

Adjustable Deliberation of Self-Managing Systems

M. Randles, A. Taleb-Bendiab, P. Miseldine and A. Laws
School of Computing and Mathematical Science,
Liverpool John Moores University, Byrom St. Liverpool, L3 3AF, UK
{ cmsmrnd, a.talebbendiab, cmppmise, a.laws }@livjm.ac.uk

Abstract

In this paper a cybernetics-based viable system architectural model is introduced, which provides a blueprint for high-assurance systems and a meta-control model necessary for the adjustable deliberation and autonomy of self-managing systems. The logical formalism is provided by the Situation Calculus and underpinned by an Enhanced Belief-Desires-Intentions (EBDI) framework to facilitate the representation and reasoning of scaleable autonomic computing systems.

1. Introduction

The more recent developments in software design involve the concept of autonomic computing [1, 2] capable of self-organisation, adaptivity, control and management, where complex computer systems will offer users advanced interaction models and services ranging from the users' task automation to the runtime adjustment of their applications autonomy. Such a design model was motivated by the fact that the complexity and the distribution of systems often outstrip the power and cost-effectiveness of manual systems' management. Thus there is value in delegating most of systems management to the software itself.

Currently design models of "autonomic systems" including the IBM blueprint [3], do not employ "cognitive" (including deliberative independent) behaviour to support autonomy, but rather use explicitly managed autonomy via policies and rule sets that predefine and predict all extraneous behaviour specified at design time often using Event Condition Action (ECA) constructs [4]. Whilst, this is a well tested and understood design principle the authors argue that often this has limited reach when applied to open, ad-hoc and evolving software systems. Thus, in this paper we outline a design

model for adjustable deliberation and autonomy, based on well-established cybernetics design model providing a blueprint for the architecture of self-managing systems (Sec. 2). This is underpinned by a formal deliberative reasoning logic, extending the Belief-Desire-Intention framework, where a Situation Calculus specification is proposed to enable the understanding of the necessary architecture and the dynamic nature of its behaviour (Sec. 3, 4).

Section 5 introduces a case study, from the medical informatics domain showing how self-management deliberation can be specified in the proposed logic system of the situation calculus in the EBDI framework. This leads to a practical implementation using the cloud architecture with the custom designed meta-language Neptune mapped from the Situation Calculus. The paper concludes (Sec. 6) with a brief summary and indication of future research direction.

2. The Viable System Model for Deliberation

Based on a well known cybernetics design and management theory A. Laws *et al.* [5] have contended that for systems to exhibit autonomy - independent existence- they are required to possess six high-level systems types namely; Operations, Coordination, Control, Audit, Intelligence and Normative Policy. They proposed a Viable Intelligent Agent Architecture - VIA-A [5]. The latter mimics the operations of the human brain, in that autonomic responses are those that require no conscious effort by an organism. The model identifies five identifiable Systems within a running system: The effectors (S^1), coordination (S^2), management (S^3), an audit component (S^{3*}), deliberation on future scenarios (S^4) and system identity (S^5).

Comparable to the Real-time Control System (RCS) reference model architecture, which at "... each level of the hierarchy is composed of the same

basic building blocks. These building blocks include behaviour generation (task decomposition and control), sensory processing (filtering, detection, recognition, grouping), world modelling (knowledge storage, retrieval, and prediction), and value judgement (cost/benefit computation, goal priority).” [6], the VIA-A model provides mechanisms for the self-governance and runtime adjustment of autonomy and deliberation, which is detailed in the following sections.

Whilst, the application of the VSM model to guide the design of autonomous systems is not new [7], the main concern of this paper is its use to provide a template of autonomic system design, whereby the functions of an autonomic system can be mapped onto the architectural model of the VSM, using a deliberative formalism, without the need for a large and complex production system. The ability to make decisions without the intervention of human users is directly related to the level of autonomy of an agent operating in the system [8].

In these applications decision making VIA-A agents must select and execute actions at each moment in time. If the environment changes during either the selection or execution the agent needs to deliberate to reconsider previous decisions (intentions) or continue its commitment to these intentions. The methods considered of making decisions are crucial to stabilize the behaviour of the decision-making agent with bounded resources [9].

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 [10]. Where agents are cooperating to achieve a representation of an autonomic computing environment this gives rise to the proposed adjustable autonomicity. A view of the ‘freedom’, in multi agent systems, that sets the level of autonomy that the agent has over its actions and decision-making processes can be specified [12]. However, from an adjustable autonomy viewpoint, the agents’ levels of autonomy describe the degree of independence an agent has over its cooperating agents.

