management, enables the exchange of market information (*R2*) like bids, matches, specific market data about available auction

[4] `http://www.rz.uni-karlsruhe.de/~Nikolay.Borissov/sorma/BidGenerator/docs`

**Table 1.** Message Protocols for Bids

| Message Type | Element | Description |
|---|---|---|
| EJSDLPrivate | *valuation* | consumer valuation or provider reserve price |
| | *strategy* | preferred consumer or provider bidding strategy |
| | participant | unique identifier of the consumer or provider |
| | timeToLive | the unit of time a bid is valid |
| | duration | upper bound job/resource duration |
| | serviceType | service type i.e. batch job or web application |
| | paymentType | payment type i.e. *before* or *after* execution |
| | currency | currency of the monetary values |
| | technicalDescription | technical description of the service e.g. in JSDL |
| EJSDLBid | type | the type of bid message, "bid" or "offer" |
| | participant | unique identifier of the consumer or provider |
| | bidPrice | the generated bid price |
| | timeToLive | the unit of time a bid is valid |
| | duration | upper bound job/resource duration |
| | serviceType | service type i.e. batch job or web application |
| | paymentType | payment type i.e. *before* or *after* execution |
| | currency | currency of the monetary values |
| | technicalDescription | technical description of the service e.g. in JSDL |
| EJSDLMarket | clearingPrice | calculated price on market clearing |
| | bidId | the id of the bid |
| | offerId | the id of the offer |
| | consumerId | the id of the consumer |
| | providerId | the id of the provider |
| | duration | fixed upper bound duration |
| | serviceType | fixed service type |
| | paymentType | fixed payment type |
| | currency | currency of the monetary values |
| | technicalDescription | fixed technical agreement JSDL/WS-Agreement |

mechanisms. The definition and implementation of strategic behavior (*R3*) is supported by the introduced methodology as well as the interfaces of the BidGenerator framework. Based on the implemented bidding strategies, interface with the Trading Management and interfaces for implementing bidding agents, the BidGenerator framework supports the automation of the bidding processes (*R4*). Moreover, the clear design of the interfaces and example implementations offer a good point of entry to reuse and adoptable the framework in further scenarios and applications *R5*.

Further analysis directions are the implementation of provider business policies into novel bidding strategies using the presented methodology. This will include the evaluation of existing utility functions and scheduling policies for providers. The scheduling policies are part of the local machine scheduling facility, which is outside the market allocation processes. Thus Q-Strategy [29] allows the configuration of utility functions by considering provider preferences, we will adopt it to evaluate provider utility functions for defined use cases.

One issue which arises when applying parametrized techniques is how to select the parameter values. This is also the case when applying Q-Learning. To find a proper calibration of Q-Strategy [29] we run several simulation settings by very the $\epsilon$ and $\alpha$ parameters. $\epsilon \in [0, 1]$ specifies the probability that the Q-Strategy is in *exploration* or *exploitation* phase. The learning rate $\alpha \in [0, 1]$ determines the weight of newly observed rewards against past aggregated rewards. Higher values of $\alpha$ increases the learning rate for newer observations, lower value decreases it. The parameter values are set to vary for $\epsilon \in \{0.3, 0.7\}$ and $\alpha \in \{0, 0.3, 0.7, 1\}$ respectively. The cases of $\epsilon = 0, 1$ are left, because in that cases the Q-Strategy is either in exploration phase – generating only random bids of the interval $b \in [0.1 * v, v]$ – or returning always the true valuation $v$ of a job. The resulting 8 settings are evaluated with two utility functions, $max(u_{j,i})$ with $u_{j,1} = v_j - b_j$ and $u_{j,2} = -(v_j - b_j)$. For each setting we run a workload of 1000 similar jobs with equal valuations $v = 100$, resulting in 1000 generated bids per setting.

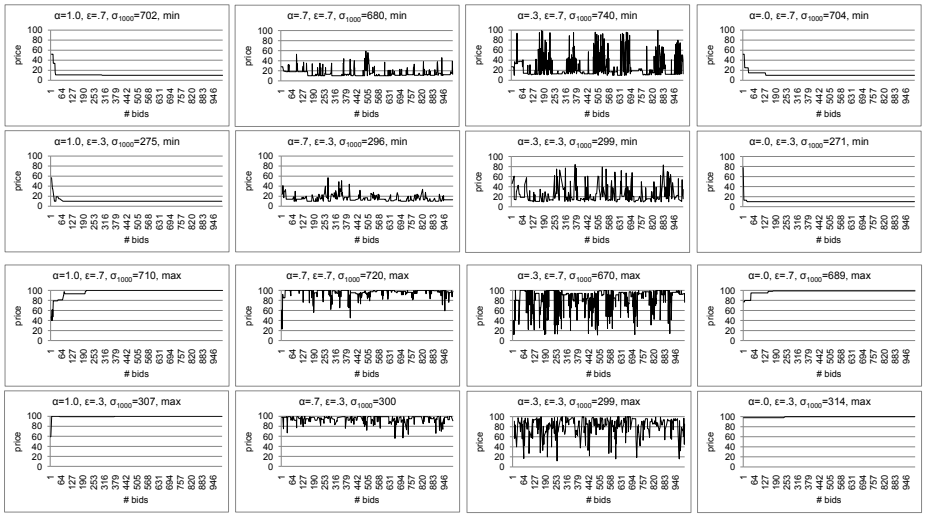


**Fig. 4.** Evaluation of the Q-Strategy $\epsilon$ and $\alpha$ parameters

Figures in 4 shows the results of the 8 different settings for both utility functions, where $\sigma$ is the number of generated bids in the exploitation phase. In a static environment with no changing conditions and rewards we observed that with $\alpha = 1$ the bids are converging quickly to the target values $u_{j,1} \to 10$ and $u_{j,2} \to 100$. Thus $v_j = 100$ is an initial value, it is founded immediately by $u_{j,2}$, second part. By settings of $\alpha = \{0.7, 0.3\}$ we observe higher exploration speed of the environment by $\epsilon = \{0.7\}$. For $\alpha = 0$ no learning is taking place, thus the payoffs are not aggregated with the time.

This observations will flow into the overall simulation scenario, to evaluate agents adopting the Q-Strategy with fixed against varying $\alpha$ and $\epsilon$ parameters of the cases between $\{\alpha = 0.3, \epsilon = 0.7\}$ and $\{\alpha = 0.7, \epsilon = 0.3\}$, which offer a good trade-off between exploration and exploitation phase. This trade-off is especially required in dynamic environments, where consumer requirements and provider offers may fluctuate in the time.

## 8 Conclusion

In this paper, we evaluated existing agent frameworks for their applicability for implementing bidding strategies and autonomous bidding agents. In the related work part we discussed existing market mechanisms and bidding strategies founding out that there is a lack of methodologies for engineering bidding strategies. In this context, we presented a novel methodology for definition and implementation of bidding strategies. Based on the presented scenario for automated provisioning and usage of Cloud-based services, the derived requirements of such a bidding framework and the analysis of existing agent frameworks, we found out that existing frameworks do not satisfy the requirements or are designed for evaluation purposes than for real deployment. According to the proposed methodology, derived requirements and inspiration from existing frameworks, we present a novel framework for implementing bidding agents and bidding strategies. Moreover, we specified message protocols for communicating consumer and provider economic preferences, which are applied on top of standardized protocols.

To complete the evaluation process, we developed a prototype of the proposed framework and performed a technical analysis of its design and proof-of-concept implementation. With the simulation results we evaluated the implementation of the Q-Strategy, analyzing the dynamics of its parameters, needed for running the identified evaluation scenarios. The discussion pointed of further evaluation scenarios and development steps, which need to be performed in order to automated the processes of preference elicitation, evaluation of suitable market mechanisms and learning optimal policies.

The future focus is on the evaluation of the market mechanisms and bidding strategies for market-based scheduling considering more the provider side. There is a lack of analysis by the interaction of provider local machine scheduling policies and provider bidding strategies in terms of utility functions, scoring rules for utilization, classification of business consumers and profit. Such an analysis will enable to measure the impact to real systems and identify optimal configurations of market mechanisms and bidding strategies for providers and consumers.

## References

1. Xhafa, F., Abraham, A.: Meta-heuristics for Grid Scheduling Problems. Metaheuristics for Scheduling in Distributed Computing Environments (2008)

2. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud Computing and Grid Computing 360-Degree Compared. Grid Computing Environments Workshop (2008) 1–10
3. Macías, M., Rana, O., Smith, G., Guitart, J., Torres, J.: Maximizing revenue in grid markets using an economically enhanced resource manager. Concurrency and Computation: Practice and Experience (2008)
4. Nimis, J., et al.: D2.2a: Final specification and design documentation of the sorma components revised version. Technical report, `http://www.im.uni-karlsruhe.de/sorma/fileadmin/SORMA_Deliverables/D2.2a_final.pdf` (2009)
5. FIPA, T.: Fipa abstract architecture specification. Technical report, Foundation for Intelligent Physical Agents (2002)
6. Wellman, M., Walsh, W., Wurman, P., MacKie-Mason, J.: Auction protocols for decentralized scheduling. Games and Economic Behavior **35**(1-2) (2001) 271–303
7. Vytelingum, P., Cliff, D., Jennings, N.: Strategic bidding in continuous double auctions. Artificial Intelligence (2008)
8. Milojicic, D., et al.: MASIF: The OMG mobile agent system interoperability facility. Personal and Ubiquitous Computing **2** (1998) 117–129
9. Bergenti, F., Poggi, A.: LEAP: A FIPA Platform for Handheld and Mobile Devices. LNCS (2002) 436–446
10. Robles, S., Borrell, J., Bigham, J., Tokarchuk, L., Cuthbert, L.: Design of a trust model for a secure multi-agent marketplace. (2001) 77–78
11. Poslad, S., Laamanen, H., Malaka, R., Nick, A., Buckle, P., Zipf, A.: CRUMPET: Creation of user-friendly mobile services personalised for tourism. (2001) 28–32
12. Helsinger, A., Thome, M., Wright, T., Technol, B., Cambridge, M.: Cougaar: a scalable, distributed multi-agent architecture. **2** (2004)
13. Winikoff, M.: JACK Intelligent Agents: An Industrial Strength Platform. Chapter 7 in Multi-Agent Programming, edited by Rafael H. Bordini, Mehdi Dastani, Jrgen Dix, and Amal El Fallah Seghrouchni (2005)
14. Milano, M., Roli, A.: MAGMA: A Multiagent Architecture for Metaheuristics. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **34**(2) (2004) 925–941
15. Wurman, P., Wellman, M., Walsh, W.: The Michigan Internet AuctionBot: a configurable auction server for human and software agents. (1998) 301–308
16. Fasli, M., Michalakopoulos, M.: e-Game: A platform for developing auction-based market simulations. Decision Support Systems **44**(2) (2008) 469–481
17. Wolski, R., Plank, J., Brevik, J., Bryan, T.: Analyzing market-based resource allocation strategies for the computational grid. International Journal of High Performance Computing Applications (2001)
18. Nassif, L., Nogueira, J., de Andrade, F.: Distributed resource selection in grid using decision theory. 7th IEEE International Symposium on Cluster Computing and the Grid (2007)
19. Grosu, D., Das, A.: Auctioning resources in grids:model and protocols. Concurrency and Computation **18**(15) (2006)
20. Bapna, R., Das, S., Garfinkel, R., Stallaert, J.: A market design for grid computing. (2005)
21. Schnizler, B., Neumann, D., Veit, D., Weinhardt, C.: Trading grid services-a multi-attribute combinatorial approach. European Journal of Operational Research, forthcoming (2006)
22. Lai, K., Rasmusson, L., Adar, E., Zhang, L., Huberman, B.: Tycoon: An implementation of a distributed, market-based resource allocation system. Multiagent and Grid Systems **1**(3) (2005) 169–182

23. Heydenreich, B., Müller, R., Uetz, M.: Decentralization and Mechanism Design for Online Machine Scheduling. METEOR (2006)
24. Sherstov, A., Stone, P.: Three Automated Stock-Trading Agents: A Comparative Study. LNCS (2005)
25. Pardoe, D., Stone, P.: An autonomous agent for supply chain management. In Adomavicius, G., Gupta, A., eds.: Handbooks in Information Systems Series: Business Computing. Elsevier (2007)
26. Wellman, M., Greenwald, A., Stone, P.: Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition. MIT Press (2007)
27. Tesauro, G., Das, R.: High-performance bidding agents for the continuous double auction. Proceedings of the 3rd ACM conference on Electronic Commerce (2001)
28. Phelps, S.: Evolutionary mechanism design. (2007)
29. Borissov, N., Wirström, N.: Q-Strategy: A bidding strategy for market-based allocation of grid services. In: OTM Conferences (1). (2008) 744–761
30. Gimenez-Funes, E., Godo, L., Rodriquez-Aguilar, J., Garcia-Calves, P.: Designing bidding strategies for trading agents in electronic auctions. (1998) 136–143
31. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transactions on software Engineering and Methodology **12**(3) (2003) 317–370
32. Vytelingum, P., Dash, R., He, M., Jennings, N.: A framework for designing strategies for trading agents. Proc. IJCAI Workshop on Trading Agent Design and Analysis (2005) 7–13
33. Sardinha, J., Garcia, A., Lucena, C., Milidiú, R.: A Systematic Approach for Including Machine Learning in Multi-Agent Systems. (2005) 198
34. Myerson, R., Satterthwaite, M.: Efficient Mechanisms for Bilateral Trading. Journal of Economic Theory **29**(2) (1983) 265–281
35. Luo, X., Zhang, C., Jennings, N.: A hybrid model for sharing information between fuzzy, uncertain and default reasoning models in multi-agent systems. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10**(4) (2002) 401–450
36. Stoesser, J., Neumann, D.: A model of preference elicitation for distributed market-based resource allocation. In 17th European Conference on Information Systems (ECIS-2009) (2009)
37. Wurman, P., Wellman, M., Walsh, W.: A parameterization of the auction design space. Games and Economic Behavior **35**(1-2) (2001) 304–338
38. Friedman-Hill, E., et al.: Jess, the rule engine for the java platform. Distributed Computing Systems (2003)
39. Anjomshoaa, et al.: Job Submission Description Language (JSDL) Specification, Version 1.0. (2005)
40. Andreozzi, S., et al.: GLUE Specification v. 2.0. GLUE WG (2008)
41. DMTF: Common Information Model (CIM) v2.19.1. Distributed Management Task Force (DMTF). (2008)
42. Andrieux, A., et al.: Web services agreement specification (WS-Agreement) (March 2007)

# A Nature-inspired Approach for Large-Scale Pervasive Service Ecosystems

Cynthia Villalba and Franco Zambonelli

Department of Engineering Sciences and Methods
University of Modena and Reggio Emilia
Via Amendola 2, 42100 Reggio Emilia, Italy
{cynthia.villalba,franco.zambonelli}@unimore.it

**Abstract.** Innovative frameworks have to be identified for the deployment and execution of pervasive services made up of a massive number of components, and able to exhibit properties of *self-organization* and *self-adaptability*, and of *long-lasting evolvability*. This paper discuss how such frameworks should get inspiration from *Ecological systems*, modeling and deploying services as autonomous individuals (i.e *agents*), spatially-situated in an ecosystem of other services, data sources, and pervasive devices, all of which acting, interacting, and evolving according to a limited set of *"laws of nature"*. In this context, we present a reference architecture to frame the concepts and components of *eco-inspired* systems, discuss the characteristics of the ecological approach, and exemplify it with the help of a representative case study. Preliminary simulation results show the potential effectiveness of the approach.

## 1 Introduction

Pervasive and mobile computing devices increasingly populate our environments [5, 18]. These, together with the increasing amount of Web tools that make it possible to produce and access spatially-situated information about the physical world [3], will define a global-scale and very dense, decentralized infrastructure for general-purpose usage. At the user level, the infrastructure can be used to access innovative services for better perceiving/interacting with the physical world and for acting on it. It is also expected that users themselves will be able to personalize the infrastructure by deploying customized services over it. In addition, the infrastructure will be used as a way to enrich traditional classes of services with the capability of dynamically and autonomously adapting their behavior to the context in which they are exploited.

The effective development and execution of services in the above infrastructure calls for a deep rethinking of current service models and of service frameworks, in order to: (*i*) Naturally match the spatial nature of the environment and of the services within, and rely on mostly localized spatial interaction to provide support for massive scalability. (*ii*) Inherently exhibit properties of self-organization, self-adaptation and self-management that are required in highly-decentralized and highly-dynamic scenarios. (*iii*) Flexibly tolerate evolutions of

structure and usage over time. This is necessary to account for increasingly diverse and demanding needs of users as well as for technological evolution, without forcing significant re-engineering to incorporate innovations and changes.

