

A Stochastic Situation Calculus Modelling Approach for Autonomic Middleware

M. Randles

School of Computing and
Mathematical Science,
Liverpool John Moores
University,
Byrom St. Liverpool,
L3 3AF, UK
cmsmrand@livjm.ac.uk

A. Taleb-Bendiab

School of Computing and
Mathematical Science,
Liverpool John Moores
University,
Byrom St. Liverpool,
L3 3AF, UK
A.Talebendiab@livjm.ac.uk

P. Miseldine

School of Computing and
Mathematical Science,
Liverpool John Moores
University,
Byrom St. Liverpool,
L3 3AF, UK
cmppmise@livjm.ac.uk

Abstract- The ever growing complexity of modern computer systems to cater for users increasing demand for higher software functionality, reliability and security, whilst lowering maintenance, administration and access costs, has provided a springboard for exploring new models for distributed software engineering and lifetime management. A number, of these models, are inspired by biological concepts thus requiring software to possess awareness capabilities to continuously monitor its own operating, security and/or environment's conditions and to plan and execute changes to adjust its behaviour for safe and optimal operation. Much related research work can be found under a variety of headings including: self-adaptive software, proactive computing, reflective middleware and autonomic computing. The latter, championed by IBM sought to adopt a rule based approach to underpin the "sensor-effector" mechanism for autonomic behaviour. This paper, however, presents a formal semantics for the event-situation-condition-action sequence, via the stochastic situation calculus, which is used to formalise the "adjustable" governance models for autonomic software behaviour. In addition, the paper presents the use of the stochastic situation calculus together with an automated deliberative reasoning mechanism to develop autonomic middleware architecture.

Keywords: Stochastic Situation Calculus, Adjustable Autonomy, Autonomic Middleware, Normative Systems.

1 INTRODUCTION

The more recent developments in distributed software design involve the biologically inspired concept of autonomic computing [1], which is anticipated to exhibit self-organization, adaptation, management and protection capabilities. In such a model computer systems can assume much of their own administration, maintenance and management operations. Thus alleviating the effects of the growing systems complexity, management and administration costs. The literature is full of applications of autonomic computing in a variety of domains including; grid computing and networked appliances, with scenarios ranging from on-demand composition of intelligent home appliances [2], self-

healing grid computing to zero-maintenance home networks.

In particular, this paper outlines the results of an investigation into formal modelling of autonomic software including self-governance constructs and reasoning models for predictable self-adaptive software. In this work, distributed software objects are modelled as a federation of autonomous software agents or peer-to-peer systems. Where agents are considered as active entities whose behaviour can usefully be described in terms of mental notions such as knowledge, obligations, to itself and other agents, goals and abilities.

In addition the ability to make decisions without the intervention of human users is directly related to the level of autonomy of an agent [3]. Thus the authors contend that autonomous software agent theory provides a very useful insight and applicable framework for the reasoning process and design models for autonomic computing. In particular, the methods considered in the decision making process are crucial to self-stabilizing the agent with bounded resources [4].

In order to represent and validate this approach the model must be underpinned by a logical and rigorous formalism, which in this work is achieved through the use of a variant of first order predicate calculus – stochastic situation calculus [5-11]. In the context of distributed reflective software, meta-reasoning mechanisms are required to facilitate automated governance of autonomic software applications insuring predictable, safe and/or efficient operation.

The motivation for this paper is to gain a fundamental understanding of the design, running and maintenance of self-adaptive systems, as a facet of autonomic computing, through the understanding of how the software agents can govern and operate within such a system. This requires a study and application of distributed artificial intelligence and underlying theories of deliberative software models to move towards the designs and specification for a new generation of self-governing software. The paper will start with a brief introduction of the stochastic situation calculus with concepts of autonomous agency, followed, in Section 3, by the proposal for an

extensible agent architecture in which reasoning can take place. In Section 3.3, the programming model and middleware architecture are presented. This approach is illustrated, in section 4, using a self-healing scenario of mobile computing applications.

2. BACKGROUND

This work takes a logical representation of bounded autonomous agents operating under a deliberative normative architecture to facilitate an autonomic computing paradigm. The first requirement is to look at logical specification of systems using the situation calculus. A full presentation of situation calculus is out of the scope of this paper and can be found in [5]. However the main constructs and concepts are outlined below.