Stemming from the use of the Extended BDI model and the viable systems architectural model, deliberative agent activities are viewed as intentional situated actions thus the next section introduces the EBDI, as an adjustable deliberation framework, where the Situation Calculus (Sec. 4) may be used to model and test the processes to enable the specification of agent systems in the VSM model to facilitate autonomic computing.

3. An Adjustable Deliberation Framework

A number of agent architectures have been proposed to enable deliberation and agent autonomy. The most common representation is the Beliefs-Desires-Intentions (BDI) deliberative framework [13]. This sought to avoid an agent becoming weighed 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 agent’s ideal world. The mismatch, between these representations, triggers the intentions to rectify the current state to the ideal state.

There have been numerous proposed extensions to the BDI model to include normative behaviour and thus cooperation and coordination in multi-agent systems. For instance, the Belief-Obligation-Intention-Desire model (BOID) [14] and the Epistemic-Deontic-Axiologic (EDA) model [15]

However, the Extensible BDI elaborated and proposed here [16], is based on an enhanced Epistemic-Deontic-Axiologic (EDA) model [15]. In the EBDI agent intentions are triggered by a discrepancy between the believed state of the world and the goal state as represented in the desires. These intentions are constrained by the component modules within the model:

- The Epistemic component holds the representation of beliefs, in a logical form, cognitive norms and the inference capabilities. Belief statements are simple propositions believed true by the agent. Cognitive norms are conditional beliefs that are based on the communicated beliefs of other agents.
- The Deontic component holds all the possible agent’s behaviours, which can be represented as plans at various abstract levels. For instance, an agent goal is a highly abstract plan such as see to it that A holds, for some proposition A . Alternatively, a sequence of elementary executable actions can form the course of a completely specified plan. If plans are conceived of as agent behavioural norms then the executable plan steps can be obtained from high-level goals by backward chain reasoning. That is the abstract goals entail the formulation of more specific goals, via the reasoning processes, until executable tasks are identified in the agent’s behaviour.

- The Axiologic component gives a utility to the intentions. Since some agent committed intentions may be in conflict, the intentions executed ought to be the ones that minimizes a cost function.

The system norms represent a shared ontology for the agent federation. In terms of computational economy an agent is best served in following the system norms. Norms can be viewed in a number of ways as in [17]. They can be separated into categories:

- Constraints on behaviour
- Goals
- Obligation

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 safe, predictable autonomy. A system action can be seen as a derivation of *resources* from an actor-action triggered *role*. The S^3 management marshals the necessary S^1 units as *resources* to accomplish the required action. The S^2 management coordinates the action execution in a prescriptive manner allowing predictable autonomy, in that;

- The model is triggered by an agent's role through perception filters, subject to the agent's ontology, using monitoring and utility norms to update any of the components.
- The reward component decides which domain specific factors to perceive, providing environmental variety attenuation.
- The epistemic knowledge base is where the agent stores its beliefs both explicitly and via logical inference.
- The responsibilities represent sets of plans that the agent can choose to execute to amplify its variety to the environment.
- The resources may constrain the agent's actions via availability.

Thus a framework is established whereby agent intention can be constrained, via the defined model components, to output the committed intention best suited to the situational trigger gained through the event-situation-rule-action procedure. Hence a flexible and expressive language is required in which the intentional normative deliberation can be represented, reasoned with and modelled.

It is proposed to use the Situation Calculus because of the modelling formalism it provides for dynamic worlds. Physical systems consist of many successive, action triggered, transitions between many situations or snapshots of the world. In the Situation Calculus situations are sequences of actions or action histories. Transitions are allowed, subject to suitable preconditions that alter the value of situation dependent functions. Thus the Situation Calculus provides similar features to the well known Finite State Machine (FSM) methods. However, in the Situation Calculus, there is no requirement for an enumeration of states prior to runtime. The Situation Calculus also gives a richly descriptive formal 'paper-based' specification that uses; precondition axioms to ensure the correct execution of actions, successor state axioms to concisely represent the effects of events and complex action theory to express sequential, conditional and iterative execution patterns.

The next section gives an overview of the Situation Calculus that provides the requisite features.

4. Modelling Autonomy via a Situation Calculus Language

The Situation Calculus used here [18] formalizes the behaviour of dynamically changing systems, and is derived from the original formulation of [19]. It can support concurrent actions and timing constraints [20]. It is useful in that each situation can be viewed as a history of previous actions. A formalisation of action and time can be presented to model the deliberation points in an autonomic setting.

