Tangible Toolkits for Reflective Systems Modeling

by

Timothy M. Gorton

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 21, 2003

Author _____

Department of Electrical Engineering and Computer Science

May 21, 2003

Certified by _____

Bakhtiar Mikhak

Thesis Supervisor

Accepted by _____

Arthur C. Smith

Chairman, Department Committee on Graduate Theses

Tangible Toolkits for Reflective Systems Modeling

by

Timothy M. Gorton

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 2003

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

Understanding dynamic systems is an important but difficult task across a wide range of disciplines. Building such intuitions is particularly difficult for novices, for whom traditional representations of systems models are often opaque and confusing. Yet these novices often have deep, situated knowledge that is crucial to building powerful models. This thesis leverages work in the field of tangible interfaces to create an infrastructure for building domain-specific toolkits for models of dynamic systems that are transparent and actively support collaboration in their use. We report on two case studies for which we built toolkits based on this infrastructure. Drawing on these studies, we develop criteria for the design of toolkits that provide support for reflection by novices and experts alike. We further outline a framework for the effective use of reflective systems modeling in both academic and corporate settings.

Thesis Supervisor: Bakhtiar Mikhak
Research Scientist, MIT Media Lab

Acknowledgements

My thesis advisor, **Bakhtiar Mikhak**, has been a constant source of support, warmth, insight, inspiration, and motivation throughout the last five years of my association with the Media Lab. He has both worked tenaciously to create an intellectual space for his young research group and held the bar high for our achievements. My work would not be nearly what it is without his high expectations, guidance, insight, and direct involvement.

The other members of the **Grassroots Invention Group**, Dan Bryan, Sara Cinnamon, Margarita Dekoli, Chris Lyon, Michael Rosenblatt, Andrew Sempere, and the talented undergraduates who've worked with us have made the lab a energetic, fun, successful, and wacky environment to spend the last two years. Particular thanks go to **Chris Lyon**, who has repeatedly assisted this work through last-minute involvement, technical advice, and sustained verbal sparring.

Mitchel Resnick and the past and current members of the **Lifelong Kindergarten Group** provided a very supportive UROP experience, especially Bakhtiar Mikhak, Fred Martin, and Rick Borovoy for the opportunity to bite off just barely what I could chew.

Kenny Paul was an integral part of the genesis of this work, and his support, insight and humor helped guide it through our work with the Postal Service. He also provided a rare tour of the Boston plant to a small group of wide-eyed and wildly appreciative guests.

Benny Penta and **Greg Love** devoted their time during and after the workshop described in this thesis to help guide the development of this work, keeping it grounded in real-world challenges with enthusiasm and humor. The USPS management staff of the Boston distribution facility also generously donated their time to provide feedback on this work.

Adentunji Onamade, the manager of the Chelsea, MA Computer Clubhouse, helped gather and energize Clubhouse members for the workshops described in this thesis, and the **Chelsea Clubhouse members** provided an exhausted graduate student with hours of insanity and fun and were excellent hosts in between.

This documents has been immeasurably improved by the thorough reading and thoughtful feedback of **Bakhtiar Mikhak**, **Margo Dekoli** and **Dan Bryan**. **Sam Davies**, **Chris Lyon**, and **Andrew Sempere** also provided crucial feedback on portions of the document. Both I and everyone perusing these pages owe these kind souls a great deal of gratitude for their assistance.

My **friends** and especially my **family** have been an inexhaustible source of support, humor, and good-natured abuse, and I certainly would not have completed this work—or done a great many other more and less notable things—without them.

Table of Contents

Figures

Tables

1 Introduction

There is little question about the importance or potential impact of systems modeling and simulation in our understanding of dynamic systems—systems with behavior that changes over time—in a wide variety of fields, including manufacturing, business, science, and engineering. However, the only introductions to systems modeling in any of these fields generally occur in specialized university classes, only available to those pursuing one of these disciplines. The lack of transparent means of constructing and manipulating such models can limit the range of individuals who can be actively engaged in systems modeling, leaving most as mere consumers of the results of these simulations. There have been efforts to provide means for introducing broader audiences to systems modeling of ecological systems and the spread of epidemics through science education, as well as materials for introducing general systems modeling to elementary school students. However, with a few notable exceptions described in section 2.1.2, these efforts have remained constrained by traditional representations used in desktop software packages for simulating such systems.

This work focuses on the rich domain of building tools for reflection on dynamic systems. Though constructing any such tools will necessarily involve technical issues relating to building systems simulations, the primary challenges in this domain deal with how we can learn, think, and communicate with these tools. Like Senge, [34] we focus on building microworlds in the spirit of Papert's work [25] in order to provide somewhat structured environments in which a wide range of individuals can build and manipulate system models in a particular subject domain. There are primarily two activities that must be supported by such tools: reflection through manipulating models and reflection through building these models. In both cases, there is clear value for collaboration in these activities, whether they take place in a corporate office or an elementary school classroom.

These tools must support a variety of levels of engagement and reflection in order to provide a comprehensive environment for exploring dynamic systems. It must both be

possible to reflect on the system as a whole and examine the behavior of individual elements. This can provide a new perspective for both individuals deeply enmeshed in a system such as a workflow process and students approaching a subject domain for the first time. In order to support non-traditional audiences for systems modeling, especially those without formal training in systems modeling, we explore the approach of building tangible models of dynamic systems by using embedded computational modules to represent objects in a simulation. Compared to traditional on-screen diagrammatic representations, this has both the possibility to change the relationship of individuals to objects in the simulation and provide a means for small groups to collaboratively build and manipulate simulations in their shared, physical space.

This thesis involves two case studies in two specific domains in order to provide concrete proof-of-concept instances of the model-building infrastructure developed in this work. The first domain involves building models of workflow processes such as manufacturing, supply chains, responding to customer-service issues, and many others. These processes are the focus of nearly every business, and thus these processes are already often the subjects of intense and ongoing systems modeling efforts in order to increase efficiency in ways that directly impact the business's profits. However, achieving concrete results from these efforts may prove difficult due to challenges in communicating their results to others who are not involved in the modeling effort but are deeply invested in—and often responsible for—the processes in question. Despite such efforts, it remains difficult for many of these individuals in an organization to develop robust understandings of these systems.

This domain has been explored through a close collaboration with engineers from the United States Postal Service's Boston Processing and Distribution Facility, who worked with us to build models of the flow of pre-barcoded mail through their facility in order to provide a means for USPS managers and supervisors to collaboratively explore alternatives in scheduling operations under normal and extraordinary conditions. Engaging a broader audience within an organization in collaboratively building and manipulating models of these systems has the potential to leverage a much wider range of

deeply situated knowledge that individuals across an organization can bring together and share.

The second domain we have explored is computer networks and network applications, a type of distributed system that has long been the subject of simulation by engineers. The use of the Internet and private networks has become increasingly commonplace, but the mechanisms involved remain opaque to end-users at a variety of levels. In fact, this opacity is in some sense the goal of much of the engineering that has produced our modern networks and applications; the structure of these networks and applications may only be apparent or interesting to end-users during failure conditions such as, for example, when all of the individuals using a chat system from the same high school are suddenly disconnected. Yet during these conditions users have little information and little understanding of the system that they may spend a great deal of time using. Modeling such a network and network applications can provide a transparent way for individuals to experiment with different network structures and factors that can affect their performance and fault tolerance. To this end, we have created a toolkit for modeling a computer network and a chat system to run on it. Our evaluation of this modeling activity has involved a group of 10- to 12-year-olds who were frequent Internet users but had little understanding of the workings of the networks or applications they used.

The structure of the remainder of this thesis is as follows:

Chapter 2 examines related concepts and approaches to modeling dynamic systems in order to develop design criteria for our work, followed by a discussion of our approach to supporting reflective systems modeling.

Chapter 3 discusses our two case studies, exploring the need for a new form of systems modeling in each domain and describing our work with the groups involved and the application-specific technology which was required for each domain.

Chapter 4 describes the design and implementation of the infrastructure which we have created in order to support the creation of domain-specific toolkits for this style of

systems modeling. This chapter also outlines a methodology for using this toolkit to create application-specific toolkits.

Chapter 5 contains our reflections on the design of our infrastructure, the impacts we have observed in our case studies, and the roles of practitioners which have emerged from our case studies.

Chapter 6 describes a framework for the practice of reflective systems modeling in a variety of contexts, situating the work described in this thesis, other related work using pieces of this infrastructure in our research lab, and several scenarios for future work.

Chapter 7 summarizes the results of our work and potential future directions.

The **appendices** provide deeper technical details about the implementation of the modeling infrastructure and also reproduce two other short papers written about tangible modeling efforts that are discussed in this thesis.

2 Approaches to Systems Modeling

2.1 Related Work

Numerous researchers have asserted that modeling dynamic systems constitutes a powerful means of understanding complex systems in the world around us. Jay Forrester, the inventor of system dynamics, which is a set of techniques for systems modeling, has argued that systems modeling should be an essential component of pre-college education and, in fact, can offer a framework for bringing cohesion, meaning, and motivation to education from kindergarten onwards. [7] Forrester and many others have worked to create tools and methodologies that can enable relative novices to build models of complicated systems in order to provide a broad audience with a means of understanding complex, real-world systems. This section examines some of these researchers' work, as well as several others whose writing provides a conceptual framework for our work in developing tools and methodologies for systems modeling.

2.1.1 Conceptual Frameworks for Systems Modeling

There have been several key writers whose insights have guided our work in developing environments for novices to engage in systems modeling. Seymour Papert's work on constructionism [25] can provide insight for the design of these environments. Papert, building on Jean Piaget's work, has shown that children refine their models of the world most effectively during the construction of personally meaningful, public artifacts. Papert describes a learning process of "getting to know" an idea rather than learning specific facts or skills, and he suggests that particular environments, which he calls "microworlds," may be designed with intrinsic properties which place ordinarily obscure concepts and entities directly within a learner's manipulation. [25] Turkle and Papert have also stressed the importance of multiple representations and means of access to these microworlds as crucial elements to ensuring that they reach a broad audience. [36] Papert's notion of a microworld as an environment for "getting to know" new and counterintuitive systems has been appropriated by numerous researchers in their efforts to introduce novices to systems modeling.

Donald Schön's descriptions of the "reflective practitioner" [32] may also guide our work in properly situating systems modeling in the everyday practice of the workplace. In contrast to his portrayal of a "technical rationality" professional as a problem solver using the results of science in a stable context, Schön describes the reflective practitioner as being necessarily embedded in complex situations and systems that he seeks to understand. Often, the practitioner may directly modify the situation in order to understand its workings. In many cases, Schön writes, "virtual worlds" may prove a more effective environment for hypothesis testing. He describes these virtual worlds as the "medium" of reflection-in-action, saying that practice in the construction, maintenance, and use of these virtual worlds develops a capacity for reflection-in-action which he describes as artistry. In order to support development in a particular domain, these environments must be closely linked to the context of the practitioner's actions. Schön also envisions a substantial impact for making the interactions between professionals and clients much more open if reflective professionals are open about their reasoning and understanding of these systems with clients, who also work to build their understanding of the situation in order to make well-informed decisions. [32]

Peter Senge [34] stresses that understanding complex systems is also important in management, where, as he writes, the real leverage often lies in understanding "dynamic complexity not detail complexity." Senge stresses "systems thinking" as a discipline for seeing structures that underlie complex situations and a language for describing these structures. He also emphasizes the importance of learning for individuals and teams. Senge says that the appropriate environment for this learning is experimentation within Schön's virtual world, for which he then uses Papert's term microworld. Senge asserts that learning through experimentation only works when the results of a decision are rapid and unambiguous, and he uses microworlds to compress time and space in order to enable this speed and transparency. He sees the personal computer as a platform for creating powerful and expressive microworlds where groups of individuals can expose, test, reflect on, and improve their mental models. In many of Senge's case studies, his focus is not on the operation of the software models themselves but rather on how the participants were able to use the models to spark conversations that illuminate previously hidden assumptions from the participants. Another remarkable feature of Senge's view of

these microworlds is that he describes using them as "play" for the participants, because their decisions can be decoupled from the real-world consequences of decision-making, and he asserts that this may be the most powerful method for learning in an organization. [34]

Michael Schrage focuses on this notion of "serious play" as the foundation for an organizational culture of prototyping, based on his assertion that "innovative prototypes generate innovative teams." [33] Schrage's writing provides some guidelines for how organizations should treat systems models in their work practices. He describes the value of "quick and dirty" prototyping using rapid iterations to explore alternatives and refine designs. They may also serve as shared space between organizations and clients or between individuals within a team. Like Senge, he emphasizes the importance of models' impact on human understanding and relationships rather than their technical features as the measure of success. Schrage also describes the importance of enabling others to play with a prototype in a way that creates an "innovative team" of collaborators, often drawing from unexpected sources, such as customers, vendors, supervisors and colleagues. In this case too, the transparency of the prototype is paramount in order to engage these constituencies. [33]

2.1.2 Existing Tools for Systems Modeling

Jay Forrester explicitly states that merely talking about systems models is far from sufficient; students must have a deep involvement in creating models of systems that are both interesting and relevant to their lives. These models may draw from topics as varied as biology, economics, social systems, and even character motivations in literature. [7][8] To support this work, Forrester and his team have created a set of "Road Maps" as a self-guided introduction to system dynamics using selected literature and modeling exercises. [24] Forrester reports favorable, though anecdotal, results from a number of schools pursuing system dynamics within their curricula. Perhaps the most interesting of Forrester's results from these studies is that students' past academic success does not appear to predict how they respond to the systems modeling material, an interesting sign

that this constitutes a substantially different means of allowing learners to explore material across a variety of disciplines. [7]

In addition to the traditional software representations used by Forrester's work, such as STELLA™ [12] and Verisim™, [38] numerous researchers have created other environments to introduce systems modeling to novices. These even include simulation games such as SimCity, which has introduced a wide audience to manipulating a complex simulation in a gaming environment with few explicit goals. SimCity's designer describes playing the game as "the process of figuring out how the model works." [35] Paul Starr, however, expresses his concern at this, citing certain assumptions and simplifications in the model that are not transparent to users in SimCity and SimHealth. [35] Starr's comments remind us of the importance of transparency in model construction, an idea also found in the writings of Papert, Schön, Senge, and Schrage. Researchers have attempted numerous representations to fulfill these goals, including programming, tangible representations, and participatory simulations.

