Viable Components:
A Cybernetic Organization for Computing

Charles Herring and Simon Kaplan
Department of Computer Science and Electrical Engineering
The University of Queensland
Brisbane, Queensland QLD 4072

ABSTRACT

In this paper we explore the Viable System Model (VSM) as a basis for design of a component framework. The goal is to address complex application requirements by increasing the level of abstraction and through principled consistency in a component architecture. We develop a scenario around a Smart Lecture Room. This provides a hardware and software setting to illustrate how the VSM is used to develop an advanced application. Following the scenario an overview of the VSM is given. The model is then used to show how "Viable Components" can address the complexity of building such environments.

1. INTRODUCTION

A host of new computing and communications applications are on the horizon. Examples include ubiquitous computing, heterogeneous wireless networks and smart environments. The rapid convergence of existing systems and the proliferation of diverse embedded devices are driving this new class of application. We refer to these as **complex** applications. Also, component-oriented software development has emerged as an approach to building large-scale systems. We think new abstractions and principles on which to base component software architectures are required to build these complex systems. In this paper we explore the application of a Cybernetic model of component organization as the basis for complex system architectures.

We begin by characterizing the evolving technological environment that motivates our investigation. Computing and communications technologies are converging to form a world wide system-of-systems that will soon result in a global integrated information infrastructure. Major trends driving this merger are:

- Integration of telephony, television, and computer networks
- Integration of wireline and wireless networks
- High bandwidth, low latency communications
- Ubiquitous computing based on embedded systems
- Smart buildings and smart appliances: Smart Environments.

These developments are prompting a rethinking of the fundamental structure of basic support systems such as operating systems and network protocols. The traditional boundaries between many of these systems are shifting. Also, greater demands are being placed on applications to support new and complex domains. The desired qualities of these applications are often described as self-organizing, self-configuring, adaptive, etc. Projects typifying the goals and requirements of these new domains include the DARPA funded "Smart Spaces" [1] and the BARWAN heterogeneous, wireless overlay network project at UCB [2]. These are examples of the class of complex applications motivating our search for new software architectures.

Recently, the component-oriented approach to software development has emerged with the hope of addressing long-standing software engineering problems. However, few examples exist and little is known about how to construct component systems [3]. We think two things are necessary for component systems to meet the needs of advanced applications described above. First is the recognition and promotion of new levels of abstraction in component frameworks and system architectures. Second is the pervasive application of a small number of key principles in order to obtain a high degree of consistency throughout the design.

In this paper we explore the Viable System Model (VSM) [4] as a basis for design of a component framework to meet the goals of increased abstraction and principled (Cybernetics) consistency. The next section develops a Smart Environments scenario around a Smart Lecture Room. This provides a hardware and software setting to illustrate how the VSM is used to develop an advanced application. Following the scenario an overview of the VSM is given. The model is then used to show how "Viable Components" can address the complexity of building such environments.

2. SCENARIO: THE SMART LECTURE ROOM

Smart Environments are a nexus for technologies and disciplines: both old and new. These include human-computer interaction, embedded devices, ubiquitous computing, networking, wireless systems, and any number of specialized applications such as image and voice processing. This area provides a rich setting and a big challenge to software architecture. Much of the early work naturally focused on individual elements for specific purposes such as Active Badges, tracking, various aspects of image processing to recognize human gestures, etc. Now researchers are assembling these disparate components into synergistic systems of systems.