To reach this goal, we should no longer conceive services and their interactions as in usual service-oriented architectures [10], where services are simply functional entities orchestrated according to mostly static patterns and with the help of specific middleware services. No one can rely on ad-hoc one-of solutions to achieve features of self-* properties. Rather, the most promising direction is that of taking inspiration from natural systems [17, 9], where spatial concepts, self-organization, self-management, and long-lasting evolvability are inherently there because of the basic "rules of the game". We are aware that nature-inspired solutions have already been extensively exploited in the area of distributed computing for the implementation of specific middleware solutions or of specific distributed services [12]. Here we go further, by arguing that natural ecosystem can act as the key metaphor around which to conceive, model, and develop a fully-fledged pervasive service framework and all the components within.

Although one can think at different classes of natural systems and from different perspectives (e.g., physical, chemical, biological, or social) one can always recognize the following characteristics: above a spatial environmental substrate, autonomous individuals (i.e., agents) of different kinds interact, compete, and combine with each other in respect of the basic laws of nature. Accordingly, in our scenario, the shared pervasive infrastructure substrate will have to be conceived as the space in which bringing to life an ecosystem of service agents, intended as individuals whose computational activities are subject to some basic laws of the ecosystem, and for which the dynamics of the ecosystem (as determined by the enactment of its laws) will provide for naturally enforcing features of self-organization, self-management, and evolvability.

In this context, the paper provides the following contributions: at first, we introduce a reference architecture for nature-inspired pervasive service ecosystems, to show how ecosystem concepts can be framed into a unifying conceptual scheme, and briefly discuss the possible approaches to realize the architecture and the related works in the area (Section 2). We detail the specific ecological approach that we have started investigating. The approach abstracts the components of the ecosystem as sorts of "hungry" goal-oriented organisms that, driven by laws of survival, interact with each other and self-organize their activities according to dynamic food-web relations (Section 3). A simple yet representative case study is also introduced to present via test, performed in a simulation environment, a preliminary assessment of the potentials of our approach (Section 4). Section 5 concludes and outlines directions for future work.

## 2 Unified Architecture for Pervasive Service Ecosystems

A unifying reference architecture can be identified around which to frame the key abstractions and the conceptual structure for spatial pervasive service ecosystems, independently of the specific metaphor adopted (see Figure 1).

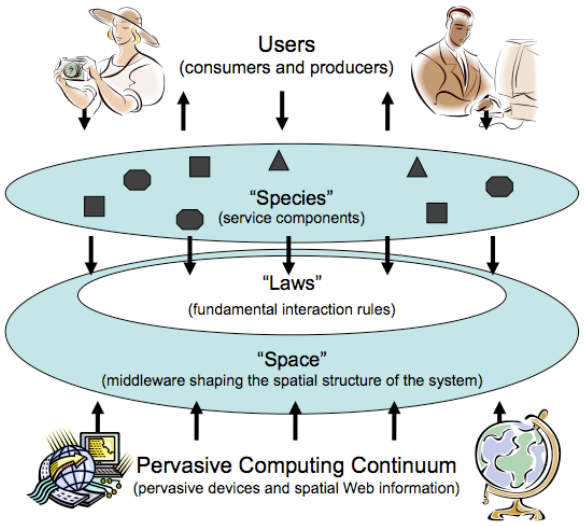


**Fig. 1.** A Reference Architecture for Pervasive Service Ecosystems

At the lowest level is the physical ground on which the ecosystem will be deployed, i.e., a very dense and widely populated infrastructure (ideally, a worldwide pervasive continuum) of networked computing devices (e.g., PDAs, tags) and information sources (Web fragments). At the highest level, service developers, producers and consumers of services and data, access the open service framework for using/consuming data or services, as well as for producing/deploying in the framework new services and new data components. At both levels, the architecture exhibits a high-degree of openness and dynamics, as new devices, users, services, data components can join and leave the system at any time. Between these levels, there are the components of the pervasive ecosystem architecture.

The "Species" level is the one in which physical and virtual devices of the pervasive system, digital and network resources of any kind, persistent and temporary knowledge/data, contextual information, events and information requests, and of course software service components, are all abstracted as "living individuals" (or agents) of the system. Although such individuals are expected to be modeled (and computationally rendered) in a uniform way, they will have specific characteristics very different from each other, i.e., they will be of different "species". The dense population of devices and actors involved at the highest and lowest levels, together with their dynamics, reflect in the presence of a very massive and dynamically varying number of individuals and species.

The "Space" level provides the spatial fabric for supporting individuals, their spatial activities and interactions, as well as their life-cycle. From a conceptual viewpoint, the "Space" level gives shape to and defines the structure of the virtual world in which individual lives. What the actual structure and shape could be, might depend on the specific abstractions adopted for the modeling of

the ecosystem. From a practical viewpoint, the spatial structure of the ecosystem will be implemented by means of some minimal middleware substrate supporting the execution and life cycle of individuals, and will enforce concepts of locality, local interactions, and mobility, coherently to a specific structure of the space.

The way in which individuals live and interact (which may include how they produce and diffuse information, how they move in the environment, how they self-compose and/or self-aggregate with each others, aggregate, how they can spawn new individuals, and how they decay or die) is determined by the set of fundamental "Laws" regulating the eternal service ecosystems model. Such laws, or "eco-laws", are expected to act on the basis of spatial locality principles, as in real laws of nature (which is also what makes real ecosystems scalable): the enactment of the laws on individuals will typically affect and be affected by the local space around them and by the other individuals on.

The dynamics of the ecosystem will be overall determined by having individuals in the ecosystem act based on their own internal goals, yet being subject to the eco-laws for their actions and interactions. The fact that the way eco-laws apply may be affected by the presence and state of other individuals, provides for closing the feedback look which is a necessary characteristic to enable self-* features. As far as adaptation over time and long-term evolution are concerned, the very existence of the eco-laws can make the overall ecosystem sort of eternal, and capable of tolerating dramatic changes in the structure and behavior of the species. Simply said in ecological terms: while the basic laws of life (i.e., the basic infrastructure and its laws) are eternal and do not change (i.e., do not require re-engineering), the forms under which it manifests continuously evolve (i.e., the actual service and data species), naturally inducing new dynamics for the interactions between individuals and for the ecosystem as a whole.

### 2.1 Metaphors and Related Work

The key difference in the possible approaches that can be undertaken towards the realization of eco-inspired service frameworks stands in the metaphor adopted to model the ecosystem, its individuals, the space in which they live, and its laws.

Physical metaphors consider that the species of the ecosystem are sort of computational particles, living in a metric space of other particles and virtual computational fields, which act as the basic interaction means. In fact, all activities of particles are driven by laws that determine how fields should diffuse and how particles should be influenced by the local gradients and shape of some computational field. Physical metaphors have been proposed to deal with several specific middleware-level aspects in dynamic network scenarios [2, 13] or in the area of amorphous computing [16]. However, they appear not suitable for general adoption in large-scale pervasive service ecosystems, due to the fact that they hardly tolerate diversity and evolution (i.e., the number of behaviors and interactions enforced via fields is limited).

Biological metaphors typically focus on biological systems at the small scale, i.e., at the scale of individual organisms (e.g., cells and their interactions) or

of colonies of simple organisms (e.g. ant colonies). The species are therefore either simple cells or very simple (unintelligent) animals, that act on the basis of very simple reactive behaviors and that are influenced in their activities by the strength of specific chemical signals (i.e., pheromones) in their surroundings. Biological metaphors are very similar to physical ones. Indeed, they too have been proposed to solve specific algorithmic problems in distributed network scenarios [15], and they too can hardly be suitable for general service ecosystems due to the same shortcomings of physical systems.

Chemical metaphors consider that the species of the ecosystem are sorts of computational atoms/molecules, living in localized solutions, and with properties described by some sort of semantic descriptions which are the computational counterpart of the description of the bonding properties of physical atoms and molecules. The laws that drive the overall behavior of the ecosystem are sort of chemical laws that dictate how chemical reactions and bonding between components take place to realize self-organizing patterns and aggregations of components. Chemical metaphors have been proposed to facilitate dynamic service composition [15], and they appear to well tolerate diversity and evolution. However, they cannot easily apply in large-scale and spatially distributed systems.

In the end, only ecological metaphors which focus on biological systems at the level of animal species and their interactions (as described in the following and along the lines envisioned in [1]), promises to be suitable for large-scale pervasive service ecosystems. In fact, other than supporting adaptive spatial forms of self-organization based on local food-web interactions, they also inherently support diversity and evolution. Others possible approaches that can be suited to model pervasive computing scenarios and that are somewhat similar to the ecological metaphors are market based models [4], these are based on the classical model of consumers, producers and purchasing goods and services. The problem with these approaches is that they can be fitted to pervasive computing, but still there are much work to do to prove this.

## 3 The Ecological Approach

### 3.1 Key components

Ecological metaphors focus on biological systems at the level of animal species and of their interactions. In our specific approach, the components of the ecosystem are sort of goal-oriented animals (i.e., agents) belonging to a specific species (i.e., agent classes), that are in search of "food" resources to survive and prosper (e.g., specific resources or other components matching specific criteria) that is, individuals have the ego-centric goal of surviving by finding the appropriate food and resources. The laws of the ecosystem determine how the resulting "food web" should be realized and ruled, that is, they determine how and in which conditions animals are allowed to search food, eat, and possibly produce and reproduce, thus influencing and ruling the overall dynamics of the ecosystem and the interaction among individuals of different species.

The shape of the space is typically organized around a set of localities, i.e., of ecological niches (think at a set of local pervasive computing environments), yet enabling interactions and diffusion of species across niches. Each locality will determine how the different species organize to live, will describe how individuals of each species respond to the distribution of resources and other species, and how they alter these factors.

Such Ecological metaphor promises to be very suitable for local forms of spatial self-organization (think at equilibria in ecological niches), and are particularly suited for modeling and tolerating evolution over time (think at how biodiversity has increased over the course of evolution, without ever mining the health existence of life in each and every place on earth). However, understanding how to properly control the local and global equilibria of real ecological system is a difficult task, and it would probably be very difficult also in their computational counterparts, yet we think that is an interesting approach to explore.

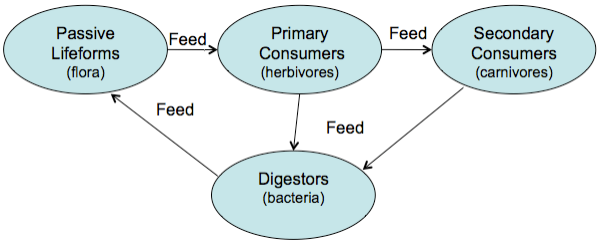


**Fig. 2.** Key Elements for an Ecological System.

In general, an ecological system can consider the presence of different classes of living forms (see Figure 2). Passive life forms (i.e., the flora system) do not actively look for food, although their existence and survival must be supported by nutrients that are in the space. Primary consumers (i.e., herbivores) need to eat vegetables to survive and prosper. Secondary consumers (i.e. carnivores) typically need to eat other animals to survive, though this does not exclude that can also act as primary consumers (eating vegetables too). The result of the metabolization of food by both primary and secondary consumers ends up in feeding lower-level "digesters" life forms (e.g., bacteria), densely spread in space, and that in their turn produce and diffuse resources and nutrients for the flora.

Let us now translate the above concepts in computational terms. Passive life forms represent the data sources of the ecosystem, which are not to be considered proactive computational entities. Primary consumers represent those services that require to digest information to be of any use, and yet are computationally autonomous. Secondary consumers, instead, are those services that, to be of any use, need the support of other services, other than possibly of information sources. Digesters can be generally assimilated to all those background computational services that are devote to monitor the overall activities of the system,

and either produce new information about or influence the existing information. This closes the food-web loop that can support self-organization.

### 3.2 Modeling of individuals

Let us describe the elements that define an individual of the eco-system. These include: its needs, a happiness function and a set of actions it can perform.

Each individual needs food and resources to survive. These needs are specific for each individual/species. These typically will be represented by a sort of "templates" (possibly a semantic representation) describing the characteristics of the needed resources or individuals. Of course, this also requires that each individual exposes its own characteristics.

Based on the need of an individual, and of the characteristics of the individuals around it, a process of matching takes place. The general template for the match function is given by: $match(individual, individual\_need)$ . This function should return some value, expressing the degree of match if in a given niche there is an *individual* and an available *individual_need*.

Based on this match, individuals can, according to specific behaviors, start interacting with each other (e.g. a primary consumer having found matching information, can decide to absorb or consume it). The happiness function tells us the satisfaction degree of each individual, in other words how well an individual satisfies himself. In general, the happiness of an individual is greater when it finds a lot of resources to eat (i.e. lot of matching individuals). Also, the happiness of an individual it matches with, can transitively influence an individual.

The individual's happiness is influenced by the different situations caused by other individuals that are in his near environment. The near environment has a limited area that is regulated by the specific characteristics of the environment. For instance, in a wireless scenario, the near environment can be defined as the space covered by the radio range of an individual.

We can measure two kinds of individuals' happiness during the time: one that is instantaneous and does not consider what happened before and the other is the overall happiness, that considers the instantaneous happiness and the happiness happened in the past.

The instantaneous happiness of an individual is related to his needs in a given moment, which means that it is directly proportional to the match function and the number of individuals that are in his near environment. The general function to obtain the instantaneous happiness of an individual in a niche, where there are $n$ individuals that influence in his happiness is given by:

$$H_{inst}(t) = f(match(individual_i, individual\_need_i), i = 1, ..., n).$$

The general template for the overall happiness function is given by:

$$H_{overall}(t) = f(H_{inst}(t) + H_{overall}(t-1)).$$

The actions are the third element that defines an individual. The actions that an individual perform during the time are always looking to increase his happiness. These actions can be move, reproduce, migrate, dye, etc.

The degree of happiness influences in the actions that each individual performs. Each individual moves through the space until he finds an appropriate niche where he feels well, gets a proper food and reaches some kind of happiness. In other words, if an individual is not happy or wants to be happier (find a better food), then he will move (perform some actions) from one niche to another until finds the appropriate one.

Here we have presented a simple model based on the individuals' happiness in order to choose the appropriate behavior at each tick. There are others similar models [6, 11, 14] based on agents' satisfaction or motivation, but they have not been applied to pervasive computing yet.

### 3.3 A Case Study

To better clarify this idea, let us present a simple case study we are currently in the process of developing and testing, which has been inspired by [7, 8].

Consider a scenario with diverse kind of devices on it, like a thematic park or an exhibition center, densely pervaded with digital screens where to display information, movies, advertisements, or whatever. We can consider each of these screens (i.e., the computational resources associated with each of them) as a spatially confined ecological niche. Different classes of visitors will watch these screens to look for different information (intended as passive life forms). Thus, we can think at sort of "user agents" executing on the users' PDAs that, once in the proximity of a screen (i.e., while finding themselves into that specific ecological niche) start looking for specific information to eat (i.e., to have it displayed). User agents would thus act as primary consumers with the goal to find the information required by users. Concurrently, we can think at "advertising agents" that, acting on behalf of some advertising company, roam from screen to screen in search of specific classes of user agents (i.e. those interested in specific types of information), with the ultimate goal of displaying advertisements where they could be more effective. Advertising agents would this act as secondary consumers. Background monitoring agents, executing on each ecological niche and possibly interacting with each other, can contribute replicating and spreading information there where it appears to be more appreciated, and can also contribute in supporting the spatial roaming of advertiser agents by directing them there where they could find more satisfaction. Thus, they would act as digesters.

The resulting food web can be summarized as follow (see Figure 3). Information agents feed the user agents, and these feed the advertise agents. Display agents (monitoring agents) maintain the feedback in the system (between information agents and user agents; and user agents and advertise agents), in order to maintain the equilibrium in the ecosystem. It is expected that the whole system continuously move during the time following this food web. In particular, user agents will keep moving trying to follow users movements.

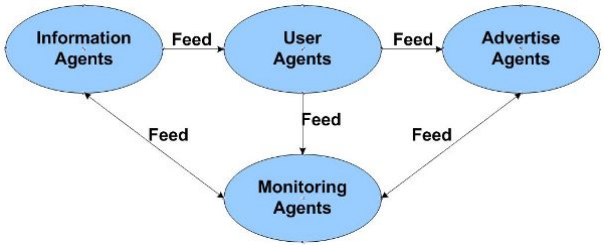


**Fig. 3.** Food web for an Ecosystem of Displays.

The feedback loop that derives from the above activities can contribute to properly rule the overall dynamics of the screen ecosystem, by continuously self-organizing and self-adapting the way information flows in the system, as well as the way advertising agents move, act, and coordinate with each other. The possibility of exerting control over the dynamics of the system is ensured by the possibility of injecting in the system additional classes of "digester agents" that can radically influence the dynamics of information diffusion and the activities of advertising agents. The adaptation of the system over time is ensured by the fact that it is mostly irrelevant, for the overall functioning of the system, what specific classes of information user agents want, or what the specific goal of advertising agent is. In fact, independently of the specific species of life forms that will populate the system, the basic eco-laws will ensure that such life forms will either find their way of living and their role in the system (e.g., as it can be the case of useful information and of advertising agents that find appropriate users to which to display their ads), or will simply disappear (as it can be the case of useless information or of advertisements no user is interested in).