All situations emanate from an initial situation, S_0 , where no actions have yet occurred. A situation is transformed to a new situation by means of a named action. A possible history is a sequence of actions called a situation. This is built up by means of a constructor operator *do*. So that $do(a, s)$ means the situation where action *a* has been performed in situation *s*. Then $do(b, do(a, s))$ means the situation where action *b* has been performed in situation $do(a, s)$. So a situation denotes a history consisting of a sequence of actions occurring in order from right to left in the situation term.

Relations where truth-values change from situation to situation are relational fluents. They are denoted by predicate symbols with a situation term as their final argument. Similarly functional fluents are functions whose values change according to situation. They are denoted by function symbols with an extra argument for the situation term.

4.1. The Situation Calculus Language

So, more formally, using the methods of [18] who in turn took their description from [21], the situation calculus has the usual logical connectives and quantifiers. Using these features the situation calculus takes the following form:

- A constant symbol S_0 denoting the initial situation
- A function symbol $do : action \times situation \rightarrow situation$ so that $do(a, s)$ represents the successor situation resulting from doing action a in situation s .
- A predicate symbol $\subseteq : situation \times situation$ which orders the situations treated as histories of actions. Thus $s \subseteq s'$ is interpreted as s is a sub-history of s' .
- A predicate symbol $poss : action \times situation$. $poss(a, s)$ means it is possible to do action a in situation s .
- Predicate symbols of sort $(action \cup object)^n$ to denote situation independent relations
- Function symbols of sort $(action \cup object)^n \rightarrow object$ to denote situation independent functions.
- Function symbols of sort $(action \cup object)^n \rightarrow action$ to denote action functions such as $drop(x)$, $move(X,Y)$, etc. There is a distinction here between the functions whose range is action values and those with a range in object values. The action functions need to be axiomatized with action precondition axioms.
- Predicate symbols of sort $(action \cup object)^n \times situation$ to denote relational fluents.
- Function symbols of sort $(action \cup object)^n \times situation \rightarrow action \cup object$ to denote situation dependent fluents or functional fluents

By convention relational or functional fluents have one situation argument that is placed as the final term.

There are foundational axioms for the uniqueness of situations, induction on situations, existential initial situation and ordering of situations.

In order for an action to occur a set of propositions must be true at the instance of the action. Thus there are precondition axioms for any domain action.

$$poss(A(x_1, x_2, \dots, x_n), s) \Leftrightarrow F_A(x_1, x_2, \dots, x_n, s)$$

where $F_A(x_1, \dots, x_n, s)$

is an expression specifying the preconditions for the action $A(x_1, x_2, \dots, x_n)$.

It is necessary to state how the actions affect the world via effect axioms, which 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 [22]. 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 [23], that is,

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

Combining these ideas provides a complete specification of any general domain.

5. Case Study-Implementation

It is proposed to investigate this approach by an implementation that initially specifies an autonomic control service, based in the middleware, using Situation Calculus to specify the deliberative functions and meta-model. The control service incorporates three core services (agents), embedded in a three-layered model (Fig. 1), provided by: the service manager agents' team, the distributed shared system space 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.

In terms of the VSM the application level services are the S^1 systems. The service manager agents, identified with the S^1 management functions, are used to adapt the structural components and the dynamic behaviour of the provided services. The S^2 system acts as a distributed system space or shared memory, initially the JavaSpace but later, in a more flexible implementation, as an XML space. This achieves coordination by each service manager posting its service state to be used by other agents or the system controller for conflict resolution. This can also gain fault tolerance as other service manager agents, based on the reading of the shared space, can adopt unconnected services, as an emergency measure. The S^3 system is a system controller, responsible for the dynamic management of the entire distributed application. It also coordinates the activities of the individual service manager agents, through the S^2 system. 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, are part of the S^4 system. It is here that the resolution strategies (intentions) are chosen based on the agent deliberation of its role, responsibility, reward and regulatory strictures through the EBDI. The S^5 system gives the high level system desires.

Beliefs correspond to service information derived from a range of sources, including the environment (gathered by the S^4 system) or the communicated beliefs of other agents via the shared space.

