Model-Based Self-Managing Systems Engineering

A. Taleb-Bendiab
School of Computing, Liverpool John Moores University, UK
cmsatale@livjm.ac.uk

D.W. Bustard
School of Computing, University of Ulster, UK
dw.bustard@ulster.ac.uk

R. Sterritt
School of Computing, University of Ulster, UK
r.sterritt@ulster.ac.uk

A.G. Laws
School of Computing, Liverpool John Moores University, UK
cmsalaws@livjm.ac.uk

F. Keenan
Dept. of Maths & Comp Dundalk Institute of Technology, Ireland
Frank.keenan@dkit.ie

Abstract

For many years, the vision of smart computing where systems can function and/or manage themselves independently from human intervention has provided numerous theoretical challenges to research communities ranging from intelligent systems and cybernetics to AI communities. These research trends have now been further fuelled by the IBM autonomic computing initiative, where biologically inspired concepts inform the development of systems that can adapt autonomously to their users' requirements and environments. This paper considers the extent to which well-established general systems concepts might be valuable in the design of autonomic systems. The main two approaches considered are Checkland's Soft Systems Methodology (SSM) and Beer's Viable Systems Model (VSM). The paper summarizes the relevant aspects of each approach and demonstrates their potential through the provision of an illustrative case study. Moreover, the paper illustrates how SSM and VSM approaches facilitate autonomic systems engineering by the capture of functional and non-functional application requirements such as lifetime self-management policies and operational tolerances.

1. Introduction

Recent developments in software design involve the concept of autonomic computing capable of self-organization, adaptation, control and management [1]. Here, complex computer systems, offering users intuitive interaction with the system, without any involvement in the systems running, can effectively manage themselves. Thus, as the complexity of systems outstrip the human ability to manage them, so systems themselves can automatically take care of the majority of associated mundane management tasks. Previous work has focused on the design of autonomic elements in a reductionist manner. Here, autonomic components are analyzed as separate communicating systems within the overall system. The intention here is to consider autonomic concepts as integral to the whole system. A systemic view is taken to produce a procedure for the modelling and development of an autonomic computing system.

Soft Systems Methodology [2] is a well-established process to elicit user requirements and system functionality. Likewise, the Viable System Model [3] has been widely used and credited with providing basic robustness to a system structure. Although both methods are concerned with human systems, it is thought that software systems, with autonomic capabilities, are susceptible to the analysis afforded by these system models.

A range of notable research work related to autonomic computing has adopted control theory [4, 5], AI-based planning [6] and/or software reflection techniques [7] to provide application-level self-adaptive mechanisms and/or heuristics. Here the focus is on software engineering concerns including; a generative programming model and/or software engineering support for finer-grained dynamic and predictable software adaptation. Using an architecture-driven approach incorporating probes and gauges enables the software to interact with the executing system and collect raw measurement data for translation into suitable metrics for system performance tuning and/or error recovery through adaptation.

Much insight into system meta-control and management has been developed from policy-based management and context-awareness systems. This enables systems to operate independently of direct human control yet remain in harmony with their users command and controls settings. However, very little work has focused on the application of systemic approaches to model and capture stakeholders concerns and requirements and so provide a guiding framework for the development of the adjustable autonomic behaviour to be exhibited by target computer applications.

The remainder of the paper is organized as follows. An introduction to the VSM and SSM and where they fit into the autonomic computing paradigm is presented in the following sections. This is followed by the specification of an approach to bring SSM and VSM to an iterative autonomic computing system design process. This is illustrated with a practical, currently running implementation. Finally, the paper concludes with a discussion of the contribution made by this work and the future development opportunities it affords.