Let us now detail the modeling of the various agent classes, i.e. of this is the specific happiness evaluation function.

The users' happiness is directly proportional to the match of their information requests, if they were satisfied or not. Then, the formula that gives us the happiness of each user is given by the followings:

$$H_{user}(t_0) = 0. \tag{1}$$

The instantaneous happiness for one user is the following:

$$H_{user}(t) = \sum_{k=0}^{n} match(user, displayedTask_k(t))/n. \tag{2}$$

In this formula, we define $n$ as the quantity of displays that are around the user in a specific time, which are going to influence in his happiness. The *match* function returns the value of the user's happiness when he sees something in a given display ($k$) and in a given time ($t$). This function returns different values depending on what the $displayedTask$ is.

The overall happiness for one user agent is:

$$H_{user}(t+1) = \sum_{k=0}^{t-1} H_{user}(k) + H_{user}(t) - \triangle. \quad (3)$$

The equation has the sum of user's happiness in the past (before $t$) and the instantaneous happiness. We define the symbol $\triangle$ as a constant value that represents the user's unhappiness, makes his happiness decrease during the time. This value is useful when the user does not see something that he is interested in, in this case, the current happiness will return 0 and this constant will make the happiness decrease.

Advertise agents are happy when they show their publicity in an environment with happy users that want to watch their advertisements. The formula that evaluates the happiness of advertisements is given by:

$$H_{adv}(t_0) = 0. \quad (4)$$

The instantaneous happiness for one advertisement agent is:

$$H_{adv}(t) = (\sum_{k=0}^{n} match(adv, dT_k(t))/n) * (\sum_{k=0}^{m} H_{user}(t)/m). \quad (5)$$

In this function $n$ and $m$ are defined as the quantity of displays and users respectively that are around the advertisement in a specific moment $t$ and influence in the advertisement's happiness. $dT_k(t)$ refers to the $displayedTask$, that is the currently task showed in the displays. The function $H_{user}(t)$ can evaluates the overall happiness average or the instantaneous happiness of the user. If we evaluate the overall happiness then, the advertise happiness will be influenced by all the happiness that had happened before (for example if the user was happy or not in the past). If we evaluate the instantaneous happiness of the user then, no matter what happened before, just matter the current time that we are evaluating. Both functions have their advantages and disadvantages, these will be analyzed with the results of the simulation of each function.

The *match* function returns the value of the happiness of the advertise agent when he sees something in a given display ($k$) and a given time ($t$). For example, if the advertise agent sees information in the display, this function should return a positive value, making the advertisement's happiness increase a little, because he is interested in happy users.

The overall happiness for one advertisement is given by:

$$H_{adv}(t+1) = \sum_{k=0}^{t-1} H_{adv}(k) + H_{adv}(t) - \triangle. \quad (6)$$

As in the users' happiness, this function includes the happiness in the past and the current one. $\triangle$ is the value that decreases the advertisement happiness.

The display agents just want to show advertisements because they pay for the service. The happiness of the display is directly influenced by the amount of

advertisements showed and by the happiness of their clients (advertise agents). The formula that evaluates the happiness of the displays is given by:

$$H_{dis}(t_0) = 0. \tag{7}$$

The instantaneous happiness for one display agent is:

$$H_{dis}(t) = (\sum_{k=0}^{n} H_{adv}(k)/n). \tag{8}$$

We define $n$ as the number of advertises that are around the display. Here, we have the same problem that we described before. We have two kind of advertise agent happiness that can be analyzed here, the overall and the instantaneous happiness. We plan to analyze it with the results of the simulations.

The overall happiness for one display is given by:

$$H_{dis}(t+1) = \sum_{k=0}^{t-1} H_{dis}(k) + H_{dis}(t) - \triangle. \tag{9}$$

This function is defined as the overall happiness described before.

A possible reason that it causes the decrease of the display agents happiness is that in their niches, there are not enough advertise agents that want to show their publicity, because there are not enough happy user agents because there are not enough information agents. To overcome this situation and reach the happiness, the display agent has to generate more information agents, in order to have happy users and indirectly happy advertisements in his niche.

Displays agents have two main functions: to select the task (information or advertisement) to display and to provide the necessary information to the user and advertise agents, in order to they can move towards the niches where they can find what they need (information and user agents corresponding).

The selection of the task to show is made in each niche by each display agent. This selection tries to maximize the displays' happiness and consequently, the happiness of the advertisements that are around the display (see Formulas 8 and 9). In simple words, the selection consists of choosing the task that will make the advertisements happier.

Display agents examine their niche to provide the necessary information about user and information agents. Before the user and advertise agents decide where to move, they evaluate which is the best next position to find what they want. The evaluation can be based on several considerations:

How? Advertise and user agents require to the near display agents the necessary information to decide in which direction (toward witch niche) move. If they do not find the appropriate users/information around, they move in randomly direction. Where? Just move to the position that is next to their current position. The migration is not considered. When? Advertisements and users remain in their current position unless their happiness is equal to zero ($H = 0$). If their happiness is zero, they evaluate the niches around.

# 4 Experimental Results

## 4.1 The simulation environment

We have implemented a simulation environment to assess our approach in the context of the case study, within the Recursive Porous Agent Simulation Toolkit (Repast).

As from the Section 3.3, the scenario considers a 2D space (the thematic park), where different spatially situated agents (users, advertisements, information and display) act, interact and evolve given the eco-laws (trying to maximize their happiness). Simulations initialize user, advertise and information agents with randomly positions. User and advertise agents can move during the simulation to find what they want: information and user agents respectively. Displays agents have a fixed position during the whole simulation. We particularly study the behavior of users, advertise and displays.

The simulation environment has currently some limitations that will be overcome in the future, and that nevertheless enable us some preliminary assessment of our approach. The current limitations are: (*i*) User agents do not move continuously but stop when they find their appropriate niche. (*ii*) Agents do not contemplate the dynamical actions of joining or leaving the system and the overall number of agents is static. (*iii*) Information and advertise agents do not self adapt to the users requirements. (*iv*) Agents cannot migrate, reproduce or dye.(*v*) Display agents do not interact with each other.

## 4.2 Results

Several experiments have been performed to test the behavior of the eco-system of displays in our simulation environment.

Figures 4 and 5 show two screenshots of a simulation scenario with 100 advertise agents, 100 user agents, 100 information agents and 25 displays. The squares represent the displays and the little bodies represent the users. Displays can show different colors that represent what they are showing. In order to make clearer these Figures, we do not show advertise and information agents.

These Figures show how the user agents' behavior changes during the time, particularly how they group around displays. The Figures were taken at two different moments of the simulation: the first one at the beginning, when users move through the space (Figure 4) and the second one almost at the end, when users find the niche where they feel happy (Figure 5). After a period of time, we can see that agents always get grouped in the niches where they find what they need, i.e. they feel "happy". Agents move, flow through the space until they find their appropriate niche.

To evaluate the whole behavior of the system, we measured the happiness of agents and their process of convergence. Figures 6 and 7 show how the current average happiness (between 0 and 1) of users and advertisements varies over time until it reaches the equilibrium. The happiness of display agents is not showed, because their happiness is proportional and quite similar to the advertise

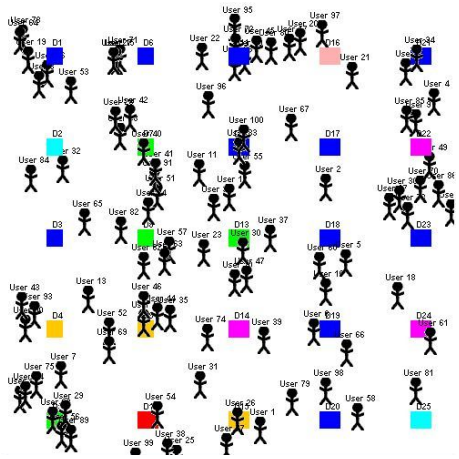

**Fig. 4.** Space View for 100 agents (few ticks).

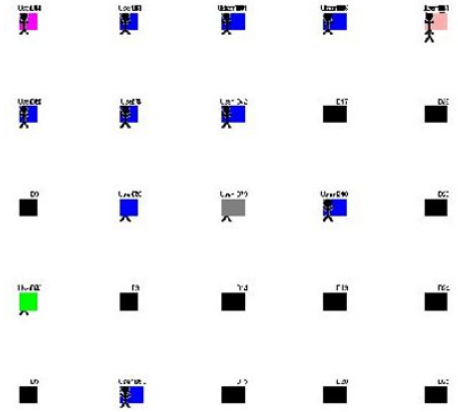

**Fig. 5.** Space View for 100 agents (the equilibrium).

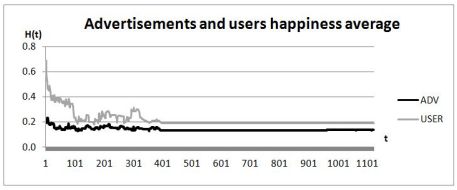


**Fig. 6.** Happiness average for 100 agents.

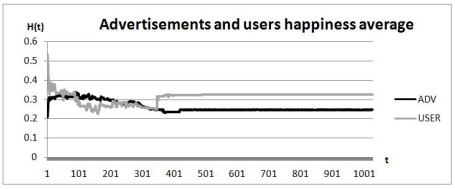


**Fig. 7.** Happiness average for 500 agents.

agents happiness. We can see that the curve fluctuates at the beginning, as user and advertise agents are looking what they want (information and users respectively), when they begin to find their appropriate niches, the curve begins to stabilize, until arrives to be almost a constant (not change during the time), that means that the systems arrives to a balance. These Figures show clearly how the agents can organize by themselves and drive the system to a balance state, independently of the actual size of the system. The simulated eco-system has a great potential to deal with the global self-organization of the activities of their individuals (agents).

By aggregating the results of several experiments with varying number of agent, we had similar results and conclude that the system always reaches the equilibrium, no matter the number of components. In other words, we can say that system promises to be scalable. This is showed in Figures 8 and 9. Figure 8 shows how many ticks approximately the systems need to reach the equilibrium under different number of agents. Advertisements and users almost always arrive at the same time to equilibrium. Figure 9 shows the happiness values of users and advertisements when the system reaches the equilibrium state. It seems like the happiness of both agents grows if the quantity of user agents grow, we need

to do more test in order to confirm this trend. This would be a good point for the scalability of the systems: the more user agents are, better the overall organization of the system.

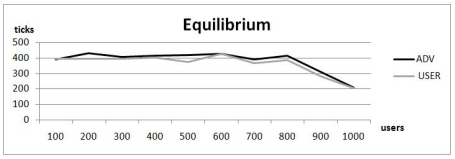


**Fig. 8.** Quantity of user by quantity of ticks to arrive the equilibrium.

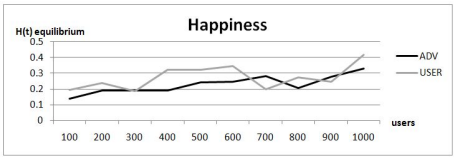


**Fig. 9.** Quantity of user happiness in the equilibrium tick.

## 5 Conclusions

In this paper, we have elaborated on the idea of model and develop next generation of pervasive service framework inspired in the ecological model. That is, of conceiving future pervasive service frameworks as a spatial ecosystem in which services, data items, and resources are all modeled as autonomous individuals (agents) that locally act and interact in accord to a simple set of well-defined "eco-laws".

We attempted to clarify these ideas through a simulation of a case study related to an adaptive advertisement displays. The development of this simulation is not finished yet, however the first results give some feedbacks about the self-organization, the scalability and process of convergence of the system.

Nevertheless, there are still a lot of experiments to do and several open questions to answer to better assess our approach: (*i*) Concerning to the simulation environment: we have to extend it to overcome its current limitations. (*ii*) Concerning to the general applicability of approach: we have to experience with a larger variety of case studies. (*iii*) Finally, we have to better formalize and generalize the model and realize a prototype implementation.

## References

1. Gul Agha. Computing in pervasive cyberspace. *Commun. ACM*, 51(1):68–70, 2008.
2. Jacob Beal and Jonathan Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems*, 21(2):10–19, 2006.
3. Gabriella Castelli, Alberto Rosi, Marco Mamei, and Franco Zambonelli. A simple model and infrastructure for context-aware browsing of the world. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 229–238, Washington, DC, USA, 2007. IEEE Computer Society.

4. David Cornforth, Michael Kirley, and Terry Bossomaier. Agent heterogeneity and coalition formation: Investigating market-based cooperative problem solving. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 556–563, Washington, DC, USA, 2004. IEEE Computer Society.
5. Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.
6. Daniel Thalmann Etienne de Sevin. An affective model of action selection for virtual humans. In *AISB '05: Proceedings of Agents that Want and Like: Motivational and Emotional Roots of Cognition and Action symposium at the Artificial Intelligence and Social Behaviors*, 2005.
7. Alois Ferscha, Andreas Riener, Manfred Hechinger, and Heinrich Schmitzberger. Building pervasive display landscapes with stick-on interfaces. In *Workshop "Information Visualization and Interaction Techniques", associated with CHI'06 International Conference, Quebec, Canada*, page 9, April 2006.
8. Alois Ferscha and Simon Vogl. The webwall. In *Proceedings of the Ubicomp 2002 Workshop on Collaboration with Interactive Walls and Tables*, Göteborg, Sweden, September 2002.
9. Sebastian Herold, Holger Klus, Dirk Niebuhr, and Andreas Rausch. Engineering of it ecosystems: design of ultra-large-scale software-intensive systems. In *ULSSIS '08: Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, pages 49–52, New York, NY, USA, 2008. ACM.
10. Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.
11. Pattie Maes. The agent network architecture (ana). *SIGART Bull.*, 2(4):115–120, 1991.
12. Marco Mamei, Ronaldo Menezes, Robert Tolksdorf, and Franco Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8):443–460, 2006.
13. Marco Mamei and Franco Zambonelli. *Field-Based Coordination for Pervasive Multiagent Systems (Springer Series on Agent Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
14. Jacques Ferber Oliver Simonin. Modeling self satisfaction and altruism to handle action selection and reactive cooperation. In *SAB '00: Simulation of Adaptive Behaviors*, pages 314–323, 2000.
15. Raffaele Quitadamo, Franco Zambonelli, and Giacomo Cabri. The service ecosystem: Dynamic self-aggregation of pervasive communication services. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 1, Washington, DC, USA, 2007. IEEE Computer Society.
16. David Servat and Alexis Drogoul. Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence? In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 441–448, New York, NY, USA, 2002. ACM.
17. Mihaela Ulieru and Steve Grobbelaar. Engineering industrial ecosystems in a networked world. In *5th IEEE International Conference on Industrial Informatics*, pages 1–7. IEEE Press, 2007.
18. Roy Want. An introduction to rfid technology. *IEEE Pervasive Computing*, 5(1):25–33, 2006.

# Dross into Diamonds: Efficient Use of Untrustworthy Agents in a Large Scale Environment

Chris L. D. Jones[1] and K. Suzanne Barber[1]

[1] The University of Texas at Austin
Laboratory for Intelligent Processes and Systems
1 University Station C5000, Austin, TX, 78712-0240
{coldjones, barber}@lips.utexas.edu

**Abstract.** Self-interested agents in multi-agent systems have traditionally tried to find the most trustworthy and reliable agents, either via experience or reputation, to partner with. This paper seeks to demonstrate that multi-agent teams in large scale environments can improve their utility by using redundant groups of untrustworthy agents rather than using a single, most-reliable partner. This paper first describes a large scale market environment where contractees and contractors operate according to pure self-interest to increase their utility. The paper then describes a strategy for substituting teams of redundant, low reliability agents for individual, highly reliable partners, and finally demonstrates the superiority of such an approach via a multi-agent simulation. Results from this simulation demonstrate that agents utilizing redundant teams of unreliable agents are both more profitable and more successful in completing tasks.

**Keywords:** Coalition formation, Large-scale environments, Markets, Trust and reputation, Redundancy

## 1 Introduction

Recent advances in computational engineering have uncovered multiple real-world domains where multi-agent systems have the potential to be extremely useful. For example, grid-based computational systems may be enhanced by overlaying virtual organizations of software agents to efficiently manage cooperation and resources [1]. Similarly, distributed communication networks may use large swarms of software agents to optimize path length and increase overall efficiency [2]. Even teams of low-reliability agents may successfully complete complex workflows by using redundant teams of agents [3].

