

Viability Systems: The Control Paradigm for Software Architecture Revisited

Charles Herring and Simon Kaplan
*Department of Computer Science and Electrical Engineering
The University of Queensland
Brisbane, Queensland QLD 4072
{herring, kaplan}@dstc.edu.au*

Abstract

*We identify an emerging class of software application as “complex” systems. They are complex in that they must adapt to a changing environment. This motivates us to revisit the “Control Paradigm” for software architecture. In this paper we go beyond that approach and introduce the concept of **viability** as the overall characteristic of the behavior desired in such systems. We present an architecture to guide in software engineering of this class of complex system. The architecture is based on a Cybernetic model called the *Viability System Model*. As an application of the approach we are developing a Smart Lecture Room. We report on our first efforts in employing the architecture to develop this application.*

1. Introduction

Computing and communications technologies are converging to form a world wide system-of-systems that will soon result in a global integrated information infrastructure. A host of new applications are on the horizon. Examples include ubiquitous computing, heterogeneous wireless networks and smart environments. This rapid convergence of existing systems and the proliferation of diverse embedded devices are motivating a new class of software application. We refer to these as **complex** applications to distinguish them from the familiar, algorithmic-like applications such as accounting systems.

Some characteristics of a complex system that distinguish them from the more static type of application are:

- Large numbers of heterogeneous components
- High degree of interconnections, relationships and dependences
- Dynamically changing environment and *changing internal composition*.

Given these characteristics we ask: What is the overriding capability or quality such an architecture must provide? It seems to us it is the ability to address the changing environment in which these systems are embedded. This led us to revisit Shaw’s “control paradigm” for software architecture [1]. The control paradigm proposed by Shaw was based on the insight that the problem at hand (cruise control) was a control problem. That is, the key characteristic of the application is one of maintaining system stability in a changing environment and that any suitable software architecture must take this into account.

In the time since Shaw’s article was written this idea has become even more important. To that end we searched for control theoretic models that might serve as the basis for a software architecture for complex systems. We chose a Cybernetic control model called the *Viability System Model* (VSM) [2] as the basis for our software architecture. This model goes beyond the original control paradigm, as it is an “adaptive control” model. The objective of this approach is to develop a model and methodology through which complex adaptive software systems can be dynamically assembled and evolved. Software with this capability we call a **Viability** system: it is sufficiently self-aware that it can adapt and survive in a changing environment.

The organization of the paper is as follows. We review the control paradigm approach as background and motivation. Then the concept of viability is introduced and the properties of a viability system are described. The basic structure of the VSM as an adaptive control system is given. As an example of a complex system we are building a Smart Lecture Room. This serves to illustrate how the VSM is used as a software architecture. We outline the software engineering steps to apply the architecture and discuss our first efforts in building a Smart Camera controller based on the approach.

2. The Control Paradigm for Software Architecture

Shaw [1] analyzes the “canonical” object-oriented design problem – the automobile cruise control. Her fundamental insight is that *it is a control system* and there is a body of engineering knowledge associated with such systems. She argues that the object-oriented paradigm may not be the most suitable approach in this case. She develops a software architecture, the “control paradigm”, based on classical feedback control theory as opposed to one based solely on identification of objects and their relationships. A diagram of the cruise control system is shown in Figure 1.

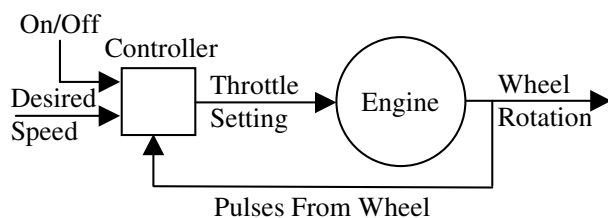


Figure 1. Control Architecture for Cruise Control

She notes the control paradigm separates the *plant* (the Engine in this case) of the main process from the compensation for external disturbances, the *control*. This separation of concern yields appropriate abstractions that lead to design issues that might otherwise be missed in the (domain neutral) object-oriented approach. In particular, performance and correctness constraints are identified early in the design process.

