

ComponentSpaces

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

ABSTRACT

New software architectures are required to meet the demands of a host of complex applications on the near horizon. These applications include ubiquitous computing, heterogeneous wireless networks and smart environments. In this paper we describe our first efforts in applying “Systems Thinking” to the design of a component framework to support these applications. First we present our motivation based on emerging technological trends. Next we give design rationale for selection of a cybernetic control model as the chief organizing principle for our proposed architecture. A mapping from design to component framework is given. An example based on Smart Environments is provided.

Keywords: Component Frameworks, Viable System Model, Cybernetics, Address Space, Smart Environments.

1. INTRODUCTION

The component-oriented approach to software construction is apparently the next stage in the evolution of software practice. However, there are few examples of component systems, i.e., component frameworks and component system architectures. Thus far developers have concentrated primarily on individual, isolated components and today there are only basic component interoperation facilities (Microsoft’s COM+ and Sun’s Enterprise Java Beans). Rapidly evolving and converging information technologies require development of new and more complex software systems. If components are to achieve the impact that many are hoping for, discovery of new organizing principles to permit design and development of advanced component systems is required. This paper describes research based on “Systems Thinking” applied to develop a component framework to meet these anticipated complex application requirements. We give the motivation, rationale, and design of a component architecture called **Component Spaces**. An example application of ComponentSpaces is provided.

2. MOTIVATION

We now characterize 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” and the BARWAN heterogeneous, wireless overlay network project at UCB [1]. These are examples of the class of problems motivating our development of the ComponentSpaces architecture.

Recently, component-oriented 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. We think two things are necessary to achieve advanced component systems. 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. Research projects such as Microsoft’s Millenium [2] operating system and Sun’s Jini [3] networking environment are pursuing this approach. We are also investigating how to raise the level of abstraction and to identify key principles applicable across a range of component system layers. This has led us to begin development of the ComponentSpaces framework. The rationale for selection of abstractions and modeling principles of our design is now described.

3. DESIGN RATIONALE: ABSTRACTIONS AND PRINCIPLES

General Systems Theory investigates structural, behavioral and developmental properties shared by classes of systems, in particular living organisms. One 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. We employ these general systems approaches to determine pervasive abstractions and key modeling principles for the development of a component system architecture suitable for a range of advanced applications. In this section we give our rationale for choosing a *cybernetic* approach to the organization of the component architecture. In the simplest terms, a cybernetic control system consists of the controller, object of control and environment. The overall control model is presented first. Next, the fundamental abstraction of *address space* is used to couple the cybernetic model to a component software implementation.

Cybernetic Control Model

We investigated the application of Miller's Living Systems Theory to component architectures [4, 5]. While this yielded some insights we were also influenced by Shaw's "Control Paradigm" [6]. Thus, we began thinking about control theory and Cybernetics. We studied Beer's Viable System Model (VSM) [7]. VSM originated in Beer's effort to develop "management cybernetics" as a science to understand and control human organizations [8]. However, it is a holistic cybernetic model and we use it as a template for a general control system. In particular, we use VSM as the structuring principle for component containers in the ComponentSpaces framework.

Figure 1 is a simplified VSM template for the control part of our component architecture. The model incorporates a set of functions that ensure the viability of the system. These functions and their interrelationships are based on its cybernetic theoretical underpinnings. The model is inherently recursive and decomposed into nested control 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:

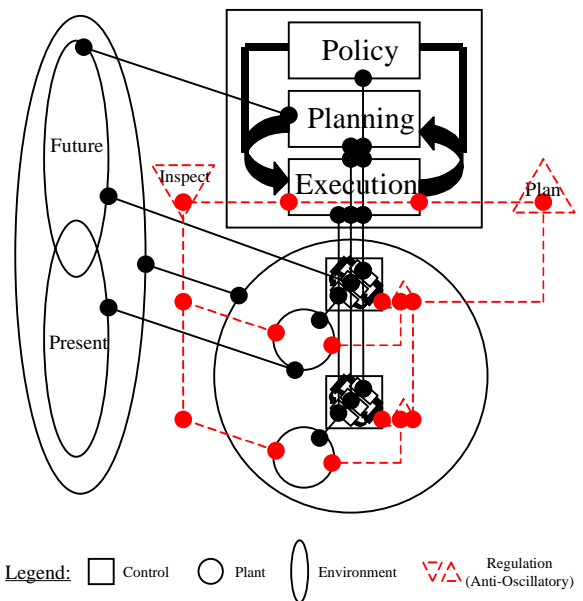


Figure 1 Viable System Model (simplified)