Multi-agent systems can benefit significantly from the use of experience and reputation information to find trustworthy agents [4, 5].

Trustworthiness may be defined as "confidence in the ability and intention of an agent to provide correct information or perform promised actions" [6]; thus, by utilizing various mechanisms to determine the most trustworthy and reliable partners, agents in a multi-agent system can find other agents who are most likely to fulfill contract agreements and provide required goods, services, or information in a prompt and efficient manner.

However, in large scale systems, many agents may be less reliable and trustworthy. In a system where agents seek to work with more trustworthy and reliable partners, less trustworthy and reliable partners may be less in demand and, therefore, more available for team formation. Furthermore, recent research has shown that less reliable agents may be more likely to abandon their current tasks in response to a new contracting request, which may make it easier to initially form teams [7].

Also, basic economic rules suggest that trustworthy, reliable agents can command higher premiums for their services, while less reliable, untrustworthy agents may be forced to take lower premiums. This begs the question of whether less reliable agents, which may be more available and less expensive to deal with than reliable, in-demand agents, may be used more efficiently.

Accordingly, this paper explores the idea of using multiple, less reliable agents, rather than single, highly reliable agents, as part of a multi-agent team to complete complex tasks in a large scale market environment. This paper is organized as follows. Section 2 of the paper describes a large scale market environment, and the behavior of selfish agents in such an environment. Section 3 provides this paper's main contribution, an agent strategy for creating multi-agent teams out of redundant, low-reliability agents, and the theoretical underpinnings of this novel strategy. Section 4 presents the results from and discusses of a set of experimental parameters designed to test the utility of the redundant strategy from section 3. Section 5 provides conclusions and possible future work.

## 2 Large Scale Market Environment

Agents in this paper operate in a large scale market environment, wherein they are divided into two groups: contractors, or foremen, who

assemble teams of agents to work on jobs comprised of multiple tasks, and contractees, or workers. In this, the simulation environment is similar to both the classic Contract Net framework and the Request for Proposal frameworks [8, 9].

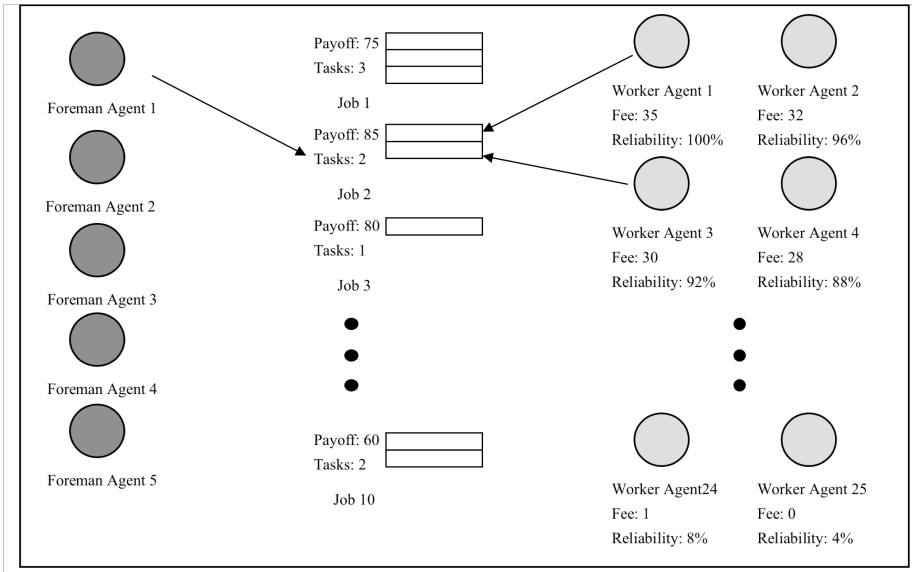


**Figure 1. Exemplary Jobs, Foreman, and Worker agents**

### 3.1 Jobs, Workers and Foremen

We begin with a broad overview of the three basic market components: jobs, workers and foremen, all of which are illustrated in Figure 1. Jobs consist of a payoff amount and a certain number of tasks which must be completed for the job to be complete. If all tasks in the job are successfully completed, the job's payoff amount is transferred to the foreman working on the job. There are a fixed number of jobs in the market at any given time, and jobs are replenished after they are attempted by teams of agents.

Workers receive fees from foremen to work on specific tasks within a job. Workers and tasks are not typed – any worker can work on any task within any job, but can only work on one task in any specific job. Workers demand fees to work on tasks, which are paid by foremen. The market environment contains a fixed pool of workers that does not

change. Different workers in the pool have different reliabilities – that is, different likelihoods of completing an assigned task. These reliabilities are set amounts for each different worker agent, and do not change over time.

Accordingly, while a worker always receives a fee to work on a task, its likelihood of actually completing the assigned task is entirely dependent on its set reliability – an agent with 50% reliability is 50% likely to finish the assigned task, an agent with 90% reliability is 90% likely to finish its assigned task, and so on. Workers also set the fees that they require to work on a team, and raise or lower their prices depending on how many offers they get, as will be described in further detail below. As will be seen, more reliable workers will logically command a higher fee to work on tasks than less reliable workers.

Foremen agents are responsible for assembling teams of worker agents to complete jobs. Foremen agents must always pay their workers in advance to work on a given job, regardless of if the agents are successful or not, but only receive payoff from the job if all tasks in the job are completed successfully. Accordingly, foremen agents look to balance the expense of paying a team to complete a job, with the probability that the job will successfully be completed.

Foremen agents therefore consider different possible combinations of jobs and worker assignments to determine which are the most potentially profitable. More specifically, foremen agents multiply the payoff of each job by the likelihood of that all worker agents on the job will complete their assigned tasks, and from this figure subtract the combined fees for all worker agents on the job to produce an expectation value for that combination job and agents. Foremen agents then take the job/team combination with the highest expectation value and try to complete that job, earning or losing profit as workers succeed or fail at their assigned tasks. Note that if foremen agents cannot find a job and team combination with a positive expected profit, they may decline to pursue any jobs.

For example, if a foreman agent in Figure 1 considers job 10, it will find several teams that are untenable for solving the job. Worker agents 1 and 2, although the most reliable agents available to fill the job's two tasks, also have combined asking fees (67) greater than the job payout (60). Worker agents 3 and 4 have a combined asking fee less than 60, but also have a combined probability of only 81% of successfully completing the job, which translates into an expected

payout value for the foreman agent of 48.6. Since this amount is less than what the agent would have to pay out, this team is likewise not feasible for the foreman.

However, as shown in Figure 1, a foreman could successfully form a team of worker agents 1 and 3 to work on the two tasks in job 2, since the foreman's expected payout would be 78.2, and the workers cumulative fee would only be 65.

### 3.2 Selection Mechanism

Jobs, foremen, and workers exist in an iterative market simulation, where, during each iteration, foremen consider their options and potentially form teams to solve jobs accordingly, while workers work on job tasks and adjust their fees according to how many offers they have recently received to work on a job.

At the beginning of each round of the simulation, foremen agents in the simulation are randomly selected to pick a job and a team to work with. The selected foremen consider all available jobs and try to determine the best team for each job using a randomized hill-climbing search algorithm. More precisely, for each job, the selected foreman generates a team of the most reliable agents available, without regard to cost. The expected earnings for that job/team combination is calculated, based on the job reward, agent reliability and agent costs. A random subset of workers is then generated for each member of this initial team. Each of these random workers is then swapped with the original, highly reliable worker, and the projected earnings of each potential new team calculated. The team which offers the best projected earnings is then retained, and the process is repeated multiple times.

The foreman agent then selects the best job/team combination out of those discovered, if any job/team combinations have an expected positive payout, and then attempts to complete the job. As described above, the foreman agent pays the asking fees of all workers on the team in advance, but only receives payoff from the job if all workers on the team successfully complete their assigned jobs. A running balance is kept for each agent's profit, so a worker agent's profits reflect the total amount it has been paid by foremen to work on tasks, and a foreman agent's balance reflects the cumulative payouts it has received from successfully completed job, less the cumulative amounts it has paid worker agents to work on jobs.

Once a job has been attempted by an agent team, it is removed from the pool of available jobs, and is no longer available to another team of agents to work on. Likewise, worker agents may only work on one job per simulation round, so that foremen agents who go later during each round have a more constrained pool of viable jobs and teams. Once all foremen have had an opportunity to form a team and complete a job, the simulation randomly generates new jobs to replace the jobs completed during the previous round, according to the parameters described below.

Finally, before the simulation advances to the next round, all worker agents in the simulation have an opportunity to adjust their fees.

Workers keep track of the last time they were hired by a foreman agent. If a predetermined period – 10 rounds – elapses without a worker receiving an offer, the worker decrements its asking fee by 1. (The fee cannot be decremented below 1, which represents the worker's fixed costs for completing the task.) Alternatively, if the worker receives multiple offers within another predetermined time period – 5 rounds – it increments its asking fee by one. Workers thereby adjust their asking fees in accordance with demand from foreman agents, which in turn is directly dependent on the reliability of the worker agents.

Accordingly, as shown below in Figure 2, worker agents in the simulation arrange themselves into a simple demand curve according to their reliability. Specifically, Figure 2 demonstrates the average asking fees of 2000 workers at the end of 24 simulations of 200 rounds apiece. (Additional parameters are identical to those found in Table 1 below). Worker agents were assigned reliabilities evenly distributed between 0.1% and 100%. As can be seen below, the demand curve rises sharply as agents become more reliable, such that highly reliable agents (>85% reliable) command an asking price around 95, and low reliability agents are essentially unwanted.

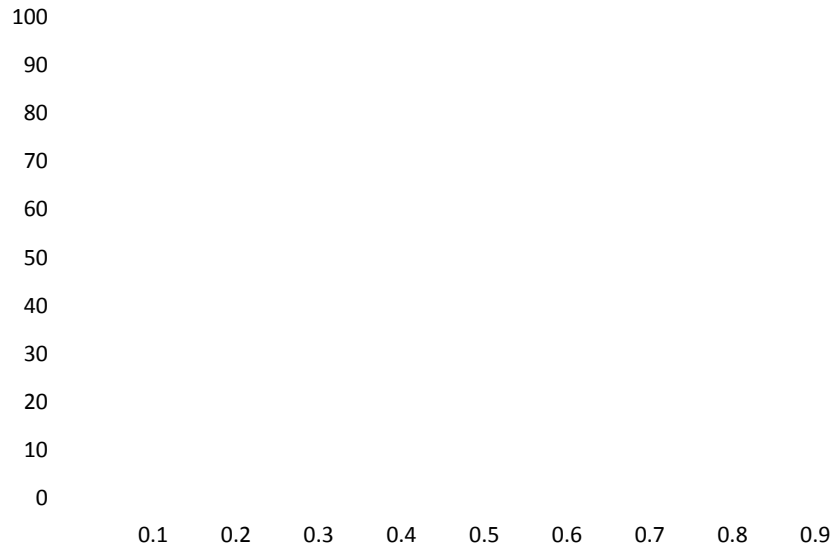


**Figure 2. Worker Asking Fees at the end of 200 simulation rounds**

## 4 Redundant Low-Reliability Workers

From the shape of the demand curve in Figure 2, it is obvious that only relatively reliable worker agents are in demand by foremen. This makes intuitive sense, since unreliability by workers on different tasks is compounded. While an individual reliability of 80% might be considered reasonably sufficient for most purposes, six 80% reliable agents working on six separate tasks in a job give an only slightly better than 25% chance that the job will be completed.

Accordingly, if highly reliable agents are the only way that a large team can form, but highly reliable agents are also highly expensive in a market environment, it would desirable to find another way to build large, multi-agent teams, ideally making use of otherwise under-utilized low-reliability agents. We therefore approach this problem by allowing foremen to create teams of redundant, low-reliability agents. Note that while this approach is somewhat similar to the one used by Stein, Jennings, et al. in [3], our approach differs in that it explicitly deals with agents of differing reliabilities, and seeks to use many such

agents in a team. In contrast, Stein, Jennings, et al. examine domains where agents all have a similar reliability, and deal with unreliability by determining how many basic agents should be added to a team to produce an acceptable cost/reliability tradeoff.

We therefore allow a selected foreman in the market simulation to hire multiple worker agents to complete a single task, as opposed to one agent per task. Under this modification to the simulation, any worker assigned to a task may successfully complete it, and the task is uncompleted only if no agent assigned to the task is successful. It remains the case that workers must be paid their asking fee in advance to work on a task, and that workers can only work on one task per round.

Foremen agents using this redundant strategy select teams differently than regular foreman agents. Like regular foremen agents, agents using redundant teams of workers determine the best team for every available job, and then select the best job/team combination to work on, if one exists with a positive expected profit for the foreman. However, in addition to the randomized hill-climbing search described above in section 2, redundant foremen agents utilize an algorithm that tries to replace each individual selected agents with a team of low reliability agents.

More precisely, the team is assembled by randomly selecting any agent, and then repeatedly adding increasingly reliable workers to the team until the team is determined to be more expensive or more reliable than the agent selected by randomized hill climbing. To maximize the diversity of teams considered, low reliability agents are selected by randomly selecting an agent, and then jumping forward a random number of agents to select the next worker. This mechanism therefore produces both relatively large teams of very unreliable agents (e.g. several agents under 50% reliability) and relatively smaller teams of fairly reliable agents (e.g. two agents of roughly 80% reliability). Whichever is cheaper – the single most reliable worker or the team of low reliability workers – is selected for the task and the process repeats until all tasks on the job are filled.

Note that, unlike the previous foreman strategy which primarily uses reliability data to form teams of highly trustworthy agents, the redundant strategy utilizes information about the reliability of different agents to form teams composed of agents with by high and low reliability. Although this paper assumes such knowledge is commonly

known and does not simulate an experience or reputation-based mechanism to gather such data, it should be clear that a strategy of building teams of low reliability agents would work with almost any existing mechanism that provides accurate reliability data on all agents in a system. Furthermore, in a large-scale system with hundreds or thousands of available agents, the method described above utilizes many agents that would otherwise be unused using more traditional methods.

## 5 Simulation Results and Discussion

The simulation was conducted in accordance with the parameters given below in Table 1. All differences that are noted as statistically significant have been differentiated by a t-test with an alpha of .05. The experiments below were run three times, the first time with all 200 foremen agents using the standard foreman search algorithm described above, the second time with 200 foreman agents using the redundant team formation algorithm described above, and the third time with a mixed population of 100 standard foremen and 100 redundant foremen.

**Table 1. Experimental parameters**

| Parameter | Value |
|---|---|
| Experiment types | 200 standard foremen / 200 redundant foremen / 100 standard and 100 redundant foremen |
| Number of worker agents | 2000 |
| Initial asking price per worker | 10 |
| Worker agent increment period | Less than 5 rounds between hires |
| Worker agent decrement period | More than 10 rounds between hires |
| Number of jobs | 200 |
| Tasks per job range | 5 to 10 |
| Payoff per job range | 500 to 1000 |
| Rounds per simulation | 200 |
| Number of simulations run | 24 |

Table 2 shows average final results gathered from the experiments described above in Table 1. More specifically, the average number of jobs attempted and jobs successful are displayed for experimental runs where the population of foremen exclusively followed each individual strategy, and where the population of foremen was evenly divided

between the two strategies. Although all foremen attempted the same number of jobs, redundant foremen completed significantly more jobs than standard foremen in all cases. As can be seen from the results in Table 2, redundant foremen operating separately from standard foremen completed nearly 33% more jobs, while redundant foremen operating concurrently with standard foremen completed more than 10% more jobs.

**Table 2. Experimental results**

| Metric | 200 standard foremen | 200 redundant foremen | 100 standard foremen | 100 redundant foremen |
|---|---|---|---|---|
| Jobs attempted | 10.0 | 10.0 | 10.0 | 10.0 |
| Jobs successful | 6.8 | 9.1 | 7.7 | 8.5 |
| Success ratio | 68.2% | 91.1% | 77.0% | 84.8% |
| Foreman profits | 2969 | 5078 | 3972 | 5528 |
| Profit per job | 434.2 | 557.2 | 513.8 | 650.0 |
| Workers used | 16.1% | 96.4% | 98.4% | N/A |
| Total worker profits | 475541 | 370191 | 297403 | N/A |

Foremen following the redundant strategies have a statistically significant advantage in total profits, both when executing separately and concurrently with standard foremen. Redundant foremen operating separately from standard foremen earned nearly 2000 points more in total profit, roughly 71% more in total, while redundant foremen operating concurrently earned 1300 points more, or roughly 39% more. This advantage in total profits traces not only to a greater success ratio among redundant foremen, but a significantly greater average profit per job.