Shaw offers the following “rule of thumb” for use of her control paradigm architecture: *When the execution of a software system is affected by external disturbances, forces, or events that are not directly visible to, or controllable by, the software this is indication that a control paradigm should be considered for the software architecture.*

One of the main reasons problems become difficult is they are formulated in a manner that frustrates solution. This is what Shaw is saying. The object paradigm is not the correct way to analyze this problem. Software architecture is concerned with the fundamental components that make up a system, how these components relate to each other and how the system will evolve over time. Object analysis does not lead to a suitable architecture for this particular problem. However, it may be a useful aid in subsequent steps of implementing the software architecture.

In analyzing the cruise control problem Shaw recognized that control theory was the most appropriate context for the problem. *We claim her rule of thumb is the general case for “complex” software systems.* We

maintain that these types of software systems should be designed to adapt to external disturbances. Control theory in general and Cybernetics in particular offers an approach to doing this. If the underlying theory and principles of Cybernetics are in fact general, then there is no choice – they cannot be avoided in any non-trivial information processing system. For example, we have shown that the most successful design pattern, Model-View-Controller, is a closed-loop control system [3].

3. The Viable System Model

Our model of a viable system is based on work from cybernetic control theory on *viable systems*, due to Stafford Beer [2], specifically, his *Viable System Model* (VSM) [2]. (A thorough literature review of the VSM is available at [4].) The VSM provides a high-level architectural framework for complex adaptive systems. In this model, a set of components (the *plant*) is *controlled* by a set of meta-components (the *controller*), which monitor the behavior of the plant using various feedback protocols. The resulting information is used to adapt the system to changing conditions in the environment. A viable system is stable precisely because it can adapt. In this section we describe the properties of a viable system and then the structure of the VSM.

3.1 Properties of a Viable System

Adaptation can take several forms, and is related to internal variables or to system structure. *Variable*-related adaptation – called *homeostasis* – is the maintenance of critical variables in the plant within certain limits to maintain the stability of a system in response to changes in the environment. *Structural* adaptation can take two forms. *Morphostatic* systems maintain the integrity of their internal component structure in the face of environmental change, usually through homeostasis. *Morphogenetic* systems maintain meta-properties of the system in the face of environmental change, through evolution of the structure and/or components of the plant. A morphostatic system might, for example, change its internal behavior to maintain a particular structure, while a morphogenetic system might change the structure. In short, an “adaptive” controller for a system implements certain homeostatic, morphostatic or morphogenetic policies to manage (maintain the stability of) the plant comprising that system, and the viability of a system is a measure of how well these policies can be realized in a particular environment.

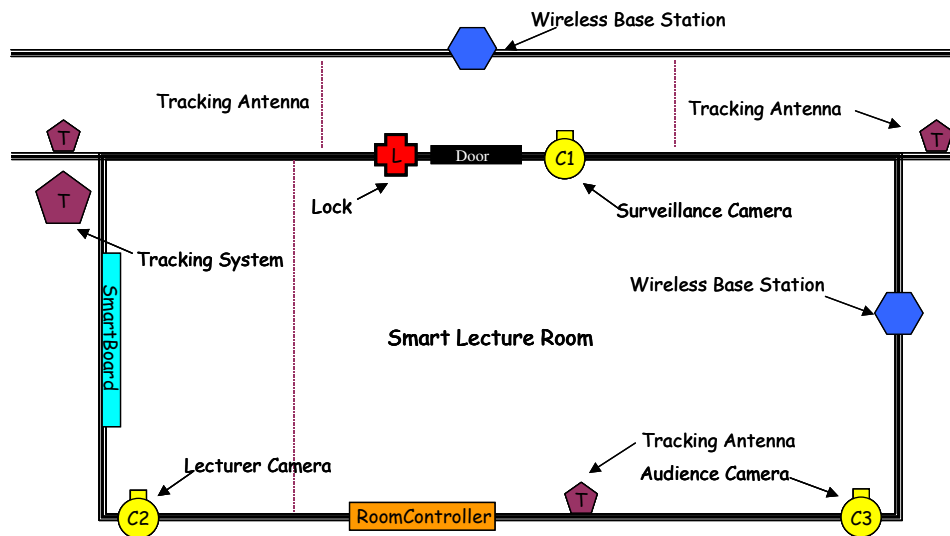


Figure 3. Smart Lecture Room

3.3 Viable Software Systems

Our goal is to use the VSM as a high-level software architecture. However, not all systems need to exhibit all these kinds of adaptability. Some systems might be homeostatic only, while some might exhibit a mixture. Nor is it necessary that the kinds of adaptability exhibited by a system are static; a system might be only homeostatic under some circumstances, but become morphogenetic under others. Some current software systems already exhibit these properties. The great strength of the system of network infrastructure and protocols supporting the internet, for example, is that it exhibits both morphostatic (the network can adjust to different loads because it can sacrifice packet delivery quality) and morphogenetic (the network structure can be extended, by introduction of new routers, to maintain throughput) properties, which the systems administrators can use to their advantage as necessary. This makes the Internet a highly viable network. However, viability as a property of software systems is by far the exception rather than the rule.