Some researchers have provided environments to make it easier to write computer programs to explore ordinarily opaque systems and situations. Mitchel Resnick invented StarLogo, a massively parallel variant of the Logo programming language that allows users to easily create thousands of "turtles" and write simple programs to control their movement and interactions with each other and with their environments. This allows users to model dynamic systems such as traffic jams, flocking birds, and termites building dirt mounds; Resnick demonstrates that this environment can help break the "centralized mindset" whereby people believe that a single cause or leader exists for these decentralized systems. The nature of the language makes it difficult to create a single leader but easy to create thousands of turtles following the same program, encouraging users to explore such decentralized systems. [31] Uri Wilensky also used the StarLogo programming environment to allow individuals to explore issues of probability. Wilensky's work uses the StarLogo language as an environment for users to answer probability questions by setting up experiments. Especially interesting of his work is that his subjects often learned not by the results of these experiments but from the simple necessity of concretely formulating the problem as it was created in StarLogo, such as

one subject's realization that two different means of picking a "random chord" of a circle produced very different probabilistic effects.

Other researchers have instead utilized tangible representations of systems in order to make it easier for novices to interact with abstract models. These tangible interfaces can utilize users' ability to manipulate physical objects in space as a means of manipulating digital objects. These interfaces can also take advantage of the physical properties of the objects, such as making it impossible to place two objects in the same location, as well as placing digital objects in the user's personal space instead of a computer screen. [15] Based on these ideas, Patten, Ishii, et al. have created a system called the Sensetable, an object-tracking platform for such interfaces. One of the first applications built upon this platform was a system dynamics simulation, in which a traditional system dynamics representation was projected onto the Sensetable, and users were given dials that could be placed atop projected simulation objects to modify simulation parameters. Using the position and orientation of multiple objects placed on the table, the system was able to zoom in on areas of interest to the user. However, this system only permitted users to manipulate such models, not create them. [27][26] Kobayashi, et al. have also used the Sensetable to manipulate a projected computer network model. This system also did not support the creation of such models. [18] Following the completion of the case study with the US Postal Service described in section 3.1, Oren Zuckerman began using some of the infrastructure built for this thesis to build a tangible toolkit to allow young children to build and manipulate system dynamics models. [39] Zuckerman's work is described further in relation to this thesis in section 6.1.

Another approach to involving students in simulation modeling has been to make the students themselves elements of the simulations. Vanessa Colella's work in "participatory simulations" enabled a group of 15-30 individuals wearing computational "badges" to take part in a simulation of the spread of an infectious disease through a community. Participants could move around, and might catch a disease when two badges were brought near each other. By providing similar but non-identical experiences among participants, each participant could become both a collector of data and a designer of experiments relating to their own behavior. This immersion proved effective in

stimulating discussion and collaborative experiment design among students. [2] Following Colella's work, Eric Klopfer has continued to develop other simulations using similar badges based on Cricket technology, [21] including the "prisoner's dilemma" and simulations exploring concepts in genetics. Researchers at the University of Toronto have built additional simulations for this platform. [17]

2.2 Our Approach

Our aim is to provide transparent tools for small groups of individuals to collaboratively build systems models that are deeply relevant to their work or their interests. We strongly agree with the importance placed upon participants' construction of such models by Papert, Senge, and Schrage, and we also recognize the need to support manipulation of the model to draw others into the system under investigation to help create the innovative teams that Schrage describes. Multiple levels of structured microworlds are involved in any such project, including a domain-specific toolkit of model elements that a group of individuals could use to create a model, as well as a particular model acting as a microworld for others to manipulate and explore. In order to support designers of a variety of domain-specific toolkits, we add a third structure to this process: an infrastructure which provides support for building models in a style which aims to satisfy the demands discussed here of transparency, collaboration, and a variety of levels of access.

In designing this infrastructure, described in detail in chapter 4, we have created the means for building tangible representations of objects in a model and bringing the simulation into participants' physical space in order to make the model more transparent and facilitate collaboration. In order to avoid complicated projection systems while providing an infrastructure for active simulation elements, we have used embedded computational elements, connected together using wires to explicitly connect elements that exchange information, as shown in Figure 1. This creates a computationally distributed model of the system under study.

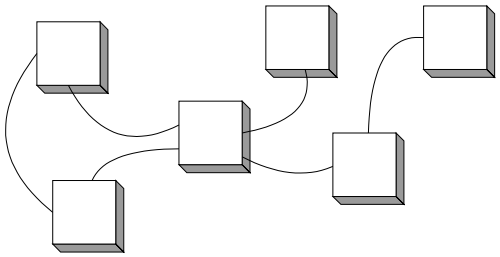

Figure 1: A network of computational elements for a tangible system model

In addition to creating this infrastructure to support a broad range of domain-specific toolkits, we have also performed two case studies involving the creation and evaluation of domain-specific toolkits. These two case studies, described in chapter 3, have focused on modeling US Postal Service mailflow in collaboration with USPS engineers and modeling computer networks with 10- to 12-year-old youth. These studies have demonstrated the usefulness of this type of infrastructure and toolkits for reflective systems modeling in two very different environments, a workplace and an informal educational setting. Additionally, two projects in our research lab that are not directly part of this thesis work have explored two other settings: computer network research and engineering and exploring system dynamics with very young children. These projects are described in more detail in section 6.1. Taken together, these case studies span a wide range of systems: manufacturing systems, computer networks, computer hardware, and formal system dynamics. Other areas with clear potential to benefit from these tools and methodologies include social systems, other business practices, economics, and biological studies such as the classic predator/prey model.

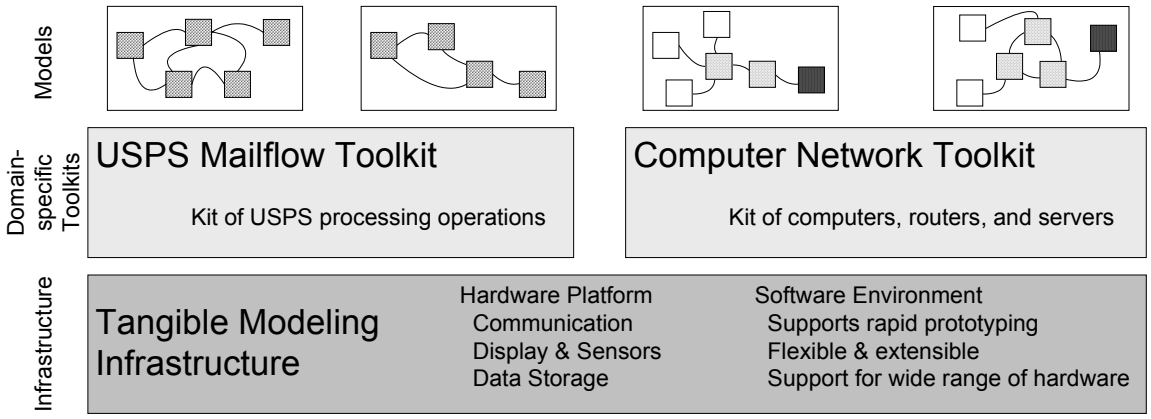

Figure 2: Modeling methodology overview

In order to implement the network of computational objects described above, we have used and extended the Tower system, a modular system for designing embedded computational systems, developed by Chris Lyon, Bakhtiar Mikhak, and myself. [20] The Tower, shown in Figure 3, consists of a set of computational Foundations and a set of layers which can be stacked above a Foundation to add various types of other functionality, including storage, communication, sensors, and actuators. In order to support the needs of this type of network, a new Foundation was developed for the Tower system using a Rabbit Semiconductor™ microcontroller module, [29] supporting multitasking and string manipulation. This Foundation is described in section 9.1, and the Logo virtual machine created for this platform is described in section 9.2. The desktop programming environment created for this platform grew into a general platform for allowing users to program Tower Foundations and write assembly code for Tower layers; this environment is described in section 9.3. The need to connect Towers to several other Towers in a simulation led to the creation of a serial communication layer for the Tower, described in section 9.4, which contains four buffered, bi-directional serial ports. Packet parsing and routing libraries were also developed for this platform; these are described in section 9.5.

Figure 3: The Tower with many of the layers used in this thesis

Other components of the Tower system, including sensor layers, tricolor LED modules, display layers, and PS/2 keyboard layers, were also added to these Towers in creating the particular models described in this thesis, with these components providing ways for the simulation objects to display information to users and ways for users to interact with these objects. The Towers were connected together using wires to build a simulation model, such as the one shown in Figure 4. Logo code was written for the Towers in order to enable them to act as elements in the simulation model, providing appropriate displays and interactivity and behaviors appropriate for a particular domain. A laptop computer was also used in some cases as one node in the network, enabling users to visualize data and aggregate in ways not possible on individual Towers. The individual models created for this thesis are described in more detail in chapter 3, and more information on the modeling infrastructure is included in chapter 4.

Figure 4: Towers used in the USPS simulation model described in section 3.1

3 Case Studies

This section discusses the two case studies which we have conducted using the Tabletop Process Modeling Toolkit. The first to be discussed is the modeling of a workflow process in collaboration with the United States Postal Service. The process that we examined together was the flow of pre-barcoded mail through the USPS Boston Processing and Distribution Facility. This case study benefited from the input and involvement of individuals in a wide variety of roles and locations from the Postal Service, and the following subsections will detail their reactions and input.

Our other case study has involved building models of computer networks with youth who spend time at the Computer Clubhouse, [3] an after-school center which encourages disadvantaged youth (called "members" of the Clubhouse) to use technology creatively. These members use computers, file servers, and numerous Internet sites daily at the Clubhouse, so they were a natural choice for exploring the operation of computer networks.

For each of these two case studies, we first provide background information on the systems under study and motivation for the study's inclusion in this thesis. We then discuss the nature and results of participants' interactions with the models in each study, followed by a description of the application-specific toolkits created for the study. As discussed in section 2.2, these studies both support the validity of our approach to tangible systems modeling and provide important cases in different types of systems and environments for this work.

3.1 Workflow Models

3.1.1 Background: Managing Postal Mailflow

The Postal Service manages 38,000 post office locations, 740,000 employees, and dozens of processing facilities. Each facility contains 20-50 mail-processing machines. Analyzing the flow of mail in the Postal Service is very challenging due to the varying ways that mail moves through a processing plant. Different types of mail (e.g. advertising

mail, first class mail, parcels, periodicals, etc.) have different delivery standards, and thus these different types of mail have separate streams within a facility. All of these mailflows, which move through automation and mechanization within a postal facility, create a complicated web of physical and information flows. [11]

Systems modeling plays a major role in the management of the postal service on every level; however, according to Postal Service engineers and managers, the effectiveness of these simulations is limited due to the challenges in communicating their results to the managers and supervisors who need to utilize these results in real-time decision-making. There is a great deal of anecdotal evidence from these engineers and managers that managers often find the results of these simulations counter-intuitive and untrustworthy, and, as a result, may ignore them entirely. [11]

The existing simulation model used in the Postal Service requires a week of training to learn to use and requires someone who already has a very good understanding of mailflow. The user—a postal engineer responsible for managing mailflow—enters volumes of each mail type, operation numbers, arrival profiles, maintenance schedules, percentages of mail that flow from one operation to another, and so on. The simulation produces equipment requirements in order to process the associated volumes of mail. The user must trust the simulator since it produces only the final results in table format, without any way for a user to build his intuition about the underlying model. Others in the postal facility who depend on these results certainly have no way to understand the model's workings. Additionally the tool does not calculate people requirements (staffing), let alone attempt to optimize staffing according to varying workloads. [11]

It is important to note that the problem is not with the existing simulation tools' technical merits or fidelity in capturing the subtleties of the situations they are used to model but rather the accessibility of their results to managerial decision makers and supervisors who create day-to-day schedules. At the root of the problem, the interaction between the engineering team and the management team often consists largely of opaque written reports because the simulations models used are too abstract and inaccessible for non-experts to understand or manipulate. [11] In fact, expertise and intuition about the

workings of the mailflow is very limited in the Postal Service, limited to several individuals in a large distribution facility such as Boston's, for example. [28]

In our discussions with a wide range of engineers and managers from the Postal Service, it has become clear that the Postal Service represents a particularly good context for exploring the potential impact of tangible simulation tools, both in the tools' ability to provide a more intuitive means of manipulating models of dynamic systems and because of the collaborations possible among a small group of individuals working on such a model together. Despite the importance attached to modeling and simulation efforts, the lack of understanding and communication about these models substantially hinders the efficiency of mail processing. This is particularly apparent when discussing proposed changes in the mailflow or scheduling, because it is difficult to enact change without widespread understanding of a proposed change's potential effects among supervisors and managers, who are unable to utilize the simulation models used by the plant engineers.

At the suggestion of Kenny Paul, the Postal Service's liaison to the MIT Media Lab and an engineer at the USPS Boston Processing and Distribution Facility, we chose to focus on the flow of pre-barcoded mail through the Boston facility. This flow contains five operations between the mail entering the facility and being placed in trucks for dispatch to other local post offices and other processing facilities. The Postal Service refers to this type of mail as Facing Identification Mark (FIM) mail, and USPS engineers commonly use diagrams similar to the one in Figure 5 to illustrate such flows.