Similarly, overall usage of all workers in the simulation tends to be much greater in homogeneous populations of redundant foremen and in mixed populations of standard and redundant foremen. Furthermore, the total worker profits, or the total cost of using the system of agents, is significantly lower when all redundant or mixed redundant/standard foremen are used, likely because competition between foremen for reliable workers is decreased.

Figures 3 and 4 provide additional evidence for this hypothesis. Figure 3 shows the average final asking fee for worker agents in the three experiment types, while Figure 4 shows the average total earnings for worker agents. As can be seen below, average asking fees and

earnings are both dramatically lower when redundant foremen are used. This is almost certainly due to the fact that a much greater range of worker agents are used by foremen using the redundant strategy, thereby creating less competition between foremen for highly reliable agents, and lower worker asking prices.

Lack of competition also seems to be responsible for increased foreman profits in the "mixed" redundancy experiments, where redundant strategy foremen executed concurrently with standard foremen. As can be seen below, although the standard strategy foremen drove the price of highly reliable agents up, and redundant strategy foremen drove the price of semi-reliable agents up, the fact that the two groups of foremen were largely not in competition with each other kept overall prices down.

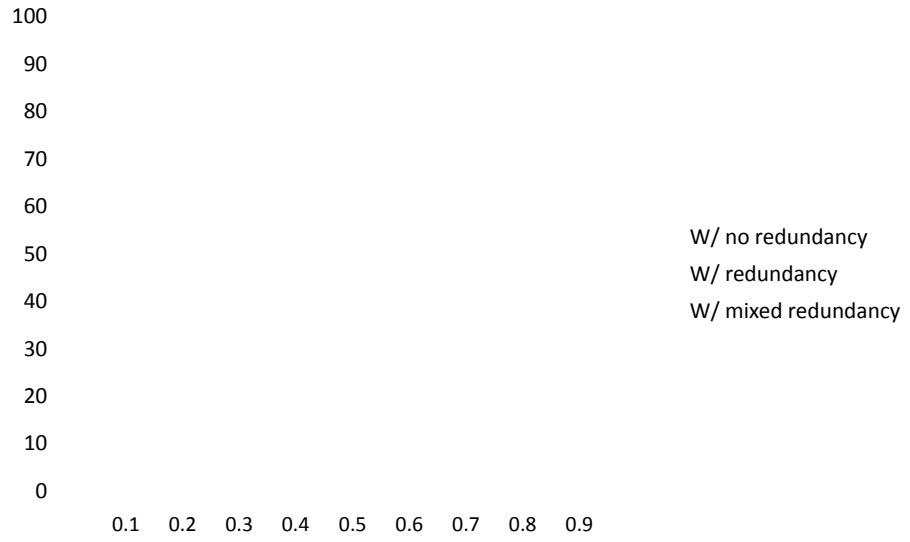


**Figure 3. Final average worker agent fees**

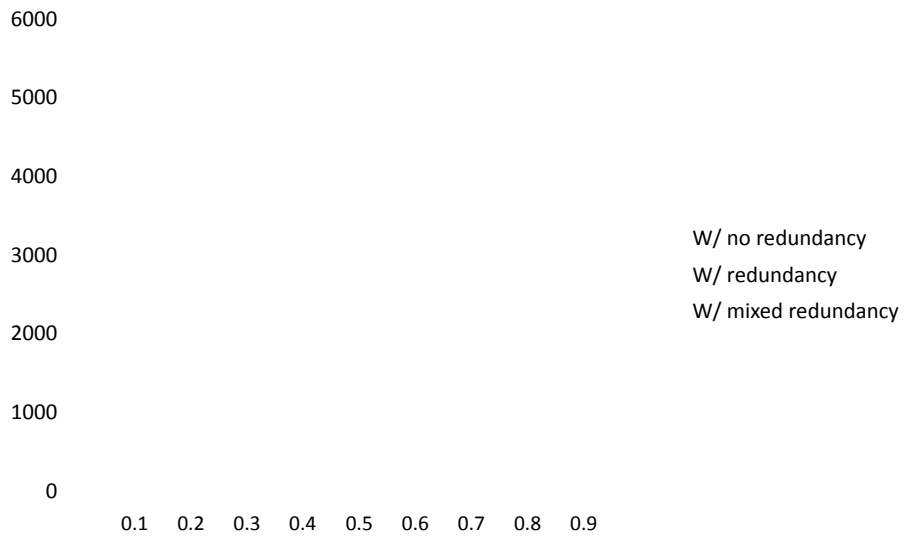


**Figure 4. Final average worker agent earnings**

Finally, it should be noted that there are some obvious domain limitations with the redundant strategy presented here. Although there are some domains where multiple agents can simultaneously attempt the same task, which can be considered successfully completed if any one of them is successful, there are other domains where only one selected agent may attempt a task at a time. For example, while low-level economic activities such as ditch digging might allow for multiple attempts, with no real loses incurred by switching between workers, other activities, such as brain surgery, clearly cannot be attempted over and over again.

In several large-scale computational domains where software agents might attempt to form teams to complete tasks, however, redundant teams of less reliable agents might prove useful. For example, it is possible to imagine a cloud computing environment where data storage services are represented by agents of differing reliability and costs. In such a domain, it might be better to attempt to store data on several cheap, low-reliability services, rather than a single expensive high-reliability service. (Obviously such an example assumes that data is either completely successfully stored or not. However, while the possibility of data being corrupted might be handled by encrypted

checksums, which would be used to verify that returned data is likely to have been unchanged.) Likewise, message passing services might be improved by using multiple transmission attempts through less reliable channels rather than single attempts through highly reliable channels.

In addition, it should be noted that many interactions between foremen and workers do not demand complete payment up front b the foreman, regardless of the worker's ultimate success. (Indeed, one of the primary functions of trust and reputation gathering mechanisms is to know under what circumstances a worker or contractee is sufficiently reliable to warrant being paid up front.) However, there are many situations where utilizing a contract worker involves at least some up front, unrecoverable expense for the contractor. Insofar as this up front expense is a smaller portion of a contractor's total expenses, the results here will be less relevant, but still valid. More importantly, this work shows that even in situations where a foremen entirely loses its up-front payment for the vast majority of non-performing workers, the redundant strategy may still be more cost-effective than hiring a single, highly reliable, highly expensive worker.

## 6 Conclusions and Future Work

This paper successfully demonstrated that multi-agent teams operating in a large-scale environment can improve their utility by using redundant groups of untrustworthy agents rather than using a single, most-reliable partner. In doing so, it showed that trust and reputation information about less reliable agents could be used just as effectively as information about reliable agents.

The paper first described a large-scale market environment where contractees and contractors operate according to pure self-interest to increase their utility. The paper then described a strategy for substituting teams of redundant, low reliability agents for individual, highly reliable partners, and finally demonstrated the superiority of such an approach via a multi-agent simulation. Results from this simulation demonstrated that agents utilizing redundant teams of low-reliability agents are both more profitable and more successful in completing tasks over time. Results also suggested that a more widespread use of redundant team formation could lead to a more efficient use of all agents available in a system, and not merely a subset

of highly reliable agents.

There are numerous ways in which this work could be extended. First, although the simple redundant strategy used here has been shown to be viable, it has not been shown to be optimal. Additional research should determine a mechanism to build an optimally inexpensive, highly reliable team of agents out of the agents available. Such research may also explore how effective redundant team formation is with different distributions of reliable agents (e.g. a stepped distribution of multiple groups of agents with identical reliability, rather than an even distribution of agents across the entire reliability space.)

Second, although this work shows how trust and reputation information can be utilized, it has not yet explored how such strategies would influence actual trust and reputation mechanisms. More specifically, many existing trust and reputation mechanisms cease to find out information about agents as it becomes clear that those agents are unreliable, instead focusing on exploiting and refining their knowledge of reliable agents. However, if low reliability agents may be used just as productively as high reliability agents, the exploration vs. exploitation decisions carried out by existing trust and reputation mechanisms may have to be modified to ensure that the reliability assessment of all agents is accurate.

**Acknowledgments.** This research is sponsored in part by the Defense Advanced Research Project Agency (DARPA) Taskable Agent Software Kit (TASK) program, F30602-00-2-0588. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

## References

1. Patel, J., Teacy, W. T. L., et al.: Agent-based virtual organizations for the Grid, Multiagent and Grid Systems, 1(4/2005), IOS Press, 237-249.
2. White, T. and Pagurek, B.: Towards Multi-Swarm Problem Solving in Networks. In Proceedings of the International Conference on Multi Agent Systems (Paris, France, July 3-7, 1998), 333-340.

3. Stein, S., Jennings, N., et al.: Flexible Provisioning of Service Workflows. In Proc. 17th European Conf. on AI (ECAI-06), Riva del Garda, Italy, pages 295-299. IOS Press 2006.
4. Fullam, K.: Learning Trust Decision Strategies in Emerging Reputation Networks. Doctoral Dissertation, Electrical and Computer Engineering, University of Texas at Austin, (2007).
5. Sabater, J. and Sierra, C.: Reputation and social network analysis in multi-agent systems. In Proceedings of the first International Joint Conference on Autonomous Agents and Multiagent Systems (Bologna, Italy, July 15-19, 2002), AAMAS 02, ACM Press, New York, USA, 475-482.
6. K. Barber, K. Fullam, and J. Kim.: Challenges for Trust, Fraud, and Deception Research in Multi-agent Systems. In *Trust, Reputation, and Security: Theories and Practice*, volume 2631 of LNCS, pages 8--14. Springer, 2003.
7. Jones, C. L. D., Fullam, K. K., et al.: Exploiting Untrustworthy Agents in Team Formation, In Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (CA, USA, Nov. 2-5, 2007) IAT 07, IEEE Computer Society Washington, DC, USA, 299-302.
8. Smith, R. G.: The contract net protocol, IEEE Transactions on Computers, C-29(12), 1104-1113. (1980).
9. Shehory, O. and Kraus, S.: Methods for task allocation via agent coalition formation, Artificial Intelligence. 101(1-2), Elsevier, 165-200. (1998).

# Towards a Transparent Middleware for Self-Organizing Multi-Agent Systems on Clusters

Paulo Motta, Maíra Athanázio de Cerqueira Gatti and Carlos José Pereira de Lucena

Departamento de Informática – PUC-Rio,
Rua Marques de São Vicente, 225, 4o andar RDC
Rio de Janeiro, RJ, Brazil
{pjunior, mgatti, lucena}@inf.puc-rio.br

**Abstract.** This paper presents the architecture to deliver parallel processing to a self-organizing multi-environment multi-agent system in order to achieve better performance. Parallelism is used as a tool for processing either larger instances of a problem or to deliver a faster solution to a given problem. The architecture chosen is both parallel and distributed; however, one of the main goals was to provide a transparent implementation that frees the developer to handle all the complexities associated with this kind of environment. The solution is to provide a set of libraries that work actively checking the agent requests against the distributed environment.

**Keywords:** Agents, Parallelism, Distribution, Simulation, Self-Organization.

## 1 Introduction

Like many systems today, agent environments are reaching their limits when it comes to performance. Agents are used in many interesting ways, and notably on systems that either do not have the complete description of the task to perform or systems that will evolve through time. Simulation systems can then benefit from these characteristics of agent-based software. In particular, autonomic distributed systems are designed with self-organizing mechanisms in order to achieve self-* properties [16] and before being deployed they are simulated to be validated.

On the other hand, self-organization is a dynamic and adaptive process where components of a system acquire and maintain information about their environment and neighbors without external control [16]. This makes it more difficult to distribute in a cluster since the environment is not centralized. The environment is locally observable to agents and if multiple distributed environments exist, an agent can only exist in one environment at a time. In self-organizing systems, the environment acts autonomously with adaptive behavior just like agents and interacts by means of reaction or through the propagation of events.

In [1] a multi-environment simulation framework is proposed that provides higher abstractions and components to support the development of self-organizing systems with multiple environments, which can be situated or not. In 2D or 3D situated environments, regardless of the visualization process, one requirement is the space management. In a sequential simulation this does not incur in a problem because each request is treated in the order of arrival. However, in a distributed parallel environment requests may arrive at any time, so it is necessary to provide a solution for concurrency.

To overcome this problem we are proposing a transparent distributed and parallel architecture for a middleware specifically targeted at performance for agents. Thus, our motivation is to provide the capabilities for the multi-environment simulation framework to become a distributed and parallel solution, in the sense that it could work on a cluster environment in order to achieve better processing times or larger problem sizes.

This paper is organized as follows: the next Section describes the problem through the case study being developed. Section 3 presents the proposed solution with the architecture description. Section 4 presents the architecture evaluation and some discussion. Section 5 presents the related work. And, finally, section 6 presents the conclusions and future work.

## 2 Stem Cell Simulation - The Problem Description

Our case study is a Stem Cell Simulation [3] that models the growth of stem cells. It is agent-based and runs over the self-organizing framework [1] which provides:

- Discrete time control for verification – all the processing is controlled with a global *step* that allows the framework to provide verification mechanisms [20].
- Visualization occurs during the processing – the self-organizing framework provides a 3D Continuous Space based on sparse fields [1] in addition to many graphical capabilities for visualization inherited from Mason [2].

In a discrete event simulation system, an entity is allowed to behave from time to time. These slices of time are called steps. Thus, a basic entity will usually implement the step method where it will perform its activities. The agent or environment has a set of events, i.e., the information provided for the underlining self-organization and implicit coordination, to handle on each time step of simulation. An agent can behave and execute actions over the environment where it resides or over itself.

The environment uses a sparse field-based grid [1] to realize the several strategies for self-organizing patterns, such as the atomic ones Replication, Death, Diffusion, and Aggregation [19], or combined ones, such as Gradient Fields and Pheromone Path [16], for instance.

At the stem cell computational model, an agent represents each cell and in so doing, the number of agents grows tremendously. One requirement regardless of the visualization process is of space management (be it 2D or 3D). Since Stem Cells are

multiplying during the process, it is necessary to manage the space that will be allocated to each cell. In a sequential version this does not incur in a problem since each request is treated in the order of arrival. However, in a distributed parallel environment requests may arrive at any time, so it is necessary to provide a solution for concurrency.

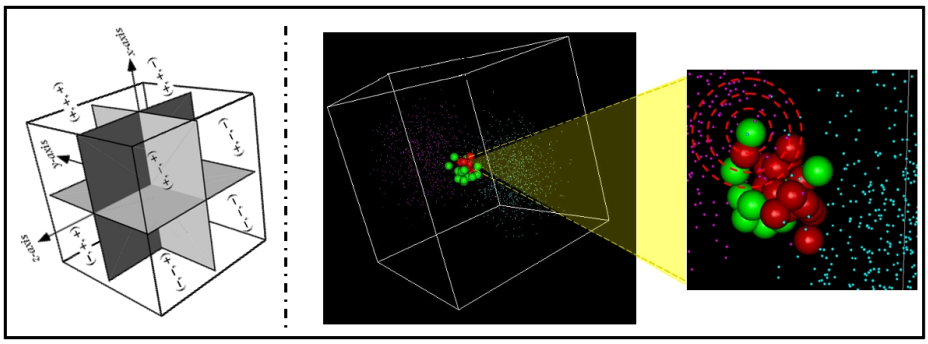

**Fig. 1.** The Stem Cell Problem Description

Furthermore, stem cell behavior is a self-organizing system. In self-organizing systems the agent interaction occurs through the environment (e.g. gradient fields, pheromones) [16],[1]. The real life spatial self-organization of stem cells is explained as follows. The stem cell divides itself into two news cells, due to the mitoses division process which can be induced by the environment (niche) or a reaction of a grown factor in the neighborhood (Figure 1, right side). Both cells assume new positions: one assumes the parent position and the other an adjacency position. Before the division process, the cell grows up, pushing its neighbors away and occupying the required space for the new cell. The differentiation division process happens in the same way, but creating a specialized cell. At the simulation startup the cells are in the center of the 3D continuous grid. More details about the 3D spatial stem cell self-organization can be found in [3].

The application is effective, however, due to its processing requirements it does not achieve the size problem desired. To overcome this problem we are proposing a distributed and parallel architecture specifically targeted at performance for agents. The goal for this architecture is to provide the user with: the solution in less time, or; a solution to a larger instance of the problem in the same time frame. Specifically for the case study a single machine implementation is capable of 10 thousand agents, with our implementation we are expecting to raise this number to 70 thousand agents in parallel.

However, it is important to pay special attention to the representation of the 3D space. Regarding this issue, there were two choices that would affect the cluster evolution:

i) if we represent the 3D space in a distributed manner, each computing node would be responsible for an octant of the space (Figure 1, left side). This could

improve performance when communication between two agents at the same octant takes place, but the growth would be eight machines each time, one for each octant.

ii) the other approach would be to have the 3D space virtually represented in a centralized way, at first on a single machine, this way we can add more processing nodes easily and one at a time. Although this solution is limited by the network bandwidth and the central node throughput, it was the approach that was selected and is described in the next section.