4. Smart Environment Example

We are building a “smart environment” as part of the Information Environments program at the University of Queensland. We have found the smart environments research area to be a nexus for technologies and disciplines: both old and new, including human-computer interaction, embedded devices, ubiquitous computing, networking, wireless systems, and any number of specialized applications such as image and voice

processing. This provides a rich setting and a significant challenge to software architecture. Much of the early work naturally focused on individual elements for specific purposes such as Active Badges, tracking, image processing to recognize human gestures, etc. Now we are assembling these disparate components into a synergistic system of systems based on our VSM architecture.

4.1 The Smart Lecture Room

The Smart Lecture Room is one element in our overall environment and is shown in Figure 3. There is a hallway shown at the top with the room below. There are a number of systems installed in the hallway and room that contribute to the smart environment. The hall contains a Tracking System and antennas. There is also a Wireless LAN running throughout the building. People using this facility have Active Badges that periodically broadcast a signal that is sensed by the tracking system. This supports tracking the location of people within the building. There is a door opening into the room from the hall, with a Lock. Outside the room and near the door is a Surveillance Camera. There is a RoomController that coordinates the action of the other components. These include a Lecturer Camera that focuses on the lecturer and based on gestures and voice commands direct the SmartBoard presentation system. There is also an Audience Camera that monitors the audience. In the figure, the dotted vertical lines are virtual boundaries. These are boundaries registered with the tracking system to alert other components within the room as people cross these boundaries.

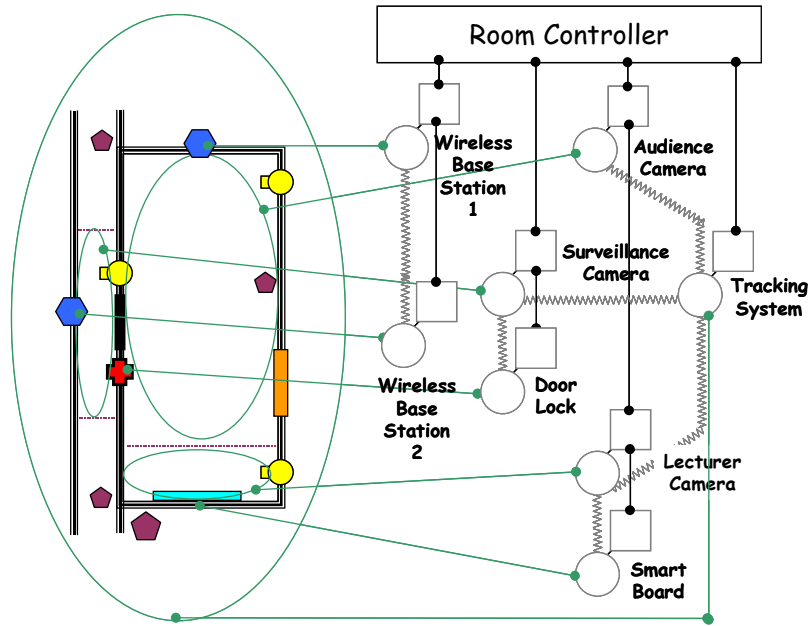


Figure 4. Viable Component Model applied to the Smart Room

Now we consider the operation of the smart environment as people enter into the system. As they walk down the hall and cross the virtual boundary the Tracking System notifies the Surveillance Camera. The Surveillance Camera begins monitoring them. If they approach the door, the Surveillance Camera will try to identify them based on its face recognition software. It consults a database of facial images it has been given by the Room Controller as authorized to enter the room. If positive identification is made the Surveillance Camera sends a message to the Lock opening the door. In this way a number of people and the lecturer enter the room. The Lecturer goes to the front of the room (crossing the virtual boundary) and begins the presentation. When he crosses the boundary, the Tracking System notifies the Lecturer Camera. The Lecturer Camera focuses on the lecturer. It identifies the lecture and automatically downloads the presentation for the talk. It also recognizes gestures and voice commands that are used to operate the SmartBoard presentation system. The Audience Camera monitors the students. Here there are number of conditions, based on the purpose of the class (lecture versus exam) that the camera's software can detect, record, or bring to the lecturer's attention.

