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
{ cmsmrand, a.talebbendiab, cmppmise, a.laws}@livjm.ac.uk

Abstract

The provision of complex software systems that are self-configuring, self-healing, self-optimizing and self-protecting is the basic tenet of autonomic system design. The realisation of such goals within a complex system is by itself a very complex theoretical challenge. That is, because of the variety, variability and scope of the behaviour, exhibited by such systems, it is unfeasible to model and thus predict every conceivable event or state that may arise. The complexity of the observed behaviours ought to be matched by the provision of a meta-framework that seeks to enable system self-governance as an instance of deliberative agency regulation. Whilst, much research is underway within the emerging area of autonomic computing, much remains to be done to achieve true autonomicity in large-scale computer systems including the development of systems engineering support for the design and lifetime management of autonomic systems. In this paper a viable system architectural model is introduced, which provides a blueprint for high-assurance systems and a meta-control model necessary for adjustable deliberation and autonomy of self-managing systems, which is underpinned by an Enhanced Belief-Desires-Intentions (EBDI) model. This paper starts with a brief description of the proposed viable systems architecture, which will be followed by the adjustable deliberative model -- EBDI and the use of the situation calculus to specify a given system's states (situations) of interest including; norm-based governance, delegation and deliberation logics. In addition, the paper presents a qualitative evaluation of the proposed theoretical models of adjustable deliberation through a software implementation of a grid-based medical decision-making system. Finally, the paper concludes with general remarks and a statement of further works.

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. Here 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. There are, for instance, some aspects of the system's operation that should be visible to the user while others should run transparently, such as the mundane and time consuming systems operation interactions. Thus, as the complexity of the systems outstrips the power of human ability to manage them, the systems themselves can automatically take care of the majority of the associated routine management tasks. So, for example, in future intelligent embedded applications, within domestic and personal appliances, a personal smart network of services can be easily composed, delivering what is needed when it is needed, to the user, without any physical or mental exertion by the user.

Currently design models of "autonomic systems" including the IBM blueprint [3] do not employ "cognitive"/deliberative autonomous behaviour to support autonomy, but rather use explicit managed autonomy via policies and rule sets that predefine and predict all extraneous behaviour at design time using Event Condition Action (ECA) constructs [4]. Whilst, this is a well tested design principle the authors argue that it is limiting and not scalable. Thus, in this paper we outline a design model for adjustable deliberation and autonomy based on well-established cybernetics design model (Sec. 2), formal deliberative reasoning and logic (Sec. 3, 4). A situation calculus specification is proposed to enable the understanding of the necessary architectures to model software self-adaptation through an Extensible Belief-Desire-Intention framework. In this manner normative, social and utilitarian intentions are considered in addition to the agents' solipsistic attitudes.

The paper is organised as a series of introductions to the concepts involved. Thus the next Section, 2, gives a brief overview of a Viable System Model (VSM) with the Viable Intelligent Agent-Architecture (VIA-A) and continues with a discussion of how best to realise autonomic computing within a VSM model blueprint using the idea of autonomous agents. Section 3 introduces

the Extensible-Beliefs-Desires-Intentions (EBDI) model as an intentional deliberative framework, for agent autonomy, leading to an overview of the situation calculus (Sec. 4) as a method of specifying the logical deliberation in a dynamic system. Section 5 introduces a case study, from medical informatics, and it is shown how the deliberation can be specified in the previously defined 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 JBel (Sec.5). The paper concludes with a brief summary and indication of future research direction.

2. The Viable System Architectural Model

Based on 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. As illustrated in Figure 1, the model identifies five identifiable Systems within a running system and no system can act in isolation:

- System 1 (S^1): is effectors space where the actual operations are carried out.
- System 2 (S^2): is the coordination of the operations.
- System 3 (S^3): is the 'fire-fighting' on the run management in real time.
- System 3* (S^{3*}): is an audit component for S^3 giving quality assurance.
- System 4 (S^4): is the development system for future scenarios.
- System 5 (S^5): is the coherence functions where plans are created and system identity is formulated.