2. A Self-Managing System Architecture

The Viable System Model [3] provides a theoretically supported cybernetic model of organization. Viable systems may be defined as being robust against internal malfunction and external disturbances and have the ability to continually respond and adapt to unexpected stimuli. The model specifically attempts to imbue the system with the ability to adapt to circumstances not foreseen by the original designer and identifies the *necessary* and *sufficient* communication and control systems that must exist for any organization to remain *viable* in a changing environment. The major systems (i.e. S1s, S3, S4 and S5) are structured hierarchically and connected by a central 'spine' of communication channels passing from the higher-level systems through each of the S1 management elements, as shown in . These provide high priority communication facilities to determine resource requirements, accounting for allocated resources, alerts indicating that a particular plan is failing and re-planning is necessary and the provision of the "legal and corporate requirements" or policies of the system.

The systems shown in Figure 1 concern the management structure at one level of the system, and consequently specify the communication and control structures that must exist to manage a set of S1 units. However, the power of the model derives from its recursive nature. Each S1, consisting of an operational element and it's management unit, is expected to develop a similar VSM structure, consequently, the structure of systems is open ended in both directions and may be pursued either upwards to ever wider encompassing systems or downwards to ever smaller units. However, at each level the same structure of systems would occur although their detail would necessarily differ depending on context.

The value of assuming such a viewpoint is in the immediate provision not only of the outline architecture that the autonomic software system itself must assume, namely that of the Viable System Model, but also the identification of the requisite communication links to bind the system to the organization.

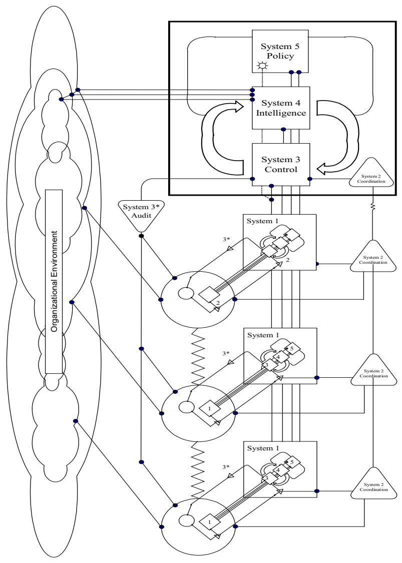

Figure 1. The Viable System Model [3]

We now extend and apply this cybernetic approach and consider an S1 of the VSM in terms of an autonomic software system. To demonstrate, a conceptual, architectural outline of such a system is determined, using both the principles of the VSM and the terminology and design of a classical Artificial Intelligence design, namely Bratman *et al.*'s Intelligent Resource-Bounded Machine Architecture (IRMA) [8] as a constructional guide. As shown in Figure 2, the developed J-Reference architecture embeds a Beliefs, Desires, Intentions (BDI) unit at the S5 level representing;

- Desires - or what the agent wants to do and is taken as a given for the moment.
- Beliefs - or what the system currently knows and is represented by two structures. A model of the external world and a model of the current internal status of the architecture.
- Intentions - or what will actually be done, is determined by a process of deliberation, which interprets desires in the light of current beliefs about both the environment and the 'stance' of the system.

S3, using a reasoning process supported by a plan library and the capacity to audit the current status of operational S1 units, structures the intentions into plans, these are then passed to a scheduling process. The scheduling process, in cooperation with a resource bargaining process, responsible for negotiating resource deployment and usage monitoring, schedule the enactment of the plan. The schedule passes to the coordinating S2 channel for dissemination to participating S1 elements.

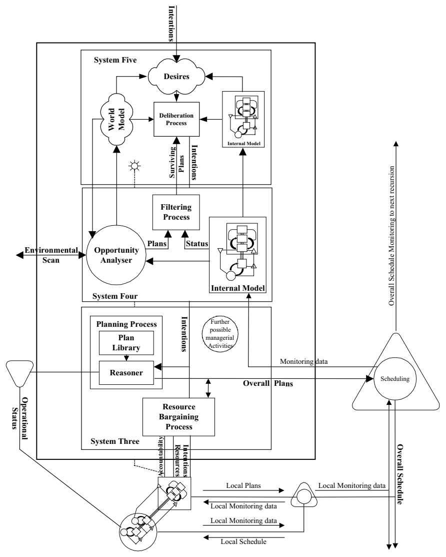

Figure 2. The J-Reference Model

Environmental change is addressed by S4, which equipped with an Opportunity Analyzer and guided by the S5 desires model, scans the environment for detrimental events or beneficial opportunities. There are two outcomes of this process, the first is the formulation of a view of the outside world which is provided to S5 in the form of the World model. The second outcome is the production of development plans for the future of the system, either exploiting advantageous opportunities or avoiding detrimental occurrences. Plans are then passed to the deliberation process to begin the intention forming cycle again.