Based on our VSM architecture it is easy to continue to add complex components into this already complex setting. For example, the students all have wireless PDAs that are dynamically added to the buildings wireless network as they enter. The students can communicate with each other (or not depending on the situation) and project assignments onto the SmartBoard.

4.2 Viable Smart Room Architecture

Now we show how the VSM is used as a high-level software architecture. Figure 4 is a simplified VSM diagram for the room. The Room Controller organizes the sub-systems into a viable system, implementing overall adaptive control management functions to achieve the synergistic behavior of the room. There are many operational sub-components: three cameras, lock, presentation system, tracking system and two wireless base stations. (Not shown in the diagram, but included are the HVAC and lighting system controls.) The diagram shows the basic structure and relationships of the Room Controller, its operational components and the environment. The vertical lines between the Room Controller and the operational sub-components represent the feedback loops (interfaces) for communicating policies, plans, commands, control instructions and resources. The squiggly lines between the sub-components indicate they have a direct relationship with each other. For example the Surveillance Camera has a working relationship with the Lock and the Tracking System. Finally, the environment in which each system is embedded is shown as ovals, with links to the relevant plant.

Now we illustrate the adaptive behavior we intend to gain by this approach. Consider the situation where an unusually large number of people turn up for a meeting. This places stress on several sub-systems that were either programmed for different conditions or originally sized for lower demands. The RoomController is receiving status reports from all of the sub-components and running

a simulation of the room. It uses the simulation to anticipate the heating and cooling needs and sends control information to the HVAC system. This is classical control of internal variables within pre-defined limits. As the large number of wireless users begins to make demands on the available bandwidth, the base station controllers change the slice-allocation algorithm to provide constant bandwidth to certain users (the lecturer) and at the expense of the other users (students). This is an example of substitution of behaviors to maintain stability. Finally, as even more people enter the room, the RoomController realizes that more wireless bandwidth is required that can be supplied by its base station. It asks the LevelController for use of the base station in the hallway. The LevelController transfers use of the hallway base station to the RoomController. This shows changing structure to adapt and how dynamic component assembly is accomplished.

Thus, the smart room example illustrates how the VSM organizes many different devices, made by different vendors, into a complex adaptive system. The room controller used information from these devices to make decisions. The controller began running a simulation when it realized the number of people reached a critical threshold for the communications sub-system. The results of the simulation indicated that additional resources were required. These resources were obtained from its controller and reassigned for the duration of the meeting. This example illustrates the range of adaptation in this approach:

- Homeostatic: the room adjusts heating and lighting to changing loads.
- Morphostatic: the base stations change their slice allocation algorithms.
- Morphogenetic: the RoomController acquires a new resource, the wireless base station in the hallway.

4.3 Implementing the Smart Room

Here we sketch out the steps in a software engineering methodology based on the VSM. As our approach is grounded in control theory, there is a large body of literature available on designing controllers that we can make use of [5-7]. The basic steps in a software methodology to implement a VSM-based system are as follows:

1. First the VCS architecture is applied to identify the controllers, plants and their relationships. This results in a basic “wiring diagram” design for the system. (An example is shown in Figure 4.)
2. Next obtain or build a model and a simulation for each of the plants. This identifies the basic input and output requirements for selecting the control strategies.

3. Then the developer selects appropriate controllers from a palette of generic controller templates. A software “wizard” steps the developer through the process of generating the software classes for each function in the controller and generate all the communications “plumbing” code.
4. The generated system is compiled and run in simulation mode to ensure proper functionality.
5. The system software can now be loaded into the various hardware components of the room.
6. Finally, the initial policies, plans, rules, and conditions are sent to the various controllers and the room begins real-time operation.