2.1 The Situation Calculus

The situation calculus presented in [5] formalizes the behaviour of dynamically changing systems, which provides a particularly useful instrument to model autonomous, mobile, distributed applications, including intelligent networked appliances. The situation calculus is derived from the original formulation of [6]. The formalism is based on the notion of a situation, a snapshot of the state of the world. Each situation is defined by the value of the fluents, the situation dependent functions and predicates, in the situation. A situation is transformed to a new situation by a named action that changes the value of one or more fluents.

- **Situations:** which all emanate from an initial situation, S_0 , where no actions have yet occurred. A possible history is a sequence of actions called a situation.
- **Fluents:** are relations or functions where truth or function values change from situation to situation. They are denoted by function symbols with a situation term as their final argument
- **Actions:** change one situation to its successor situation. Each named action has an action precondition axiom stating the conditions under which the action can occur. In simple cases these can be single actions with a linear ordering. However [7] shows how concurrency and time can also be introduced to the representation. A treatment allowing the use of a flexible and very applicable stochastic situation calculus is given in [8].

2.1.2 The Language and Foundational Axioms of the Situation Calculus

A formal rigorous specification of the situation calculus language based on [9] is given in [5].

2.1.3 Effect, Frame and Successor State Axioms

It is necessary to state how the actions affect the world via effect axioms. These show the change in

value of a fluent when an action causes the situation to change.

However to reason about change in the system these axioms are not sufficient. It is necessary to add frame axioms that state when fluents remain unchanged by actions.

This gives rise to the frame problem [10]. The quantity of frame axioms is very large (of the order of two multiplied by the number of actions multiplied by the number of fluents). The solution is to combine the frame and effect axioms into a single successor state axiom [11].

i.e. TRUE in next situation \Leftrightarrow (an action occurred to make it TRUE)

\vee (TRUE in current situation and no action occurred to make FALSE).

2.2 Adjustable Autonomy

In a coalition of agents following the proposed EBDI model (section 2.3.1) the norms are represented as a shared ontology. The agents must balance their individual desires against these norms, with associated responsibilities and obligations. Since the domain of interest is a dynamic environment the degree of solipsistic behaviour versus social behaviour should not be fixed beforehand. So agents ought to be able to adjust their autonomy at runtime [12]. Where agents are cooperating to achieve a representation of an autonomic computing environment this gives rise to the proposed adjustable autonomicity.

An agent's autonomy is necessarily bounded since it is situated [13]. The agent is not autonomous in any abstract sense but is autonomous with respect to another object such as other agents or the environment. A view of autonomy for multi agent systems that sets the level of autonomy that the agent has over its actions and decision-making processes is given in [14]. However, from an adjustable autonomy viewpoint, the agents' levels of autonomy describe the degree of independence an agent has over its cooperating agents.

2.2.1 Norms

In terms of computational economy an agent is best served in following the system norms. Norms can be distinguished in a number of ways. Norms may be categorized as constraints on behaviour (e.g. never delete a file marked important), goals (e.g. open a file) or obligations (e.g. retrieve a file for a user) [15].

For the normative deliberation, to be proposed in the EBDI model, norms are best seen as obligations. This can render norms as situation dependent behaviour rules, for the computational economy, and as a description of the distribution of problem solving by agent roles. The norms thus enable an agent to expect fellow agents to behave in a prescribed manner allowing predictable autonomy.

2.3 Agent Architectures

A number of agent architectures have been proposed to enable deliberation and agent autonomy. The most common representation and the basis of most models is the Beliefs-Desires-Intentions (BDI) deliberative framework [16]. This sought to avoid an agent becoming bogged down with many competing action options by constraining the formulation, over which the agent must reason, to intentions and commitments.

In essence the BDI agent has beliefs representing the current state of its world and desires representing the agents ideal world. The mismatch, between these representations, triggers the intentions to rectify the current state to the ideal state.

2.3.1 Extending BDI

There have been numerous proposed extensions to the BDI model, most notably to include normative behaviour and thus cooperation and coordination. (e.g. The Belief-Obligation-Intention-Desire model (BOID) [17] .) However the Extensible BDI proposed here, is based on an enhanced Epistemic-Deontic-Axiologic (EDA) model [18].

In the EDA model intentions are constituted via a filtering process. The agent has its own beliefs in an epistemic component and sets its goals via self-obligations. Additionally it has obligations to its fellow agents. These self and social obligations are set in the deontic component. The axiologic component provides a means of dynamically setting the importance an agent assigns to norms. The constituted obligations are thus assessed through the axiology and the committed intentions established.