## 3 The Proposed Solution

Our solution is based on a cluster environment that uses processing nodes with the same hardware and software configuration and fully dedicated to the solution processing. This is better suited when trying to achieve improved parallel performance for a great number of agents that need to share or compete for some resource. Over a cluster, we have a software architecture that provides network transparency to the user.

We use an architecture based on classical solutions of parallelism to provide the user a better alternative for solving problems faster or larger problems. The solution rely on a master node that control the processing nodes and its goal is to deliver parallelism over distribution being cost effective since it allows for the growth of processing nodes, limited only by the master node processing power.

### 3.1 Technology Choices

Since the framework is Java based, the solution for distribution is RMI, which is provided with the standard Java Development Kit. This is the same approach used for the WebHLA by DoD[4] and is the best solution while we still have other aspects to optimize; only when all of the alternatives have been explored should we try to change the standard libraries for improved specific solutions.

For concurrency we chose to model the Sequential Object Monitors [5], which are special active objects that can control a given resource by providing a request queue to the clients and that use a processing thread that consumes the queue in the arriving order. This allows clients to work asynchronously with a non-blocking request call and at the same time provides a means to control requests, so that when a request gets served it is guaranteed to receive a unique 3D space location, for instance.

### 3.2 Distributed and Parallel Architecture

The proposed solution creates an overlay network of brokers interconnected through a communication bus that is provided by the central node, which is responsible for step control and other services.

The goal is to provide a virtual environment that works exactly as the local version, at least for the application programmer. The framework's infrastructure is responsible for connecting remote nodes that should have their broker elements initialized at startup.

Figure 1 shows the architecture proposed for distribution and parallelism. As we can see, the implemented pattern is a Master/Worker [8], where the master node is responsible for control activities and the worker nodes compute the simulation tasks. The network bandwidth may become a bottleneck for this solution; however, a hierarchy of central nodes may be provided if necessary.

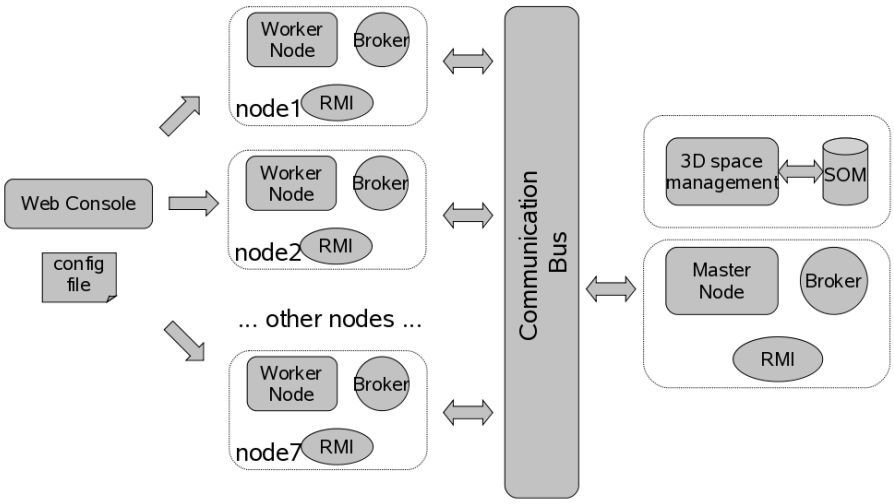


**Fig. 2.** Distribution and Parallelism Architecture

A key concept here is that we use distribution as a mean to parallelism, since it is more cost effective to have more machines than a single multiprocessor. However, we have some communication and coordination cost between nodes.

### 3.2.1 The Remote Steppable Interface

The framework offers the *IAgent* and the *IEnvironment* interfaces [1]. They are steppable interfaces which are seen by the simulation as entities that may execute a simulation step and being so, they reside on a list of objects to receive a step notification. When providing a distributed environment we needed to create a similar interface that would allow the master node to communicate with remote nodes and notify them to execute a new step.

However, due to Java technology limitations we needed to implement a local adapter on the master node that represents the remote node. That was necessary

because the remote interface must throw a *RemoteException* that is not supported by the original version.

### 3.2.2 3D Space Management as a Service

Many types of simulation will need not only 2D or 3D space management, but also many other types of supporting services, such as diffusion algorithms, force calculations and unique Id generation, the latter already being used and provided on our framework. Seeing these as services becomes much easier not only to provide them to the application through the framework, but also to evolve them in an orthogonal fashion.

Using a Service Locator [6] Design Pattern the application code simply has to make a call to find the service needed. Implementing the correct interface, framework developers may deploy as many services as needed and may provide new abstraction levels to the application programmers.

We chose a centralized representation of the 3D space to be used by the stem cells. That way we may have many working nodes processing the simulation, although we will end up with many requests for 3D locations. To avoid concurrency problems we use an implementation of a Sequential Object Monitor [5], to hold a queue of agents' requests. Furthermore, exposing this facility as a service makes it easier for agents to use it as if it was a local resource.

### 3.2.3 Infrastructure Transparency

One major design goal is to provide the communication means to application programmers in a way that it is not necessary to know where an agent resides nor even that there is distribution, besides the start up process.

That said, all communication occurs through each node's broker. If the target agent resides on the same machine, the local broker is allowed to route the message on its own. On the other hand, if it is a remote target, the local broker routes the message to the master broker through the communication bus. The central broker is then responsible for finding the target location on its agents table and delivers the message to the target agent's local broker, which in turn will deliver it to the agent.

### 3.2.4 The Simulation Engine

The *MasterState* class, shown in figure 3, represents the concrete and complete simulation. Once the simulation class is ready, we use the infrastructure to deploy it to the multiple remote worker nodes. To this end, each worker node's *Broker* must be instantiated at startup and wait for the *Central Broker* to start and connect to each worker node. The worker nodes must be configured at the central node that will

connect to them in order to establish a star-like overlay network through which all the communication will take place.

Every time an agent or environment is created it must register at the *Central Broker* to receive a unique id. The *Central Broker* can then register this at a local table in order to keep the physical location of every entity created. The *Remote Broker* for the requester will also keep a local table of agents and environments that it keeps in order to enhance communication performance. Figure 3 illustrates the sequence diagram for this process.

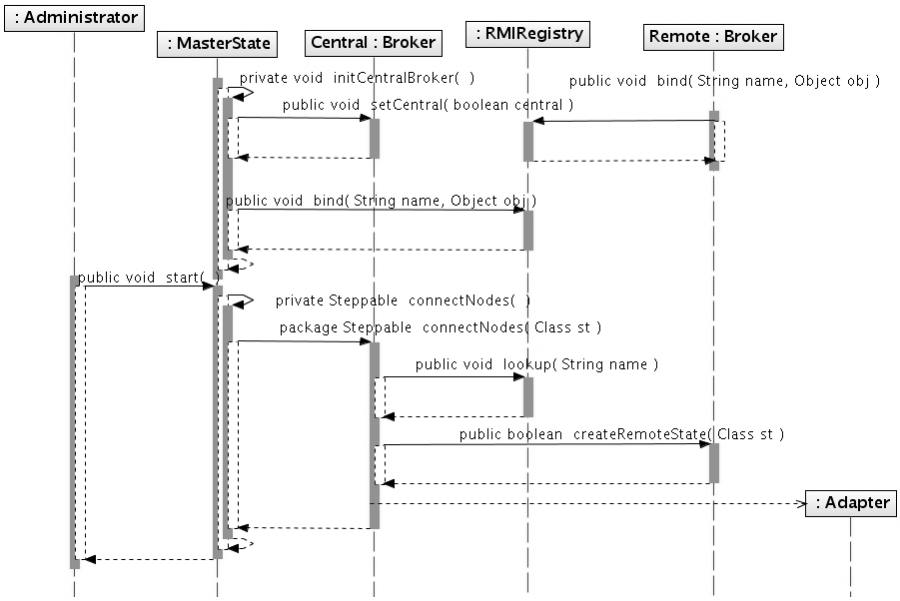


**Fig. 3.** The Master node sends the Simulation class to the Worker nodes

The *Central Broker* and the *Remote Broker* must register at Java's RMI register (*RMIRegistry*) service in order to establish the distributed Java environment for the application to start.

### 3.2.5 The Communication Bus

Since the proposed architecture uses many processing nodes it would be a major issue if each node could be connected to its neighbors in order to communicate. Since we can grow the cluster with nodes, this would saturate the network with packages and it would degrade the application's performance.

To overcome this problem, we provide a communication bus based on the central broker, which is capable of routing any message to any agent. Thus it is possible to find any entity with fewer steps although communication may become a bottleneck if the cluster grows beyond a certain point.

### 3.3 Virtual Space – Distributed Parallel

Creating a virtual space gives the programmer the feeling that she is working on a single centralized machine, and all the distribution and parallelism is controlled and handled by the middleware. It is necessary, however, to configure the working nodes to bring up the broker element and also to configure the nodes' addresses on the central broker.

Figure 4 below shows the layers that are part of the proposed solution. Using brokers to create an overlay network allows the construction of a distributed agent environment with centralized step control. Over the distributed agent environment, agents are able to communicate the same way they do on a single machine using the event diffusion mechanisms, according to each self-organizing pattern and its strategies.

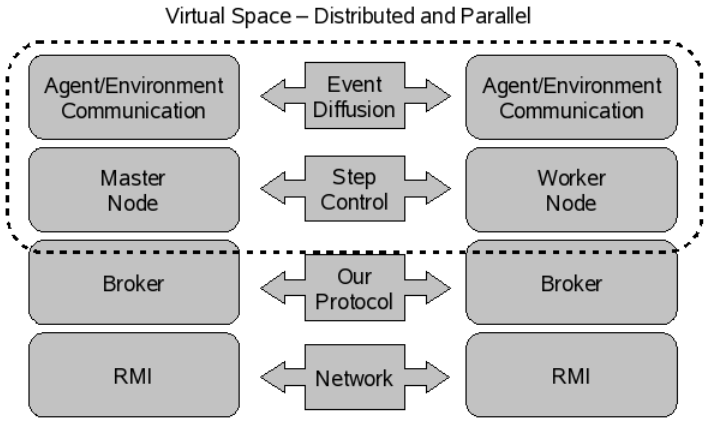


**Fig. 4.** Virtual Environment provides the user with a perception of a single machine

Furthermore, having the framework divided in layers allows us to evolve each part independently; it is possible to optimize each component and have the measure of its benefits available to the whole architecture.

## 4 Discussion

Currently, we have a working prototype that is able to connect to a certain set of fixed nodes. However the simulation is started independently on each node. In order to coordinate the remote nodes the 3D space must be fully operational.

The use of a centralized step control allows us to coordinate the execution on the multiple nodes, and to this end the simulation is not started on each node through the mechanisms provided by the original self-organizing framework engine. The scheduler for each machine is disabled and each machine is represented at the master

node through an adapter that implements the *Steppable* interface becoming a simple bridge to reach remote nodes.

Another interesting issue is that, on the master node, the adapters are held in a *ParallelSequence* structure; this means that they receive their steps in parallel, each adapter on its own thread. This allows for the cluster to work with all nodes at the same time. The first, naïve, implementation used a different data structure, which led to a side effect of a sequential processing cluster, i.e., each node executed one at a time according to its adapter order on the master node.

## 5 Related Work

Similar solutions have been proposed to allow distribution on agent environments, namely Jade [7] and Octopus [11], which use the same technology to achieve the connectivity necessary to communicate between processing nodes. Both solutions offer a general solution that allows nodes to interact, the communication is handled by the user and, to achieve better performance, the programmer must implement optimizations on her own. Moreover, Jade, has a complete infrastructure for message passing and domain name service to find and re-route messages to agents. It has neither an infrastructure for discrete event simulation nor for self-organizing patterns.

The FLAME framework [17] has been used in a similar case study presented in this work. It uses the MPI (Message Passing Interface) [18] framework for C. The agents are specified using the X-Machines formal method and are translated into C, which is then adhered to the MPI interface. The communication synchronization points ensure that concurrently executing agents remain in sync with each other.

Farm [21] is an environment that allows programmers to create their own simulations and that provides means for distribution and parallelism similar to what we intend. However the meta-agents would become a problem since our goal is to achieve a great number of agents, in order of 70 thousand. Furthermore, Farm has its own infrastructure based on Java, our solution provides parallelism for an already evolved environment based on Mason. Finally, Farm uses a real-time approach, while we are using a discrete-time step-controlled simulation approach.

Mason [2] offers a mechanism to allow parallel execution on a single machine based on threads. For the use of *ParallelSequence*, a data structure provided by Mason's library, the programmer fires many threads, each one handling a *Steppable* entity. The first change that a programmer should implement in her application is to use this resource in order to allow for better performance. A problem arises when we mix up *ParallelSequence* with other types of entity holders available, because the *ParallelSequence* behaves in a parallel fashion only within its borders. Hence there is no way to distribute it over a network, since the framework is a target on a single machine.

The major difference is that we seek to improve performance by the means of distribution and parallelism, and so the middleware is adapted to execute different elements in parallel on many machines. The idea is to hide this kind of detail from the

programmer, which in turn allows greater freedom to focus on the modeling problem; once the application is complete, the programmer may configure how many agents must be set on each working node and, based on the coordination of the 3D space, position the application flows in parallel.

## 6 Conclusions and Future work

We have presented an architecture to deliver parallel processing to a self-organizing multi-environment multi-agent system in order to achieve better performance. To achieve that we used means of distribution over a cluster using broker elements to create an overlay network seeking the creation of a virtual space over which the programmer has a single-machine perception.

The main contribution of this solution is to provide transparency for the complexities regarding distributed and parallel processing. The use of parallelism is targeted specifically at performance improvement, whether the processing of larger instances of the given problem at the same processing time or faster processing times for the given instance size.

The solution is layered, which allows independent evolution of each layer. This capability is crucial for the benchmark of each optimization contribution for the performance improvement. There are some optimizations specifically for the communication, as follows: 1) changing from the standard RMI library to a specialized communication infrastructure; and 2) using bulk communication packs between remote nodes and the master node. Since each communication will respect the step control, we can send a set of requests for 3D space locations on a single network package and this, in turn, will improve the network usage and traffic.

Other improvements may be provided as different services like different interaction algorithms.

As a means of monitoring we plan to provide a web console that will display the state of each processing node in terms of its health regarding the distribution infrastructure; for agent behavior the user should refer to the framework agent features. It is important for the application administrator to be able to check what is going on with the machines since simulations may require long processing times, depending on the problem instance size.

**Acknowledgments.** This work was supported by MCT/CNPq through the "*Grandes Desafios da Computação no Brasil: 2006-2016*" (Main Computation Challenges in Brazil: 2006-2016) Project (Proc. CNPq 550865/2007-1).

## References


1. Gatti, M.A. de C., Lucena, C.J.P.; A Multi-Environment Multi-Agent Simulation Framework for Self-Organizing Systems. In the Workshop MABS at AAMAS'09, May 2009.
2. S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan. MASON: A New Multi-Agent Simulation Toolkit. Proceedings of the Eighth Annual Swarm Users/Researchers Conference (SwarmFest 2004).
3. Faustino, G.M. ; Gatti, M. A. C. ; Bispo, D. ; Lucena, C.J.P. de . A 3D Multi-Scale Agent-based Stem Cell Self-Organization. In: SEAS 2008 - Fourth Workshop on Software Engineering for Agent-Oriented Systems, 2008, Capinas. XXV SBES, 2008.
4. Dongarra J Et al, SourceBook of Parallel Computing, Morgan Kaufmann, ISBN 1-55860-871-0
5. Caromel D., Mateu L., Tanter E. Sequential Object Monitors, In Proceedings of the 18th European Conference on Object-Oriented Programming (ECOOP 2004), number 3086 in Lecture Notes in Computer Science, Springer-Verlag, 2004, pages 316-340.
6. Alur D., Crupi J., Malks D. Core J2EE Patterns: Best Practices and Design Strategies, 2nd Edition, Prentice Hall / Sun Microsystems Press, June, 2003, ISBN:0131422464
7. Bellifemine, F., Poggi, A., Rimassa, G.: Jade, A FIPA-compliant Agent Framework. Proceedings of PAAM'99, London, UK (1999)
8. Mattson, T.G.; Sanders B.A. And Massingill B.L.; Patterns for Parallel Programming, Addison Wesley, ISBN 0321228111
9. Quinn M.J., Parallel Computing Theory and Practice, McGraw-Hill, ISBN 0-07-051294-9
10. Zomaya, A.Y., Parallel Computing Paradigms and Applications, Thomson Computer Press, ISBN 1-85032-188-4
11. Uhruski P., Grochowski M., Schaefer R., Octopus - Computation Agents Environment, Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial, ISSN 1137-3601, Nº. 28, 2005 (Ejemplar dedicado a: Nuevas tendencias en Sistemas Multiagente y Soft Computing), pags. 55-62
12. Ishida T., Parallel, Distributed And Multi-Agent Production Systems, Lecture Notes In Computer Science, Vol. 878, 1994, Xvii, 166 P.
13. Muthukrishnan C., Suresh T.B, A Multi-Agent Approach To Distributed Computing, Workshop Agent Based High Performance Computing, 1999
14. Brauer W., Weiß G., Multi-Machine Scheduling - A Multi-Agent Learning Approach, ICMAS Proceedings of the 3rd International Conference on Multi Agent Systems, IEEE Computer Society Washington, DC, USA, 1998, p.42, ISBN:0-8186-8500-X
15. Grochowski M., Tuska E., Uhruski P., Influence of Inter-Agent Communication Cost to Diffusion Scheduling in Irregular Parallel Computations, Revista Iberoamericana de Inteligencia Artificial, Nº. 28, 2005, pp.93-100, ISSN 1137-3601
16. De Wolf, T.; Analysing and engineering self-organising emergent applications, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May, 2007, 183.
17. Mariam K., Coakley, S., Walkinshaw, N., McMinn, P., Holcombe, M.: Validation and discovery from computational biology models. Biosystems 93(1-2): 141-150 (2008).
18. Foster, Ian. Designing and Building Parallel Programs (Online) Addison-Wesley ISBN 0201575949, chapter 8 Message Passing Interface, 1995.
19. Gardelli, L., Viroli, M., Omicini; A.; Design Patterns for Self-Organizing Multiagent Systems. 2nd Int. Workshop on Eng. Emergence in Decentralised Autonomic Systems (EEDAS 2007). To be held at the 4th IEEE Int. Conf. on Autonomic Computing (ICAC 2007). June 11th, 2007, Jacksonville, Florida, USA.