Roles represent the state of affairs in an ideal world that often maximize the agent's own goals in terms of fulfilling a desire. By comparing the system belief set (observed system states) against its stated role, the system may detect a mismatch and instantiate a set of intentions. So the system high-level desires are propagated throughout the system to lower level management systems in a form that sets desires local to the services. For instance the high level goal of availability manifests itself in the goal of setting a repair strategy for services the agent believes to be damaged.

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) including acting as a resource for fellow agents and maintaining system norms.

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

Reward 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 [24] 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.

So a direct mapping between the components in an autonomic control service and the system levels in a VSM system model can be established using the EBDI framework to control the variety available and amplify the output variety to and from the system levels conceptualised as agent teams.

This is the generic autonomic control service based on the VSM. The case study, under consideration here, takes this architecture and applies it to a medical decision support and information system. Various treatment guideline models are available within the system that returns treatment options based on varying patient data variables. The user of the system ought not to be aware of the autonomic nature of the systems running.

Now to formalise the deliberation procedures required for intention resolution the situation calculus is used to illuminate the EBDI processes. For instance, a report of an unavailable service in the system space triggers a situation whereby the role of service reconnect is activated in the system controller (S^3 level). When the system detects a failure to connect to a service it automatically retries as a responsibility to the connecting agent. On failing a predetermined number of times it then attempts to connect to an alternative service and/or starts a diagnostic process assembled from its available resources resulting in a repair strategy committed intention. The Situation Calculus representation of this example of EBDI deliberation process is detailed in the example rules shown below (Fig. 2). In this way the agent team is defined by logical expressions for agent federations, via joint persistent goals, with the domain goals and intentions for the team members set as Situation Calculus language logical expressions. Similar representations, to the formulae, detailed in figure 2, of self-governance rules have been derived to give a complete specification and are used in the resulting implementation described below.

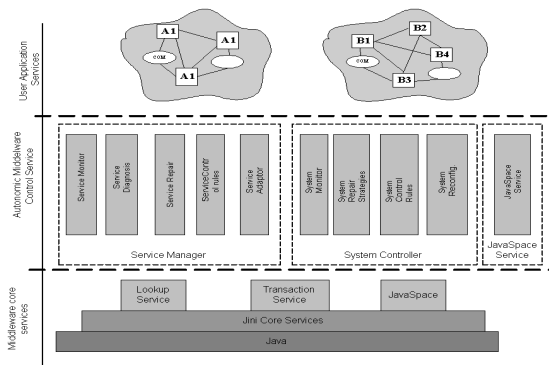


Figure 1. Autonomic Control Middleware Service

```

retrial(id, do(a,s)) ⇔ retrial(id, s) ∧
    -∃ t(a=read(space)→decision(id,t))
    ∧-retried(id,s) ∨ a=retry(id)

with poss(retry(id), s) ⇔ -available(service, s) ∧ -retried(id,s)
  where id=(session,doctor,patient,service)
giving rise to additional successor state axioms:

retried(id, do(a,s)) ⇔ retried(id, s) ∨
    [(a=read(space)→ -available(service, s))∧
    retrial(id,s)]

available(service, do(a,s)) ⇔ available(service,s) ∧
    - (a=read(space)→ -connected(service,s))
connected(service, do(a,s)) ⇔ ( connected(service,s) ∧
    a= fails_connect(service)) ∨
    a= succeeds_connect(service)

with
poss(fails_connect(service),s) ⇔ remote_service_exception(s)
poss(succeeds_connect(service),s) ⇔ request(id, s)

If the system was still unavailable after retrying a specified number of times then a
diagnostic state would be entered leading to repair strategies based in the
implemented scripts

diagnosing(service, do(a,s)) ⇔ (diagnosing(service,s)∧
    -∃fault(a=service(fault))∨
    (a=findfault(service))

with
poss(findfault(service),s) ⇔∃ id (retried(id,s)∧-connected(service,s)
leading to
repair(service, do(a,s))⇔[repair(service,s)∧-∃ a(a=repaired(service))∨
    (a=repair(service))

with
poss(repaired(service),s)⇔available(service,s)
poss(repair(service),s)⇔diagnosed(service,fault,s), etc....

```

Figure 2. A Situation Calculus Representation

The application is a proposed theoretical model of adjustable deliberative autonomy using an implementation of a grid-based medical decision-support system -- Clouds [25], a self-regenerative architecture in which self-governance logic modelled in the Situation Calculus is expressed and enacted through a bespoke declarative meta-language, Neptune [26] that can be compiled, deployed, inspected and modified at runtime. (The full description of this meta-language is outside the scope of this paper. However further details can be found by following the stated references.)