- Control, shown in a rectangle, performs all aspects of policy, planning and execution. Policy 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 and Execution 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. Execution is focused on the accomplishment of current plan within some time constraints. Planning and Execution is an ongoing, iterative process as indicated by the bold arrows in the figure.
- Plant, shown in a circle, is the "object of control", i.e., the actual operation under consideration. Because of the recursive nature of the system, plant in turn repeats the structure of the higher levels. Dependencies among levels are shown using solid lines.
- Regulation. Plan and Inspection 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 Plant. 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 shall return to the cybernetic control model in the next section where we show its use as the organizing principle for component containers in our ComponentSpaces architecture. Next we present the primary abstraction in the software domain that permits implementation of ComponentSpaces.

Address Space

An examination of many levels of computing and communications systems reveals the concept of address space to be fundamental and pervasive abstractions. This is true as all entities in computing and communications systems are referenced by numeric labels defined in a standardized manner. Also, addressing is at the heart of many critical issues such as scalability and interoperability. Given the above, the concept of “address space” is chosen as a pervasive abstraction for ComponentSpaces.

Accordingly we have developed a *Generalized Address Space Model*[9]. This model captures the essential features of addressing as exhibited in a range of real world systems. It is based on examination of the major subsystems, e.g., operating systems, distributed systems and networking, as well as catering for certain application domains. The central concepts of the address space model are centers, boundaries, boundary crossing, and hierarchy. The characteristics of the general address space are:

- Supports hierarchical, layered, and/or nested configurations of component subsystems
- Boundaries can be identified and defined
- Supports an addressing or indexing scheme.

The general address space model was initially developed to support *cellular architectures* common to many types of communications systems. In that work we used a spatial data structure as the underling addressing system. We chose a geometric coordinate system based on the *Generalized Balanced Ternary (GBT)* [10]. GBT is a hierarchical, uniformly adjacent, cellular-aggregate addressing system defined on Euclidean n-space. It was developed to support a range of scientific and engineering applications. GBT is an algebraic system permitting the construction of vector spaces in n-dimensions that support vector addition and multiplication as well as aggregation operators. The result of combining this geometric addressing scheme with the general addressing model results in a hierarchical, n-dimensional address space that is also a vector space. Other possibilities for geometric addressing include data structures such as quad-trees, R-trees, etc.

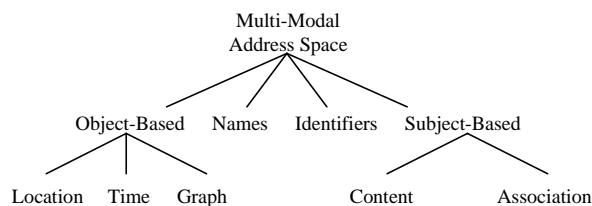


Figure 2 Multi-Modal Address Space

Based on new application domain requirements [], we extended our view of the ideal address space for ComponentSpaces. We call this a “multi-modal” address space. It incorporates the four major modes of addressing in computing and communication systems. The scheme is shown in Figure 2. The spatial or vector-based address space is shown on the left as object-based. This type of addressing covers the n-dimensional case (e.g. latitude-longitude-altitude; fifo, lifo; internet routing). Another common form of addressing is by Name (character strings, symbols). Names are convenient for humans but may not be guaranteed to be unique, or are unique only within some naming context. Identifiers are also widely used. Identifiers are guaranteed to be unique, but they are generally “opaque” in that they carry no information on the entity so identified. Finally we incorporate subject-based addressing. We distinguish content-based addressing and associative addressing as follows. Content-based address spaces permit specification of search or matching criteria over the possible values of the address space. Associative addressing is used in systems such as neural networks and holographic images [11]. In these systems *the address is the data*.

This completes the design rationale for ComponentSpaces. To summarize, we chose a cybernetic control model adopted from Beer’s VSM as the central organizing principle for a component architecture. The chief abstraction relevant to a software implementation is that of the address space. To that end we described a generalized address space model. In the next section we show how to map these constructs onto a component framework.