20. Soares, B.C.B.A., Gatti, M.A.C., Lucena, C.J.P; “Towards Verifying and Optimizing Self-Organizing Systems through an Autonomic Convergence Method,” The Workshop SEAS 2008, Campinas, SP, Brazil.
21. Horling, B., Mailler R., Lesser,V., “Farm: A Scalable Environment for Multi-Agent Development and Evaluation,” In Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003), Oregon, 171-177, 2003

# Resource Management and Adaptive Replication for Fault-Tolerant MAS

Sylvain Ductor[1], Zahia Guessoum[12], and Mikal Ziane[13]

[1] LIP6 - Université Pierre et Marie Curie (Paris 6)
104 avenue du President Kennedy 75016 Paris, France
[2] MODECO-CReSTIC - IUT de Reims, 51687 Reims Cedex 2, France
[3] Université Paris Descartes, Paris, France

**Abstract.** Distributed cooperative applications are now increasingly being designed as MAS. Such applications may be massive, open and very dynamic: new agents can join or leave, they can change roles, strategies, etc. This characteristics create new challenges to the traditional approaches of fault-tolerance. In this paper, we focus on a replication-based preventive approach. The aim is to dynamically and automatically adapt the agent replication strategy (e.g. number of replicas and their location), in order to maximize the MAS reliability. We describe a negotiation protocol, supporting adaptive replication. This protocol provides a distributed solution that uses local decision but guarantees global MAS performances. We report on experimental results using a first implementation of the protocol.

## 1 Introduction

Multi-Agent Systems (MAS) have generated lots of excitement in recent years because of their promise as a new paradigm for conceptualizing, designing, and implementing software systems, ranging from manufacturing to process control, air traffic control, and information management. They are particularly attractive for creating software that operates in distributed and open environments, such as the Internet. Being both decentralized and self-organized, these systems consist of autonomous entities called agents that are designed to solve tasks by cooperating with each other. These systems thus suffer from all the problems associated with building traditional distributed systems as well as the additional difficulties that arise from having flexible and sophisticated interactions between autonomous and adaptive components. It means that MAS are non-deterministic, and a specific behavior is hard to guarantee, especially in fault situations. Since some changes are unpredictable, there exists no generic way of describing the global state of the MAS. A fault-tolerant infrastructure that could detect and adapt to these failures to provide continuity of processing is thus crucial to MAS.

To build a fault-tolerant infrastructure, several projects have used replication mechanisms [7], [13]. Replication of data and/or computation is an effective way to achieve fault-tolerance in distributed systems. However, replicating every

agent on each host is not feasible in general because the performances of the MAS would be affected. Several approaches have thus been proposed to identify what critical agents should be made robust and how to configure their replication [7]. Nevertheless, the replication is decided statically before the application starts. Since new cooperative applications are much more dynamic and large scale, it becomes very difficult, or even impossible, to identify in advance the most critical software components. Other approaches have thus been proposed to let the MAS dynamically identify the most critical agents [13] [11]. These approaches are based on two levels of information: system-level information, like communication load, and application-level/agent-level information, like roles, interdependences or plans. However, to the best of our knowledge, no solution has been proposed to deal with the problem of dynamically defining the most suitable replication configuration to replicate agents of a large-scale MAS.

In this paper, we introduce a negotiation protocol and an operational framework to manage these resources and improve the availability of a large-scale and open MAS. This protocol can be tuned to have limited impact on the perfomances of the MAS. This paper is organized as follows. Section 2 deals with related work. Section 3 introduces the problem and principles of our approach. The architecture of the framework is described in Section 4 while the the negotiation protocol is described in Section 5. Finally, Section 6 reports on the experiments.

## 2 Related Work

There are two major approaches to fault-tolerance in MAS: diagnostics and redundancy. Diagnostics tries to identify the faulty elements in MAS [14][15]. This is done by deduction, based on information on the form of results of tests applied to the elements. Once the faulty elements have been identified, the MAS is able to isolate them, to ignore their output and to initiate a repair operation such that the reliability of the MAS can be maintained in the long run.

Redundancy tolerates faults of the components in the system [4][7]. It aims at preventing failures by creating two or more representation of an element on differents nodes. A special mechanism ensures the consistency of the representations [18].

The design of a fault-tolerant MAS by redundancy must take into account the cost of the redundant elements: not only in the initial cost of the redundant components of the system, but also in the support needed to use these components or simply to have them in the system. Furthermore, the redundant components may not necessarily benefit the system when the faults that they were designed to tolerate do not occur. The fault-tolerance research community has developed preventive solutions (algorithms and architectures), mostly based on the concept of replication. Replication of data and/or computation has been proved to be an effective way to achieve fault-tolerance in distributed systems. A replicated software component is defined as a software component that pos-

sesses a representation on two or more hosts [8]. Many toolkits (e.g., [8], [20], [7]) include replication facilities to build reliable applications.

Several approaches have been proposed to study the resource allocation problem (see for instance [16],[2],[22]). These approaches present useful solutions to replicate multi-agent systems. Kraus et al. [16] propose a solution to deciding the allocation of extra resources (replicas) for agents. They proceed by reformulating the problem in two successive operational research problems (knapsack and then bin packing) statically before the application starts. Their approach and results are very interesting. However most of those approaches are centralized and executed statically before the application starts. The acquired information is typically used off-line to explain and to improve the MAS robustness. Also, the considered application domains typicaly only involve a small number of components with non adaptive behavior. These centralized mechanisms are not suited to large-scale and complex MAS where resource allocation needs to be analyzed in real-time to adapt the multi-agent system to the evolution of its environment.

A lot of research has addressed resource allocation problems and has provided results to resource management (see for instance [6],[2] [5],[3]). However, those contributions are theorical and based on several hypotheses to simplify resource allocation (e.g. indivisibility of resources, specific preference structures, ...). Applying these theories to a real application such as replication management thus remains a challenge.

## 3 Problem Statement and Global Objectives

A replication-based fault-tolerant MAS consists of a set of agents, $\mathcal{A}gents$, that run and can be replicated on a set of hosts, $\mathcal{H}osts$. In this work, we focus on host failures. When a host fails, all the hosted replicas are lost. Each host $h$ is thus associated with a failure probability, $P_h(t)$. Agent $a$ fails, if and only if, all its replicas are lost

The availability of Agent $a$, $\mathcal{D}_a(t)$, is its non-failure probability. In case of independent failures:

$$\mathcal{D}_a(t) = 1 - \prod_{h \in \mathcal{R}eplicats_a} P_h(t)$$

where $\mathcal{R}eplicats_a$ is the set of hosts where $a$ is replicated. Note that an agent does not need more than one replica on a given host.

The first purpose of the safety engineering mechanism is to minimize the failure probability of each agent. However, since some agents may have more important roles than others, their availability is more important to the system. Informally, the criticality of an agent, $\mathcal{W}_a(t)$, is a measure of the impact of its failure on the MAS. [12] and [10] introduced metrics to compute agent criticality.

Our proposal aims at efficiently allocating agent replicas. An allocation $\mathcal{A}$ is modeled by a bipartite graph

$$\mathcal{A} = (\mathcal{A}gents, \mathcal{H}osts, \mathcal{R})$$

where each vertex $(a, h) \in \mathcal{R}$ represents the replica of Agent $a$ on Host $h$. Agents are characterized by the reliabilities provided by their replicas and hosts by their current loads. Resource allocation is a dynamic process that initializes and updates this graph: it decides which agent to replicate, how many replicas each agent is granted and where to locate them.

The next two subsections present the selected criteria to evaluate the two main parameters (host load and agent reliability). We then present the global evaluation of MAS reliability which defines the optimality objective. Section 5 presents the proposed protocol which uses local preferences to optimize MAS reliabiliy while controling the impact on the system performances.

### 3.1 Host Load

The host load depends on the local resources used by the agents. In this work, we focus on two important resources: the processor and the volatile memory.

Let us note $Proc_h(t)$ and $Mem_h(t)$ respectively, the processor use rate and the memory use rate for Host $h$. The host performances depend on both these parameters; any overload causes their degradation.

**Definition 1 (Host Load $\mathcal{C}_h(t)$).**

$$\mathcal{C}_h(t) = \max(Mem_h(t), Proc_h(t))$$

Note that the replication service evaluates the load of *all* the processes running on the host.

Our first objective is to avoid replications that overload hosts. Let $\alpha_{user}$ be a parameter provided by the user to control the maximal load that can result from the replication service; in case of heterogeneous hosts, $\alpha_{user}$ may vary from one host to another.

**Definition 2 (Host overload).**

$$h \textit{ is overloaded} \Leftrightarrow \mathcal{C}_h(t) > \alpha_{user}$$

Meanwhile, we want to optimize the distribution of resources. Indeed, some hosts, due to their characteristics may be more attractive than others. It may then happen that the resources of the attrictive hosts are overly consumed while the resources of other hosts are not used. Our purpose here is thus to allocate host resources as uniformly as possible.

### 3.2 Agent Reliability

We informally defined the reliability of Agent $a$, $\mathcal{F}_a$, as an aggregation of its criticality and its availability. Informally, $\mathcal{F}_a$ defines a target availability (e.g. more critical agents reach higher availabilities) and evaluates how close $a$ is to this goal. Besides, each agent, even a not critical one, may have a role that is not negligible in the application. It is therefore safer to keep every agent available,

as much as possible. Reliability computation must thus permit every agent to reach at least a minimum level of availability.

In order to evaluate the reliability of an agent on an allocation, fuzzy logic provides a good decision-making process [21]. Each agent associates an estimated reliability status (*not reliable*, *reliable* and *too reliable*) to each allocation. The *reliable* status is the goal of agents; both the *not reliable* and the *too reliable* status are undesired. Concerning the latter, the aversion for a massive creation of replicas implies that agents are cooperative : they will save resources that they do not really need to the benefit of other agents and system performances.

The criticality, the availability and the reliability values are represented as fuzzy granulations. Criticality variation is described by three symbolic values (*rather critical, critical, very critical*), availability variation by four (*not available, rather available, available, very available*) and reliability variation by three (*not reliable, reliable, too reliable*). We analyzed these various classes to define the decision rules. Table 1 gives the adopted rules; each case contains the conclusion of the rule which premisses are the names of the column and the row (e.g. Agent a is not available and rather critical implies that it is not reliable). These

**Table 1.** Rules for defining reliability

| $D_a$ \ $W_a$ | Rather critic. | Critic. | Very critic. |
|---|---|---|---|
| Not avail. | Not rel. | Not rel. | Not rel. |
| Rather avail. | **Rel.** | Not rel. | Not rel. |
| Avail. | Too rel. | **Rel.** | Not rel. |
| Very avail. | Too rel. | Too rel. | **Rel.** |

rules are compliant with our proposal. Firstly, a lowly available agent is considered as *not reliable*, whatever its criticality is. Secondly, consider the diagonal drawn by the *reliable* state: An agent "less available than critical" is considered as *not reliable* and tries thus to replicate. Whereas a lowly critical agent never tries to reach a highly available state.

### 3.3 MAS Reliability

In case of limited resources, massively replicating the agents is not possible. Our proposal is to find an allocation that maximises the reliability of all the agents. Indeed, the reliability of an agent defines its availability objective which relies on the necessity of replicating this agent. We want thus to converge toward allocations where each agent is the closest to its objective. To characterize such an allocation at the MAS level, we use an approach from collective decision theory [1] [19]: each agent is endowed with a preference order over allocations. The best allocations are defined by an aggregator over the agent preferences. We describe here the properties of the chosen aggregator.

The best allocations belong to those which respect the "Pareto optimality" principle [1]. This unanimity principle states that if all the agents prefer an altenative $x$ to an alternative $y$, $x$ should be chosen. Thus, an allocation is Pareto-Optimal if there exists no other allocation which is better for every agent.

**Definition 3 (Pareto optimality).** *Let $\mathcal{F}_a(\mathcal{A})$ the reliability of Agent $a$ in Allocation $\mathcal{A}$. $\mathcal{A}$ is Pareto-Optimal if and only if:*

$$\nexists \mathcal{A}', \begin{cases} \forall a \in \mathcal{A}gents, \mathcal{F}_a(\mathcal{A}') \geq \mathcal{F}_a(\mathcal{A}), \\ \exists a \in \mathcal{A}gents, \mathcal{F}_a(\mathcal{A}') > \mathcal{F}_a(\mathcal{A}) \end{cases}$$

Also, agents take into account their criticalities when defining their targeted availabilities. The aggregator must thus characterize allocations where each agent is close to its objective. Such an allocation minimizes the differences among agents reliabilities. The leximin-order criterion[1] achieves Pareto-optimality and minimizes inequities. It consists in selecting allocations where the reliabilities of the less reliable agents are highest.

**Definition 4 (Leximin order).** *Let $\vec{\mathcal{F}}(\mathcal{A})$ be the vector of sorted agents' reliabilities for Allocation $\mathcal{A}$. $\mathcal{A}$ is prefered over $\mathcal{A}'$ if and only if $\vec{\mathcal{F}}(\mathcal{A})$ is lexicographically superior to $\vec{\mathcal{F}}(\mathcal{A}')$,* i.e.*:*

$$\mathcal{A} \succ^F \mathcal{A}' \Leftrightarrow \exists m > 0, \forall i < m, \begin{cases} \vec{\mathcal{F}}_i(\mathcal{A}) = \vec{\mathcal{F}}_i(\mathcal{A}') \\ \vec{\mathcal{F}}_m(\mathcal{A}) > \vec{\mathcal{F}}_m(\mathcal{A}') \end{cases}$$

The sorted vector of agents reliabilities is associated with each allocation and defines the MAS reliability. Moreover, the use of a defuzzificator (e.g. gravity center) allows to compute agents reliabilities as numeric values. These values are used to evaluate more precisely the reliability of an agent. The leximin order is used in the negotiation protocol (Section 5) to characterize optimal allocations for the reliability criteria.

## 4 Overview of the Monitoring Architecture

DIMA [9] and DarX [18] have recently been integrated to build a flexible dynamic fault-tolerant multi-agent framework (named DimaX). DimaX provides MAS with several services such as distribution, replication, and naming [18]. Moreover, to dynamically control the replication, a new monitoring multi-agent architecture has been introduced [13]. One of the prime motivations behind the proposed architecture is to improve the robustness of large-scale open and distributed MAS in dynamic environments and with limited resources. Monitoring thus consists in acquiring necessary information to dynamically and automatically apply replication to agents when it becomes necessary. This information may be based on standard measurements (communication load, processing time...) or multi-agent characteristics such as the roles of agents or their interdependences.

This monitoring architecture distributes the monitoring mechanism to improve its efficiency and robustness. This distributed monitoring relies on a reactive-agent organization with several roles:

- Observe the agents and their interactions,
- Build global information,
- Update local information,
- Define the agent criticality,
- Use the agent criticality to replicate it according to the available resources.