We have implemented a Smart Camera controller following the steps above. The camera controller’s software structure follows the VSM architecture almost exactly. There are separate modules for Executive, Planning, Operations, Audit, Regulator and Plant. The controller functions as a “presence” sensor. It detects when a person is sitting in front of it, or is away. The controller uses a fuzzy control algorithm to adapt to the activity level of the person it is monitoring. The controller exhibits both homeostatic and morphostatic adaptation. The basics control strategy is to conserve system resources based on the users activity level. That is, if the person remains (or Planning predicts he will remain) in front of the camera, the cycle rate of camera is reduced. Thus the Operations module adjusts the camera’s strobe rate based on measured activity level of the subject. This is homeostatic adaptation. The Planning module maintains a history of observations. It uses this historical data to anticipate the probability of the subject being present or away. The Operations module uses this information to change *its own* cyclic rate of operation. This is one example of morphostatic adaptation.

5. Related Research

The VSM approach to building complex software systems touches on a number of systems and software related research disciplines, tools and techniques. In that regard it is more integrative and complementary than competitive to other approaches. It is unique in that it is based on “Systems Thinking” and Cybernetics in particular. Basically we take a cybernetic model, the Viable System Model, developed for enterprise modeling and use it as the basis for software architecture. One of the goals is to incorporate successful results from other areas. We begin this section by providing some insight into the research background that lead us to the Viable systems approach. These areas are “Systems Thinking” (General Systems and Cybernetics), Agent architectures, component software and complex applications.

5.1 Systems Thinking

The goal of determining high level abstractions and principles for complex applications led me to review work in General Systems and Cybernetics. These disciplines are based on a systems or holistic approach to understanding complex (e.g. biological) organisms and organizations. General Systems Theory [8] investigates structural, behavioral and developmental properties shared by classes of systems, in particular living organisms. The strategy is to identify phenomena found in different disciplines and build a general model relevant to these phenomena. Another approach is to organize the various systems into a hierarchy based on complexity and develop levels of abstraction appropriate to each. This leads to the hierarchical systems-of-systems model now commonly used to describe the evolving information infrastructure.

Following on with the biological metaphor, we explored Miller's Living Systems Theory (LST) [9]. LST is a general systems theoretic framework for studying biological and social systems and their ecological contexts. I summarize LST and give a literature review (57 citations) at [10]. Briefly, LST is a system-of-systems model of biological organisms and organizations. It considers them to be conceptually divided into eight levels: Cell, Organ, Organism, Group, Organization, Community, Society and Supranational. At each level, systems are composed of 20 critical subsystems. The subsystems are grouped in terms of how they process matter-energy or information. Eight subsystems process matter-energy, ten process information, and two process both. The application of LST to component software systems is described at [3].

5.2 Cybernetics

Norbert Wiener coined the term Cybernetics around 1948 from the Greek word meaning "steersman" [11]. He defined Cybernetics as the science of communication and control in mechanisms, organisms and society. Cybernetics is a general theory of control that can be applied to any organized system. Several important findings about control systems were made early. The "First Law of Cybernetics" says that any system that seeks to control another system (including self) must have a model of that system. The role of information in control systems and the concept of *feed-back* were major discoveries.

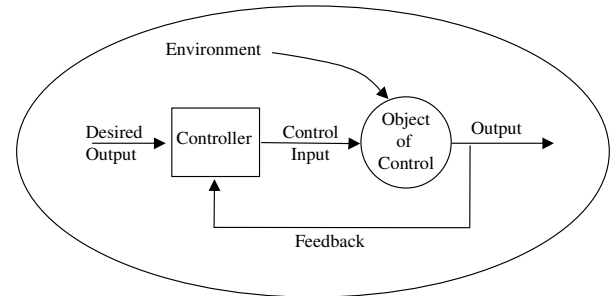


Figure 5. Closed-loop feedback control system

A prototypical control system is shown in Figure 5. Note the system is embedded in an environment that effects (disturbs) the system. Control systems fall into four major categories: stabilizing, following (tracking), programmed and optimizing [12]. Also, Cybernetics deals with adaptive and self-organizing systems wherein the system changes itself in order to achieve better performance (survive) in a changing environment. A handy on-line reference to Cybernetics is [13].

5.3 Viable System Model

Pursuing this line of thinking about control theory and Cybernetics led us to study Beer's Viable System Model (VSM) [2]. VSM originated in Beer's effort to develop "management cybernetics" as a science to understand and control human organizations [14]. However, it is a holistic cybernetic model and it provides a template for a general adaptive control system. We have chosen to use VSM as the structuring principle for component framework to address complex applications requirements. A complete literature review of the VSM is at [4].

