Demonstrating Communication Services Based on Autonomic Self-organization

A. Bernadas[2], R. Alfano[1], A. Manzalini[1], J. Solè-Pareta[2], S. Spadaro[2]
[1]*Telecom Italia, e-mail: antonio.manzalini@telecomitalia.it*
[2]*Universitat Politècnica de Catalunya, e-mail: pareta@ac.upc.edu*

Abstract

This paper reports the main results achieved in the development of a prototype for demonstrating communication services, based on the principles autonomic self-organisation. In particular, the prototype has been designed and developed as a distributed adaptable complex system, realized by means of a population of lightweight autonomic components interacting with each other through self-organizing algorithms. A demonstration prototype to show the collaborative ambient of a rescue team in a critical situation with limited connectivity, such as mobility, data distribution and high probability of disconnection, has been simulated. This prototype show both how these factors represent strong challenges for current software architecture and how the distributed lightweight components can self-organize themselves in order to face these challenges.

1. Introduction

The definition of the autonomic system is taking inspiration from the self-governing behaviors of some natural autonomic systems, such as the human autonomic nervous system. Once launching the Autonomic Computing initiative, IBM defined four general properties a system should have to constitute self-management: self-configuring, self-healing, self-optimizing and self-protecting. Since the launch of Autonomic Computing initiative, the self-* list of properties has grown substantially. Now it includes also features such as self-anticipating, self-adapting, self-critical, self-defining, self-destructing, self-diagnosis, self-governing, self-organized, self-recovery, self-reflecting and so on. The extension of the autonomic technology principles from computing to network and services resources has still the meaning of developing solutions that are capable of hiding operational complexity to both Operators and Users. Autonomic systems are capable of making decisions on their own, by using high-level policies from operators, checking and optimizing their status in order to adapt themselves to change environment conditions at the same time.

In Framework Program VI, the European Commission has launched the Situated Autonomic Communication Initiative (Future Emerging Technologies): its vision refers to the self-* features of the distributed network and service resources, systems and infrastructures fostering the development of the Information Communication Society.

In this context, the IST Integrated Project Cascadas [http://www.cascadas-project.org/] has the main goal in identifying and developing distributed component-ware architecture for development of innovative situation aware and autonomic services. Basic concepts and algorithms adopted for the development of the prototype have been inspired by the activities carried out in the project.

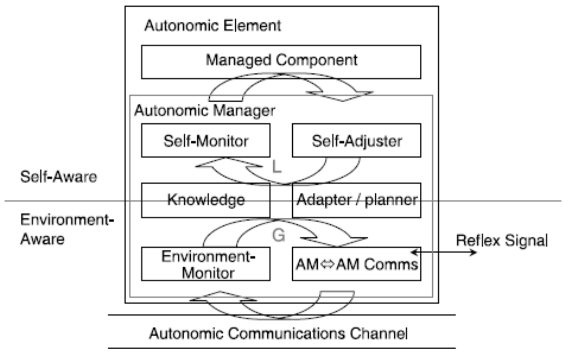


Figure 1. Example of an autonomic element

0-7695-3109-1/08 $25.00 © 2008 IEEE
DOI 10.1109/CISIS.2008.143

In IP Cascadas, a distributed autonomic system can be seen as a framework of autonomic components (figure 1) [1], dynamically interacting with each other and self-organizing their activities to achieve certain goals. Specifically, autonomic features of the components can be exploited through the introduction of goal-oriented knowledge reasoning capabilities. Regarding the self-organization capabilities, the second key aspect of the prototype, biological algorithms have always been a key source of inspiration [2]. As an example, swarm intelligence principles have been widely used for modeling problems through some simple interactions of a collection of agents cooperating to achieve a common goal. In these systems, problems are "self-solved" in real time through the appropriate collective behavior, as consequence of interactions occurring between the agents and the environment.