The self-regenerative architecture, Clouds, is defined to enable the self-adaptive framework to function using the scripting language, Neptune that allows management objects to be compiled and inspected at runtime. The autonomic control service

has been implemented as a Cloud system. Neptune exposes policies and decision models for system governance, derived from the situation calculus/EBDI model, as compiled objects that can be inspected, modified and executed at runtime. The mechanics are specified and modelled logically and the implemented Cloud architecture contains associated Neptune objects that are addressable and adaptable at runtime. Thus the system can evolve as modelled by the logical specification in a safe and predictable manner giving the adjustable self-management required.

The Clouds concept is conceived as a system with fluid and flexible boundaries that can interact and merge with similar system architectures. The cloud can be thought of as a federation of services (agents) and resources controlled by the system controller and discovered through the system space. The system space provides persistent data storage for service registration and state information giving the means to coordinate the application service activities.

The system controller, as the S^3 level management, controls access to and from the individual services, the S^1 systems, and resources within the Cloud. It brokers requests to the system based on system status and governance rules, in Neptune objects, derived from the EBDI deliberative process as exemplified in figure. 2. The system controller acts as an interface to the Cloud. It can function in two roles, either as an abstraction that inspects calls between the System Space, acting as an S^2 level management, and services, or as a monitor that analyses the state information stored within the system space.

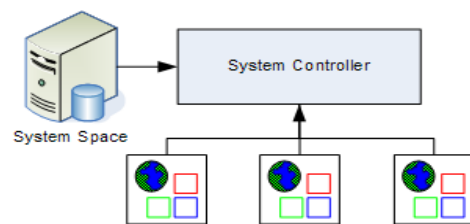


Figure 3. System Controller in Charge of System Space

In its first role (Fig. 3), the controller marshals and inspects calls to and from the System Space from services. In its second role (Figure 4), the controller delegates storage responsibilities to the services, that inherit the capabilities to interact with the System Space at design-time through inheriting base classes that encapsulate some of the controller's functionality.

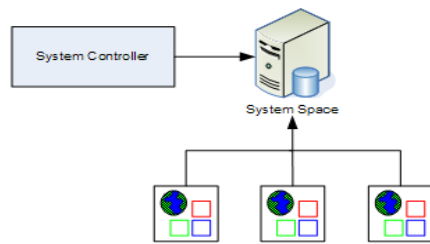


Figure 4. System Controller as System Space Monitor

Each service and resource when it first registers itself to the Cloud sends a meta-object serialized from an XML definition file. This meta-object contains the properties and state data of the service it is describing and is stored within the System Space at registration. Each service maintains its own meta-object and updates the System Space when changes in state occur.

The XML definition file contains all information required for the Cloud to discover the service through registration contained in the service element. This allows the querying of the service status through the published state properties contained within the state element. As the meta-objects are stored within the System Space, they enjoy system wide scope, allowing any other service or resources to request read and write access to the properties of a service through the guard of the system controller.

The Neptune scripting language is structured in terms of rules, conditional statements and variable assignments, directly mapped from the logical specification, which are parsed from script form into software system objects, encapsulating all the logical inference processes and variable instantiations. Thus allowing the Neptune object to be inspected, modified, recompiled and re-evaluated at runtime. In this way the base rules for deliberation in the EBDI to control the Cloud architecture, based on the VSM, have been transcribed, from the Situation Calculus reasoned representation, into Neptune objects that can be modified as a result of agent deliberation on system events.

For example in the Situation Calculus representation shown (fig. 2) an availability rule is defined together with resolution strategies in the case of a service being unavailable. This specifies that if the service is not available for calling, then it first queries the Cloud status to see if a service instance alternative is available. If no instance is found, the repair strategy is service regeneration. Calls are then rerouted to the newly generated service, or the

alternative service instance if located. The mapped Neptune script is shown in Figure 5:

```

define service s
if (service.availableServices.likeMe.count = 0)
    service.s = regenerate(me,machineID)
else
    service.s = services.availableServices.likeMe[0]
end if
rerouteCalls(s)

```

Figure 5. Neptune Repair Strategy Script

6. Conclusions and Further Research

The paper outlined how a generic, well-tried and tested model for viable system architecture can have its systems mapped to autonomic control service architectures. This has two immediate consequences. The first is that a blueprint can be established for a viable autonomic system. So designers can take the model and structure their service requirements within it. Secondly existing architectures that facilitate autonomic computing can be validated against the VSM model. Additionally the systems deliberation is addressed at each level of the VSM, mapped to the autonomic architectural components. A logical deliberation framework is established with the Situation Calculus representation of an EBDI agent model. This specifies the deliberative normative intentions of the autonomous, cooperating and coordinating agents charged with the execution of system tasks. The environmental variety is attenuated by using agent roles. The decision process uses behavioural deontics, axiologic utility with the agent knowledge base to constrain the intentional choices to a set of non-conflicting actions.