This cybernetic perspective is utilised in the VIA-A model (Fig. 1) to consider each S^1 system as an intentional intelligent software agent.

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, without the need for a large and complex production system.

For this reason it is intended to view the systems in a VSM as a federation of cooperating and coordinating autonomous agents. Where "agents" are here considered to be active entities whose behaviour can be described and modelled in terms of mental notions such as knowledge, obligations to themselves and other agents, goals and abilities. The ability to make decisions without the intervention of human users is directly related to the level of autonomy of an agent [8].

Furthermore it is the reasoning process undertaken by the agent that is of major interest in this work. The decision-making capability of the agent emerges from the deliberation cycle taken to decide which action to perform next. The actions performed are actually based on many high level decisions such as whether the agent stays with its current plan to reach a given goal, whether it needs to change the goal, whether to follow a system norm or not, etc. The decision procedures are required to operate in highly dynamic environments. In these applications decision making 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].

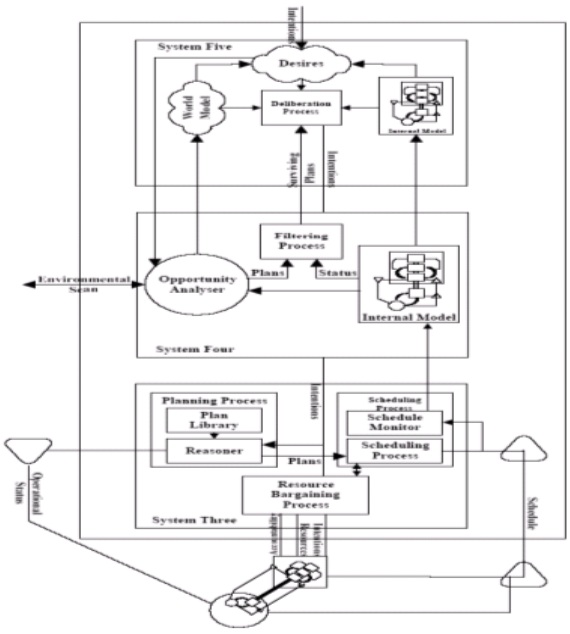

Figure 1. The VIA-A Model

Thus, it is proposed to define how coalitions of agents behave, within a deliberative intentional framework, where system norms describe the agents' 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 [10]. 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 [11]. The agent is not autonomous in any abstract sense but is autonomous with respect to another object such as other agents or the environment. [12] gives 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. 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], which models intentions as a filtering process. The agent has its own beliefs in an epistemic component and sets its personal 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.

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. Typically the high level plan will be formulated in the S^5 system of the VSM with the desires propagated through the system until the plan steps can be enacted at S^1 level. 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 proposed EBDI provides a highly suitable architecture for the design of situated intentional software agents that continuously monitor and/or observe their environment. 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. 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, and which goals to forward for inclusion in the potential intention agenda.
- 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. 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. There are three types of sort: *action, situation* and *object.* The *action* and *situation* sorts are the actions and situations that occur. The *object* sort is the elements of the domain in which the reasoning is taking place. Using these features the situation calculus takes the following form:

- A constant symbol S_o denoting the initial situation
- A function symbol do : *action* x *situation* $\rightarrow$ *situation* so that *do(a, s)* represents the successor situation resulting from doing action a in situation s.
- A predicate symbol $\subset$:*situation* x *situation* which orders the situations treated as histories of actions. Thus $s \subset s'$ is interpreted as s is a sub-history of s'.
- A predicate symbol *poss : action* x *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 $(action \cup object)^n$ x situation to denote relational fluents.
- Function symbols of sort $(action \cup object)^n$ x 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.

The foundational axioms are:

1. Situations ought to be unique That is two situations are the same if and only if they are the same sequence of actions:

$do(a_1,s_1) = do(a_2,s_2) \Leftrightarrow (a_1=a_2) \wedge (s_1=s_2)$

This means that two situations may be different yet assign the same truth-values to all the fluents. So the situation calculus constructed in this way does not formally equate a situation with a state. A situation is a sequence of actions.