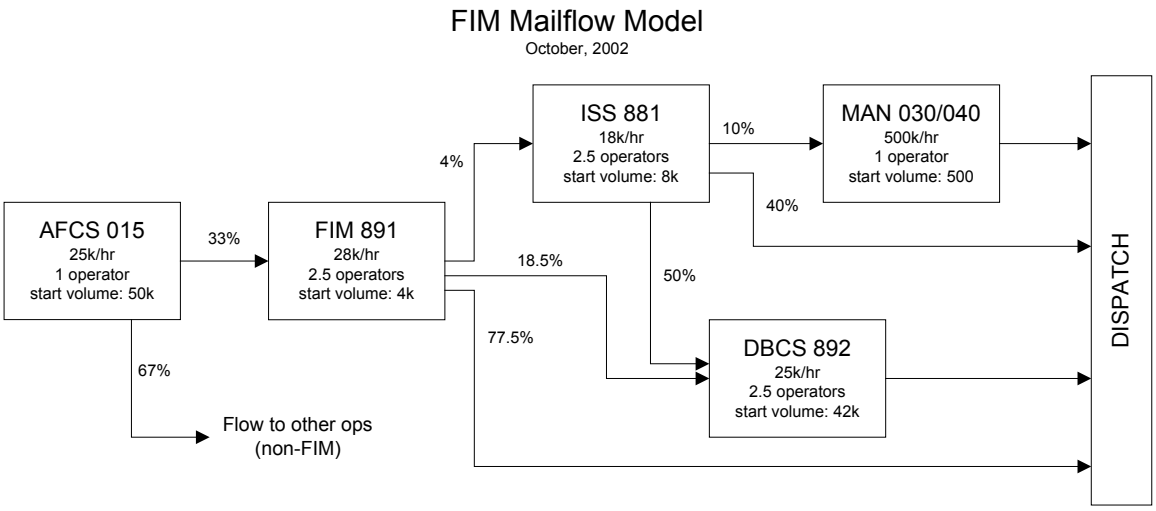

Figure 5: USPS Facing Identification Mark (FIM) Mailflow Diagram

3.1.2 Initial Prototype

In September, 2001, we constructed an initial prototype intended to model the processing of pre-barcoded mail in the Postal Service's Boston Processing and Distribution Facility. Our initial prototype was constructed using the Cricket system, developed by the MIT Media Lab's Lifelong Kindergarten Group. [21] This model was designed to simulate the processing of mail in the plant by passing messages between computational objects indicating how much mail was flowing between different operations.

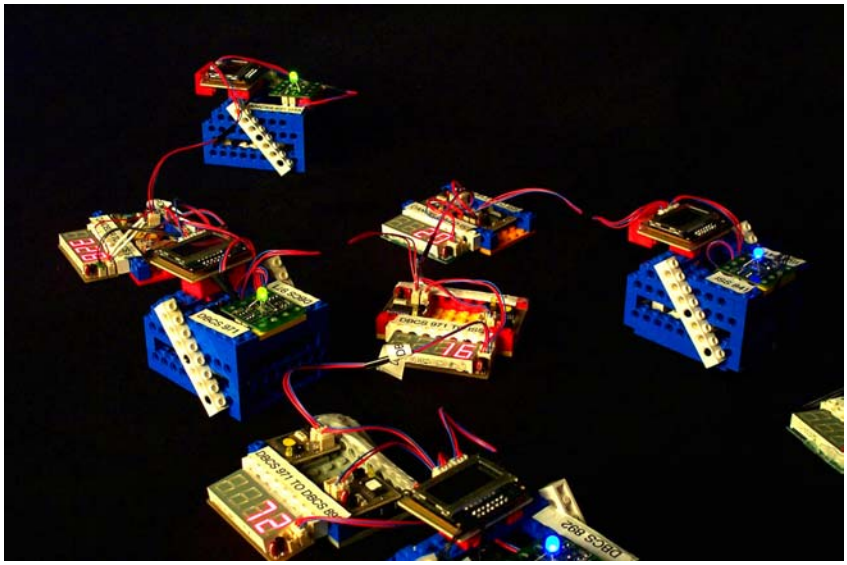

Figure 6: Initial prototype of USPS FIM mailflow model

In our initial prototype, single Cricket modeled each processing operation in the facility, with a number of "bus devices" [21] attached for display and communication. Each Cricket, contained within the blue LEGO structures in Figure 6, maintained data about the operation's queue size, throughput, start volume, the number of machines operating, and workhours used in the operation. The Crickets were connected to each other in order to send information about how many letters were being passed between operations. These connections were created using pairs of infrared transceiver bus devices, one connected to the Cricket on either side. Each Cricket had two means of displaying information about the operation's status: a small, two-line LCD display and a tricolor LED. The LCD display was used to display the operation's current throughput and current queue size (that is, the amount of mail awaiting processing at the operation.) The LED was used to indicate the state of the operation, as listed in Table 1. Finally, a laptop computer acted as the delivery trucks, injecting mail into the model according to a mail arrival profile from a typical day at the Boston plant. After a night's processing was complete, the laptop could also query each node to obtain statistics about the operations to aid in reflection about the model's operation.

Color	Description
Blue	Operation has not started.
Dim or flickering green	Operation is running at partial capacity.
Bright green	Operation is running at full capacity.
Dim of flickering orange	Operation has received its last incoming letter and is running at partial capacity.
Bright Orange	Operation has received its last incoming letter and is running at full capacity.
Red	Operation has finished processing.

Table 1: LED colors used to indicate operation status in USPS FIM models

Though the embedded Logo virtual machine allowed us to quickly begin prototyping our application, the Crickets' limited speed (at most 300 logo instructions per second) and memory (sixteen 16-bit signed numbers, plus several thousand bytes of much slower EEPROM) [21] created serious challenges to constructing our model. But the most severe limitation was the communication between Crickets using the infrared transceiver bus devices. These devices were not designed to implement high-reliability communications

and are limited by the necessity of interrupting the receipt of data when the Cricket uses the bus protocol to initiate communication with any device. [21] Through resending bytes until the receiver echoed them, we were able to implement the reliable receipt of a single byte between Crickets, but this severely limited the amount of data we were able to send between operations in the model.

As a result of these limitations, the model did not allow users any interaction with the model's parameters because the volumes of mail sent between operations were essentially hard-coded, since the actual values were, in fact, too large to be represented in the eight available bits which could be communicated between Crickets. Instead, messages were sent either to indicate that a batch of mail was received or to reset the operations' parameters. However, the batch sizes and other parameter values were merely constant values contained in the program loaded onto each Cricket.

Yet the prototype served its purpose well in allowing us to evaluate the potential impact of tangible systems modeling to affect the practices of individuals and organizations, both within the Postal Service and in a variety of other sponsor companies of the MIT Media Lab, who regularly tour our lab space. Though the model did not yet have sufficient fidelity or interactivity to be useful for reflection about the USPS engineers' challenges, it did serve as a provocative example of possible means of sharing systems models, engaging individuals from a variety of organizations and levels within the Post Office and other companies in reflecting on how their organizations could benefit from better communication about and around system models.

Even this early model elicited a wide variety of feedback from individuals in different areas and levels of these organizations. Our early feedback came from the Postal Service itself. When the model was demonstrated to a group of Postal Engineers, despite overall positive reactions, side conversations focused on calling the model "simplistic." Jeff Freeman, a USPS manager listening to the demonstration and these comments, later noted that many of these engineers were comfortable working with on-screen simulation models and tables of results. However, Mr. Freeman also described the potential for others in the organization to benefit from being able to "watch the mail flow" through a

tangible model, as well as the possibilities for a group of engineers to gather around a tabletop model to manipulate and explore the mailflow process. [10] Other reactions included suggestions about other scenarios, such as including traffic between major processing facilities and adding new delivery products to the Postal Service, and other audiences who could benefit from a more transparent model of the work processes, such as union leaders who need to understand the reasons behind changing staffing needs and work conditions. Feedback from other organizations included interest in modeling manufacturing processes and some novel ways that individuals interact, such as one Media Lab professor's suggestion that it would be enlightening to model how the Media Lab's staff manages interactions with the lab's sponsoring companies. Following substantial interest from the USPS and the willingness of a pair of engineers to devote some time to helping us drive this project further in the context of modeling USPS mailflow, we organized an in-depth workshop in September, 2002.

3.1.3 Results of Workshop in September 2002

Before an in-depth workshop with USPS engineers was possible, a great deal of work was needed on the technical infrastructure to support the creation of robust, high fidelity models with the ability to rapidly modify simulation parameters. By mid-summer 2002, we had developed the modeling infrastructure described in chapter 4 using Towers to the point that we were able to work with Kenny Paul, the USPS liaison to the Media Lab and a USPS engineer at the Boston facility, to update the previous model, using the new infrastructure to create the model shown in Figure 7 and set up a workshop with two USPS engineers in order to understand how both this particular model and this type of model in general could engage USPS personnel in reflective systems modeling. This workshop took place over three days in September 2002. The first two days took place in the MIT Media Lab, and the third day involved the two engineers traveling with Bakhtiar Mikhak and I to the USPS processing facility in Boston to discuss the possibilities of these models with the management staff of the plant. The USPS participants in the workshop were Kenny Paul, Benny Penta, an Operations Support Specialist at the Boston facility, and Greg Love, an engineer at the Boston facility. The description of the results

of the workshop, and especially the conversations which occurred, are based on our notes written during the workshop and subsequent conversations.

Figure 7: Second-generation USPS FIM mailflow model

On the first day of the workshop, we began with a demonstration of the model and a discussion of the engineers' work and how tangible models might be useful. Despite their interest, several tensions were immediately apparent in these discussions. First, Benny and Greg were reluctant to consider the part of the process that we had chosen to model in isolation, insisting that the model would need to encompass the entire plant—approximately 100 operations—in order to be useful to engineers or managers, even though Kenny Paul, a fellow USPS engineer, had suggested this particular segment of the process as a good candidate for the model and worked with us on its implementation. The nature of the interface to the model was also an interesting issue, with both Benny and Greg suggesting an on-screen representation more similar to the flowcharts that they currently use.

But the most interesting tension in the first day's discussion was the challenge of how to structure the interactions possible with the model to allow non-engineers in the Postal Service to benefit from it. Both Greg and Benny wanted to prevent managers from choosing parameters that might, for example, exceed the actual possible throughput of a machine. However, Benny was also interested in preventing the managers from choosing

parameters that would violate the planned end times of the process, while Greg thought that the plan itself could be open to evaluation and manipulation. However, Greg also suggested that another process, Delivery Point Sorting, need not be included in the processes to be manipulated by the managers because it didn't "need" to be modified.

One other noteworthy challenge emerged on the first day of the workshop involved how to control the times at which the various machines would start. Benny and Greg focused on simple rules to control when operations started and when machines were added as volume increased. Though while looking at the model they asked questions about what would happen if machines were added at specific times, they decided to use rules about how much mail accumulated in front of an operation before starting the operation or adding machines. Though they acknowledged that this method might not cover all situations, their intent may have been to more directly lead their manipulations of the model to rules that supervisors could use on the floor.

Despite their reservations, both engineers did expect these tangible models to be useful in conveying the results of their work to others. As Greg pointed out, the Postal Service has a great deal of detailed historical data regarding their process, "just not in a format we can use to convince managers and unions." Specific explorations that they suggested for this particular model included staff scheduling, how to add new operations to the mailflow, the effects of different arrival profiles, and the effects of different machine throughputs due to wet mail, machine downtime, and poor address recognition by automatic equipment.

On the morning of the second day of the workshop, we had made the changes to the model that the engineers requested, including displaying more data on the operations and allowing users to set the throughput, start volume, and volume to add another machine on each operation. (Ideally, the infrastructure would allow engineers to make these changes themselves, though further development will be required before this is possible. More discussion of these needs is found in sections 3.1.5 and 4.4.) Benny and Greg also arrived with a scenario to explore based on a crisis that had occurred the previous night at the processing facility. Ordinarily this plant receives about 800,000 pieces of mail on a

Monday night, and the previous night an extra 200,000 pieces had arrived in the final deliveries to the plant. After this additional volume occurred, as they put it, "everything blew up." Data from the run indicated that all of the operations had finished much later than normal, preventing the on-time delivery of much of the mail to area post offices. The engineers were very concerned with the end times of the operations since this also impacted other processes performed on those machines later in the evening. The engineers also criticized a decision by the floor supervisor to move a large amount of the unexpected mail into a different process in order to limit the extra time that the operations were used that evening. This alternative process used extra time at downstream facilities and was not represented in our model.

We began by using the model to simulate the effects of the added volume on a nightly run using the ordinary parameters. We found that the later finish time predicted by the model for the first operation exactly matched the actual finish time and that if the supervisor had not removed the mail from the FIM machine, the process would have lasted until about 3am, two hours longer than usual. The engineers suggested adding another FIM machine at a particular time, and this allowed the operation to finish by 1am, causing the whole process to finish only a half-hour later than usual. By using the laptop computer to query the Towers, we are able to determine the extra workhours that this solution would have required. The engineers were very excited to show these options and consequences to management the following day, because they often have a hard time convincing supervisors and managers of the validity of their predictions.