3 The EBDI MODEL

Central to the provision of a formalism, within which to reason, is the model architecture. The BDI model was first proposed by Bratman [16] as a design for deliberative software agents. Various extensions to the BDI model were proposed to address some of its documented weaknesses, such as the inability to dynamically set intentions and the lack of social interaction leading agents to blindly follow their own (selfish) intentions. The proposed EBDI provides a highly suitable architecture for the design of situated intentional software agents that continuously monitor and/or observe their environment.

3.1 Normative Structure

Norms arise from a social situation in that they involve providing rules for a community of more than one individual. They express a desire external to the agent yet an agent is charged with the fulfillment of that desire. A norm can take virtually any cognitive form. A typical, social psychological, classification partitions norms into four types: perceptual, evaluative, cognitive and behavioural. These translate into four distinct attitudes:

Ontological, axiologic, epistemic and deontic [21]. In other words: to recognizing the existence of an object, applying a value system to decide in favour of or against something, to possess a degree of belief or disbelief in something and to be obliged to act in some way.

Thus it is proposed to use the EDA approach in [18] to extend the agents knowledge base to include:

- An epistemic component that stores the statements accepted as true by the agent. This can be as declarative or procedural statements, such as plans and the know how of the agent.
- A deontic component where there is an internalization of duties or social obligations. An agent's personal goal can be seen as a self-obligation.
- An axiologic component that acts as an agent's value system.

3.2 Social Agency Intentions

The distinction between social and selfish obligations is carried forward to the agents' desires. The deontic component is based on the concept of a generalized goal that combines the social and individual goals.

A notation may be established whereby $B_\alpha(p)$ indicates that p is one of agent α 's beliefs, $O_{\alpha\beta}(p)$ is the assertion that agent α must *see to it that* proposition p is true for agent β . So the individual goals may be represented as interests, that the agent is not necessarily aware of, maybe originated externally by other agents, or desires, interests that the agent is aware of. So not all interests become desires but all desires are agent interests. In the notation desires may be represented as: $O_{\alpha\alpha}(p) \wedge B_\alpha(O_{\alpha\alpha}(p))$.

There are also corresponding social forces on the agents' achievement agenda. Duties are social goals that the agent is not necessarily aware of. Demands are duties that the agent is aware of. In the notation a duty is denoted by $O_{\alpha\beta}(p)$ in that α must do p on behalf of β .

A demand is $O_{\alpha\beta}(p) \wedge B_\alpha(O_{\alpha\beta}(p))$.

These individual and social goals in turn become intentions if they are part of the preferred model, representing a set of non-conflicting goals. These intentions may be placed in the achievement agenda. For various reasons these intentions may also be discounted as they may be overridden by higher priority intentions, the timing may not be right or the resources may not be available.

3.3 Autonomic Computing

Within the distributed system community there is a need to design and build computing systems capable of running themselves, adjusting to unpredictable changes and handling resources efficiently [19]. The two main elements of autonomic management are the

functional unit and the management unit. The functional unit performs the main operation and is provided by elements such as web services or databases, etc. The management unit is responsible for system resources and operational performance and hence the reconfiguration of resources according to adaptive changes [20].

The problem to be addressed here is to provide a logical and rigorous formalism for the event-situation-condition-action sequence that occurs to facilitate the maintenance of distributed computer systems, over a wide network, by autonomic methods.

3.4 EBDI in an Autonomic Middleware Control Service

It is proposed to investigate the model with an autonomic control service based in the middleware. The control service incorporates three core services, embedded in a three-layered model (Fig. 2), provided by: the service manager agents, the javaSpace service and the system controller agent. The architecture is based on a control service model that continuously monitors the specified service for non-ideal behaviour, to identify conflicts and errors, prescribing repair plans and performing reconfiguration.

3.4.1 The Service Manager Agents

The service manager agent is used to adapt the structural components and the dynamic behaviour of the provided services. Structural components can evaluate their behaviour and environment against their goals and restructure accordingly. The separation of concerns for the control service is achieved by assigning a service manager agent to each application level function. The service manager agent performs four tasks: Service monitoring, service diagnosis, service repair and the provision of an adaptor to receive messages from the repair processes to enable the adaptation of the application level services attributes.