As noted above, the power of this approach lies in the recursivity of the underlying model. Figure 2, indicates that the entire architecture described above is repeated in the client S1 unit in the next layer. Consequently, the intentions channel at one recursion informs the desires model in the next, thus allowing an autonomous response to local conditions at each level while remaining within the purpose of the overall organization.

3. SSM and Autonomic Systems

Soft Systems Methodology (SSM) seeks to utilize the basic principles of systems thinking to resolve soft or poorly defined problem situations [2]. Here it is proposed to bring SSM into the problem domain of specifying self-governing distributed software systems in line with the IBM autonomic computing paradigm. A full description of SSM is beyond the scope of this paper, however a brief diagrammatic outline of the seven stages of SSM is provided by Figure 3 below. The interested reader is directed to the references provided at the end of the paper for further details.

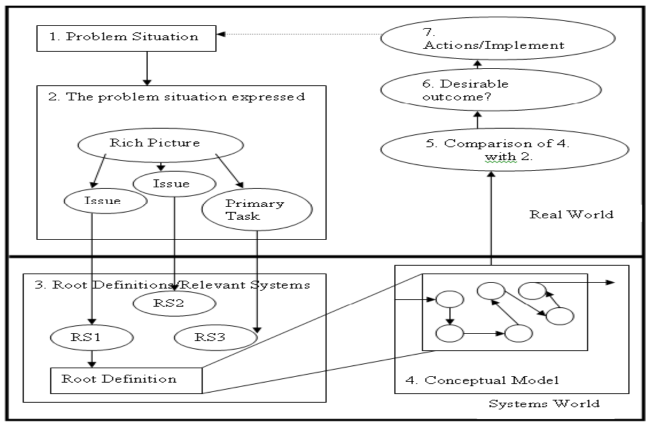

Figure 3. An SSM Process Summary.

4. A “Lean” SSM and VSM-Based Process

To design an autonomic system or imbue a distributed system with autonomic functionality via an SSM approach, it is first necessary to produce a clear picture of the components in the system (including the human users) and the interactions they have with each other. This picture ought to raise the issues that need to be addressed in order to produce the autonomic system. For instance, a certain component that must always be available to the system may be identified as a specific issue. Such issues then lead to the development of a relevant system for each issue. So, in the example, a system is conceived that always keeps the specified component available with the root definition:

An autonomic system, which under the following environmental constraints <list domain properties> transforms an input <component unavailable> into this output <component available> by means of the following activities <retry component, if component retried then seek alternative component service and enact component repair or replacement routine>. The transformations are carried out by these components <component manager, system controller> and directly affect <clients>. The concerns that make this transformation meaningful contain these elements <autonomic function, reliability, availability etc.>.

This root definition would then be used with other root definitions to form the conceptual system with deliberation techniques logically defined. Then, the resulting systems’ architectural positions would be defined, in the whole overall system, by their classification according to the VSM model.

Figure 4 illustrates a kind of “agile” systems’ engineering process, which will be necessary to define and develop a required autonomic system including its lifetime self-management capabilities. The requirements are captured through the SSM cycle and embodied in a VSM-based conceptual model resulting in an evolvable system. This will generate an abstract baseline systems architectural model together with its associated systems’ governing norms including; rules, policies for self: -management, -healing, -configuration, -tuning and -protection.