During the second day, I observed a rapid change in the engineers' engagement with the model. After their insistence during the first day of the workshop that the model would need to be complete in order to be useful, in exploring the situation described above both engineers began to develop a deeper understanding and more trust in the model's operations. They also became much more willing to take control of running different scenarios through the model, as well as identifying specific data to get from the model, when similar data might not be available from the actual process. For example, Greg asked during one discussion, "What was at the machine at 10:30pm when he shut it down? We should be able to simulate that." Kenny gestured towards one Tower as the

simulation ran, saying "it'll be interesting to see when this one comes down." The engineers' growing trust in the model was evident in a number of noteworthy comments they made while running and manipulating the model. Perhaps most memorable was Benny's exclamation as he leaned forward to watch the simulation begin, "I've got mail to process!" He repeated this later while starting the model, telling Greg and Kenny, "We're running mail here, you two can take a break." There was, however, no doubt that the engineers were aware of the dangers of mistaking simulations for reality, as Greg noted during one discussion, "remember, it's just a simulation."

While experimenting with different start volumes and volumes at which to add machines, we began to see the effects of relationships between operations. These effects were familiar to the engineers but are rarely seen so clearly. These included the effects of operations' finishing times on downstream operations as well as the ability to optimize an operation to improve its workhours or finishing time, but not both simultaneously.

The presentation to the plant management went quite smoothly, and was, remarkably, driven almost entirely by the two engineers who had spent the previous two days at our lab. They demonstrated the model, showed the possible choices for handling the increased volume they had experienced two days earlier, and discussed with the managers how this compared to other internal projects in the Postal Service to obtain better data from their processing machines. The staff members were generally interested in the possibilities of this approach, but their interests varied widely. Several members of the management staff said that they'd like to have a copy of the model on their desks or the supervisors' desks in order to experiment with possible scenarios and examine the previous night's operations. Much of the interest also focused on obtaining real-time data from their processing machines, which has been the subject of at least one largely unsuccessful internal initiative inside the Postal Service. One staff member suggested that these goals could be combined, allowing a supervisor to view the night's operations thus far and experiment with possible choices for the remainder of the shift.

Following this experience, Benny was enthusiastic about the possibility of expanding our work within the facility and our suggestion that we consider getting a group of shift

supervisors together to experiment with different scenarios. Benny was quick to point out that machine operators and supervisors of a specific operation don't understand other supervisors' operations or the effect that their decisions can have on others, but "I think they need to understand." He further explained his frustration with the challenges posed by a lack of broad understanding of the mailflow: "We tell them not to run [a certain machine], and they don't for a couple of weeks, and then we come back and they're running it anyway. They're afraid they won't finish. This [the model] would show them they don't need the machine." In a visit to our lab several weeks after the workshop, Benny offered the following explanation of the appeal of these tangible models: "What I like about the model is that it's all together, it's not on different floors, and you can touch it, you can feel it. You can see how the handshake between operations is. You can also see what effect one operation has on another operation." After running a final simulation through the model on that visit, he excitedly told us: "Every time I see this I think of new things to try!"

3.1.4 Application-Specific Technology

For the second-generation USPS model, we constructed a toolkit for modeling USPS operations atop the infrastructure described in chapter 4. This consisted of a program for the Tower which maintained a queue of mail to be processed and handled starting and adding machines, as well as recording relevant data about the run to be reported to the laptop after the simulation run is complete. Each Tower represented a single processing operation, displaying data on the operation's status and maintaining a queue of mail waiting to be processed. Wires connecting Towers together allowed them to send mail to downstream operations.

In order to set parameters on each Tower, we created a simple menu interface which used a dial to set parameters values and a pushbutton switch to scroll through a list of parameters to modify. The final set of parameters that could be modified in this model included:

- Operational throughput (pieces of mail processed per hour)

- Start volume (the amount of mail required in an operation's input queue before the operation would begin)
- Volume to add a machine (the amount of mail in the queue required to add another operator or machine)
- Number to start (the number of machines or operators to start when an operation starts)
- Number to add (the number of machines or operators to start when "volume to add" is exceeded)
- Maximum number of machines or operators

Data about the simulation run was saved in RAM so that it could be sent through the Tower network when requested by the laptop. Another internal array, this one located in non-volatile flash memory, stored each Tower's default configuration values, as well as configuration data such as the operations to send outgoing mail to and the fraction to send to each operation. This configuration data allows the same program to run on all five Towers, making development much easier, while also allowing technical users to easily change the values of this internal configuration data. In fact, by the end of the workshop, Benny and Greg were able to easily modify these parameters on the mailflow model.

The laptop computer used a simple user interface built in LCSI's MicroWorlds software. [19] The main screen of this interface, shown in Figure 8, allowed the user to start and stop the flow of mail into the model, as well as modifying the mail arrival profile, shown on the left of the screen. Since the Towers in the model measure time by simply advancing their clocks by one half-hour each time that a batch of mail arrives, pausing the flow of mail into the model effectively pauses the model, allowing users to examine its operation. The second screen of the software interface, shown in Figure 9, allows the users to obtain data from the Towers after the simulation run is complete. The final screen in the interface, shown in Figure 10, is used to load configuration data onto each Tower. Since the Towers retain this configuration data even when they are turned off, unlike the data for each run that is lost during a power cycle, this screen is mainly used when initially constructing a model to provide the Towers with configuration information about the specific model being constructed.

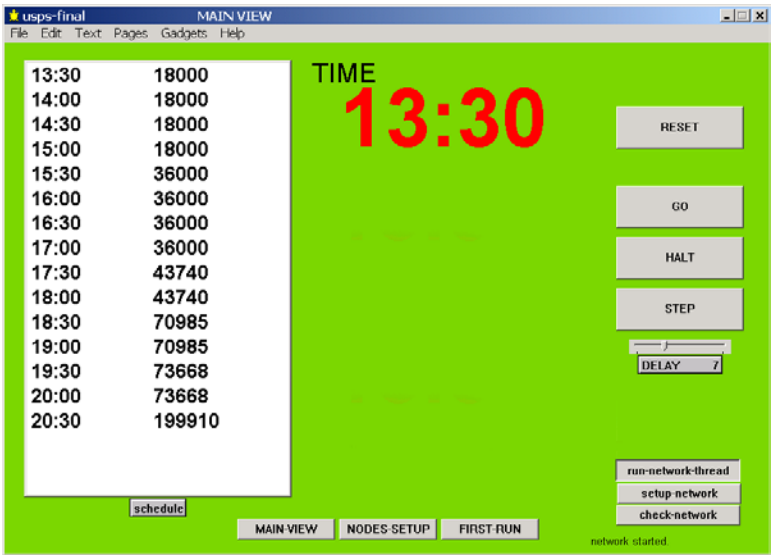

Figure 8: Control screen of USPS FIM model software interface

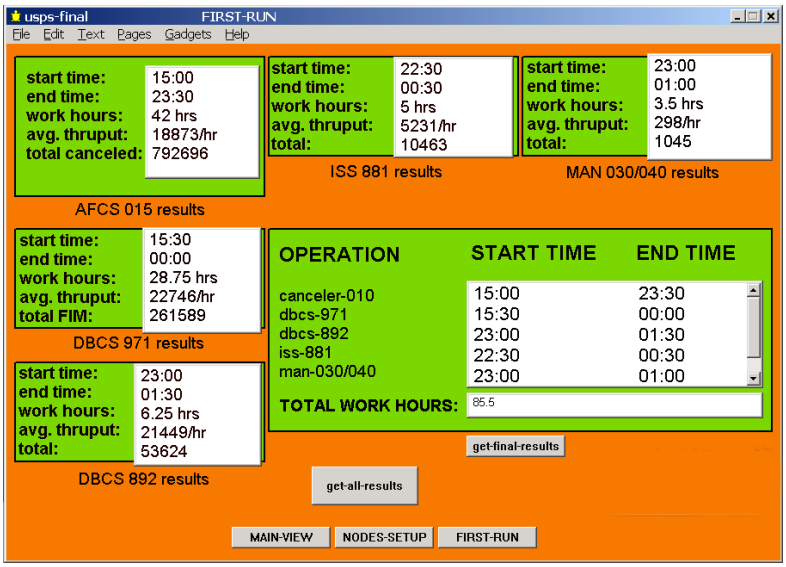

Figure 9: Data collection and analysis screen of USPS FIM model software interface

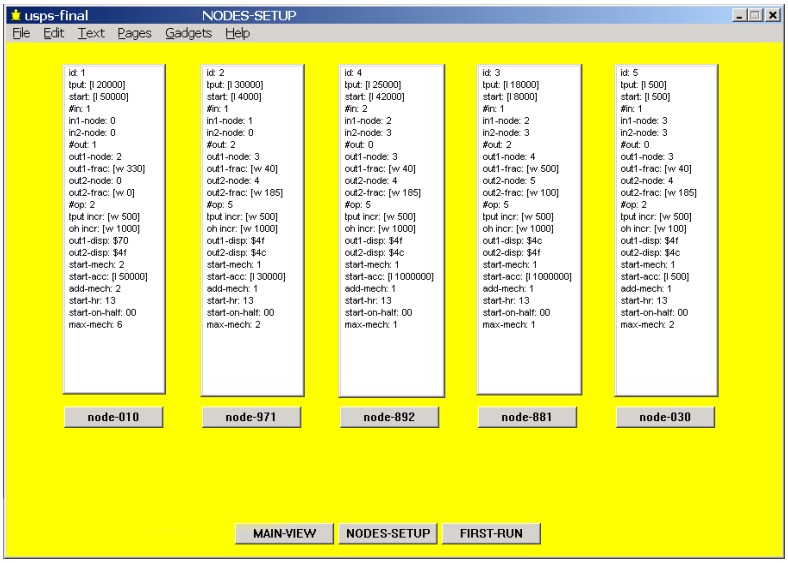

Figure 10: Configuration data screen of USPS FIM model software interface

3.1.5 Further Development

The next steps in our work with the Postal Service will involve expanding our work with individuals in the Boston facility to involve shift supervisors who manage pieces of the mailflow process that we've modeled with the USPS engineers. As described by Benny Penta, the supervisors could greatly benefit from using the model as a medium for reflecting about their individual work and its relation to other parts of the process. This could also involve other stakeholders in the Postal Service, such as union representatives. In order to provide a model of sufficient completeness to be useful for these supervisors, Benny suggested that the current model be expanded to include several other operations in order to completely model the evening shift's mailflow. This would involve adding several more Towers and changing the data-collection portion of the laptop's software to pool data from all of the Towers.

There is also a great deal of work to be done in bringing this type of modeling into the practices of the Postal Service in a deep and sustainable way, taking guidance from the framework for the sustainable use of reflective modeling presented in chapter 6. Several members of the plant management staff who attended the presentation and discussion on the third day of the workshop indicated that they thought it would be valuable to have such models on their desks or the shift supervisors' desks in order to examine the

previous night's work or explore potential scenarios for the coming night. This indicates potential both for widespread use and impact, but also the interest and willingness of audiences within the Postal Service who are not traditionally involved in systems modeling to become active investigators.

The Postal Service managers at the concluding presentation were very enthusiastic about the possibility of integrating real-time status information into the model. Though the technology is not yet present on the mail-processing machines to do so, this would present an interesting opportunity to integrate status information of a complex real-time system into the model to investigate potential choices' effects precisely when these insights have the potential to have the greatest impact—in the middle of a shift at the facility when a supervisor or manager needs to make a decision about how to run the evening's operations. This would, of course, require changes in the model's operation to be able to take advantage of real-time data, but once the technology in the processing machines and the model is available, this could enable a new range of activities in real-time decision-making supported by transparent computer models.

The use of these models in the Postal Service would also greatly benefit from the work described in section 4.4 to further develop the modeling infrastructure so that USPS engineers, or perhaps just a few system designers, can continue the development of USPS-specific toolkits for use in analyzing a variety of USPS systems in a variety of contexts, from training to real-time decision making. More work would also be needed to make such toolkits easier to reconfigure than the current toolkit, preferably without requiring the step of entering configuration data on a laptop as the current model requires. More detail on how such toolkits could be designed and supported in a sustainable way inside the Postal Service or other similar organizations can be found in section 6.1.

3.2 Computer Network Models

3.2.1 Background

According to the UCLA Center for Communication's recent study of Internet use, over 70% of Americans went online in 2002, with almost 60% of these users having access at

home and rising from previous years to an average of over 11 hours per week spent online. The study also found that almost 74% of students use the Internet at school, and 50% of those employed use the Internet at work. It also found that with 97% of youth from twelve to eighteen years old went online in 2002. [37] There is no question that the Internet now plays a role in the lives of most Americans, and the high fraction of those using the Internet at school and work suggests that many also use resources on local networks such as networked printers and file servers. However, the only window that these end-users have to the computer networks which they use so regularly are the software applications residing on their computers. The network itself has been designed to be invisible to users as long as it works, much like the complicated global telephone network.

Efforts exist to enhance users' understanding of the workings of the Internet, such as howstuffworks.com's extensive set of articles on the workings of the Internet, including explanations of computer network infrastructure and various types of online applications and services. [14] However, despite the comprehensiveness of these articles, there is great potential to provide engaging, interactive means for novices to build and use their own computer networks in order to provide an exciting and accessible environment for reflecting on their workings. There has been a great deal of work in inexpensively simulating computer networks for engineering education, beginning as early as 1981 [22] and continuing to the present, [1] as well as a recent effort to bring such models into a tabletop space by projecting traditional representation and providing tangible means of interacting with them. [27] Yet despite these models' time-tested utility as engineers design networks, the representations of aggregate traffic used in these models are far removed from the experiences of individual users, and in order to introduce Internet users to the workings of these networks, the model must be directly connected to their everyday experiences.

This broader understanding could galvanize efforts to ease the creation of web services and network applications by novices. Our work in developing the Rabbit Tower Foundation, described in sections 9.1 and 9.2, provides means for novices to easily create web servers which interface to real-world sensors and devices. A broader understanding

of the workings of the Internet and network applications could substantially broaden the range of individuals interested in undertaking such projects as well as efforts to build infrastructures allowing novices to construct these applications.