3.4.2 The JavaSpace Service

This service provides the means of communication by configuring a distributed shared memory space based on distributed tuples.

3.4.3 The System Controller Agent

The system controller agent is responsible for the dynamic management of the entire distributed application. It also coordinates the activities of the individual service manager agents. The service managers communicate via the JavaSpace to post their application level service states. The system controller contains three main components: The system monitor, the system reconfiguration module and the system repair strategies consisting of resolution actions determining when, where and how the repair is applied. It is here that the resolution strategies (intentions) are chosen based on the agent deliberation of its beliefs and obligations. The lack of norm based reasoning, dynamic adjustment and cooperation/coordination in the BDI model make it necessary to use the proposed EBDI.

Beliefs correspond to service information derived from a range of sources, including domain, environment or the communicated beliefs of other agents.

Desires represent the state of affairs in an ideal world that often maximize the service's own goals. By comparing the system belief set (observed system states) against its desires, the system may detect a mismatch and instantiate a set of intentions.

Situated intentions represent action sets for the system to undertake in a given situation to achieve its specified desires and/or to address the mismatch between the system environment (beliefs) and the system's desires (goals).

Normative intention represents a set of actions to be undertaken to ensure a specified set of norms, including obligation and responsibility, rules are observed, via a dynamic axiologic representation, before a given intention is enacted.

Utility intention represents a set of system actions constituted through the axiological filtering to optimize its goal-oriented intentions such as minimizing costs or optimizing quality of service.

In this implementation and in accordance with [22, 23] the beliefs, desire, goal and intention can be described as being collections of constraints, each of which represents distinct pieces of beliefs, desire or goal and so on. These constraints are generated using service beliefs and desires. In this implementation, beliefs are a runtime service's states such, as a service is available for client requests or not. Intentions are the system actions (execution), which are, for instance, triggered because of a mismatch between the system beliefs and system desires sets using the norms.

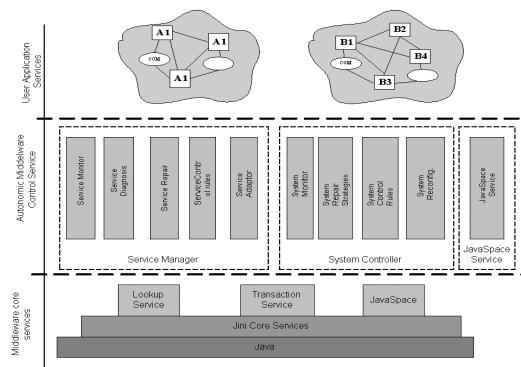


Figure 2: The control service architectural layers

4. CASE STUDY

The autonomic control service has been demonstrated through an industrial case study, with Merseyside Fire Brigade, namely EmergeITS. A service offered by EmergeITS is a 3-in-1-phone service that allows a mobile phone or PDA device to be used in one of three ways: A cellular phone, a WAP phone or a walkie-talkie.

```

service_max(do(a, s)) ⇔ ¬ service_max(s) ∧
    ¬ ∃ client(a=remove(client)) ∨
    ∃ client(a=add(client) ∧
    service_num(s)=N-1)
service_num(do(a,s))=n ⇔ ¬ service_num(s)=n ∧
    ¬ ∃ client(a=add(client) ∨
    a=remove(client)) ∨
    (service_num(s)=n-1 ∧
    a=add(client)) ∨
    (service_num(s)=n+1 ∧
    a=remove(client))
poss(remove(client), s) ⇔ (service_num(s)=N ∧
    low_priority(client, s)) ∨
    requesting_disconnect(client, s)
poss(add(client), s) ⇔ service_num(s)<N

```

Figure 4: The successor state axioms for $service_max(s)$, $Service_num(s)$ and the action precondition axioms for removing or adding a client.