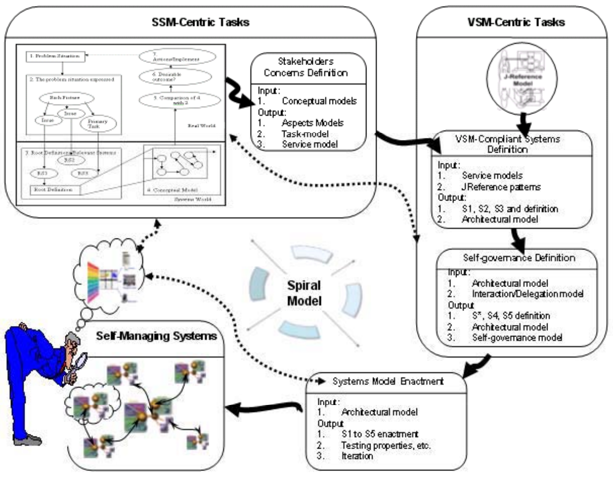

Figure 4. "Lean" ways in which they might be used.

The Normative units S4 and S5 [9] provide a deliberative and intentional meta-system, which regulates and controls the running of operational systems (S1). The deliberative process specifies the behavioural output via the cooperation and coordination of system services.

The process is envisaged as an iterative/spiral model and can use a variety of techniques including; aspect-oriented requirement capture, service-oriented architecture and generative programming. The process can be described as follows:

- Phase I: This follows the SSM cycle (Step 1-7) to define the conceptual model of a considered autonomic system.
- Phase II: In line with the separation of concerns design principle, different functional and non-functional system goals are separated under different aspects.
- Phase III: a user task model will be listed. From which a software service model can be generated through task to service mapping.
- Phase IV: Following the VSM-model, S1 (operational systems) then S2 and S3 etc. are defined.
- Phase V: The self-governance is defined, using the appropriate pattern, obtained by capturing the rules from the logical model.
- Phase VI: Validation of the model and refinement.
- Phase VII: Systems generation and deployment.
- Phase VIII: Systems testing, and policy deployment, etc
- Phase IX: runtime adaptation if and when necessary. This might require refactoring, etc.

5. Evaluation

To evaluate the proposed abstract agile design process for autonomic systems engineering, this section outlines the design of an implemented grid-based medical decision-making system.

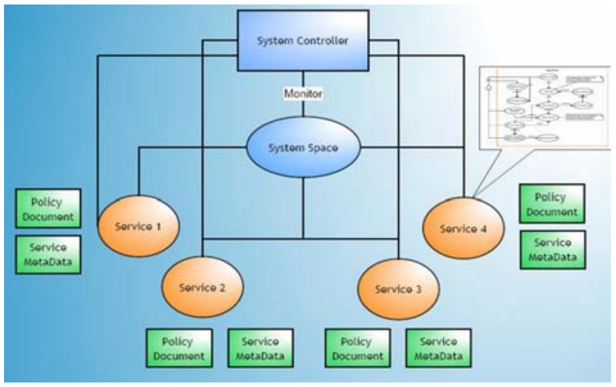

Figure 5. The Service Oriented Architecture of the Cloud Application.

As illustrated in Figure 5, the implementation is based on a “Cloud” framework [10], in which the notion of a Cloud represents a federation of application services and/or computational resources regulated by a system controller, which comprises the S3/3*/4/5 control, and discovered and coordinated services respectively. As illustrated in Figure 5, the Cloud’s coordination and communication is achieved via the shared memory, which maps to an S2 level function.

In this example, all S1 level units Services 1 to 4 are medical domain specific applications such as; decision-tree, medical data access, each of which exposes some of their states and their identity, roles to other services in the same Cloud by publishing their metadata and policy documents via the System Space.

Rather than expressing autonomic norms through traditional rules, Clouds uses a custom designed meta-language, JBel to define and deploy the systems’ governance norms (policies) and decision models as compiled objects that can be inspected, modified, and executed at runtime.