We have created a toolkit based on the infrastructure described in chapter 4 to allow novices to build a computer network out of a set of small routers, computers, and servers, all shown in Figure 11. We have also written a simple chat system to run on this network. We also added the Tower system's tricolor LED's, [20] which are capable of generating millions of colors, and extended the chat system to allow users to send colors to each other's computers. The design and implementation of the computer network toolkit will be described in more detail in section 3.2.3.

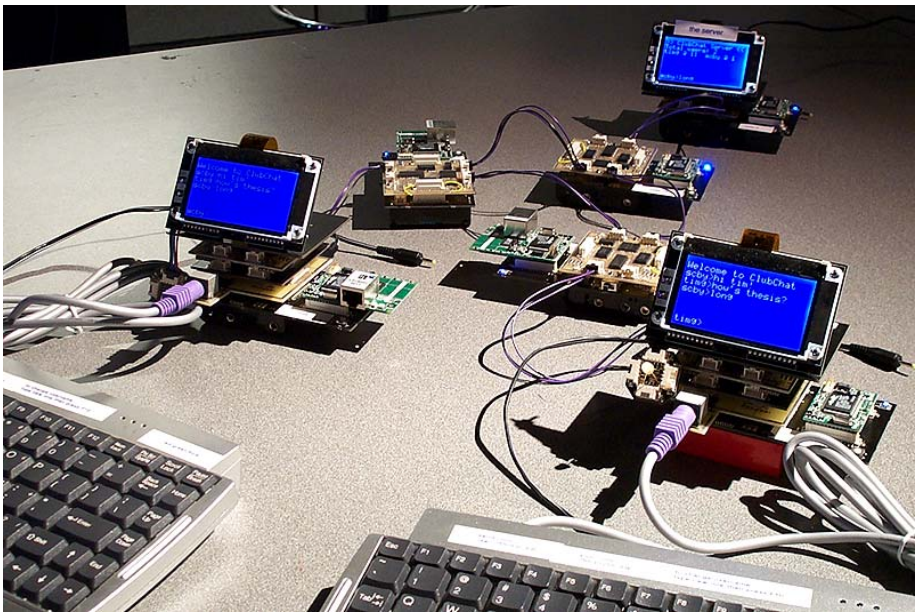

Figure 11: Computer network model

To evaluate this model, we set up a series of workshops in the Computer Clubhouse in the Boys and Girls Club of Chelsea, Massachusetts. The Computer Clubhouse Network [3] consists of a number of after-school centers around the world that encourage disadvantaged youth to be creative with technology. No classes are taught at these centers, but instead young people at the Clubhouse (called "members") are provided with resources including computers, a music studio, and professional design, animation, web site creation, and music and video editing software. Working with the Clubhouse

members allowed us to evaluate our toolkit with a group of young people who were sophisticated in their use computers and computer networks. The Clubhouse's informal environment also allowed us to work with a group of young people who were free to choose how to spend their time in the Clubhouse and to provide honest feedback.

The work of Amon Millner of Georgia Tech's Electronic Learning Communities Lab provided some context for our work at the Clubhouse. [23] Millner interviewed Computer Clubhouse members at the Atlanta, Georgia Computer Clubhouse in order to investigate their conceptions and experience of the Internet. The members involved in his study ranged from 9 to 14 years old, and they displayed a remarkable degree of sophistication in describing their experience of the Internet as end users, with many including descriptions of web sites, search engines, uploading, downloading, and using HTML code to build web sites. It was also clear that these youth found value in their use of the Internet, including comments such as "some stuff you can only find on the Internet" and "I like it when I find something I've never seen before like some new songs." Also interesting was the value that some children placed on the Internet as a place to meet others, as one girl said that the Internet is "a place where you can meet people from all over the world, you can interact with them." However, the young people that Millner interviewed were much less confident when describing the Internet itself; in fact, drawings that Millner asked them to make of the Internet are particularly telling about their ideas of the system that they seem to use so often. None of the drawings included more than two computers, and most were simply drawn as if the observer were sitting in front of a computer. The globe was a prominent feature in four of the seven drawings, in two of which it simply appeared in the middle of the computer screen. Two other images showed only a computer screen with a web browser. This echoes one girl's description of the Internet as residing inside the computer in front of her: "It exists in your computer…. The Internet is in all computers as long as you have Internet access. As long as you have a telephone you got the Internet." Some comments by the children interviewed reveal a sense that the Internet itself is beyond their understanding or explanation, including one member's drawing of the Internet, which simply said, "so many things I can not explain it!" [23]

3.2.2 Results of Workshops in May, 2003

We organized a series of three workshops in May, 2003 at the Chelsea Computer Clubhouse, shown in Figure 12, to evaluate our computer network toolkit. Overall the workshops included approximately 20 Clubhouse members, all aged 10-12, except for one 13-year-old and one 9-year-old. The workshops were very free-form, with the computer network toolkit elements on a table in the center of the clubhouse and members joining and leaving the group as they moved between activities in the Clubhouse as they wished. Each workshop was approximately three hours long, and individuals spent between twenty minutes and two hours using, manipulating, or discussing the model. I was the sole facilitator of the workshops, and this section is based on notes I took during these sessions.

Figure 12: The Chelsea, MA Computer Clubhouse

I began the first workshop with a discussion among the ten members who attended the session about their use of the Internet and the computers in the Clubhouse. The members described making and visiting web sites, playing online games, and obtaining music. I asked whether anyone had used a chat program, and a number of members said they used the AOL Instant Messenger™ chat program. I asked the members how AIM worked, and they said people could talk to each other "through the Internet wires" "like email." One boy said that "I have to be connected and so do you or we can't chat." An 11-year-old boy brought up the idea of a "server," which he described as a "big computer." He said that, when using AIM, "the server connects us." Others quickly agreed. When I asked for clarification on their description, asking, "I talk to the server and you talk to the server?"

they were quick to say no, repeating that the server "connects us." While overall this discussion seemed in line with Millner's work, these young people's idea of a server seemed more sophisticated than that of the children in Millner's study, and the members at the Chelsea Computer Clubhouse focused on the idea of being "connected" rather than the Internet existing inside the desktop computer. However, because these questions were discussed as a group before the start of a collaborative activity, it was difficult to isolate individual members' ideas. Interviewed individually, they might espouse ideas similar to those described in Millner's work. This issue is discussed further in section 3.2.4.

Figure 13: Computer Clubhouse members using the computer network model

After this discussion, I told the members that they were free to use the chat system on the table in front of them. They quickly discovered that the system didn't work, since I had not connected any of the network elements together. As the participants began to arrange the routers, four computers, and a server on the table in the middle of the Clubhouse, as shown in Figure 13, they continued to focus on the notion of connectivity. They quickly tried several arrangements, including connecting two computers directly together, making a chain of computers, and connecting two chains of two computers each to the server. This last approach, shown in Figure 14, allowed them to start to use the chat system, and

they were quick to articulate that the server had to be connected in order for the chat to work. However, they were hampered by the occasional failures of a computer when its power cord was slightly touched, and they said that a far computer would no longer be connected when the computer connected to the server shut off. I asked them if this was also true when a computer in the Clubhouse was turned off, and they said that other computers would still be able to access the Internet. This prompted them to ask what else they could do to arrange the computers. I suggested the use of a router, and they created the arrangement shown in Figure 15.

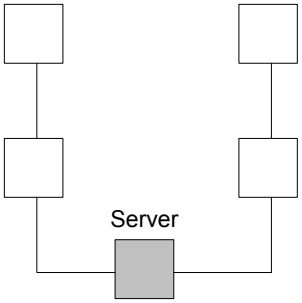

Figure 14: Early arrangement of computer network model elements

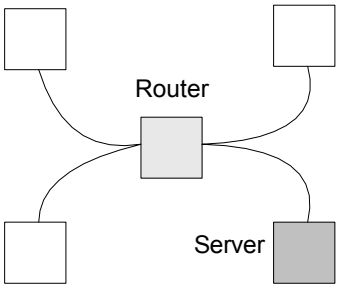

Figure 15: Later arrangement of network model elements

This rearrangement of the network proved much more reliable, and the members spent some time simply chatting back and forth to each other. After a while I decided to probe their understanding of the operation of the chat system's behavior—specifically, whether the network provided ways for them to observe that when they pressed enter on one of the computers, the message first went to the server and then was sent to each of the computers. The members were still using the network in Figure 15, and they explained that the message went from their computer to the router and then simultaneously to the server and other computers. We watched a single message travel while no one else was typing, and their conclusion was the same. I then added another router between the central router and the server, as shown in Figure 16. We again watched a single packet travel, and this time the members were able to see it traverse the new router towards the server, and then back towards the other computers. They concluded that the packet first went to the server and then to the other computers. The boy who had introduced the idea of a server in the initial discussion quickly said, "See? I was right! It goes through the server."

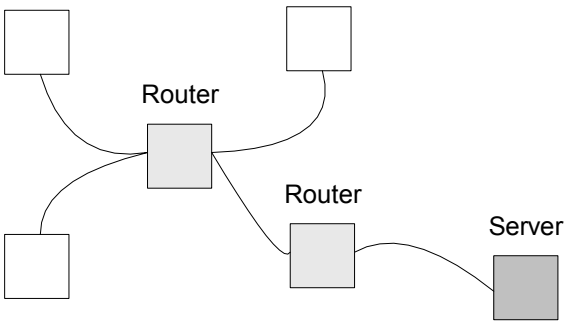

Figure 16: Modification of the arrangement in Figure 15

This arrangement also provided a means to test another interesting aspect of the network's behavior: the fact that the chat system will fail when the computers are connected together but the server is no longer connected. I turned off the router attached to the server, and the chat system stopped working. I asked the members why this happened, pointing out that their computers and the central router still seemed to be working. They were briefly confused but quickly explained that their computers were "not connected to the server," and then they promptly turned the router back on so that they could continue chatting.

After some time more, I introduced the members to the ability to send colors between nodes by entering a series of four numbers (`node-number/red/green/blue`). It was clear that the formulation of a color as three numbers was difficult for the members to use, as was the upper limit of 255 on any of the components of the color. However, the members did have a fun time sending colors to each other's computers. Their comments regarding these colors were particularly interesting during the second workshop, as will be discussed later in this section. While the members were busily sending colors back and forth, I quietly disconnected the server. Because they were sending colors based on the node numbers printed on the computers, this continued to work and they did not notice my action. After some time, I told them what I had done, holding the disconnected and powered-off server in front of them. They were very surprised, and I asked them why sending colors was still working. The same boy who had brought up the idea of a server in the initial discussion described the colors moving around as a different type of message, saying that the "color is just in the computer" while the words had to go through the server. Despite my prompting, no other explanations were forthcoming from

the other members using the network, and they were reluctant to even comment on the boy's explanation. They seemed relieved when I reconnected the server, and they returned to chatting and sending colors.

During the second workshop, I also demonstrated how to send a color by replacing the node number with someone's username. By turning off the server again, I helped the members discover that using the node number works without the server but that using the usernames does not. An 11-year-old girl explained that this occurred because "the names are in the server." She may have been helped to this conclusion by the fact that the server displays the usernames and node numbers of current users, but she did not take the next step to recognizing that the server maintained (and displayed) a crucial mapping from the username to the node number.

For most of the second and third workshops, I took a much less active role, simply letting the members build and use the model as they wished. This allowed me to make a number of interesting observations. During the second two workshops, at least one member in each workshop had been in a previous workshop, and they quickly moved to connect the computers and routers to set up the network and begin chatting. The members quickly showed each other how to change usernames, send colors to each other, and identify and restart troublesome routers and computers. As they were sending colors to each other, some interesting affective aspects of sending these colors became apparent. Members liked getting colors from other users and would often call out, "somebody send me a color!" Yet sending a color could also be an act of malice, as one girl said, "I'm going to make [another girls' name] mad" by sending her a particular color. One boy, after receiving a similarly offensive color, said to the girl who sent it, "I'm gonna get you back for that!" I also observed an emerging subversive quality to the members' interactions through the system, as the members used this new form of expression to write gossip and things like "I C U P" to each other, out of the view of the Clubhouse manager.

During these last two workshops I did not actively shut off routers and the server as I had during the first workshop, but instead took advantage of the times when these would fail on their own, either due to declining battery power or sudden, excessive packet loss.

During the second workshop, for example, several members were using the chat system when the router connected to the server failed. One of the members, a 13-year-old boy, quickly identified the offending node and reached to restart it when I asked him, "How could you change this arrangement so it doesn't matter if one of the nodes failed?" He began reaching for wires but kept stopping before attaching them, as if realizing that what he'd been planning wouldn't solve the problem. While he thought about this for a couple of minutes, another boy had connected two wires between his computer and the nearest router and said, "Look! I have two connections." At this, the first boy exclaimed, "Everyone should have two connections!" He then proceeded to ensure that every computer and the server had two connections to different routers, and that routers were connected to at least two others. This resulted in a jumble of wires, but he verified that he could shut off the previously offending router without harming the chat system. After doing so, he looked up at me and challenged me to test his design, saying, "okay, now disconnect one!" I did, and it continued to work as the boy grinned proudly. This way of engaging the members in the design of the network seemed less of a disruption to the members' chatting activity since it took place when the network was already misbehaving and had a direct relation to the members' goal of restoring a functional network and preventing future failures.

Throughout the workshop it was interesting to observe the members accumulating a vocabulary for talking about the objects in the model. The members quickly took up the word "server", as it was printed on a label on the miniature server and came up in their initial discussion. My description of the Towers with keyboards as little computers also stuck, as did their early focus on keeping the machines "connected." Even my offhand remark that a computer was "sad" as I restarted it after its program had crashed stuck in the members' heads, and as they quickly learned to restart crashed computers themselves, they would anxiously say, "Look, this one is sad!" None of these terms were printed on the models, except for the word "server." In fact, at the beginning of the third workshop I removed the "server" label as I was debugging another Tower display, and the members continued to refer to the server by that name throughout the 3-hour workshop.