The service manager monitors the requests from clients such as; $connect()$ or $disconnect()$. The capacity of the system is limited so that there is a maximum number of connections possible, N (say). So that a $remove_client()$ action may be a repair strategy if a client has low priority when the system is operating at full capacity. This raises a fluent $service_max(s)$, meaning the service is at maximum capacity in situation s , and $service_num(s)$, a functional fluent recording the number of service users in situation s , with associated actions $remove(client)$ and $add(client)$.

```

unavailable(client,do(a,s)) ⇔ unavailable(client,s) ∧
    a ≠ succeeds_connect(client) ∨
    a ≠ fails_connect(client)
retrying_connection(client, do(a, s)) ⇔
    retrying_connection(client, s) ∧
    (a ≠ succeeds_connect(client) ∨
    a ≠ fails_connect(client)) ∨
    a = retry_connect(client)
alt_provider(client,do(a,s)) ⇔
    (alt_provider(client,s) ∧
    a ≠ switch(home_provider(client))) ∨
    (unavailable(client, s) ∧
    retrying_connection(client, s) ∧
    a = fails_connect(client))

```

Figure 5: Successor state axioms for $unavailable$, $retrying_connection$ and $alt_provider$.

When exceptional behaviour prevails the system controller will trigger a conflict resolution process. For example the action $connect()$ can be represented by stochastic choices $succeeds_connect()$ or $fails_connect()$. The unavailability of the GSM service will cause a $fails_connect()$ action to occur. The system controller agent checks this exception and an appropriate resolution strategy is applied based on the EBDI model. So the strategy is first to retry to connect. Then, if this is still unsuccessful, to search for an alternative GSM service provider. The EBDI based conflict resolution strategy can be represented in stochastic situation calculus by using the fluent $unavailable(client,s)$ meaning the service is unavailable for the client in the situation. Associated fluents are raised during the action processes. The actions participating in the fluents will have associated precondition axioms. An implementation has been established using this formalism.

```

<?xml version="1.0" ?>
- <!-- Simple Description of 3in1 phone Strategies -->
- <!--
- <DOCTYPE strategy (View Source for full doctype...)
- <Strategies>
- <Strategy id="1" type="desire">
- <Action id="1" type="plan" name="Connection">
- <Properties>
- <property id="1" name="host">cmsnbadr</property>
- <property id="2" name="Location">GPS_loc</property>
- <property id="3" name="Max_connected">maxNo</property>
- <property id="4" name="Method">connect</property>
- </Properties>
- </Action>
- </Strategy>
- <Strategy id="2" type="intension">
- <Action id="1" type="plan" name="Retry">
- <Properties>
- <property id="1" name="No_trial">two</property>
- <property id="2" name="serviceStatus">not null</property>
- <property id="3" name="Method">connect</property>
- </Properties>
- </Action>
- <Action id="2" type="plan" name="Alternative">
- <Properties>
- <property id="1" name="host">cmpnbadr</property>
- <property id="2" name="New Manager Interface">ManagerProxy</property>
- <property id="3" name="Get Manager">getServiceManager</property>
- <property id="4" name="Client interface">ClientProxy</property>
- <property id="5" name="Get Client">getClient</property>
- <property id="6" name="Notify Client">notifyClient</property>
- <property id="8" name="Max_connected">maxNo</property>
- <property id="9" name="Method">connect</property>
- </Properties>
- </Action>
- </Strategy>
- </Strategies>

```

Figure 6: XML description of the self repair strategy

5 CONCLUSIONS AND RESEARCH DIRECTION

It is clear that the deliberative properties of the agent model are easily represented in the situation calculus. The agent setting of self and social obligations through to the establishment of the committed intentions are flexibly realized. While the progress of the instantiation and negation of fluents gives a path through the repair strategies prompted by the model. Additionally this flexible representation allows the creation of situation trees promoting contrafactual reasoning and the analysis of what-if scenarios. The extension to stochastic actions is also very applicable in this domain where actions can succeed or fail.

The use of EBDI gives increased system resilience with optimized repair strategies. The system was evaluated using the metrics of stability, robustness, performance profile and average latency (the measure of the average time taken for the controller to initiate

its control cycle). This evaluation revealed that this system is slower in a conflict occurrence situation. However it is possible to tune the autonomic control. Also the system without autonomic control stops, in cases of catastrophic failure, whereas the autonomic system proceeds through such failures.

This work and study is ongoing. Currently research is being conducted, jointly with the NHS, to provide a highly functional decision support system, based across a computational grid, for breast cancer patients/clinicians [24]. Also further experiments are being conducted applying the results of this research, to body area network and biofeedback. Where body sensor data (states) can be deliberated upon leading to control, activation and self-tuning of networked appliances.

ACKNOWLEDGMENT

Thanks are due to Nagwr Badr, whose past work at Liverpool John Moores University has formed the basis for some of this research.