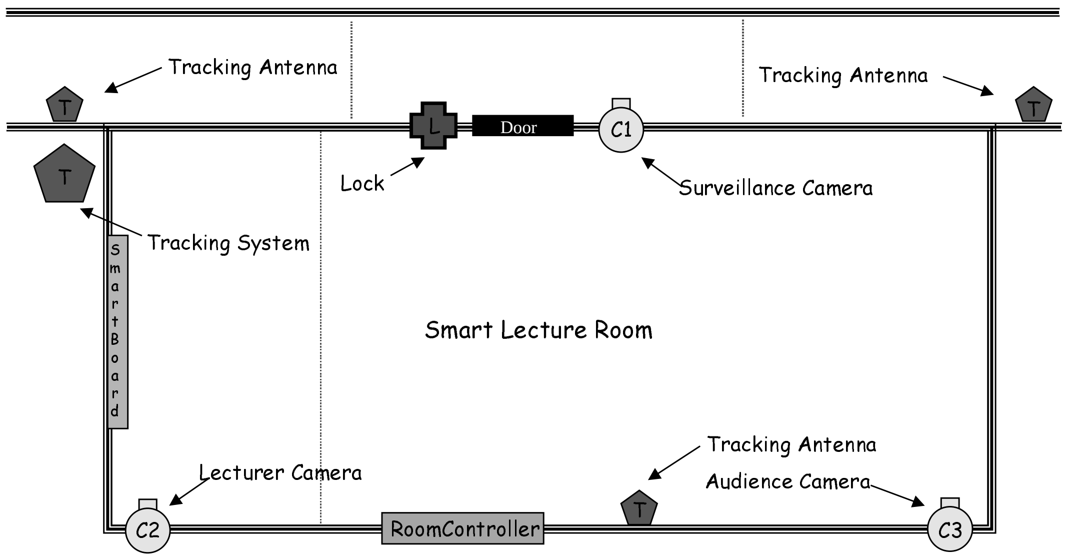

Figure 1 Smart Lecture Room and Components

The Smart Lecture Room is the setting for our scenario. Figure 1 shows a hallway at the top and the room below. There are a number of systems installed in the hallway and room that make up the overall smart environment. Starting with the hall there is a Tracking System and antennas. People using this facility have Active Badges that periodically broadcast a signal that is sensed by the tracking system. This permits location of people with the building. There is a door opening into the room from the hall and there is 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.

Now we add people 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 Tacking System notifies the Lecturer Camera. The Lecturer Camera focuses on the lecturer and 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.

We could continue to add elements to the scenario. For example, the students all have wireless PDAs that are dynamically added to the Room's network as they enter. The students can communicate with each other (or not depending on the situation) and project assignments onto the SmartBoard. But, the Smart Room as described provides sufficient complexity to illustrate our approach to designing a component software system. Next we describe the Viable System Model that will be used as an architectural template. We will then return to the Smart Room and describe how the model can be used to structure a component software system to handle the complexity of such applications.

3. THE VIABLE SYSTEM MODEL

The VSM was developed by Stafford Beer. Beer established "management cybernetics" as a science to understand and control human organizations [5]. The VSM is a holistic cybernetic model and we have selected it as a template for a general control system. In particular, we use VSM as the structuring principle for a component framework to address complex applications. A brief overview of the VSM follows.