4. COMPONENTS AND CONTAINERS

Our goal is to implement ComponentSpaces as component framework to organize both software and *real world hardware components* into computing and communicating systems. An application built using this framework will take the form of a *component container* (see Appendix on terminology). Thus a ComponentSpace is a component container wherein the overall control logic is based on the cybernetic model and whose objects of control are coordinated through a multi-modal address space. It should be noted that the objects of control may also be ComponentSpaces as well as atomic components.

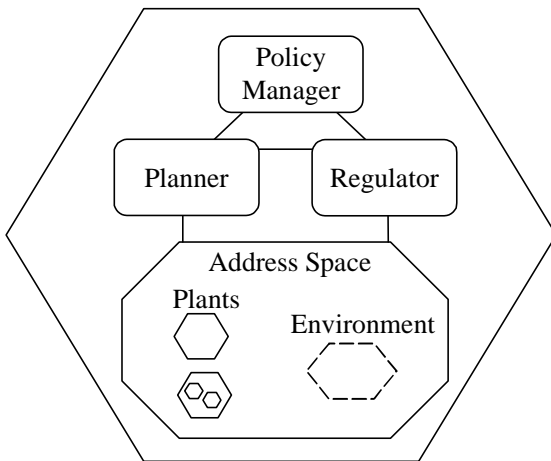


Figure 3 Generic Component Mapping

The generic mapping of the cybernetic model (of Figure 1) onto components is shown in Figure 3. In the figure a hexagon represents a component container. The basic elements of control are the Policy Manager, Planner, and Regulator. The objects of control (Plants) are shown embedded in the address space. Also shown in the address space is the Environment.

The Policy Manager implements the overriding rules of behavior for the application. These rules express the constraints and obligations imposed by higher-level (recursively embracing) applications. Examples of generic policies for overall operation of the application include security, transactions and protocols. The Policy Manager also serves to dynamically govern the action of the Planner and Regulator components.

The Planner's primary function is to assess both the environment and the internal state of the system with the goal of anticipating future actions and resource requirements. Modelling and simulation may be used to develop possible future plans. These potential actions are checked against the current plan by the Policy Manager.

The Regulator contains a representation of the current plan (or production schedule) for the operation of the Plants. The Regulator monitors feedback from the plants in relation to the accomplishment of the plan. It also conducts random audits of the condition of the plants to ensure operation.

The address space serves a number of modeling and implementation functions. First it provides a coordination medium for the interaction of the Planner and Regulator to manage the overall Plant and its recursive sub-component Plants. For example, it might be convenient to model the address space as coordinate vector space. In this case the location of components has physical significance. As a software infrastructure, the address space permits location and identification of components

for communications purposes. The multi-modal nature of the address space is designed to facilitate communications, especially messaging and event notification. In most instances we see the system as being event-driven. This includes events generated internally from the operation of the Plants as well as from the Environment. The addressing scheme is designed to allow dynamic "publish and subscribe" capabilities over geometrically defined regions, time intervals, etc.

The system as described thus far is operating as a basic homeostatic control system. We now emphasize the dynamic, self-organizing aspects. The so-called Plants represent a wide range of possible components that can participate within the applications envisioned. These include sensors, actuators, appliances, people, and robots. These real world systems are represented as proxy components within the ComponentSpace container. That is, a piece of software that provides an interface for communication and control of the actual physical component. Protocols to permit dynamic registration with the address space are under development by several groups. Examples of these protocols include Jini and Universal Plug-N-Play. We plan to evaluate these protocols within the context of our architecture. In any case, the dynamic self-configuration and self-organization of diverse components into a ComponentSpace is a major goal. Our hope is that the multi-modal address space model chosen will contribute to research in this important area.

5. SMART ENVIRONMENTS

DARPA and NIST have sponsored a number of workshops over the last two years on the subject of "Smart Environment." Their goal is to focus research directions on this complex domain. Smart Environments pulls together in one application many challenging requirements. To illustrate the intended use of ComponentSpaces we begin with a scenario based on Smart Environments.

A Scenario

On a university campus the new Information Environments Centre building is being completed -- just in time for the start of the semester. An engineer at SmartEnvironments Pty Ltd is using the ComponentSpaces™ WorkBench to assemble the component software system that will be installed in the new building. The WorkBench design package provides prototype ComponentSpace containers for buildings, levels and rooms. These component containers are based on the de facto industry standard CyberNetic™ Control Model for advanced feedback control systems. Additionally, these component containers implement the MultiModalAddressSpace® protocols that permit them to

seamlessly interoperate with other smart spaces in the system. Working from the building plans and the list of sensors, controls, furnishings and appliances, the engineer creates a model of the building. This model is easily constructed by drag-n-drop of commercial components from pallets in the WorkBench. Based on projected usage profiles developed during the design of the building, the engineer runs a simulation of the building for the next year. Satisfied with the component system design the engineer downloads the software into the building, level and room servers.