Above principles can be ideally applied to model the self-* collective behavior, which can be observed also in human social relationships. Cities, for example, can be easily recognized as self-* ecosystems. Nowadays, with the wide adoption of digital devices, communications are generating data clouds overlooking modern cities whose patterns show self-adaptive and self-organizing properties.

The aim of this paper is demonstrating, through the development of a prototype, the applicability of the key principles of autonomic self-organization for the development of solutions enabling communication services even in critical disconnected situations (such as a catastrophic event in a city).

A brief video clip (.ogg) of the demonstrator is available at the Cascadas web-site [http://www.cascadas-project.org/].

2. Autonomic self- aggregation

This section provides an introduction about the two basic foundations of autonomic self-organization, as developed into the prototype: i) autonomic agent environments; ii) self-organising algorithms. Attention has been mainly focused where innovation is needed, for instance on the applicability of self-organising algorithms in autonomic agent environment. Given the richness of available results, references to literature are provided for further details.

2.1. Autonomic Agents

Autonomic systems are typically distributed, complex and concurrent, comprised of multiple interacting autonomic elements and all the resultant issues have already been faced in different fields of autonomous agents.

Agent-based approaches have been, and remain, a rich area for the study of the emergence of self-organisation. For example, "artificial markets" have been studied for their potential in market-based control. The aspiration is that if the appropriate interaction/trading rules are encoded into a population of agents, then the agents will be able to self-organise into "useful" structures/networks, where "useful" is defined in terms of an application context, such as supply chains or trading markets. Di Marzo et al. [3] reviewed different aspects of self-organisation in Multi-Agent Systems. They show how inspiration derives from natural systems (complex physical systems as well as natural systems). For example, the concept of stigmergy, derived from the behaviour of social insects, has also been important in inspiring the design of Multi-Agent Systems. Bernon et al. [4] review several examples of applications of self-organising multi-agent systems. They show how Multi-Agent Systems can self-organise themselves to carry out tasks, even though individual agents have very simple properties. Moreover, the emergent properties of the self-organising system support each application.

2.2. Self-organising algorithms

This section describes two examples of self-organizing algorithms [6]: passive clustering [6] and on-demand clustering [6]. In the context of this paper the term "aggregation" refers to the process by which nodes form associations ("links") with each other. It is a "clustering" process during which each node, characterised by a certain "type" establish links with nodes of the same type. From this point of view, the efficiency of a self-aggregation algorithm can be read in terms of convergence capabilities of increasing the fraction of links connecting nodes of the same type.

2.2.1. "Passive" clustering. A first set of basic local rules has been devised requiring only direct interaction between first neighbours yet susceptible to give rise over time to spontaneous system-wide aggregation of elements. The basic idea involves two nodes being notified by a third (the "match-maker"), which are interconnected through an overlay network, even if those two nodes have no direct part in the decision process ("passive" clustering). The rules are as follows:

– match-maker node is randomly selected. This is equivalent to say that every node in the system has a chance of "waking-up" and initiating a rewiring procedure, provided that this procedure is brief enough (and/or infrequent enough) that a situation in which two concurrent rewiring affect the same nodes is extremely unlikely, and so every attempt can be considered as an independent event.

– match-maker randomly selects two of its own neighbours and, if they happen to belong to the same type, instructs them to link together
– if the two chosen nodes were not already directly connected (through the overlay) a new link is established between them (i.e. they become first neighbour of each other).
– if conservation of the total number of links is in force (optional) and a new connection is successfully established, the match-maker terminates one of its own links with one of its two selected neighbours.