The best way to capture the members' understanding of the model was through asking them to explain it to parents, staff members, and other kids who wandered by and asked what they were doing. Most of the participating members were quick to explain the way they were able to chat with each other and send colors back and forth, though a few were hesitant to explain anything even after using the network for some time. The members' description of the relationship between the model and the computers in the Clubhouse around them was simple and direct; as one boy explained it, "You know how real computers are all connected to each other. This is the same, only smaller."

One important aspect of the members' engagement of the model that has not yet been made explicit thus far is the deep engagement and excitement the members displayed for simply using the functioning network to chat together and send colors back and forth. While designing the system, it was not at all apparent to us that this would be the case; after all, the members were chatting on computer screens that are at most about two feet apart, and they were surrounded by far more powerful computers in the Clubhouse. Yet the members would often fight for control of a keyboard to type or send colors to others. Both the highly interactive nature of the chat system and the previously mentioned subversive possibilities of self-expression may have been factors, as well as the small, manageable scale of the model and the simple novelty of having a new system in their space to work with. But though the cause of this engagement was unclear, it was certainly apparent that this engagement was crucial in creating the opportunities for the experiences described in this section.

3.2.3 Application-Specific Technology

To create the computer network model atop the infrastructure described in chapter 4, we needed to create the three types of nodes in the model network: routers, computers, and the server. Since they simply needed to forward packets that contained chat messages and color information, routers required almost no modification upon the basic infrastructure. A router, shown in Figure 17, consisted of a Rabbit Tower Foundation and a serial layer, and the only addition to the packet parsing and routing code was to save the node's

address in flash memory. The computers and server were somewhat more complicated in both hardware and software.

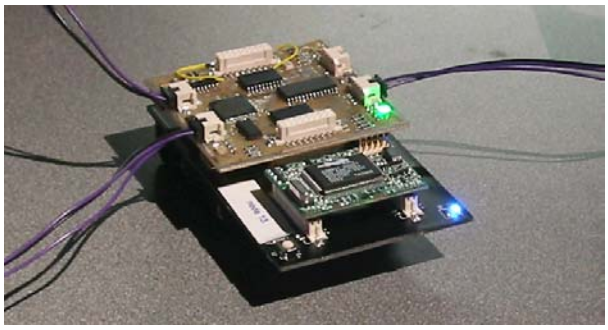

Figure 17: Computer network model router

In order to allow the computers and server to communicate, we built a chat protocol on top of the packet parsing and routing software, which is described in section 9.5. This protocol allows the messages listed in Table 2 to be transmitted between nodes. Each message is represented by a different packet opcode and contains the data listed in Table 2. The packet colors are used to make it easier to identify different types of data flowing through the network in this system. The protocol supports sending chat text between the server and computers, computers joining, querying the server to find a user's node given his username, and sending colors between computers. Sending text during a chat session has a simple pattern: a user types a line and presses enter. His computer sends it to the server, which sends a "chat string reply" packet to all the users currently in its list. The color packet is even simpler: a single packet containing the color is sent from the originating computer to the target computer. However, the sequence for changing a username, shown in Figure 18, is more complex, in order to ensure that a user does not change their username to a name that another user is already using. In order to prevent this, the user's computer first attempts to find the node number of the new username before sending a "join" packet. However, nothing in the server's code prevents duplicate usernames; in fact, this happens initially as every node starts with "????" as the default username.

Packet type	Packet color	Data payload	Use
Chat string	Purple	Text of the line entered by a user	Sent from a computer to the server
Chat string reply	Purple	Text of the line entered by a user	Sent from the server to users' computers
Join	Green	Username	Informs the server of a username to node mapping
Node query	Green	Username	Asks the server for the node of a given user
Node query reply	Green	Username and node id	The reply from the server to a node query. If there is no such user logged in, the node id is 255.
Color	Blue	Red, green, and blue color components	Sent between computers (not the server) to change another computer's LED color

Table 2: Messages used in chat model

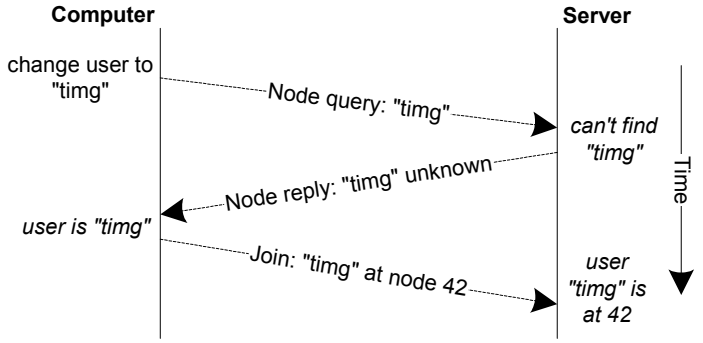

Figure 18: Communication sequence during change of username

The server, shown in Figure 19, adds a display layer, developed by Chris Lyon [20], for the Tower in order to provide status information about the currently logged-in users, as well as the last line broadcast through the chat system. The code for the server maintains a pair of arrays that maps usernames to node id numbers. The communication system for the server interfaces with the routing software in order to provide the chat protocol functionality described above. If the server does not receive a periodic "join" message from a node in its list after a certain period of time, that node is removed from the list.

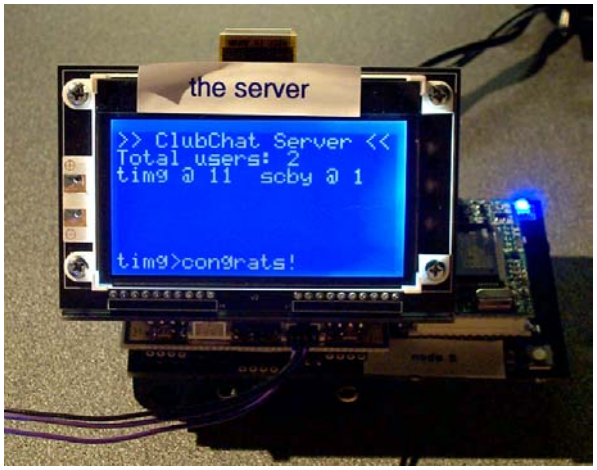

Figure 19: Computer network model server

The model computers, shown in Figure 20, included the serial layer and display layer, as well as a PS/2 keyboard layer, a tricolor LED module, and the I2C layer needed to connect the tricolor LED. The PS/2 layer, tricolor LED, and I2C layer were all designed and implemented by Chris Lyon. [20] The computer is controlled through the keyboard and connected to the network through the serial layer. Using the keyboard, the user can type a line of text and press enter to send the line to the server. The user can also enter a four-character username and press F10 to change to that username via the protocol shown in Figure 18. The user may also enter a three numbers to enter a color (`red`/`green`/`blue`) and press F9 to set the user's own LED to that color, or enter a node number or username before the color (`node-number`/`red`/`green`/`blue` or `username`/`red`/`green`/`blue`) and press F8 to send the color to another computer.

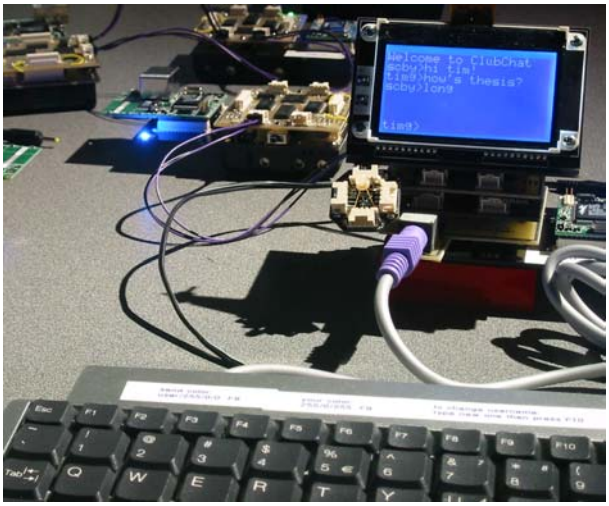

Figure 20: Computer network model computer

3.2.4 Further Development

There are a number of improvements to the computer network model that would facilitate the understanding of its operation and possible uses. In addition, the system would benefit from further thought work on how to make the most effective use of such a system in a formal or informal learning setting.

In its current form, the chat system takes approximately 2-3 seconds between a user pressing enter to send a line of text and it appearing on all the users' screens. This appears very sluggish from a user's perspective, but it remains difficult to see the path of an individual packet while it travels through the network. The current speed of the model is both too slow to use comfortably and too fast for analysis of a single packet; the solution seems to be improving the routing infrastructure as described in section 9.5.3 and providing means for users to set the model's speed depending on their particular activity or exploration. Two options for implementing this are in a system-wide basis, where every packet's speed would be controlled uniformly, or a speed field could be added to a packet so that the network knows to slow its delivery. However, the network applications would have to be sure to preserve this field during packet exchanges so that, for example, if a user sent a slow packet with a line of text, the packets that the server would send to the other users would also move slowly for analysis.

Though the users' direct interaction with the network does provide relatively easy means for users to examine the effect of individual actions and node failures on network performance, it remains difficult to understand longer-term effects of different network arrangements. Some means of visualizing aggregate traffic would also be useful in order to help users understand the long-term traffic patterns across network arrangements. It may also be useful to provide a means to create artificial, automatic traffic flows between nodes in order to see different steady-state behavior.

There are also some possible user interface improvements to the chat system that has currently been written for the computer network model. One particular challenge for the children in our case study with the chat system was entering colors in the three-number form required by the current system. Three dials for the different colors or some other, more intuitive way of choosing colors could replace this interface. The screen could also be enlarged, perhaps by using a PDA as an interface, though it seems possible that the members were more willing to work with and debug issues with a set of objects with no relation to traditional computer graphical user interfaces.

In addition to simply improving the model, there is a great deal of work to be done in expanding methodologies for use of this type of model in both formal and informal learning settings. Individual interviews before and after group workshops would provide a much more comprehensive view of the changes in participants' ideas of computer networks. In our case study, I briefly tried two approaches as a facilitator, one involving a very active role and another taking advantage of situations when they occur. This may be difficult to replicate in either a classroom or an informal setting without a high facilitator-to-participant ratio, but it can provide insight into the types of questions that students should be asked to explore or how to engineer the kind of "teachable moments" that teachers and facilitators can take advantage of. Though in our case studies these commonly occurred as nodes failed due to battery failure or other glitches, carefully creating rules which generate such failures intentionally could provide a good opportunity to guide participants' explorations toward ways of working around such failures without external intervention. It may also be useful to create entirely new applications other than a chat system for different contexts. Other network applications

that could be engaging for the model network include web servers displaying sensor data, shared whiteboard applications, other forms of actuators other than tricolor LED's, and peer-to-peer file sharing.

4 Technical Infrastructure

4.1 Technical Requirements for Modeling Infrastructure

Our early work using the Cricket system [21] to build a prototype model with the Postal Service illuminated a variety of important design considerations for a infrastructure which could be used to speed the development of such models. These requirements included the following:

- **Computation**
  - Rapid development process allowed by an embedded, interactive virtual machine (VM)
  - Substantially larger RAM, in the thousands of bytes
  - Similarly sized non-volatile memory, also in the thousands of bytes
  - Faster VM execution speed (at least 1000 Logo instructions/sec) and support for preemptive multitasking
- **Communication**
  - Robust, packet-based wired communication infrastructure, ideally including multi-hop routing between model elements and an attached PC
- **Input and Output**
  - Larger, backlit LCD display
  - Multiple sensor ports for modifying parameters
  - Ability to utilize simple visual cues like a tricolor LED to indicate node status at a glance

4.2 Infrastructure Design and Implementation

A natural direction for us to consider was the Tower system developed by our research group, [20] which was approaching stability in late 2001 as we evaluated our initial prototype with the Postal Service. This platform provided a number of already-designed modules as well as a robust structure for adding new communication functionality that would be required. However, the Microchip PIC-based Foundation being developed at the time would not have the required RAM or non-volatile storage capabilities required. We also initially hoped to avoid creating a new layer for inter-Tower communication, which would certainly not be possible using the PIC Foundation. We were unable to find a system of comparable flexibility and power, and so decided to extend the Tower system

by building a new Foundation based on a more powerful processor in order to support multitasking and string manipulation, both important for the communication support we required. Based on the requirements of this project and our independent research interests in supporting Ethernet connectivity to Tower systems, we chose the pin-compatible Rabbit Semiconductor™ RCM2200 and RCM2300 modules [29] for the new Foundation, built a circuit board for it and ported a Logo virtual machine into the Rabbit's C language. The design of the Rabbit Foundation hardware is described in detail in section 9.1, and the Rabbit Logo virtual machine is described in section 9.2. During the course of this project, the desktop application for programming the Rabbit Foundation grew into an integrated environment for programming our various Tower Foundations, an assembler for the PIC microcontrollers and a programmable data graphing toolkit. This expansion occurred with the direct involvement of Brian Silverman, Chris Lyon, and Lele Yu, and the result is described in section 9.3.

We then wrote several layers of Logo code to run on the Foundation to support packet-based communications using the Rabbit's serial ports. The first layer supports sending and receiving packets (including a one-byte checksum to help guard against malformed packets) on the serial ports and passing them to the next layer. The second layer we wrote provides support for providing each Tower with a node id number and routing packets between nodes as intermediate nodes read the destination address and forward them towards the destination node. This layer also includes a simple ah hoc routing protocol so that the nodes can determine how to route packets to each other automatically. The packet parsing and routing algorithms and their implementation are described in section 9.5. This layered approach is familiar in the development of network protocols for larger computers networks, [16] and in the context of a network of embedded microcontrollers, it provided the ability to test and debug each software layer independently.