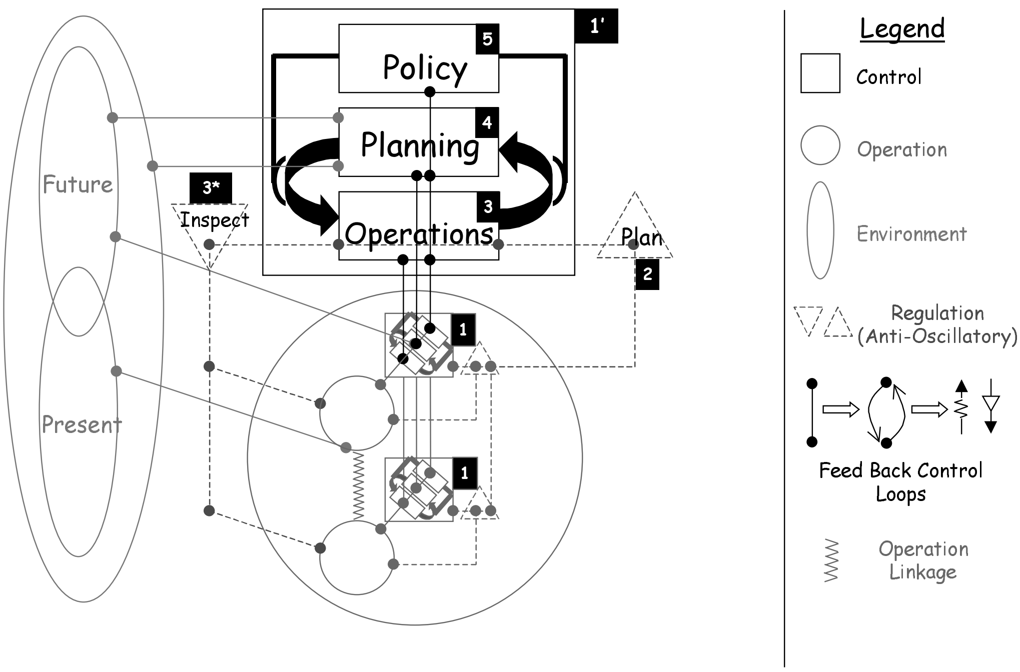

Figure 2 Viable System Model

Figure 2 is the diagram of the VSM. The model incorporates a set of functions that ensure the viability (independent survival) of the system. These functions and their interrelationships are based on its cybernetic and information theory underpinnings. The model is inherently recursive and decomposed into nested (viable) systems, each responsible for a portion of the system and each fitting into the greater whole. In this architecture, a system is viable if and only if it disposes of a set of management functions:

- Control (**1'**), shown in a rectangle, performs all aspects of policy, planning and operations (or execution). Policy (**5**) includes overarching system requirements that are mandated by higher levels in the system. These are implementation of strategies, obligations and constraints. Policy serves to balance the Planning (**4**) and Operations (**3**) function and to keep it with system-defined boundaries. Planning focuses on developing future courses of action. In this regard it makes use of relevant environmental information as well as tools and techniques such as modeling and simulation. Operations is focused on the accomplishment of the current plan within some time constraints. Planning and Operations is an ongoing, iterative process as indicated by the bold arrows in the figure.
- The Operation (**1**), shown in a circle, is the "object of control", i.e., the actual operation under consideration. Because of the recursive nature of the system, operation in turn repeats the structure of the higher levels. Dependencies among levels are shown using solid lines.
- Regulation. Plan (**2**) and Inspection (**3***) provide the needed feedback loops (shown through the dotted lines) to ensure smooth operation of the overall system. The Plan is the agreed upon production schedule of the Operation. Specifically, this "circuit" eliminates oscillations that can arise in complex dynamical systems.
- Environment, shown in an ellipse, represents the external factors and influences (present and future) of which we take account. Each recursive element of the system performs its function at a different time scale. That is, each level has different planning and execution time horizons.

We conclude this section with some remarks on the attributes of this model and why it is attractive to us. First, we think it makes specific, within a software architecture, many of the desired characteristics of advanced systems such as Agents. For example, Planning (4) and Operations (3) are designed to be in homeostatic balance (indicated by the broad curved arrows). This homeostat is the "organ of adaptation." Planning (4) is self-referential (maintains a model of itself) and simulates possible futures in order to develop plans for future operations and its own internal organization. Operations (3) is responsible for self-organization and autonomic regulation. These two systems must be in feedback-controlled balance to permit the system to operate and adjust itself to its environment. Policy (5) embodies the rules and constraints that govern the 3-4 homeostat thus forming a second level homeostat. The information flows among all these systems are to be designed based on four "Principles of Organization." These principles (from information theory and cybernetics) are state equations that ensure the correct balance of information flowing throughout the system. Another critical feature of the model is explicit support for the "system-of-systems" concept. This is shown in the figure by the recursive, self-similar figures (viable systems) with the large circle. Space does not permit full development of the VSM. The example application of the model to the Smart Room scenario will give the reader a better feel for its use.

4. VIABLE COMPONENT FRAMEWORK

Having described a Smart Lecture Room, usage scenario and the Viable System Model, we now apply the model to develop a Viable Component Framework for this application. We begin by noting the Smart Room is embedded in a larger system of systems. It is a system within the Level which is a system within the Building, …, within the Campus, etc. We must not forget to address the constraints these relationships put on the Room Controller. But our focus will be on the Room system and its subsystems that work together to "produce" the room's behavior. We will also go inside the cameras to illustrate the recursive consistency of the design.