2.2.2. "On-demand" clustering. In passive clustering technique, in order to preserve homogeneous node degree in the realistic, local rules-based scenario, the rewiring procedure has to be modified: there is the need of eliminating the indirect positive feedback leading to the emergence of scale-free topology. It may be objected that the heterogeneous node degree can be highly beneficial to the network operation if the higher connectivity of some vertices can be made to reflect their superior capability. However, in our case, such correlation is effectively absent: the emergence of hubs in the "passive rewiring scenario" results from the amplification of random fluctuations. As it cannot be guaranteed that those nodes ending up with a higher degree effectively have some specific features that designate them as efficient "super-peers", the result could be disastrous and generate critical bottlenecks, which is why we aimed at maintaining node degree as homogeneous as possible throughout the system's history.

This has been achieved by distinguishing between the initiator of a rewiring procedure and the match-maker. Basically, upon "waking-up", the initiator requests a new link from one of its existing neighbours, which will then act as the match-maker. Since with this logic, the probability for a node to be appointed match-maker is obviously a direct function of its own degree (and the match-maker still ends losing one neighbour in the process of a successful rewiring operation), it introduces a negative, "rich becomes poorer" feedback similar to the one observed in the abstract model.

The detailed algorithm governing key node behaviour in the three roles of "initiator", "match-maker" and "candidate" involved in a rewiring operation following the "on-demand" clustering procedure is shown in figure 3. It involves exchanging five types of messages (plus the link termination message which isn't discussed here). The "neighbour request" (NRQ) message is sent by the initiator to the chosen match-maker and specifies the type of node desired. The "neighbour reply" (NRP) message is sent by the match-maker to the initiator to inform it to a potential candidate. The "link" (LNK) message is sent by the initiator to the candidate to ask for the establishment of a new link, which will only be effectively created if it is compatible with the goals of the candidate, as evidenced by the receipt of an "acknowledgement" (ACK) message by the initiator. Notice that, for most of the results presented in this section, this will always be the case as all nodes in the system share the same objective, i.e. they are all assumed to be simultaneously in clustering (or reverse-clustering) mode. Finally, after a successful handshake between the initiator and the candidate, the match-maker is informed via the "success" (SCC) message so that the match-maker can be able to determine whether or not its own connection to the candidate has to be terminated, in order to conserve the total number of links.

3. Architecture of the prototype

A prototype has been designed and developed in order to demonstrate the applicability of the main principles of autonomic self-organization in terms of interactions of a population of autonomic components through self-organizing algorithms.

The scenario used for demonstrating the prototype is a dynamic and data intensive digital environment, such as a city with pervasive digital devices, where everyone can exchange information and collaborate with each other. A scenario like this requires technologies and solutions to manage large amounts of highly distributed data items, which need to be transformed into meaningful, reliable and available information for each mobile user.

3.1. The use-case

The selected use-case is a city where some catastrophic event has impacted the communication infrastructure causing faults limiting normal wire and wireless connectivity, but with the premise that all agents are endowed with mobile devices with wireless capacity [5]. The use-case is ideal for demonstrating how autonomic self-organisation meets requirements for emergency communications between the survivors and rescue teams, with the aim of providing first aid as soon as possible.

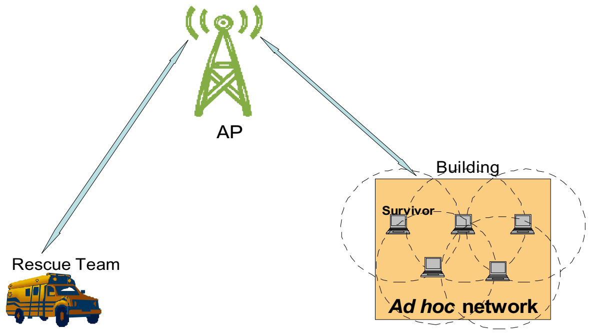


Figure 2. Main entities involved

Two basic groups are involved in the simulation: the rescue teams and the survivors. The interaction between groups of survivors and between survivors and rescue teams should test the efficiency of the self-management system. Rescue teams and survivors are placed in two differentiated interaction environments.

Inside each building the communication is managed in a completely distributed way without any central control. This is likely to be a communication network between peers based on an ad-hoc system. The last objective is to achieve the major spread degree of available information in the environment, in order to make each survivor able to provide and gather the information from its neighbours.