Another important component of the design of this Tower network was the ability to place a laptop computer as a node in the system in order to communicate with the Towers, providing a means for a laptop to aggregate data, send configuration data to a Tower, or provide the input to a model. After the packet and routing layers were completed in our RabbitLogo, we ported these layers to the Logo variant used by LCSI's

MicroWorlds software. [19] This version of the code used one of the PC's serial ports to connect with a Tower, allowing the PC to send and receive routed packets using the same protocol as the Towers. The universality of this means of communication between elements in the models has proved to be important in managing the complexity of internode communications in these models.

It became apparent after working with these libraries using the Rabbit Foundation's three serial ports that the Tower would need to support a larger number of buffered serial ports on each unit. Since no layer was available for the Tower system that supported multiple serial ports, we designed and produced a layer for our use with four independent buffered serial ports. In the most recent revision of this layer we added a tricolor LED under the onboard microcontroller's control at each corner of the board in order to color-code data flows through the model. The design and implementation of this layer are discussed in detail in section 9.4. The Tower architecture allows us to easily add multiple serial layers, and the packet communications software layer was modified to use these layers, including support for the color-coding packets, instead using the three serial ports that were available directly from the Rabbit modules.

One other common ground we identified between modeling applications was the need to display and modify simulation parameters on each node. We developed a simple menu interface for the Tower's LCD display layer. The menu allows the user to change a parameter's value by turning a small dial. This dial functions similar to a car steering wheel in that if you simply turn it in one direction and hold it still, the value will continue to change until the dial is returned to the center position. The user scrolls through the list of available parameters by pressing a pushbutton switch. This allows users to easily view a simulation's current parameters as well as changing them as appropriate.

4.3 Methodology for Building Domain-Specific Toolkits

This infrastructure provides the technical groundwork for building distributed simulation models. By programming in Rabbit Logo, a programmer can construct a model by passing data between nodes in packets and communicating with a PC, if appropriate, for later data analysis. However, evaluating the ability of relative novices to construct such

models remains beyond the scope of this thesis. A great deal of work remains to be done in building structures to make it easy for novices to build such toolkits, as will be discussed in section 4.4.

Rather than building a single model for a specific application, we found it especially powerful to develop domain-specific toolkits, allowing a wide range of possible models in each domain. In the first of the two case studies described in chapter 3, we created a toolkit of Towers that modeled USPS operations and configured and connected them in order to model the specific mailflow in use at the USPS processing facility in Boston. In the second case study, we created a toolkit of network routers and end-user computers that were then connected by Clubhouse members into networks of their own design. Two other current projects described in section 6.1 have involved building domain-specific toolkits for engineers working on computer network hardware [5] and for use by children as an introduction to system dynamics. [39]

As such, we may envision several groups of users and designers of tangible models in an organization engaged in reflective systems modeling. First, system designers would build toolkits in the style of our two case studies' toolkits described in the previous paragraph. These system designers may or may not be experts in the field of the object of study; our work involved both scenarios, as discussed in section 5.3. There would also be users of the system who would take these toolkits and construct models relevant to their particular challenges. Of course, particular care must be taken by system designers to support a wide range of flexibility for these users, who are likely to benefit a great deal from constructing their own models relevant to their particular work or interests. Finally, other individuals in an organization might interact with completed models, modifying parameters and exploring their effects in order to increase their understanding of the system under study. These roles are expanded into a framework for the sustainable use of reflective systems modeling in chapter 6.

4.4 Future Development of Modeling Infrastructure

Our work has illustrated the potential to design an infrastructure to accelerate the design of application-specific toolkits for systems modeling. However, in its current form,

designing toolkits based on this infrastructure still requires a fair amount of programming skill and understanding of the issues of programming a distributed system. It seems possible to enable much less technical end-users to build such application-specific toolkits by leveraging commonalities in their design to create a software design environment in which users could specify parameters and their default values without needing the write code to manage them. It also seems likely that the core logic of the process under study would be representing in code, though perhaps other representations are possible. Providing a simpler framework to build and modify these application-specific toolkits has clear possibilities for expanding the reach and potential impact of reflective systems modeling in organizations by allowing a broad range of individuals much greater control over their tools and means of communication around such models. Certainly much work remains at both the conceptual and technical level in examining these commonalities and designing and evaluating other representations and frameworks for constructing such toolkits.

5 Reflections

5.1 System Design

Our approach to building a infrastructure for domain-specific modeling toolkits is unusual and worth discussion. While the power of abstraction and modularity in computer science has produced a wide variety of utilities, libraries, and other forms of software infrastructure, building an infrastructure to enable a particular range of applications entails a balance between building appropriate structures to facilitate construction and providing appropriate flexibility to grant wide latitude to the designers and implementers of the application-specific toolkits.

This creates an interesting role for the designer of an infrastructure for building such toolkits, as well as the designers of the application-specific toolkits. The infrastructure designer must understand the scope of problems to be solved by the toolkit designers, making the structures necessary for these designers to rapidly prototype and iterate their toolkits, exploring a rich space to determine how to best facilitate the intended type of systems modeling. However, the infrastructure itself is not value-free; it necessarily provides a structure supporting a particular style of interactions with systems models. This extends to analysis and visualization tools used with these models. The principles which have guided the development of this infrastructure—and which we hope will be apparent to end users of the toolkits built atop the infrastructure—have been stated before in this thesis (see section 2.2) but are worth reviewing here. The toolkits should provide *simple, transparent,* and *collaborative construction, interaction,* and *analysis* of models of dynamic systems. The presence of a common infrastructure creates the opportunity for the transparency of one model to extend across multiple toolkits and projects by providing familiar means of interacting with these models. The remainder of this section discusses how the current infrastructure attempts to fulfill these goals and its strengths and weaknesses in doing so.

The hardware and software components described in chapter 4 allow a toolkit designer to easily create a network of nodes and send data between them. This focuses the designer

on the messages that are sent and received by each node. Thus far, messages have been used to simulate discrete events (in the USPS case study described in section 3.1), communication between elements (in the computer network study described in section 3.2), and updating variables in a more formal system dynamics simulation. [39] It seems clear that these messages represent a general communication medium with much more useful features than sending single bytes over the wires, but it is worth noting that other choices are possible for a communication infrastructure, such as a mechanism for transparently sharing variables across a communication channel. Message-passing does appear to be among the most basic means of communication which suits the range of projects which I intended to support, but there may well be a place for other, more specialized options in the future. One example would be explicit support for shared variables that could be accessed and modified by other adjacent simulation objects. This might, for example, allow one processing operation in a USPS simulation to directly increase the size of another operation's queue. This might also better support other types of systems, such as the classic predator-prey model, which would otherwise require the awkward step of sending a message to indicate how many predators died overnight.

The infrastructure also provides means for displaying data on Towers using LCD screens or simpler representations such as a single tricolor LED capable of displaying millions of colors. Numerous forms of input, including buttons, dials, sliders, and even PC keyboards are also supported in order to provide a variety of ways to interact with simulations. Since viewing and modifying parameters is a common need, the infrastructure also includes a simple menu system for modifying these parameters using only a dial and a switch. These options provide considerable flexibility for a toolkit designer to create user interactions that support a particular type of model, while also supporting common functionality. The Tower platform provides modularity in the hardware platform, enabling a wide range of possible configurations. One particularly clear example of the advantages of this hardware flexibility is the three types of elements in the computer network model described in section 3.2.3: computers, the server, and a number of network routers.

The use of the Tower system removes the necessity for the toolkit implementer to be familiar with hardware development; this is no small achievement, and is discussed

deeply in Chris Lyon's Master's thesis. [20] However, the current iteration of the architecture requires that the system implementer must be somewhat familiar with computer science and programming concepts, more so than the average case that Lyon discusses because of the complexity of the functionality required of each node, even given the substantial software infrastructure provided for the implementer. Though the Logo language used to program the Foundation is designed to be readily accessible even to novices, [25] the nature of the code required for these models can quickly run into difficult ground. For example, the packet dispatch routine usually runs in a separate Logo thread from the main process running on the node, and this can easily lead to race conditions when different threads try to modify values of the same shared variable unless the programmer is careful to avoid these. Such bugs are notoriously difficult to isolate in any programming language; Mitchel Resnick has done some interesting work in categorizing challenges novices encounter in this area. [30]

Despite some software complexity, the flexibility of the Tower system and the Rabbit Logo language have clearly provided the means for rapid development and iteration of toolkits for systems modeling. During the development of the USPS and computer network models, hardware and software changes were often made and tested in the dynamic system. The one challenge to the speed of this process was the need to load new code onto each unit in order to test a new system-wide change. As described in section 4.4, an integrated environment for creating such simulations could involve the ability to simultaneously load code onto all the devices through the network, further speeding development and debugging cycles.

5.2 Case Studies: The Effect of Systems Modeling

The case studies presented in chapter 3 provide windows into the effect that systems modeling can have for two very different audiences in very different learning settings. In the first, our collaboration with the United States Postal Service, connecting engineers' and managers' understandings of dynamic systems is crucial to handling new delivery products, coping with unexpected situations, and improving efficiency. In working with Computer Clubhouse members to build models of computer networks, young people

explored their ideas of how computer networks work and became more informed users of computer networks and network applications. The remainder of this section discusses how each of these cases embody the goal of supporting reflective systems modeling and what shortcomings the case studies illuminated in each case. As discussed in section 2.2, these case studies are intended to provide examples of very different types of systems and environments for systems modeling in order to provide indicators of the appropriateness of this approach for a wide range of systems and environments.

In our work with the US Postal Service, we were struck by the fact that our initial prototype, described in section 3.1.2, consistently engaged USPS visitors in discussions about their mailflow and their processes for proposing changes to their mailflow and schedules, despite the fact that it was impossible for them to interact directly with the initial prototype. We attribute this success to the direct relationship between the objects in the simulation model to the objects which USPS engineers and managers are accustomed to dealing with in a processing plant. The simulation objects displayed information that the USPS staff was accustomed to obtaining from the real machines, including operational throughputs, start volumes, mail arrival profiles, and the number of machines currently running. This use of common information was only possible because both the initial and later prototypes were built in direct collaboration with USPS engineers with a great deal of experience in this domain. As such, remarkably little explanation was needed for postal engineers and managers when seeing the model for the first time. In fact, rather than asking what was occurring, one member of a group of visitors would often quickly challenge one choice or another made in the schedule which was demonstrated to them—asserting, for example, that if mail did not flow more quickly to the manual operation by 11 PM, the mail would never leave the facility on time. This is a clear success for this model's ability to transparently facilitate conversation about the process under study without being mired in details of the model's operation.

The USPS model developed with the Tower system also allowed users to control its parameters, and this was the step which allowed the USPS engineers in our workshop to take ownership of the tool and make it part of their work and their communication with USPS management during our three-day workshop. We observed the engineers making

the model a useful tool in their analysis of the overload of mail that occurred one of the evenings during the three-day workshop, as well as using it as a communication tool discussing the mail overload during their meeting with the USPS management about the workshop. Though the engineers were not able to modify the simulation model themselves beyond its parameters, as much work remains to be done in making such modifications transparent to users, they were able to articulate changes in its algorithms and left me to make these changes. This observation is particularly promising towards this type of systems modeling becoming an important tool for organizations managing complex processes.

My work with the computer network model in the Chelsea Computer Clubhouse took place in a very different environment with a very different audience. The ten- to twelve-year-olds who participated in these workshops had a great deal of experience using Internet web sites and the server located in the clubhouse, but had little understanding of how computer networks work and no experience modeling them. Though the Clubhouse members' use of the network model had no obvious motivation comparable to the demands of the USPS workplace, there was no question that the Clubhouse members found using the model engaging, though they only stopped to examine its workings when elements failed—much like the way full-scale computer networks are taken for granted when they work. However, failures in the model due to run-down batteries or other glitches provided opportunities for children to reflect on the model and examine other possibilities. Although I had been initially worried that the children's engagement with the model would actually inhibit their ability to engage in the reflective modeling I wished to support, it became clear that the children's motivation to use the model led to a strong willingness to debug and reconfigure the model as they saw fit, providing the kind of engagement with a transparent, easily manipulable model that also proved successful in promoting reflective modeling in the USPS workshops.

However, much like full-scale computer networks, as long as the model was functioning, the children were focused on chatting with each other instead of manipulating the underlying computer network. The key challenge in this domain seems to be finding ways to create opportunities for the children to reflect on the network and try different

arrangements. In fact, it was the occasional failures of nodes that provided an opportunity to examine the causes of failure and consider possible solutions and workarounds. The fact that one boy challenged me to test his new, fault-tolerant model by saying, "okay, now try turning one off," suggests that activities for this age group might be effectively formulated as puzzles or challenges. An example of a challenge for this model might be to design a network that makes it easy to isolate a user who maliciously floods a chat channel with messages, preventing others from being heard. This type of challenge would likely be aided by the tendency of small groups of children to gather around the model and chat with each other. The model was able to engage the children, act as a platform for observing the performance of networks and network applications, and provide a means for the children to construct and evaluate alternative network topologies. However, it is clear that more work is needed in ways to promote children's interest in exploring the network in addition to simply using it to chat. Other changes are also possible to further enhance the model's ability to serve as a platform for reflection on computer networks. These are discussed further in 3.2.4, focusing on allowing the users more control over the timescale of the model's operation and better ways of visualizing the state of the routing algorithm.