5.4 Agent Architectures

In developed the proposed approach based on the VSM we began to see similarities between it and Agents Architectures. With the rise of the Internet interest in Agents has grown. The literature on Agents is vast, on the order of 10^4 articles. For example, both the March 1999 CACM and the summer 1999 ACM student magazine "Crossroads" are dedicated to Multi-Agent Systems (MAS). We compare MAS to the VSM approach.

The "Crossroads" article provides background and summary of agents and MAS [15]. A few key points from that survey article will be used to compare agents to VSM. First are the general attributes of agents: adaptive, autonomous, collaborative, inferential, "knowledge-level" communications, reactive and persistent. The VSM directly incorporates these attributes in a rigorous manner and identifies where in the overall framework they are to be found.

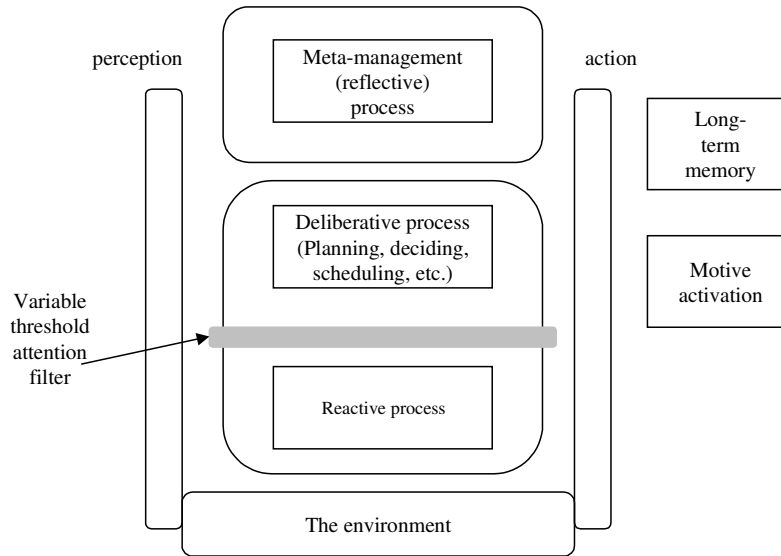


Figure 6. “Cognitively Rich” Agent Model

For example, Figure 6 shows the “Cognitively Rich” Agent Model of Sloman and Logan [16]. Compare this model with the VSM shown in Figure 8. The most obvious feature is that Sloman and Logan have divided their agent model into three distinct levels of functionality. These roughly correspond to the three major functions shown in the Control portion of the VSM. While it is interesting that the models are similar, agents are based on ad hoc anthropomorphism whereas VSM is based on cybernetic principles of biological organisms and human organization.

5.5 Application Domains

In developing the approach we investigated a number of possible application areas including operating systems, networking and finally Smart Environments. This was to ensure the approach was sufficiently general to span a large number of infrastructure layers. We settled on the Smart Environments areas as it incorporates elements of most of the areas surveyed.

In terms of networking we reviewed the work on heterogeneous, wireless overlay networks [17] and developed the cellular Component Manifold model [18]. I intend to incorporate this work into the VSM. Wireless overlay networks are one element of Smart Environments.

Two other projects that influenced our approach are Microsoft’s Millennium [19] operating system and Sun’s Jini [20] networking environment. Both Jini and Millennium are seeking to develop new architectures and system-level support for advanced distributed

applications. Jini’s vision is to bring the equivalent of a dial tone to the network. That is, a new system architecture that provides “plug-n-work” capabilities for all devices and applications in an effortless (for the user) and seamlessly expandable manner by adding new software layers that permit service federation. The ultimate goal being “the network is the computer.” The central problems Jini hopes to address are the complexity of network management and its effective use. As the number and power of hardware and software components grows, this task becomes even more difficult.

Millennium is working on many of the same problems. They maintain it is necessary to reexamine the role of the operating system in relation to distributed systems. Their goals are seamless distribution, worldwide scalability, fault-tolerance, self-tuning, self-configuring, security and resource control. To achieve these goals, they base their approach on the principles of aggressive abstraction, storage-irrelevance, location-irrelevance, just-in-time binding, and introspection. The motivation and design of the proposed VSM has much in common with both of these projects.

Smart Environments offer a rich, diverse domain that incorporates all of the above systems and applications. Smart Environments represent the intersection of human-computer interaction, mobility, networking, embedded “smart” devices and more. DARPA has pioneered this area by encouraging researchers to determine what these new environments will be like and develop prototypes [21].