Moreover, the rescue teams will interact with the different environments where the survivors are situated. Therefore, a rescue team will be able to communicate with a group of survivors when it is in the covering area where some of them are situated.

The basic rules that the agents must fulfil are:

- communicate the information to each other in the same environment.
- migrate to other environment (building).

Figures 2 and 3 provide, respectively, a representation of the main entities involved and a snapshot of the demo application developed to test the proposed technique.

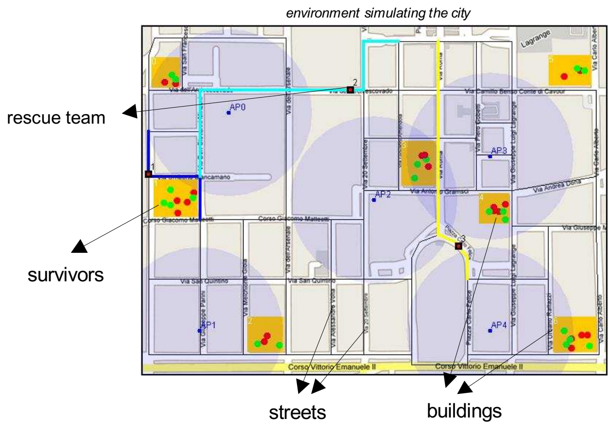


Figure 3. Demo snapshot

3.2. Main architecture

The architecture of each agent is structured into the following functional blocks.

3.2.1 Reasoning Part. It has the task of managing the lifecycle of an agent. It describes the possible state and invokes the proper specific features, if specified, running all of them as a state machine. It works within the system as a DIET's job, in a parallel execution way.

3.2.2. Communication part. It is in charge of implementing communication among the agents. It includes the three behaviours described before in the "On-demand" clustering algorithm ("initiator", "match-maker" and "candidate"). It also includes blocks that are in charge of the information exchange contemplated in reasoner's logic (messages including survivor's list, GPS position and so on). Jobs are the way of distributing the functionalities of an agent. Therefore, agents can compose their behaviour by combining multiple jobs. To implement the behaviour that responds to the communication protocol mentioned a structure of different jobs has been created. The structure is the following one:

1. *NotifyNeighboursJob*: Job that on the start-up notifies in broadcast of its type of ID to the neighbours by creating connections. It also handles notifications of the neighbours.
2. *RandomNeighbourRequestJob*: Job that initiates the request process for type items (Initiator behaviour).
3. *HandleNeighbourRequestJob*: It handles requests, returning a random address of a candidate with the type requested (Match-Maker behaviour).

4. *HandleNeighbourReplyJob*: It handles notifications of the type requested, starting the link process with the chosen candidate (Initiator behaviour).
5. *HandleLinkJob*: It handles "link" requests from the initiator in the link process (Candidate behaviour).
6. *HandleAckJob*: It handles "ack" responses from the candidate finishing the link process (Initiator behaviour).
7. *HandleSuccessJob*: It handles "successful links" notifications to the candidate from the initiator agent destroying the connection with the candidate (Match-Maker behaviour).

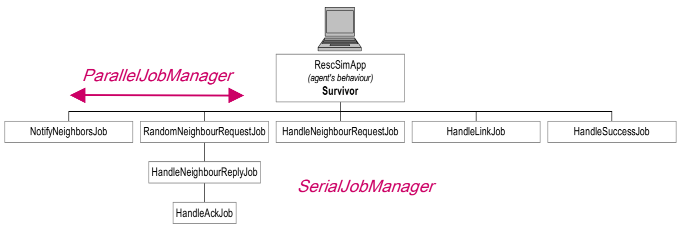


Figure 4. Jobs' Structure

The *SerialJobManager* is used to execute several jobs in sequence. Once the first job has finished, it will start the second job, the third, the fourth and so on. This is useful when an agent's behaviour can be split into various stages. This sequence system is used to implement all the steps of the initiator agent behaviour ("wake-up", link creation, send success and so on).