2. An axiom of induction can be established. So if a proposition, P, is true in the initial situation, s_0, and if P is true in situation s means that P is true in situation do(a,s) for all actions a then P is true in every situation:

$$[(P(s_0))=T \wedge (\forall a \in action,\ s \in situation$$
$$P(s)=T \Rightarrow P(do(a,s))=T)] \Leftrightarrow \forall s\ P(s)=T$$

3. It is required that the situation S_0 is the initial situation before which no situations occur (giving a big bang situation).

$$\neg(\exists s \subset S_0)$$

This ensures that each situation is a history emanating from an initial scenario.

4. To give an ordering to situations if s is a sub history of do(a,s') then either s is a sub history of s' or s equals s' [$\forall s\ s \subset do(a,s)$ as do(a,s) is the successor situation to s]

$$s \subset do(a,s') \Leftrightarrow (s \subset s') \vee (s=s')$$

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. That is an action is initially specified by stating the conditions under which it can be performed.

$poss(A(x_1, x_2, \ldots x_n), s) \Leftrightarrow F_A(x_1, x_2, \ldots x_n, s)$ where $F_A(x_1, x_2, \ldots x_n, s)$ is an expression specifying the preconditions for the action $A(x_1, x_2, \ldots x_n)$.

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. So what is required is:

1. Axioms describing the initial situation S_0
2. A precondition axiom for each action.
3. A successor state axiom for each fluent.
4. A unique name for each action
5. Axioms defining which agent performs what action
6. The foundational axioms for the situation calculus given above.

5. Agent Federation

The deliberative structure required, allowing agent autonomy and the formation of agent teams, is necessary to provide the autonomic functions of the system. The formation of an agent federation is characterised by the concept of a Joint Persistent Goal (JPG). To have a joint intention to perform an action, *a*, for the federation mutual goal, the agent team will mutually believe one of three things in line with [24]

1. That *a* has been done.
2. That *a* is impossible to do.
3. That *a* is irrelevant

So, for example, service monitor agents comprise a persistent federation with a federal intention that requires them to locate and register any stranded application level services that were connected to a federation team-mate that is no longer available. There follows a number of consequences that must hold:

- An application level service that registers or unregisters with a service monitor agent must have this status mutually believed by the federation. Thus the service monitor agent has a commitment to bring about this effect.
- When an application level service is not connected to a service monitor agent the members of the federation have a joint commitment to connect that service.
- So when an application level service is not connected to a service monitor agent all federation members also have an individual commitment to register that service.
- When a service monitor agent successfully registers a service it has a commitment to make this fact mutually believed.
- When a service monitor agent that has a commitment to register a service comes to believe that the service has registered to another federation member it gives up its commitment to register the service.

When a service monitor agent discovers an unregistered service it must arrange for this to be mutually believed by the other federation members.

Firstly considering a single agent a persistent goal can be represented by:

$$Pgoal(p, do(a, s)) \Leftrightarrow (Pgoal(p,s) \wedge \neg(\neg B(p,s) \vee K(p,s))) \vee (a=setPgoal(p) \vee K(\neg p,s)) \quad (1)$$

with obviously $poss(setPgoal(p),s) = true$

where K is the defined knowledge operator in the situation calculus with the analogous belief operator B [25]. Obviously with only one agent these are K_1 and B_1, with the number of agents, n=1. This agrees with an intuitive notion that a goal persists while an agent doesn't know p to be true and doesn't know that p cannot be true. Replacing the belief operator with its corresponding formulation using the knowledge operator in (1) and using the sensing formulation in the situation calculus [26] and working through the expression gives an equivalent formula:

$$Pgoal(p, do(a, s)) \Leftrightarrow (Pgoal(p,s) \wedge (\neg K(\neg p,s) \wedge \neg K(p,s))) \vee (a=setPgoal(p) \vee a=sense(p) \rightarrow \neg p)$$