The example System Controller JBel script below shows a simplified load balancing norm, and an application service "hot-swapping" norm (rule) to enable runtime cancer decision models (services/agents) to be "plugged in" on-demand and/or when the Cloud discovery service detects a new version of a user required decision model service. The full description of the JBel language and/or the Clouds architecture is beyond the scope of this paper and can be found in [10].

```
Rule Load-balancing
if (service.niceguidelinemodelA.cpuload >
service.niceguidelinemodelB.cpuLoad)
     delegate-
Call(service.niceguidelinemodelB)
end if
…..
Rule hot-swap
if (service.required = new)
 hotswap(service.required)
end if
```

6. Conclusions

In this paper, the authors have described the development and use of an agile, system-centric engineering process for the development of autonomic software systems. This uses a soft systems approach at the outset allowing a robust task model to emerge, which incorporates user viewpoints and establishes system policies and access rights. Applying the conceptual underpinning of the VSM and the technical blueprint supplied by the J-Reference model to that output results in the full architectural specification of the system. The self-governance aspects of the system, identified in stage one, are further refined and instigated as a situational calculus. The operational system can then be generated, deployed and adaptively refined.

A brief demonstration of these notions was presented in a case study of the development of a medical decision-support system. Certainly, more work is required to both further elaborate and detail the process and undertake a larger scale evaluation exercise before the true value of this contribution can be ascertained. However, the approach presented makes significant progress in determining a realizable, norm-driven autonomic system architecture that both clarifies and, in some respects, extends the autonomic vision [1], particularly in the higher-level, cognitive/deliberative elements of our model.

7. Acknowledgements

This research was supported by the Centre for Software Process Technologies (CSPT), funded by Invest Northern Ireland (INI) through the Centres of Excellence Programme.

Further support for this research is provided by EPSRC-funding Clouds project at JMU. The authors would also like to acknowledge the significant contributions made by both Philip Miseldine and Martin Randles to the work presented.

8. References

[1] IBM, *Autonomic Computing,* http://www.research.ibm.com/autonomic, 2002.

[2] Checkland, P., *Systems Thinking, Systems Practice*, John Wiley & Sons, Chichester, 1981.

[3] Beer, S., *Brain of the Firm*, 2nd ed, John Wiley & Sons, Chichester, 1981.

[4] Kokar, M., K. Baslawski, and Y. Eracar, *Control Theory-Based Foundation of Self-Controlling Software,* IEEE Intelligent Systems, Vol. , No. pp. 37-45, 1999.

[5] Reilly, D., A. Taleb-Bendiab, A. Laws, and N. Badr, *An Instrumentation and Control-Based Approach for Distributed Application and Management,* in *ACM SigSoft Workshop on Self-Healing Systems (WOSS'02),* Charleston, SC, 2002.

[6] Robertson, P., R. Laddaga, and H. Shrobe, *Introduction to the 1st International Workshop on Self-Adaptive Software,* in *1st International Workshop on Self-Adapative Software,* Oxford, UK, Springer-Verlag,2000.

[7] Costa, F.M., *et al.*, *The Role of Reflective Middleware in Supporting the Engineering of Dynamic Applications*, in *Lecture Notes in Computer Science*, Springer-Verlag: London,1999.

[8] Bratman, M.E., D.J. Israel, and M.E. Pollack, *Plans and Resource-Bounded Practical Reasoning,* Computational Intelligence, Vol. **4**, No. 4, pp. 349-355, 1988.

[9] Laws, A.G., A. Taleb-Bendiab, S.J. Wade, and D. Reilly, *From Wetware to Software: A Cybernetic Perspective of Self-Adaptive Software*, in *Self-Adaptive Software: Applications, Second International Workshop on Self-Adaptive Software*, R. Laddaga, P. Robertson, and H. Shrobe, Editors, Springer-Verlag: Berlin,2003.

[10] Miseldine, P., *JBel Application Development Guide,*School of Computing & Mathematical Sciences, Liverpool John Moores University, 2004.