The *ParallelJobManager* is used to run multiple jobs concurrently. For instance, an agent may use a scheduler job to manage its schedule events, and another job that implements the specific behaviour to the agent which requires scheduling functionality. Therefore, the system designed in the simulation is served by this scheduler job to manage the tree jobs structure because it requires combinations of sequence and parallel execution.

A correct implementation of these jobs provides us an important feedback about the applied behaviour of the algorithm. Figure 4 shows the tree structure distribution of the different jobs for the proper work of the communication algorithm.

3.2.3 Specific Part. It executes a normal code, depending on Reasoning Part's decisions, and returns the results so that they can be checked and sent back. For instance, there are specific functions for the survivors' list managing that confronts the information contained in them as well as the management of information that refers to the GPS positioning.

3.3. Technological approach

This section provides a description of the technology and algorithms used for the prototype development. In particular the prototype has been developed using the DIET (Decentralised Information Ecosystem Technologies) multi-agents framework and implementing the active-clustering self-organization protocol.

3.3.1. DIET. DIET Agents (Decentralised Information Ecosystem Technologies) [http://diet-agents.sourceforge.net] is a platform for developing agent-based applications. DIET platform [7], developed in the EU-funded DIET project, is an Open-source framework released under GPL license and downloadable from sourceforge web site.

DIET goal is providing an ecosystem-inspired approach to the design of agent applications [8]. In this context an ecosystem can be viewed as an entity composed of one or more communities of lightweight components conducting frequent, flexible and local interactions with each other and with the environment that they inhabit.

Although the capability of each lightweight component itself may be very simple, the collective behaviours and the overall functionality arising from their interactions exceed the capacities of any individual organism. These higher-level processes can be adaptive, scalable and robust to changes in their environment.

3.3.2. Self-organization protocol: active-clustering. In order to achieve a good level of self-organization, a bio-inspired self-organizing algorithm has been applied.

This protocol is totally based on the previous "On-demand" clustering protocol described before.

The figure 5 shows the interaction among the three behaviours included in the "On-demand" clustering protocol.

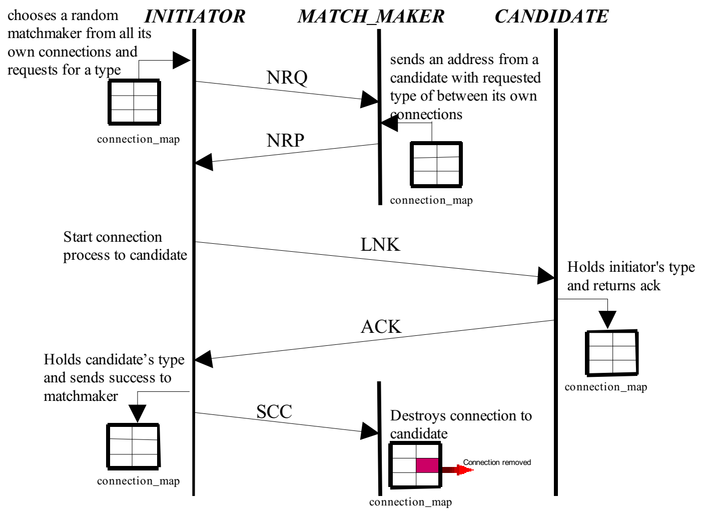

chooses a random matchmaker from all its own connections and requests for a type
INITIATOR
MATCH_MAKER
CANDIDATE
NRQ
sends an address from a candidate with requested type of between its own connections
connection_map
NRP
connection_map
Start connection process to candidate
LNK
Holds initiator's type and returns ack
ACK
connection_map
Holds candidate's type and sends success to matchmaker
SCC
Destroys connection to candidate
Connection removed
connection_map
connection_map

Figure 5. DIET behaviours interactions

4. Future work

Further investigation and development will be based on the enhancing of the Reasoner (with RuleML). In particular, the enhancements (by adding a rule engine with a set of rules) will improve the behaviour of the survivors and rescue teams in order to obtain the highest efficiency. Added rules can be created, deleted and modified by the previous rules if necessary so the rule engine is able to adapt itself to the changes in the environment. It can also be seen as a rule engine that has a set of states that an agent must achieve by fulfilling some goals.