The case study shows how an existing autonomic control service can be mapped to the VSM system levels with the deliberative components specified. Finally the Cloud architecture, as a regenerative autonomic software environment, is modelled and specified logically with a working implementation. The logical rule base and inference engine, specified and guided by a formal Situation Calculus representation, is directly mapped into the scripting language Neptune to give runtime inspection and modification of services to gain the autonomy that agents require to give an autonomic system.

The deliberation architecture is at an early stage of development. It is thought that the composition of agent teams from sets of heterogeneous agents will provide powerful techniques to forward the concept

of autonomic software systems. It is hoped the logical specification required can give a base level deliberation capability. The formation of the agent teams and the evolution by self-adaptation can provide the autonomous agent running 'intelligence'. This can then be implemented through the well defined autonomic architectures available.

Acknowledgement

This paper has been partially supported by EPSRC grant reference: GR/R86782/01, 2nrich Project, URL: <http://www.cms.livjm.ac.uk/2nrich/>

References

1. P.Horn (2001) 'Autonomic Computing: IBM's Perspective on the State of Information Technology'. IBM Corporation, 2001.
2. A.G. Ganek and T.A. Corbi, "The Dawning of the Autonomic Computing Era", IBM Systems Journal, Vol. 42, No. 1, 2003.
3. IBM Autonomic Blueprint: www-3.ibm.com/autonomic/r/downloads/blueprint/ [Accessed 31st October 2004].
4. J. Bailey, A. Poulouvassilis, P. T. Wood (2001) 'An Event-Condition-Action Language for XML' *WWW2002*, May 7-11, 2001, Honolulu, Hawaii, USA.
5. A. G. Laws, A. Taleb-Bendiab, S. J. Wade, D. Reilly (2001) 'From Wetware to Software: A Cybernetic Perspective of Self-adaptive Software'. IWSAS 2001: 257-280
6. J. S. Albus (1997) 'The NIST Real-time Control System (RCS): an approach to intelligent systems research'. *Journal of Experimental and Theoretical Artificial Intelligence* 9(2-3): 157-174
7. Beer, S., *The Viable System Model: its provenance, development, methodology and pathology*, in R. Espejo and R. Harnden, *The Viable System Model: Interpretations and Applications of Stafford Beer's VSM*, John Wiley and Sons Ltd., Great Britain, 1989.
8. 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.
9. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349-355, 1988.
10. 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.
11. 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
12. 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.
13. M. E. Bratman (1987) 'Intentions, plans and practical reason'. Harvard University Press, Cambridge, Massachusetts.
14. J. Broersen, M. Dastani, J. Hulstijn, Z. Huang and L. van der Torre (2001) 'The Boid Architecture' *Agents'01*, Montreal, Quebec, Canada
15. J. Filipe (2002) 'A normative and intentional agent model for organisation modelling' *Third International Workshop "Engineering Societies in the Agents World" 2002*, Madrid, Spain
16. N. Badr, A. Taleb-Bendiab, M. Randles, D. Reilly (2004) *A Deliberative Model for Self-Adaptation Middleware Using Architectural Dependency*. *DEXA Workshops 2004*: 752-756
17. R. Conte and C. Castelfranchi (1995) 'Cognitive and social action' UCL Press, London
18. 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/>
19. 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.
20. 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.
21. F. Pirri and R. Reiter (1999) 'Some contributions to the metatheory of the situation calculus'. *Journal of the ACM* 46(3), pp 325-361
22. 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
23. 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
24. N. Badr, A. Taleb-Bendiab, D. Reilly (2004) *Policy-Based Autonomic Control Service*. *POLICY 2004*: 99-102
25. P. Miseldine (2004) 'Cloud Architecture Schematic' <http://www.cms.livjm.ac.uk/2nrich/deliverables.asp>
26. P. Miseldine (2004) 'JBel (Neptune) Application Development Guide' <http://www.cms.livjm.ac.uk/2nrich/deliverables.asp>