Now to extrapolate this to n agents in a multi agent setting it is necessary to think of the agent federation as acting as a single entity. However this is not sufficient as it must also be possible for any agent to become aware individually that a goal has been achieved or is impossible to reach. So when a federation member comes to believe a fact, regarding the goal, which entails the dropping of the commitment by the individual member, the federation must also drop the commitment. So any federation member, upon discovering that a commitment needs to be discarded, should raise a goal to make this fact known to all other federation members. Thus each federation member has as a sort of lesser goal one of three choices. Either it believes p not to be the case but

has p as a goal or it believes p does hold and has a goal to make this common knowledge or it believes p to be impossible and has a goal to make this common knowledge. So if we have a lesser individual goal, for each agent i, relative to the other federation members, $LIgoal_i$:

$$LIgoal_i(p, do(a,s)) \Leftrightarrow$$
$$LIgoal_i(p,s) \wedge \neg((K_i(\neg p,s) \wedge goal_i(p,s)) \vee (K_i(p,s) \wedge goal_i(C(p),s)) \vee (\neg B_i(p,s) \wedge goal_i(C(\neg p),s)))$$

where C is the common knowledge operator for the situation calculus defined for modal logic in [ref]. In this way each federation member doesn't assume that the other federation members have p as a goal but rather as a lesser individual goal. Thus any federation member may have discovered p has been achieved or is impossible and be in the process of making this common knowledge in the federation.

So a joint persistent goal can be established where a single agent belief can be replaced with common knowledge and lesser individual goal as appropriate.
e.g.

$$JPgoal(p, do(a,s)) \Leftrightarrow$$
$$JPgoal(p,s) \wedge (\neg(E(\neg p,s) \vee E(p,s)) \wedge (\exists i(LIgoal_i(p,s)))) \vee (a=setJPgoal(p) \vee E(\neg p,s))$$

E is the every one knows operator as defined for modal logic in [25]. Hence an agent federation is defined by the joint persistent goals that each team member communally adopts. Thus generalised goals can be considered according to the normative, utilitarian, axiologic and situational imperatives of the system.

The next section outlines an application of the proposed theoretical model of adjustable deliberative autonomy using an implementation of a grid-based medical decision-support system -- Clouds [27], in which self-governance logic modelled in the situation calculus is expressed and enacted through a bespoke meta-language, JBel [28] to give decision models for self-governance that can be inspected, modified and executed at runtime.

6. Case Study Implementation

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 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 service managers communicate via the S^2 system to post their application level service states. The system controller, S^3, maintains communication with other 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, 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 leading to intention commitment that need to be accomplished by the S^1 systems.

Beliefs correspond to service information derived from a range of sources, including domain, 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 service'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. Agent roles provide action triggers constituted through the R^5 component model to specify the requisite resources.

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.

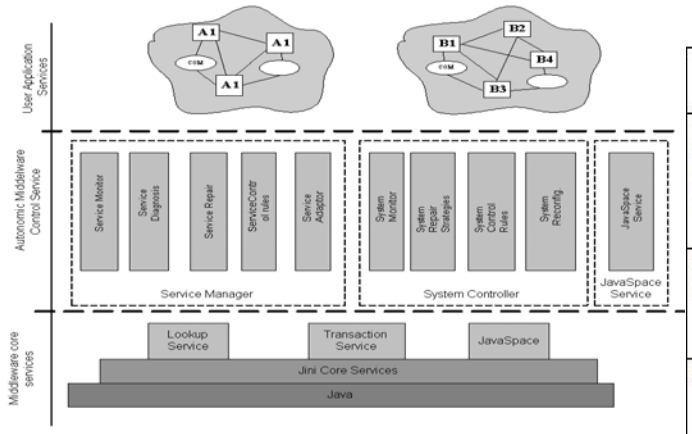

Figure 2. A Middleware Control Service

In this implementation and in accordance with [REF] 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.