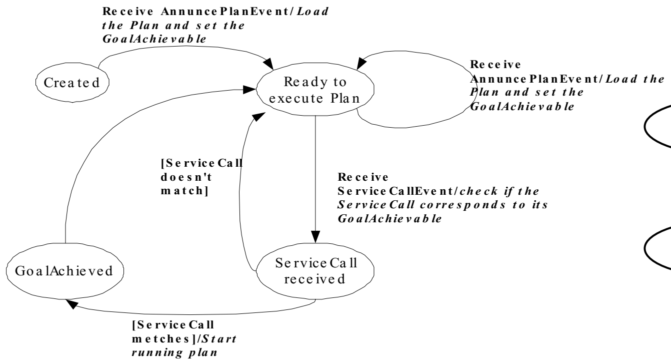

Receive AnnuncePlanEvent/Load the Plan and set the GoalAchievable
Created
Ready to execute Plan
Receive AnnuncePlanEvent/Load the Plan and set the GoalAchievable
[ServiceCall doesn't match]
Receive ServiceCallEvent/check if the ServiceCall corresponds to its GoalAchievable
GoalAchieved
ServiceCall received
[ServiceCall metches]/Start running plan

Figure 6. Basic process of reasoner

The state machine tries to understand the different changes in the environment and it gives orders to change some of the internal behaviours. Every survivor and rescue team will have his own state machine implemented in RuleML and attended by the reasoning. RuleML is a shared Rule Markup Language, which allows both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks.

The figure 6 shows the basic state-machine process of the reasoner.

Above, the simplified state-machine shows the main mechanism used by the agent to fulfil a ServiceCallEvent. The Reasoner for each plan, provided by the Facilitator, extracts a GA, representing the goal the agent achieve after the execution of the plan.

When a Reasoner with an active plan receives a ServiceCallEvent transforms the ServiceCallEvent in a GA and tries to match it with the GA of its active Plan. If the two GAs match, the reasoning on the active Plan starts. In this way, the ServiceCallEvent will be completely fulfilled when the Reasoner reach the GA.

Within the Reasoner, development has been oriented to modify the plan using a RuleML version and to introduce refactoring in the reasoning. So, Reasoning capabilities allow the agent to take the proper actions in terms of sending GAs and GNs, invoking specific functions and checking internal conditions, in order to achieve a given Goal.

The figure 7 and 8 corresponds to the state machines devised for both survivor and rescue team agents respectively.