Figure 3 is a simplified VSM diagram for the room. Here we see the Room Controller corresponds to System 1' (refer to Figure 2). That is, it implements the management functions of Systems 2-3*-3-4-5 to achieve the synergistic behavior of the room. There are five System 1 operational components: three cameras, lock, presentation system and tracking system. 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 components represent the channels (interfaces) for communicating policy, commands, control instructions and resources. The squiggly lines between the operational components show that 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 is indicated by ovals and the links are shown.

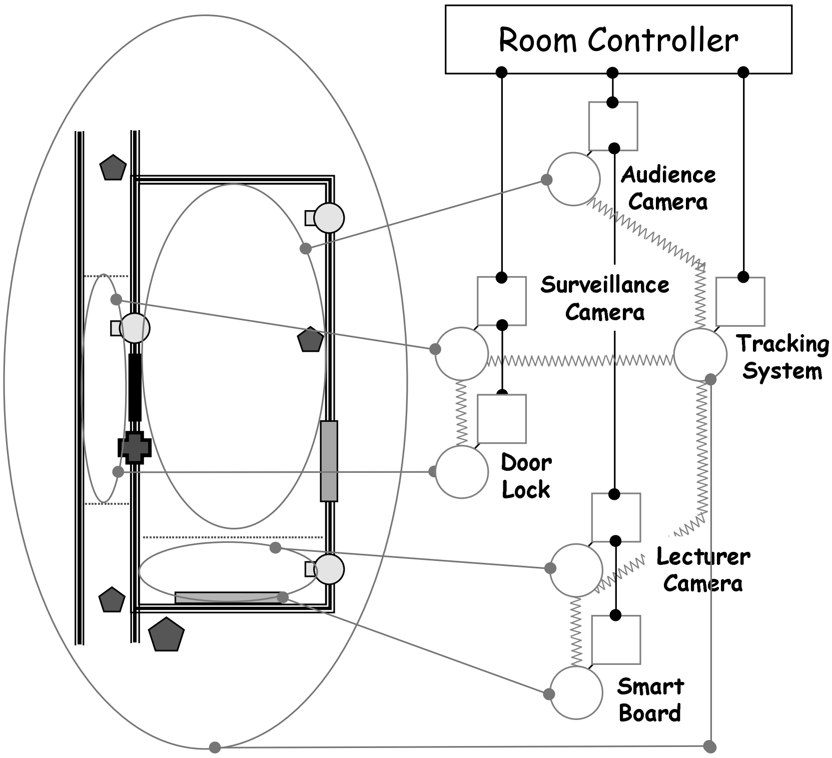

Figure 3 VSM Diagram of Smart Room and Components

4.1 Inside the Room Controller

Out next task is to design the Smart Room control software based on the VSM functional partitioning. Figure 4 shows a generic layered architecture we call the Viable *Organizing* System (VOS). The functionality of Systems 2-3*-3-4-5 is allocated to three components as follow: Policy Manager, Modeling Manager and Operations Manager. The Operations Manager is shown as containing the Auditor (3*) and Scheduler (2) functions. The VOS relies on a commercial operating system (e.g. Windows NT) to provide network communications to the various components of the room (camera, etc.), database support for storage, and so on. The VOS components can be implemented as operating system services for example.

The generic VOS architecture is now specialized for the Smart Room application. First, we need a high-level, domain specific language for describing policy, rules, constraints and commands. We will call this language *RoomTalk*. The language consists of a declarative scripting language, content format, content communications protocol and a room ontology. Examples of this of type of facility are KQLM, KIF and Ontolingua [6-8]. A functional description with examples of each of the major components of the Smart Room VOS follows.

Policy Manager: The primary job of the Policy Manager is to govern the overall behavior of the room. It does this by maintaining a rule base that represents policies common to all rooms and policies specialized to the particular room instance. These specialized policies are required (imposed) by the Level controller. The Level controller transduces these policies into the RoomTalk language format. The Policy Manager's main task is to monitor the Modeling Manager-Operations Manager homeostat to ensure any proposed changes to long term plans or internal organization are consistent with its policies. The other major responsibility of the Policy Manager is to transmit its policies to the Operational components. This requires "speaking" their language, that is, translation of RoomTalk rules into their native language. Examples will illustrate.