Later that day workmen begin delivering and installing the furnishings for the new building. In a commons room a workman hangs a picture on the wall. He hangs a clock on the opposite wall. A lounge, several chairs, and a table are also placed in the room. In an adjacent room a workman hangs a picture and clock on the wall. A desk and chair are brought in. Also a table is wheeled in and a printer is placed on it and plugged in. *[The picture is a SmartRoom server and contains various sensors that monitor and control the room. The SmartRoom server instantiates a ComponentSpace container and establishes connection with its level server. The ComponentSpace downloads specific policy, planning and execution instructions. The clock also instantiates a ComponentSpace that controls the embedded camera, motion detector and other sensor used by the room server. The other furnishings (lounge, chair, table, etc.) are also "smart" in that they contain embedded systems that communicate with the room server and the other furnishings. In this manner the room server discovers what furnishings and appliances it contains and associates specific components with them in its control model. The room server uses the MultiModalAddressSpace protocols to establish contact with adjacent rooms and learns what services they provide (e.g. the printer in the next room).]*

The next day, as students arrive, the campus packet radio network downloads class schedules, campus maps, security access codes, etc., into their SDA (Student Digital Assistant). Students in the Information Environments program check their schedules and are guided to the new building. As they approach the building, the campus packet network hands their SDA's over to the building's radio frequency network. The doors open (based on their security codes) and they explore the new building. In the commons room two groups of students spontaneously form a wireless (infrared) network to start work on their first assignment. One group finishes and sends the assignment to a printer. The smart room server locates the closest printer (the one in the next room) and sends the print job to it. The students are informed where they can pick up the print out. *[The smart building, level, and room controllers are recording and monitoring the comings and goings of the*

building's users, their activities, as well as the other environmental variables. The information is used by the room servers to learn how the building is being used and to anticipate future requirements.]

Scenario Analysis

The scenario illustrates three distinct phases of a Smart Environment: (1) design and implementation, (2) deployment and (3) use. We now discuss each phase and point out how our component framework could be used.

Design and Implementation: In the scenario, Smart Environments are designed and engineered alongside and in the same manner as other aspects of the built environment. The engineer used a CAD package to design the software for the new building. The package was (supposedly) based on ComponentSpaces. (Such component and container based packages currently exist: Microsoft's Visual Basic being the exemplar.) This package allowed him to choose from a range of third-party components to model and simulate the levels and rooms in the building. In this scenario the Plant is clearly the physical plant of a building. As it happens, most building control systems are based on control theory – so our VSM approach is a good match. The recursive nature of the software architecture supports the recursive nature of buildings, levels, and rooms. Also, the address space model chosen would most likely be a 3-dimensional coordinate system.

Deployment: A major feature of Smart Environments we intend to include in ComponentSpaces brought out in this phase of the scenario is device and service discovery, self-configuration and self-organization. As the various servers (the Picture) and sensors (the Clock) are plugged into the building network, there is a discovery/lookup/join protocol that permits them to determine their roles as participants in the overall building infrastructure. We intend that ComponentSpaces will use or improve upon such protocols. Similarly the furnishings (chairs, desks, etc.) and appliances (printer) have embedded systems that permit the rooms to register them. These protocols are used for the room server to download the component software that represents these elements. Most likely these real world components will take the form of proxies within the room services component container. Proxies provide the interface needed to communicate and control the device. We see a central role for the multi-modal address space in the task of coordination and control of the myriad devices and sensors.

Use: The daily operation of the Smart Environment building, levels and rooms is provided for by the models included in the ComponentSpace containers for each facility. In this case the Plan corresponds to the schedule

of classes, meeting rooms, special room bookings, etc. The ComponentSpace software for each of these facilities is periodically updated with policies for security, energy usage, etc. Modeling and simulation, based on records of past usage, is to predict and schedule a range of activities such as maintenance. Here the VSM cybernetic model fits in well with the application and is a meta-control system.