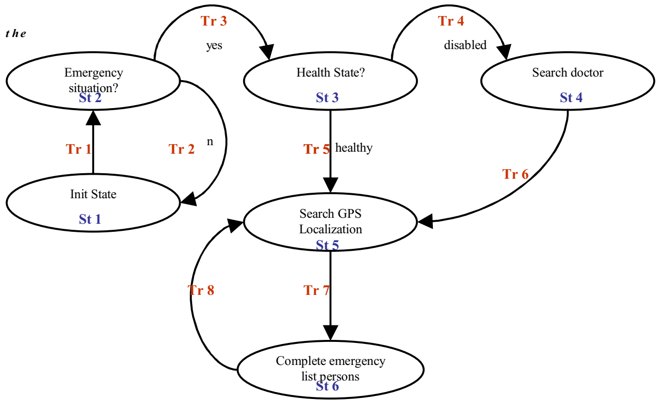
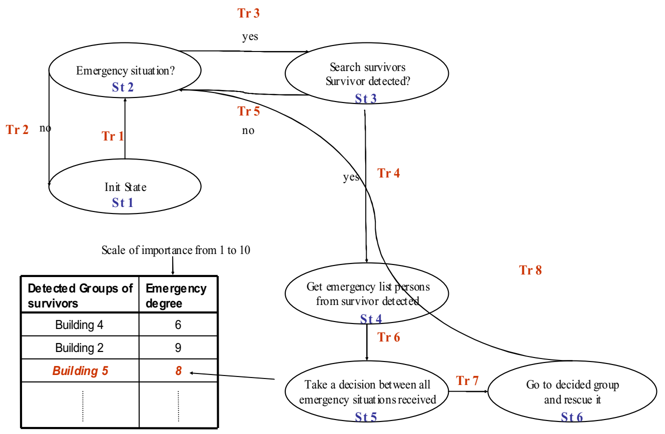


Figure 7. Survivors's Self-Model


Tr 3
yes
Emergency situation?
St 2
Search survivors
Survivor detected?
St 3
Tr 2
no
Tr 1
Tr 5
no
yes
Tr 4
Init State
St 1
Scale of importance from 1 to 10
Get emergency list persons from survivor detected
St 4
Tr 8
Tr 6
Take a decision between all emergency situations received
St 5
Tr 7
Go to decided group and rescue it
St 6

Detected Groups of survivors	Emergency degree
Building 4	6
Building 2	9
Building 5	*8*
⋮	⋮

Figure 8. Rescue Team's Self-Model

5. Conclusions

This paper reports the main results achieved in the development of a prototype for demonstrating communication services based on the principles autonomic self-organisation. In particular, the selected use-case focus on the applicability of the above principles for the development communication services, even in critical disconnected situations such as catastrophic event in a city.

The achieved results have demonstrated that the solution, designed by means of self-organising algorithms deployed in frameworks of distributed autonomic agents, is meeting some challenging requirements such as adaptability to dynamic situations and robustness, even in environments with high churn rate and/or disconnected situations.

A brief video clip (.ogg) of the demonstrator is available at the Cascadas web-site [http://www.cascadas-project.org/].
Further investigation and development will go in the main direction of enhancing reasoning capabilities of the agents, by adopting a RuleML approach for instance.

6. Acknowledgements

The authors would like to acknowledge the European Commission for funding the Integrated Project CASCADAS (Note 1) "Component-ware for Autonomic, Situation-aware Communications, and Dynamically Adaptable Services" (FET Proactive Initiative, IST-2004-2.3.4 Situated and Autonomic Communications).

7. References

[1] R. Sterritta, M. Parasharb, H. Tianfieldc, R. Unland, A concise introduction to autonomic computing, Advanced Engineering Informatics 19 (Elsevier, 2005), 181–187.
[2] A. Manzalini, P. Marrow, "The CASCADAS Project: A Vision of Autonomic Self-organizing Component-ware for ICT Services" SOAS 2006 (Erfurt, September 2006)
[3] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. LectureNotesinComputerScience, 1563:404–413, 1999.
[4]B-G. Chun, R. Fonseca, I. Stoica, and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In Proc.IEEEINFO-COM, 2004
[5] N. Ravi, et al., "Accessing Ubiquitous Services using Smart Phones", 3rd International Conference on Pervasive Computing and Communications, Kauai Island (NW), March 2005.
[6] Aggregation Algorithms, Overlay Dynamics and Implications for Self-Organised Distributed Systems. IST CASCADAS Work Package 3 Deliverable Month 12
[7] DIET –
http://diet-agents.sourceforge.net/ProjectBackground.html
[8] T. Choudhury, A. Pentland, "Modeling Face-to-Face Communication Using the Sociometer", Conference on Ubiquitous Computing, Seattle (WA), 2003.