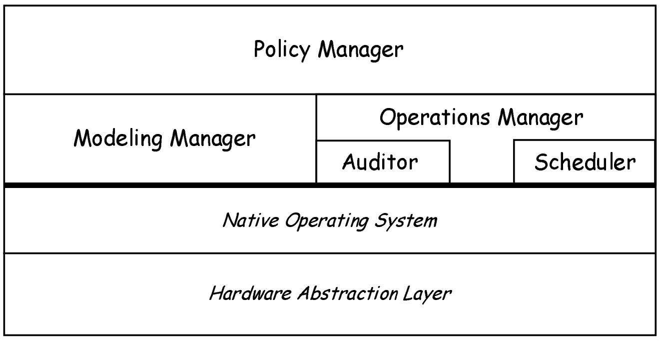

Figure 4 Viable Organizing System

The room controller comes equipped with a generic rule base: "The room is available from 0800 until 1700 Monday through Friday"; "Maintain a record of all persons entering the room"; and "Permit students to use wireless network." The Level controller has its specific policies: "No more than 30 persons allowed in this room at any one time"; "During exams no communications is allowed between students"; and "The Smart Board may use the Level server if additional storage is required." The Room's level controller provides the list of persons authorized to enter the room.

Modeling Manager: The Modeling Manager maintains one or more representations of the Room for simulation and planning purposes. These models include the spatial properties of the room, its adjacent rooms and its operational components. The operational components provide their self-models to the Modeling Manager in order for it to assemble its own self-model. Based on these models the Modeling Manager keeps a dynamic representation of the state of the room. The primary purpose of this system is to anticipate future requirements and to work with the Operations Manager in developing the Schedule.

For example, the energy required to maintain the room's temperature and humidity within certain ranges can be estimated based on historical weather data. As an example of how the Modeling Manager might work with the Operations manager, consider a request to the room to schedule a lecture requiring special setup or hardware. The Modeling Manager could simulate the impact of this special request in relation to the other scheduled lectures to determine if is possible and inform the Operations Manager.

Operations Manager: The Operations Manager is responsible for the moment-by-moment operation of the room and its components. It works with the Modeling Manager to determine any impact on the room's Schedule or internal organization generated by scheduling requests or other environmental factors. The Operations Manager handles all requests to schedule the room for use. Maintaining and executing the Schedule is a major function and requires the coordination of all the room's components. This is accomplished by transmitting to the components their specific portions of the Schedule. The Operations Manager is responsible for dynamically allocating resources to the components and sends direct commands to them as needed. In addition to the normal monitoring it also conducts random audits of selected components or groups of components to ensure they are functioning properly.

4.2 Inside the Cameras

The software controlling the cameras further illustrates the viable component design. Imagine a commercially available camera system called *SmartCam* with the following capabilities. It has a high-resolution color video camera mounted such that is can tilt, rotate, pan and zoom. The SmartCam contains a basic computer system with persistent storage and network connection running a commercial operating system (e.g. WinCE). The cameras come equipped with a range of utility software. There is also an industry standard control language *CamTalk* (similar to RoomTalk described above.) The VOS is again used as the basis for design of the SmartCam organizing system software. From the previous example of the Room controller we already have a good idea of how SmartCam's software functions. This is an advantage of the Viable Component approach; all components at all levels are conceptually self-similar. The VOS-based SmartCam organizing software is the same in all the cameras. The difference is in the policies and the particulars of the schedule. We concentrate on the operation of the three cameras and the other operational components as they are related within the SmartRoom system.

Surveillance Camera: The room controller transmits its policies regarding entry into the room to the surveillance camera in the CamTalk language. One policy is that only authorized persons may enter the room. The room controller sends the information necessary for the camera to make facial recognition of those persons authorized to enter the room. "Unlock the door if the person is authorized to enter the room." This statement implies to the room operations manager that there is a relation between the camera and the door lock. The room operations manager provides the camera with permission to send commands to the lock. Also the room operations manager connects the camera to the tracking system. This permits the camera to receive notification from the tracking system when people enter the hallway and signals it to begin observing them. The camera's modeling manager begins tracking people in the hallway and advises the operations manager to identify them in anticipation that they will approach the door with intention of entering the room.