Another important aspect of Smart Environments is they are composed of many hierarchical, nested and overlapping address spaces. Our proposed multi-modal address space was designed with this requirement in mind based on previous work in cellular architectures. Smart Environments are both systems-of-systems and systems-within-systems. In this scenario the students are in constant communication with the campus-building-level-room-personal system. At any one time they may be using one or more of these wireless and wireline systems. The wireless overlay communication system represents yet another facility that must be incorporated into the Smart Environment and is supported by the basic component architecture [9].

Although we have used physical spaces to motivate this scenario, Smart Environments are not constrained by physical boundaries. This is shown in the scenario by the groups of students establishing spontaneous networks for a collaborative task. This mode of operation will be the norm in civic places such as airports and shopping towns. Once again our address space model was developed with this in mind.

6. CONCLUSION

In this paper we have presented the design of a new type of component architecture geared for a new class of application. Motivated by the complexity of these emerging applications we have chosen what may appear to be a rather complex architecture based on the (to some) unfamiliar discipline of cybernetics. Further work is needed to implement and verify this approach. However, in many ways we are just taking Shaw's "control paradigm" the next logical step.

To restate our position, we think new requirements being placed on software architectures cannot be addressed by ad hoc means. This approach has led to arbitrary and unmanageable software. We take a Systems Thinking approach, relying on proven models based on information theory and cybernetics. Our goal is to find principles and abstractions that can be applied at all levels within a complex software system. We have adapted the VSM to this end. If this paper has made a contribution it would be in this regard. Based on a comprehensive literature review, no instance of VSM applied directly to a software architecture has been

found. Another contribution lies in the recognition of the importance of address space as a central abstraction across all systems and elevating it into the modeling domain of component-based architectures.

Our plan is to apply the ComponentSpace design to the problem of Smart Environments as illustrated in the scenario. In order to do this we will further elaborate the design and implement it in on top of one of the commercially available component frameworks.

7. REFERENCES

1. 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.
2. Millennium, *Microsoft*.
<http://www.research.microsoft.com/sn/Millennium/>, 1998.
3. Jini, *JavaSoft*.
<http://www.javasoft.com/products/jini/>, 1998.
4. Miller, J.G., *Living Systems*. 1995, Niwot, Colorado: University Press of Colorado. 1102.
5. 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>).
6. Shaw, M., *Beyond objects: A software design paradigm based on process control*. ACM Software Engineering Notes, 1995. 20(1).
7. Beer, S., *Diagnosing the system: for organisations*. 1985, Great Britain: John Wiley and Sons Ltd.
8. Beer, S., *Decision and Control*. 1956, Great Britain: John Wiley and Sons Ltd.
9. 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://www.dstc.edu.au/TU/staff/herring/awsa98-herring.pdf>).
10. Gibson, L. and D. Lucas. *Spatial Data Processing Using Generalized Balanced Ternary*. in *Proceedings of the IEEE Conference on Pattern Recognition and Image Analysis*. 1982.
11. Kohonen, T., *Self-organization and associative memory*. Springer Series in Information Sciences, ed. K.-s. Fu, T.S. Huang, and M.R. Schroeder. Vol. 13. 1984, Berlin: Springer-Verlag.
12. Szyperski, C., *Component Software*. 1998, New York: Addison-Wesley.

APPENDIX: COMPONENT TERMINOLOGY

As with any new technology, terminology can be problematic in the beginning. We include this appendix to establish a consistent set of concepts.

Components, frameworks and architectures:

Szyperski identifies three major tiers of component systems [12]. The three tiers are defined below and their relationships are shown in Figure 4. Note the layered structure within a tier.

- Software **components** are binary units of independent production, acquisition deployment and extension. (Currently there are two examples: 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.
- A **component system architecture** consists of a set of platform decisions; a set of component frameworks; and an interoperation design for the component frameworks.

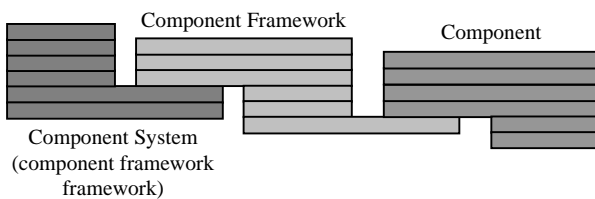


Figure 4. Component system architecture

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.

Component and containers: 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. ComponentSpaces take advantage of these commercial containers to implement an application specific component framework.