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. S^1 provides the services and their management agents. S^2 provides the coordination by means of a system space so that service agent system state is revealed to other relevant systems. Systems $S^{3\&3*}$ provides the basis of system governance encompassing both the runtime management and the monitoring of governance/performance indicators. System S^4 gives higher-level system control features where interaction with the environment feeds the belief system for in the EBDI to set system and component reconfigurations, repair strategies and predictive facilities for system protection and resilience. System S^5 is the highest-level system controller and may comprise of the human designer. It is here that the role of the system is specified and the overall system purpose is defined.

It can be seen that the high-level system role is propagated through S^4 with environmental considerations to the S^3 management functions. The S^1 services are charged as a team, with fulfilling the system role and satisfying the desires through their committed intentions. The results are available to the upper management systems through S^2 with S^3 controlling the data flow through the entire system. Thus service states can be deliberated upon and strategic management executions propagated throughout the system.

Table 1 VSM mapping to Autonomic Components

VSM System Level	System Type	System Function for Autonomic Control	Typical Autonomic Control Service System Constituent
S^1 Service & Service Management Level	S^1 performs the productive operations of the system. It may be composed of a number of S's providing distinct services. Each S^1 consists of a managed operational element in contact with the environment.	To provide access to individual services via their managers, to monitor and post service level data and provide service control at a local level.	Application layer service agent and its meta-level management control
S^2 Service Management Coordination Level	S^2 coordinates the activities of the S^1 units. It provides coordinating facilities over all S^1 that can be locally optimized by the individual units.	Distributed shared memory to coordinate the S1 level services. Each service manager posts its service state to be read by the system controller or other service managers.	A distributed system space using tuples Space such as a JavaSpace to store S^1 service states
S^3 System controller Level	S^3 brings cohesion and the option of federated behaviour to the S^1 units. It is the immediate management for the system using the existing resources.	Entire system monitoring also made available to S^4. Managing reconfigurations as required by S^4 deliberation. System repair	System Controller
S^{3*} System Controller Audit Level	S^{3*} provides an intermittent audit of S^1 units. It provides additional data to the normal monitoring.	Provides the auditing level so that quality of service and process can be ascertained through defined indicators of governance.	System Controller Subsystem
S^4	System 4 monitors environmental changes and system capabilities to plan for future trends and system functionality	Monitors the environment to make system regenerations and reconfigurations as necessary together with S3. It uses past data to align the system with predicted trends based on histories. An action history (situation) can trigger such events.	System Controller. Repair Strategies, Resolution Strategies and deliberation.
S^5	System 5 determines the purpose of the whole system, defining the tasks of the S1s. The norms of the system are formed via the sensory actions at S4	Sets the system goals in collaboration with S4 (and S3 by association)	System Controller. High-level goals (desires) and intention setting. (Maybe human controller)