REFERENCES

- 1 IBM, Autonomic Computing.
- 2 A.Mingkhwan, P. Fergus, O. Abeulma'atti and M. Merabti (2004) 'Implicit functionality: Dynamic services composition for home networked appliances', ICC2004, IEEE International Conference on Communications, Paris, France.
- 3 M. Dastani, F. Dignum and J. J. Meyer (2003) 'Autonomy and agent deliberation'. Proceeding of the First International Workshop on Computational Autonomy, AAMAS'03, Melbourne.
- 4 M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
- 5 H. J. Levesque, F. Pirri and R. Reiter (1998) 'Foundations for the situation calculus'. *Linköping Electronic Articles in Computer and Information Science*, Vol. 3(1998): nr 18. <http://www.ep.liu.se/ea/cis/1998/018/>
- 6 J. McCarthy (1963) 'Situations, actions and causal laws'. Technical report, Stanford University. Reprinted in *Semantic Information Processing*, ed: M. Minsky, pp 410-417, MIT Press, Cambridge, Massachusetts, 1968.
- 7 R. Reiter (1996) 'Natural actions, concurrency and continuous time in the situation calculus'. *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, ed: L. C. Aiello, J. Doyle and S. C. Shapiro, pp 2-13. Morgan Kaufmann Publishers, San Francisco, California.
- 8 R. Reiter (2002), 'Stochastic Actions, Probabilities and Markov Decision Processes in Situation Calculus'. Proceeding of the 18th National Conference on Artificial Intelligence, Edmonton, Alberta, Canada July 28th 2002
- 9 F. Pirri and R. Reiter (1999) 'Some contributions to the metatheory of the situation calculus'. *Journal of the ACM* 46(3), pp 325-361
- 10 J. McCarthy and P. Hayes (1969) 'Some philosophical problems from the standpoint of artificial intelligence'. *Machine Intelligence 4*, ed: B. Meltzer and D. Michie, pp 463-502, Edinburgh University Press, Edinburgh, Scotland.
- 11 R. Reiter (1991) 'The frame problem in the situation calculus: a simple solution (sometimes) and a complete result for goal regression'. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, ed: V. Lifschitz, pp359-380, Academic Press, San Diego, California.
- 12 G. Dorais, R.P. Bonasso, D. Kortenkamp, P. Pell and D. Schreckenghost (1998) 'Adjustable autonomy for human centered autonomous systems on mars' Mars Society Conference 1998.
- 13 C. Castelfranchi (1995) 'Multi-agent reasoning with belief contexts: the approach and a case study' In M.J. Woolridge and N.R. Jennings (editors). *Intelligent Agents*, Springer-Verlag
- 14 J.E Verhagen and R.A. Smit (1997) 'Multi-agent systems as simulation tools for social theory testing.' Paper presented at poster session at ICCS and SS, Siena 1997.
- 15 R. Conte and C. Castelfranchi (1995) 'Cognitive and social action' UCL Press, London
- 16 M. E. Bratman (1987) 'Intentions, plans and practical reason'. Harvard University Press, Cambridge, Massachusetts.
- 17 J. Broersen, M. Dastani, J. Hulstijn, Z. Huang and L. van der Torre (2001) 'The Boid Architecture' Agents'01, Montreal, Quebec, Canada
- 18 J. Filipe (2002) 'A normative and intentional agent model for organisation modelling' Third International Workshop "Engineering Societies in the Agents World" 2002, Madrid, Spain
- 19 P.Horn (2001) 'Autonomic Computing: IBM' s Perspective on the State of Information Technology'. IBM Corporation, 2001.
- 20 D.Chess, C.Palmer, and S.White, 'Security an autonomic computing environment'. *IBM SYSTEMS JOURNAL*, 2003. 42
- 21 R. Stamper (1996). 'Signs, Information, Norms and Systems'. In Holmqvist et al. (Eds.), *Signs of Work, Semiosis and Information Processing in Organizations*, Walter de Gruyter, Berlin, New York.
- 22 S.Rao and P.Georgeff, 'BDI agents: From theory to practice'. the First International Conference on Multi-Agents Systems (ICMAS-95), 1995(MIT Press): p. 312-319
- 23 M. Boman (1999) 'Norms in artificial decision making' *AI and Law*, 1999.
- 24 Liverpool John Moores University (2004), <http://www.cms.livjm.ac.uk/2nrich>