6. Conclusion and Future Work

In this paper we have presented a software architecture and engineering approach to “complex” applications based on the Viable System Model. We have described the desired attributes or capabilities of systems resulting from this approach as Viable software systems. In applying the VSM to software we have extended the “control paradigm” to include adaptive control. In fact we have extended Beer’s VSM to include Morphogenetic behavior as well. The Smart Environments research area was chosen as a suitably complex setting to test our approach. We have made some first steps in implementing software based on the VSM architecture and have begun to understand what a software engineering methodology based on the approach will look like.

Our near term goal is to compete development of enough components to allow us to test and evaluate their aggregate behavior. This will hopefully permit us to demonstrate a Viable system. That is, a smart environment application that exhibits homeostatic and morphostatic adaptation. Based on this we will have the structure of a Viable Component Framework on which to build software engineering support tools (e.g. code generators) to permit rapid development of viable software systems. Longer-term goals include the development of a Component Transfer Protocol to provide for dynamic system assembly. This is necessary to achieve morphogenetic behavior.

7. References

1. Shaw, M., *Beyond objects: A software design paradigm based on process control*. ACM Software Engineering Notes, 1995. **20**(1).
2. Beer, S., *Diagnosing the system for organizations*. 1985, Great Britain: John Wiley and Sons Ltd.
3. Herring, C. and S. Kaplan. *Cybernetic Components: A Theoretical Basis for Component Software Systems*. in *Component Oriented Software Engineering Workshop (COSE'98)*. 1998. Adelaide, Australia: (<http://www.dstc.edu.au/TU/staff/herring/cose98-ref.pdf>).
4. Herring, C., *The Viable System Model of Stafford Beer*. <http://archive.dstc.edu.au/TU/staff/herring/Vsm.pdf>, 1998.
5. Franklin, G.E., J.D. Powell, and A. Emami-Naeini, *Feedback control of dynamic systems*. 2 ed. Control Engineering, ed. T. Robbins. 1991, Reading, Massachusetts: Addison-Wesley.
6. Mareels, I. and J.W. Polderman, *Adaptive systems: An introduction*. 1996, Boston, Massachusetts: Birkhauser.
7. Passino, K.M. and S. Yurkovish, *Fuzzy Control*. 1998, Reading, Massachusetts: Addison-Wesley.
8. Bertalanffy, L.v., *General Systems Theory*. 1968: George Braziller, New York.
9. Miller, J.G., *Living Systems*. 1995, Niwot, Colorado: University Press of Colorado. 1102.
10. Herring, C., *Living Systems Theory a Literature Review*. <http://archive.dstc.edu.au/TU/staff/herring/Lst.pdf>, 1998.
11. Wiener, N., *Cybernetics*. 1948, New York: Wiley and Sons.
12. Lerner, A., *Fundamentals of Cybernetics*. 1975, New York: Plenum.
13. Principia Cybernetica, <http://pespmc1.vub.ac.be/Default.html>.
14. Beer, S., *Decision and Control*. 1956, Great Britain: John Wiley and Sons Ltd.
15. Flores-Mendez, R., *Towards a Standardized Multi-Agent System Framework*, in *Crossroads, The ACM Student Magazine*. 1999.
16. Sloman, A. and B. Logan, *Building Cognitively Rich Agents Using the SIM_AGENT Toolkit*, in *Communications of the ACM*. 1999. p. 71-77.
17. Katz, R. and E. Brewer. *The Case for Wireless Overlay Networks*. in *Proceedings 1996 SPIE Conference on Multimedia and Networking, MMCM '96*. 1996. San Jose, CA.
18. Herring, C. and S. Kaplan. *Manifolds: Cellular Component Organizations*. in *Proceedings of the twenty-eighth International Conference on Technology of Object-Oriented Languages and Systems, TOOL Pacific*. 1998. Melbourne, Australia: (<http://archive.dstc.edu.au/TU/staff/herring/awsa98-herring.pdf>).
19. Millennium, *Microsoft*. <http://www.research.microsoft.com/sn/Millennium/>, 1998.
20. Jini, *JavaSoft*. <http://www.javasoft.com/products/jini/>, 1998.
21. Herring, C. and S. Kaplan, *Research Directions in Component-Based Software Systems for Smart Environments*. Interagency Workshop on Smart Environments, 25-26 July 1999, Georgia Institute of Technology, 1999.