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. In this way the agent team is defined by the previously stated logical expressions for agent federations, via joint persistent goals, with the domain goals and intentions for the team members set as follows below.

```
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 3. A Situation Calculus Representation

Similar representations of self-governance rules have been derived to give a complete specification and are used in the resulting implementation described below.

A self-regenerative architecture, called Clouds, is defined to enable the self-adaptive framework to function using a bespoke scripting language, JBel that allows management objects to be compiled and inspected at runtime. The autonomic control service has been implemented as a Cloud system. JBel 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 JBel 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.

JBel objects are executed on demand through an event model exposed by the clouds architecture.

The clouds concept is conceived as a system with undefined 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 JBel objects, derived from the EBDI deliberative process as exemplified in fig. 3. 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.

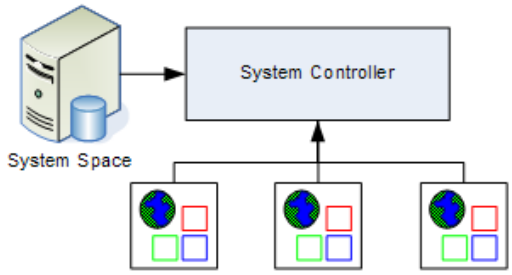

Figure 4. System Controller in Charge of System Space

In its first role (Fig. 5), the controller marshals and inspects calls to and from the System Space from services. Services use methods of the controller to register their presence, and update their state, as well as requesting brokerage of service dependency links. As such, services within the Cloud can consume the controller as a service itself meaning they can be developed and hosted in a platform-neutral manner.

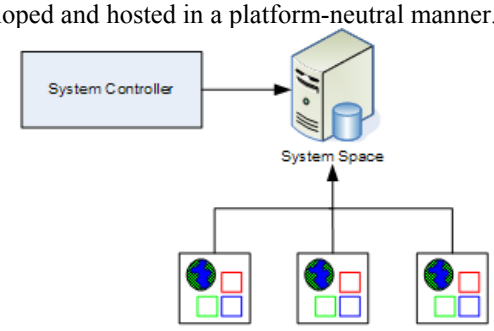

Figure 5. System Controller as System Space Monitor

In its second role (Figure 6), 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. As the System Controller is no longer marshalling calls from the services to the System Space its scalability is increased. This configuration may be suitable for Clouds with a large number of services to ease the workload on the System management. However, services must be written to inherit the controllers encapsulated functionality, such services are tied to the domain platform used to develop the Clouds.

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. Due to the generic form of these meta-objects, they provide an attribute system for services that can be queried inside JBel scripts just as an object can be queried through its published properties in traditional object orientated software

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 resource to request read and write access to the properties of a service through the guard of the system controller.

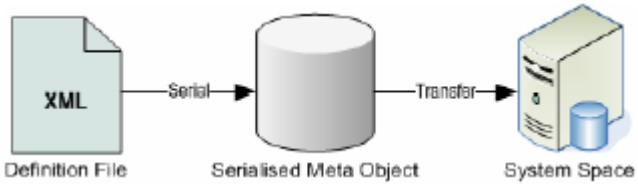

Figure 6. The XML Space

In addition to the meta-objects exposing properties of a service within the Cloud, they also describe events that can be fired, caught and handled. The event model begins by the service informing the System Controller when an event is fired, which itself marshals this event to the System Space to provide the appropriate scope. Other systems within the Cloud that have a role assigned to an event instruct the controller to inform them via message when an event is fired, which the controller duly does when the event message is received. The events themselves can pass XML files containing specific information about the event at the time of firing. For example the event occurrence, when fired, can include an XML document containing the actual data that is new to the service. This is passed along with the event message and is exposed to all agents who have declared an event handler with the controller. It should be noted however, that the event model is abstracted from the agents within the system, and is controlled by the JBel scripting language that sends and receives the appropriate event calls to the controller.

The JBel scripting language is structured in terms of rules, conditional statements and variable assignments that are parsed from script form into software system objects, encapsulating all the logical inference processes and variable instantiations. Thus allowing the JBel 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 JBel objects that can be modified as a result of agent deliberation on system events.

For example in the situation calculus representation shown (fig. 3) 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. This is shown in Figure 8:

```
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 7. JBel Repair Strategy Script

7. Conclusion and Further Research

It has been shown 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 is directly mapped into the scripting language JBel 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 evolvement 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/enrichii/

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. Poulovassilis, 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: 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. S Kumar, P. R. Cohen, and H. J. Levesque (2000). 'The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams'. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*, Boston, MA, USA, July 7-12, 2000, pages 159-166.
25. J.Y. Halpern, Y. Moses (1996) 'A guide to completeness and complexity for modal logics of knowledge and belief' Reprinted from Artificial Intelligence 54 1992 pp311-379
26. Yves Lespérance, Hector J. Levesque, Fangzhen Lin, Richard B. Scherl: Ability and Knowing How in the Situation Calculus. Studia Logica 66(1): 165-186 (2000)
27. P. Miseldine (2004)'Cloud Architecture Schematic' http://www.cms.livjm.ac.uk/enrichii/deliverables.html
28. P. Miseldine (2004) 'JBel application Development Guide' http://www.cms.livjm.ac.uk/enrichii/deliverables.html