Lecturer Camera: This camera is used in conjunction with the SmartBoard presentation system to permit the lecturer to control the presentation system based on gestures. The policies sent to this camera from the room indicate to the room operations manager that the tracking system should notify the camera when someone is at the front of the room standing near the SmartBoard. The camera uses its utility software to perform gesture recognition that is used to signal the SmartBoard (in the *PresentationTalk* language). The operations manager uses the schedule to determine what presentations should be downloaded from the level controller into the SmartBoard before the lecture begins.

Audience Camera: The audience camera monitors the students during the lecture. The policies downloaded from the room controller provide guidance for the camera. For example, during an exam, students may not communicate with each other.

4.3 Viable Component Interface Definition

We have been using the word "component" in regard to the proposed software design. We now define what we mean by a software component and define the generic interface for Viable Components.

Perhaps the simplest way to think of a software component is as a black box. That is, a component is bundle of functionality with some interface. We do not know what is inside – how it works. We can only cause it to exhibit behavior via its interface. Szyperski identifies three major tiers of component systems [3]. The three tiers are defined below:

- Software ***components*** are binary units of independent production, acquisition, deployment and extension (e.g. COM and JavaBeans.)
- A ***component framework*** is a dedicated and focused architecture, usually around a few key mechanisms, and a fixed set of policies for mechanisms at the component level (e.g. COM+ and EJB).
- A ***component system architecture*** consists of a set of platform decisions, a set of component frameworks and an interoperation design for the component frameworks.

Component frameworks may appear as components within some contexts. That is, a component framework may offer an interface for use by components or by frameworks. The distinction is arbitrary in many cases.

Software components, as binary units of independent deployment and extension, imply the existence of a runtime support environment. *Containers* are the current metaphor used to describe such environments. Currently there are two commercial component containers: Microsoft's COM+ and Sun's Enterprise Java Beans. These containers provide the basic services to support most business applications. Specifically, they provide naming, messaging, events, transactions and security. In a "three-tier" architecture, containers are the middle tier. The presentation layer being the first tier and the third tier is usually database and other backend services.

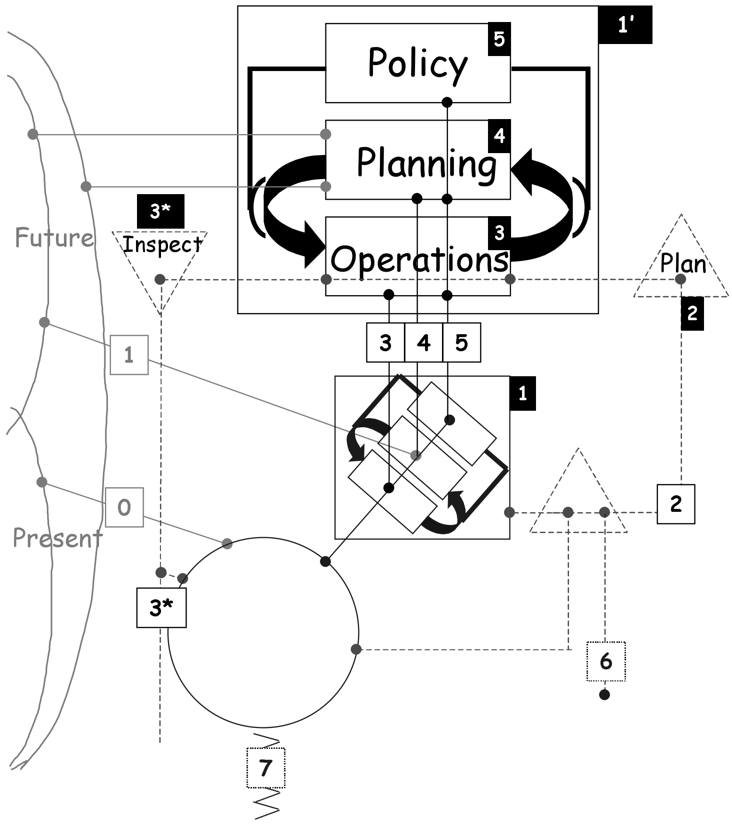

Figure 5 Viable Component Interfaces

The generic interfaces required by a Viable Component are shown in Figure 5. There are three types of interfaces: component to meta-component, component to environment and component to component (on the same level). The component to meta-component interfaces couple the component to the meta-component's control systems. These interfaces are labeled in accordance with the system numbering:

2. Provides for transmission of plans and schedules from the meta-component's scheduling system to the component.
3. Direct commands and resources flow from the meta-component's operations system to the component. Routine status reporting and requests for resources flows from component to meta-component.
3*. The sporadic audit interface permits the meta-component's opertaions (System 3) to randomly check on the component's state.
4. This is the "modeling relationship" between components. The component transmits a model of itself for inclusion in the meta-components self-model.
5. Meta-component policies and rules are transmitted to the component via this interface.

There are two component to environment interfaces:

0. This interface provides environmental information directly to the components operational element.
1. This is the planning system's coupling to the "future" environment. This may be from data projections or simulations based on the current environment.

There are also two component-to-component interfaces.

6. This interface couples related components scheduling functions. This permits coordination and synchronization between components.
7. This is the direct operations-to-operations linkage. For example, the Camera uses data generated by the Tracking System and must have a way to register its interest in events and receive the related data.

The development of a generic component interface is the first step in realizing a Viable Component Framework.

5. CONCLUSIONS AND FUTURE WORK

Motivated by the desire to identify abstractions and principles for architecting advanced component frameworks we have employed the cybernetics-based Viable System Model. The Smart Environments applications area gives us a rich setting for developing the approach. In this paper we provide the rationale and conceptual design for the Viable Component framework.

We are in the early stages of this project and our next goal is to complete the design of the Viable Component Model. Most importantly this includes the definition of a generic component interface for viable components. Also, a Component Transfer Protocol (CTP) is needed to provide for dynamic system assembly. We think the best approach to design is the development of a simulation of the Viable Components within a simulated Smart Environment. This will permit rapid verification of the needed interfaces, protocols and techniques for representation of advanced functionality with the controllers (policy, planning, knowledge representations, etc.). This approach also permits experimentation with a rich environment that includes sensors, tracking, cameras, wireless networking, etc. without the need to have the physical infrastructure. Based on the simulation results we will design and implement the framework to support a limited number of real-world components (a camera, a tracking system) within a prototypical Smart Environment.

The following contributions as a result of this work are anticipated. First is the identification of high-level architectural abstractions structuring principles for component systems. Specifically, it is hoped the work will provide verification of a cybernetic model as the basis for a component framework for complex applications. A by-product of this will be a simulation and implementation of the model. It is also hoped this work will contribute to the understanding of Smart Environments and similar complex applications.

REFERENCES

1.DARPA/NIST. *Proceedings of the DARPA/NIST Smart Spaces Workshop*. July 30-31, 1998. Gaithersburg, Maryland, USA: DARPA.
2.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.
3.Szyperski, C., *Component Software*. 1998, New York: Addison-Wesley.
4.Beer, S., *Diagnosing the system for organisations*. 1985, Great Britain: John Wiley and Sons Ltd.
5.Beer, S., *Decision and Control*. 1956, Great Britain: John Wiley and Sons Ltd.
6.Finin, T., Y. Laboru, and J. Mayfield, *KQML as an Agent Communications Language*, in *Software Agents*, J.M. Bradshaw, Editor. 1997, AAAI Press: Menlo Park , California. p. 291-316.
7.Genesereth, M. and R. Fikes, *Knowledge Interchange Format, Version 3.0 Reference Manual, Technical Report, Computer Science Department, Stanford University*, . 1992.
8.Gruber, T.R. *A Translation Approach to Portable Ontology Specification*. in *Knowledge Acquisition for Knowledge-Based Systems (KAW'93)*. 1993. Banff, Canada.