These roles are assigned to two kinds of agents: *agent monitors* and *host monitors*. An agent monitor is associated to each agent of the application (named domain-agent) and a host monitor is associated to each host (see Fig. 1). The

**Fig. 1.** Multi-agent architecture

monitoring agents are hierarchically organized. Each agent monitor communicates only with its local host monitor. This reduces replication load. Host monitors exchange their local information to build global information (global number of messages, global exchanged quantity of information...).

After each interval of time $\Delta t$, each agent monitor checks if its domain-agent is still reliable. If not, it obtains information about the different hosts of the network and asks its host-monitor to negotiate creation of new replicas. The protocol used to negotiate the replication of agents is described at Section 5. Algorithm 1 describes the behavior of agent monitors and Algorithm 2 that of host monitors.

## 5 Negotiation Protocol

In this section we introduce a negotiation protocol which uses agent interactions to reach the global criteria (see Section 3): (1) optimize the MAS reliability provided by the leximin ordering, and (2) consume resources as uniformly as possible. Moreover, it avoids the creation of any replica that would overload a host. The negotiation protocol has four steps (see Fig. 2). It mainly involves two

**Algorithm 1** Agent-Monitor Behavior

**Require:** Associated host-monitor $hm$, associated domain-agent $da$
**Ensure:** Maintain the domain-agent in a reliable state
1: Update the local data.
2: Compute the criticality of $da$.
3: **if** $da$ is unreliable **then**
4: Establish replication preferences.
5: Delegate the negotiation to $hm$.
6: **end if**
7: Inform $hm$ of important local changes.

**Algorithm 2** Host-Monitor Behavior

**Require:** host-monitors' list $hml$, local agent-monitors $lam$
**Ensure:** Update global information and negotiate the replication
1: Update MAS statistics by aggregation of $hml$ information.
2: Send to relevant $lam$ the required information.
3: Negotiate with the other host-monitors the replication of unreliable local domain-agents.
4: Send to $hml$ the observed parameters which have significantly changed.

kinds of agents: the agent monitors and the host monitors. Hosts are also in charge of negociating for the agents. However, for the sake of simplicity, we describe the protocol as if the agents were negotiating with the hosts. The following algorithm is executed by the agents every $\Delta t$:

1. **Apply** A non reliable agent selects all hosts within a certain latency and applies to those where it has not already applied or been replicated. It then sends a replication request to these selected hosts.
2. **Accept/Waiting List** After the application time (which allows every agent to apply), each host executes Algorithm 3 to update its *accepted list* and its *waiting list*. This algorithm is also executed several times during the protocol. It sends the accepted agents an accept message and informs the other agents that they are on a waiting list. Agents may answer until the end of the protocol.
3. **Confirm/Cancel** After a given timeout, Agent $a$ has received responses from the hosts (Accept/Waiting List). It selects the best allocations according to its preferences and sends a cancel to hosts that do not match them. If one of these allocations allows to reach a reliable state, it sends confirmations to the selected hosts and cancels the others. Otherwise, it waits for other potential accept messages from more interesting hosts that have put it on waiting list. At the end of the protocol, it confirms the best allocation and cancels to other hosts. Note that when an agent sends a confirm to a host, it is automatically replicated.

In the following sections, we define the decision processes adopted by the agents and the hosts. These local decision processes converge towards the objectives (see Section 3).

**Fig. 2.** Our Negotiation Protocol

### 5.1 Hosts Selection Process

In the second step of the protocol, hosts select the agents to replicate. Their first objective is to avoid any replication that would overload them. Hosts verify thus that they can replicate simultaneously the agents of the *accepted list* without being overloaded.

The selected acceptations make the MAS converging towards an optimal allocation for leximin order. For that purpose, hosts always accept all the less reliable applicants that can be replicated together without overloading them. *Accepted list* thus represents the locally optimal allocation for the leximin order. Besides, hosts keep track of the other applicants in a *waiting list*. Indeed, since some agents (in particular the less reliable ones) may receive too many acceptations they will cancel from non-prefered hosts. Resources will thus be freed and possibly reallocated to agents on *waiting list*. Note that a host cannot remove an agent, which has not cancelled, from the *accepted list*.

**Algorithm 3** Host's selection process (step 2)

**Require:** The received *requestList* and cancels
**Ensure:** Update the *acceptedList* and *waitingList*
1: Remove the canceled agents from the *acceptedList* and the *waitingList*
2: Consider the agents of the *acceptedList* as replicated (*i.e.* the resources they need are reserved)
3: **if** the host is not overloaded **then**
4: Sort the agents of both the *requestList* and the *waitingList* by reliability increasing order into an *applicantList*.
5: Determine the first agent $a$ of the *applicantList* so that replicating all agents before $a$ (including $a$) will overload the host.
6: Add all the agent strictly before $a$ on the *acceptedList* and the others on the *waitingList*.
7: **end if**

The validation of this approach is based on two theorical results:

1. Leximin is separable *i.e.* "A decision that concerns a subset of agents can be based on the utilities of those agents only" [19]. The decision mechanism may thus be distributed without loosing optimality.
2. Endriss and Al. [5] give some interesting results about convergence towards the leximin criterion with negotiation. A deal is a possible transition from an allocation to another. "Any sequence of equitable deals [4] will eventually results in an allocation of resources with maximal egalitarian social welfare."

Those results show that our protocol converges. However, further work is needed to validate the effective reach of an optimum.

### 5.2 Agents' Preferences

To select the best allocations, agents are endowed with a preference ordering. Agents' preferences take into account both agent reliability and load balancing.

The reliability preferences order over allocations, characterizes the allocations where agents are close to their objectives.

**Definition 5 (Agents reliability preferences $\succ_a^{\mathcal{F}}$).**

$$\mathcal{A} \succ_a^{\mathcal{F}} \mathcal{A}' \Leftrightarrow \begin{cases} \mathcal{F}_a(\mathcal{A}) \neq \textit{too reliable} \\ \mathcal{F}_a(\mathcal{A}) > \mathcal{F}_a(\mathcal{A}') \end{cases}$$

Agents also maintain a preference relation over load inequities among host. There exist several ways in the literature to evaluate inequity[1]. In this paper, an allocation is considered as inequitable as far as the extreme difference of hosts loads is high.

**Definition 6 (Loads inequity indicator $\mathcal{E}_a(\mathcal{A})$).**

$$\mathcal{E}_a(\mathcal{A}) = \max_{h \in Rep_a(\mathcal{A})} \mathcal{C}_h(\mathcal{A}) - \min_{h \in Rep_a(\mathcal{A})} \mathcal{C}_h(\mathcal{A})$$

*Hence, resources are equaly comsumpted when $\mathcal{E}$ is minimal.*

The preference relation over the loads aims at separating the allocations into two classes: (1) those which globally reduce inequities and (2) the others. In term of resource consumption, Allocation $\mathcal{A}$ is preferred over Allocation $\mathcal{A}$' if and only if $\mathcal{A}$ reduces inequities, whereas $\mathcal{A}$' does not.

**Definition 7 (Agents load preferences $\succ_a^{\mathcal{C}}$).** *Let $\mathcal{E}_a^0$ be the load indicator of the current allocation:*

$$\mathcal{A} \succ_a^{\mathcal{C}} \mathcal{A}' \Leftrightarrow (\mathcal{E}_a(\mathcal{A}) \leq \mathcal{E}_a^0) \wedge (\mathcal{E}_a(\mathcal{A}') > \mathcal{E}_a^0)$$

Agents' preferences are designed to choose the safest allocation among those that do not reinforce load inequities:

**Definition 8 (Agent preferences $\succ_a$ (step 3)).**

$$\mathcal{A} \succ_a \mathcal{A}' \Leftrightarrow (\neg(\mathcal{A}' \succ_a^{\mathcal{C}} \mathcal{A})) \wedge (\mathcal{A} \succ_a^{\mathcal{F}} \mathcal{A}')$$

[4] A deal that improves the satisfaction of the poorest agent involved.

## 6 Implementation and Experiments

Our negotiation protocol has been implemented as an extension of the fault-tolerant multi-agent framework DimaX (see Section 4). To validate this protocol, we carried out several experiments. The following subsections describe the experimental protocol and discuss the results.

### 6.1 Simulator Architecture & Experimental Protocol

We extended DimaX with a fault-tolerant MAS simulator. This simulator defines the resources offer by each host and the resources needed by each agent replica. These two parameters are strongly interrelated, so the simulator provides a single indicator to control them. Basically, the amount of resources of the MAS represents the number of replicas an agent can create. Its domain is $[0,\ 1]$: 0 corresponds to no possible replication and 1 corresponds to the maximum replication, each agent can be replicated on all the hosts. Faults are also simulated. Each host is endowed with a function which defines its availability over time. The availability of a host is defined as a Poisson distribution. It thus defines the probabilistic number of host failures given an interval of time. The simulator provides also a global control of the average host availabilities.

We characterized four main values for the two indicators (*i.e. MAS resources amount* and *hosts global availability*). Firstly, the minimal and maximal values (0 and 1). Secondly, two average values: 0,66 where the indicator's value is said to be high and 0.33 where it is said to be low. In addition, the simulator obtains information during the execution of an application: the average load of each host, whether an agent is reliable or not, and whether it has failed or not.

To carry out our experiments, we developed a MAS application where agent criticalities are heterogeneous. The aim of these experiments is to evaluate our proposal on three criteria:

1. The uniformity of the hosts resources consumption,
2. The robustness of the MAS provided by our resource management solution,
3. The convergence speed of the negotiation protocol towards a reliable allocation.

### 6.2 Hosts Load Repartition

Our first series of experiments consists in evaluating the evolution of load differences among hosts in different configurations. In order to globally represent loads differences, we use variance, a well-known indicator that evaluates dispersion of a sample.

Fig. 3 shows the load variances when global availability of hosts varies. The three curves stand for different configurations of MAS resources amount (maximum, high and low). In the three configurations, variance is less than 0.1. So, those experiments show that the proposed negotiation protocol allocates resources uniformly.

**Fig. 3.** Equality in resource allocation

### 6.3 MAS Robustness

This second series of experiments evaluates the robustness of MAS. For that purpose, we analysed the rate of lost agents:
Lost agents rate $= \frac{\text{number of lost agents}}{\text{total number of agents}}$.

Fig. 4 shows the evolution of lost agents rate when hosts global availability varies. Three cases are compared: (1) without replication, (2) with replication and a low amount of resource (3) with replication and a high amount of resource. We can first see that, with a global availability between 0.2 et 0.6, the curves of the experiments with replication are concaves whereas not replicating results into a convex curve. Therefore, replication quickly decreases the lost agents rate when the global availability increases. Also, in case of a high ressources amount, if hosts have an average failure probability of $\frac{3}{5}$, only 10% of agents are lost with our protocol against 90% when replication is not used. Very few are lost (less than 5%) if the failure probability is less than $\frac{2}{5}$. Our protocol results thus in a significant improvement of the MAS reliability when the hosts provide a minimal availability.

### 6.4 Convergence Speed

The last series of experiments evaluates the convergence speed of the negotiation protocol. The aim of each agent is to progress towards a *reliable* state (see Definition 5). Moreover, a MAS is considered reliable when all its agents are reliable. The time when a MAS becomes reliable corresponds to the time spent by the last reliable agent. We thus analysed the required time for agents to become reliable. Also, since the convergence time of the agents is not uniform, we provide the average time spent by the agents to become reliable. Besides,

**Fig. 4.** MAS robustness

the convergence of the protocol relies on the protocole time $\Delta t$ (see Section 5). Indeed, some agents may wait until the end of the protocol (see Fig. 2) before replicating. Finally, note that the implementation of our achitecture is not optimized. So, the time between two executions of the protocol is high.

Fig. 5 shows the maximum and mean time for agents to become reliable when the amout of MAS resources varies. Those results are presented for environments where the hosts global availability is low or high. Even if the hosts global avail-

**Fig. 5.** Convergence speed

ability and amounts of resources are low, the MAS reliabilisation time is within three execution of the protocol. Also, the average time to become reliable is very close to a single execution of the protocol. The negotiation protocol is thus efficient.

## 7 Conclusion and Future Work

In this paper, we have formalized the replicas allocation problem and proposed a distributed fault-tolerance middleware for MAS. Our proposal deals with optimizing the MAS reliability while considering the associated decrease on MAS performances. This work is related to recent work on resource management in MAS (see for instance [12],[13],[10]). We have also built a fault-tolerant multi-agent platform, DimaX, to evaluate our proposal. This platform uses a fault-tolerant framework (DARX [17]) and a multi-agent platform (DIMA [9]). It provides a generic architecture to automatically augment an already-built multi-agent system with a basic monitoring mechanism.

We have proposed to endow the monitoring agents with knowledge and a set of processes and make them negotiate in order to adapt the replication to a dynamic environment. Experiments have been carried out showing that the reliability improvement is significant and that the resources of hosts are uniformly consumed. Meanwhile, more experiments with various scenarios, including large scale, are being conducted to further validate our approach and evaluate the replication and monitoring costs.

Nevertheless, our protocol suffers from some limitations. For example, less critical agents may try to allocate many replicas whereas the amount of resources is low. For that purpose, agents need to adapt dynamically and individually their reliability objectives to characteristics of the environement (resources amount, other agents loads and reliabilities). Similarly, the placement of replicas should also consider other host parameters (e.g. bandwidth) and replication strategies.

#### Acknowledgements

The authors would like to thank all members, present and past, of the *Fault-Tolerant Multi-agent Systems* project at LIP6 for their contributions. This work is funded by ANR (Agence National de La Recherche) and Region Martinique.

## References

1. K. J. Arrow, A. K. Sen, and K. Suzumura. *Handbook of Social Choice and Welfare*, volume Vol. 1. North-Holland, 2002.
2. S. Bouveret. *Algorithmique et complexité des problèmes de partage et d'allocation équitables*. PhD thesis, Office National d'Études et de Recherches Aérospatiales, 2007.

3. Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Welfare engineering in practice: on the variety of multiagent resource allocation problems. In M. P. Gleizes, A. Omicini, and F. Zambonelli, editors, *Proceedings of the Fifth International Workshop ESAW*, volume 3451.
4. K. Decker, K. Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In *ATAL'97*, LNAI, pages 63–75. Springer Verlag, 1997.
5. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Negotiating socially optimal allocations of resources. *Journal of Artificial Intelligence Research*, 25:315–348, 2006.
6. S. Estivie. *Allocation de Ressources Multi-Agents : Théorie et Pratique*. PhD thesis, Université Paris IX Dauphine, LAMSADE, December 2006.
7. A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *AAMAS2002*, pages 373–744, Bologna, Italy, 2002.
8. R. Guerraoui, B. Garbinato, and K. Mazouni. Lessons from designing and implementing GARF. In *Object-Based Parallel and Distributed Computation*, number 791 in LNCS, pages 238–256, 1995.
9. Z. Guessoum and J.-P. Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.
10. Z. Guessoum, J.-P. Briot, and S. Charpentier. Dynamic and adaptative replication for large-scale reliable multi-agent systems. In *Proceedings of the ICSE'02 First International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'02)*, Orlando FL, U.S.A., may 2002. ACM.
11. Z. Guessoum, J.-P. Briot, O. Marin, A. Hamel, and P. Sens. *Software Engineering for Large-Scale Multi-Agent Systems*, chapter Dynamic and Adaptative Replication for Large-Scale Reliable Multi-Agent Systems, pages 182–198. Number 2603 in LNCS. April 2003.
12. Z. Guessoum, N. Faci, and J.-P. Briot. Adaptive replication of large-scale multi-agent systems - towards a fault-tolerant multi-agent platform. In *SELMAS*, pages 238–253, 2005.
13. Z. Guessoum, M. Ziane, and N. Faci. Monitoring and organizational-level adaptation of multi-agent systems. In *AAMAS*, pages 514–521, 2004.
14. S. Hagg. A sentinel approach to fault handling in multi-agent systems. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems, Methodologies and Applications*, number 1286 in LNCS, pages 190–195, 1997.
15. G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Intelligence Artificial Research*, 17:83–135, 2002.
16. S. Kraus, V. Subrahmanian, and N. C. Tacs. Probabilistically survivable MASs. In *IJCAI'03*, pages 789–795, 2003.
17. O. MARIN. *The DARX Framework: Adapting Fault Tolerance For Agent Systems*. PhD thesis, Université Du Havre, 2003.
18. O. Marin, M. Bertier, and P. Sens. DARX - a framework for the fault-tolerant support of agent software. In *14th International Symposium on Software Reliability Engineering (ISSRE'2003)*, pages 406–417, Denver, Colorado, USA, 2003. IEEE.
19. H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.
20. R. van Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, 1996.
21. L. A. Zadeh. A new direction in ai: Toward a computational theory of perceptions. *AI Magazine*, 22(1):73–84, 2001.
22. Y. Zhang, E. Manister, S. Kraus, and V. Subrahamian. Approximation results for probabilistic survivability. 2005.