One noteworthy component of both case studies was the engagement with models took on a particular tone, as if the participants were not merely interacting with a model but the real system under investigation. This was somewhat expected in the computer network study, since the model was literally a small computer network, though the level of engagement that the children demonstrated is still remarkable, considering the limitations of the model's computers and chat system. However, the degree to which the USPS engineers who worked with the model referred to the model as a real system was extraordinary, considering its abstractness. One engineer repeatedly described using the model as "running mail," eventually prompting another engineer to reply, "remember, it's just a model." During the discussion session with the Boston USPS plant's management team, many of the managers expressed a desire to more closely connect the model with the real processing operations through the use of real-time data. It appears that realizing these models in a tangible form apart from any on-screen representation can lead to a strong association between the physical objects in the model and the real-world objects

being modeled, which can be beneficial for novices to gain trust and engagement in the simulation models.

5.3 Case Studies: The Role of the Practitioner

Researchers and practitioners in both systems modeling and changing work practices [13] have often cautioned about the need to understand how this work takes place in the existing structures of an organization and how these changes may modify these structures. In fact, Senge [34] and Schrage [33] have provided evidence that effective systems modeling efforts can radically change the nature of work practices by illuminating hidden assumptions and guiding individuals, teams, and organizations into more efficient patterns of work. These changes may involve changes in the relationships between individuals, perceived status, and in some cases may lead to the elimination of some individuals' positions. These changes are all but certain to be resisted by those in the organization whose jobs are at risk, as well as those who feel that their apparent status is being eroded. Forces such as these from different constituencies can hamper or even thwart efforts to reexamine work practices, and consultants and individuals within organizations attempting to bring about such change must be skillful at understanding these forces and helping channel them into resolving the conflicting viewpoints. [13]

The case studies presented in this thesis have included elements of these challenges, especially in the case of the US Postal Service. It was clear that the engineers in the workshop began with the idea that they needed to limit the possibilities for others' interactions with the model in order to restrict the conclusions that could be reached by manipulating the model to those that had already been agreed upon by the engineers. It seemed that the engineers were reluctant to have others performing analysis similar to their work, for fear that their conclusions would be challenged, possibly by those with incomplete information. In fact, the extent of manipulation that should be possible by others was a topic of explicit discussion between the engineers, as described in section 3.1.3. But on the second day of the workshop, the engineers began to use the simulation model to explore scenarios as well as articulate changes in the model which would better support their work. In this way, the engineers began to take ownership of the model as a

tool in their work, both using it for their own analyses during our workshop and using it as a tool to illustrate the effects of choices during unusual scenarios to managers during the presentation at the conclusion of the workshop.

This interaction illustrates a valuable lesson for the use of systems models in organizational contexts: the knowledge and direction for the modeling efforts must come directly from those who will benefit from the simulations and their results. These individuals also generally have the most situated knowledge about the system and environments that are under observation, yet their mental models often include misconceptions about the systemic effects of their choices. Though understanding the systemic issues often illuminated by models could have a great impact on these individuals' work, efforts at modeling may be far-removed from them, with no direct relation to the experiences of those most deeply involved in the systems under study. The importance of involving these end users in the design, construction, and manipulation of simulation models was discussed in chapter 2, and this strategy has the potential to radically change the structure of efforts to enact organizational change by changing the sources of knowledge about the organization, expanding it from the domain of consultants and internal specialists to engage individuals at all levels of an organization in modeling and manipulating the processes they work with and within daily. It is unsurprising that these changes may illicit resistance from individuals formerly responsible for such efforts, but expanding others' involvement in the process can both broaden the amount of information available to such specialists as well as vastly increase the potential impact of their work.

These lessons may carry over somewhat into the work of facilitators in formal and informal learning environments engaging novices in systems modeling. The goal of this work in such environments is to allow novices to build their own understandings of dynamic systems by designing and building models of the systems of interest. However, in our work using the computer network model with children, we experienced a familiar problem with exposing children to manipulating systems designed to help them learn a particular set of concepts: the existence of a "right" answer. While using the model to try to determine the behavior of the chat system, on several occasions children stated a

possible explanation and turned to me, asking, “is that right?” Having designed the chat system, I naturally knew its actual operation; I found that it places a facilitator in a strange position to deny eager learners that information and instead ask, “well, why do you think that?” However, this uncomfortable interaction can be reversed by providing means for children and other novices to create network services and build their own toolkits, as described in section 3.2.4, providing authentic opportunities for children to create their own systems and examine each other’s systems. Other ways of creating similarly authentic experiences in systems modeling with children might involve local businesses. Businessmen and women could introduce students to the systems involved in their business and the challenges inherent in them. Students could then construct models of these systems and propose changes to these businesses. This approach puts the facilitator in the role of co-investigator with the learners, able to suggest alternative avenues and tools for exploration while engaged together in exploring an unfamiliar real-world system.

6 A Framework for Reflective Modeling Practice

A critical requirement of building tools and methodologies to support reflective systems modeling is the need for transparency at a variety of levels of access to different audiences of such models. In our work thus far, we have identified four levels of access to systems models built using this toolkit:

- model manipulation
- model creation
- toolkit design
- infrastructure design

These are not layers of abstraction, as a computer programmer might understand them, but rather different levels of activity in the space of system modeling. Our current work will be analyzed through this framework in the following section, but the key point is that these levels of access must inform both the design of such toolkits and infrastructures at all levels as well as informing the practices and roles of individuals within organizations and research projects engaged in reflective modeling. The goal must be for individuals at each level to experience the practice of systems modeling in a way that is malleable and transparent in order to promote reflection as they work towards some personally motivated goal with the model.

Some further description of the roles that we have observed is needed before examining these roles in the context of particular current projects. The broadest category of work is the design of the modeling infrastructure that lies beneath the particular domain-specific toolkits. Thus far, work on such infrastructures for tangible modeling toolkits has been limited to the work described in this thesis, but this infrastructure may be extended in the future by practitioners and researchers in ways that may be generally applicable or intended for a particular range of toolkit designs. Taking on this role will likely continue to require substantial computer programming experience.

The next level of work on these tangible models is the construction of domain-specific toolkits by building upon the modeling infrastructure. Though we have done much of this work in our research thus far, it is crucial to the creation of a sustainable use of such models in an organization. We have already seen the beginnings of others assuming this role in the work of other researchers at the MIT Media Lab, and there are rich prospects for this occurring in the other domains of our active work, which will be described in the following section. This kind of work currently requires some programming experience, but future development of the infrastructure described in this thesis can provide much simpler means for individuals to build domain-specific toolkits, as described in section 4.4.

The third level of engagement in this style of reflective systems modeling, creating models using the elements of a domain-specific toolkit, requires much less technical expertise and has been the focus of much of the current work that will be described in the following section. This generally involves connecting elements into a simulation model and, depending on the toolkit, possibly configuring the elements using a laptop computer or on the elements themselves. This process can certainly be collaborative to take advantage of the knowledge of several individuals in the construction of a model. Because these individuals were likely not involved in the design or implementation of the toolkit elements that they are using, it is important that the elements' operation be transparent as they are used to build a model of a particular system. Initial user-interface issues can be overcome with the help of someone experienced in using the toolkit, but it is difficult to work around missing, important information about the internal operation of simulation objects.

The final and broadest means of engaging in reflective systems modeling, manipulating an existing model, has been seen in all of the current work in this area. Allowing others to interact with a model may provide a means to communicate an idea in ways that a textual description, presentation, or video may not. However, the need for transparency is greatest for these individuals who have not been involved in the construction of the model or the toolkit used to build it. Both the individuals building the model and particularly those designing the domain-specific toolkit must be aware of the need for a

new observer of a particular model's operations to understand the operations and become convinced that it relates to the actual operation of a real-world process through observing and manipulating a particular model.

6.1 Situating Current Work

The current work to be described within this framework in this section includes the two case studies described in chapter 3, involving the Postal Service mailflow model and the computer network model, as well as two other research efforts at MIT which have used some elements of the infrastructure developed in this thesis to support reflective systems modeling in other environments. As the final project for MIT's graduate computer networking course, Margarita Dekoli and I created a tangible model of a modular network switch in order to measure performance and create a more transparent representation of its operation. [5] Oren Zuckerman has used some of the technology developed for this thesis to build a prototype of his System Blocks, designed to make it easier for children to explore dynamic systems. [39] Both of these efforts used only pieces of the infrastructure developed in this thesis, but they share important goals in using tangible representations of dynamic systems to make these systems easier to understand and manipulate, and the framework described in this chapter may shed some light on the structure of these efforts as well. Table 3 summarizes the application of our framework to these four areas of current work, and the remainder of this section expands on the roles of individuals in each case.

	USPS Mailflow Model	**Computer Network Model**	**Dekoli & Gorton's Network Switch Model** [5]	**Zuckerman's System Blocks** [39]
Model Manipulation	USPS Management	Children	Engineers & Customers	Children
Model Creation	USPS Engineers & Management	Children & Researchers	Engineers & Customers	Children & Researchers
Toolkit Design	USPS Engineers & System Designers	Children & Researchers	Engineers	Researchers
Infrastructure Design	USPS System Designers & Researchers	Researchers	Engineers & Researchers	Researchers

Table 3: Analysis of current work within the framework described in chapter 6

Our work with the US Postal Service, described in detail in section 3.1, has illuminated a wide variety of stakeholders in their efforts to understand and improve their mailflow processes. In our work thus far, we have engaged a variety of visitors from different areas of USPS management in manipulating the mailflow model, and conversations among such visitors often arise even as they merely watch one scenario run through the model. The USPS engineers and managers we've worked with have also expressed interest in involving broader audiences in this process, including shift supervisors and union representatives, in order to create widespread understanding of changes in the USPS processes. Though we worked with only USPS engineers to create the current model, as the technology and methodologies involved are developed further, managers and supervisors should be able to create models on their own from elements of a toolkit for mailflow models. This toolkit could be designed and constructed by USPS engineers or specialized engineers who focus on developing the USPS's technical capabilities for reflective modeling, whom we will refer to as "system designers." These system designers would also be able to extend the underlying modeling infrastructure in ways may be generally applicable to modeling or specific to USPS needs across individual toolkits. Ensuring that appropriate individuals are engaged in all of these roles has the potential to create a vibrant, sustained culture of reflective modeling across a wide spectrum of individuals in the Postal Service or any other organization facing similar challenges.

In our work with the computer network model and Computer Clubhouse members aged 10-13, described in detail in section 3.2, there have been two groups of individuals involved in our work: the children and us as researchers. In our workshops, the children have both manipulated and created models of different network arrangements. We have created the chat system that they've used as well as the hardware components that make up the nodes in the network. However, future development of this toolkit should certainly allow dedicated children to build their own network applications for this model network, as described in section 3.2.4, further expanding the range of explorations possible with this type of model. Though it is possible that a very enterprising teen might try to explore or modify the underlying network structure of the infrastructure, it seems likely that this will continue to be developed mainly by the researchers addressing these issues.

Margarita Dekoli and I built a prototype model of a modular network switch, shown in Figure 21, out of eight Towers, each consisting of a PIC Foundation [20] and the serial layer described in section 9.4. Our hope was that this model, in addition to serving as a useful platform for data collection about fault tolerance mechanisms inside such a switch, could also serve as a more transparent way for laymen and engineers alike to understand how different protocols function in this system. [5] In this way, customers might be able to manipulate or construct models of different switch configurations, and building such models could also be a useful way for engineers to prototype and evaluate new architectures and algorithms. Such engineers might be engaged at all levels of our framework, building and manipulating models, building toolkits for these models, and extending the modeling infrastructure. Customers and other non-engineers in their organization could also benefit from these models in evaluating designs and making purchasing decisions.

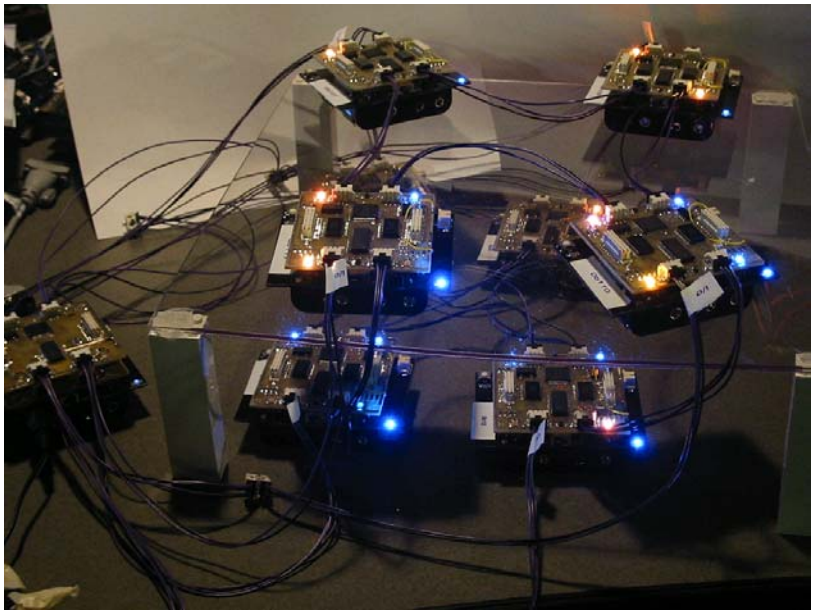

Figure 21: Dekoli & Gorton's model of a modular network switch

Oren Zuckerman's work has focused on building a toolkit of "System Blocks," shown in Figure 22, to allow children to create and interact with systems that simulate real-life dynamic systems. His work has been directly informed by research on system dynamics, and this has led him to create a toolkit using common system dynamics building blocks such as an "accumulator" and a "multiplier." [39] The work that Zuckerman has done

falls primarily in building a particular toolkit to support his research in allowing children to create and manipulate particular models using such a toolkit. Though Zuckerman used only pieces of the infrastructure developed for this thesis (specifically, the Rabbit Foundation, virtual machine, and serial layers described in sections 9.1, 9.2, and 9.4), work like his may both take advantage of and contribute to infrastructures to support the creation of toolkits like the one developed for this thesis.

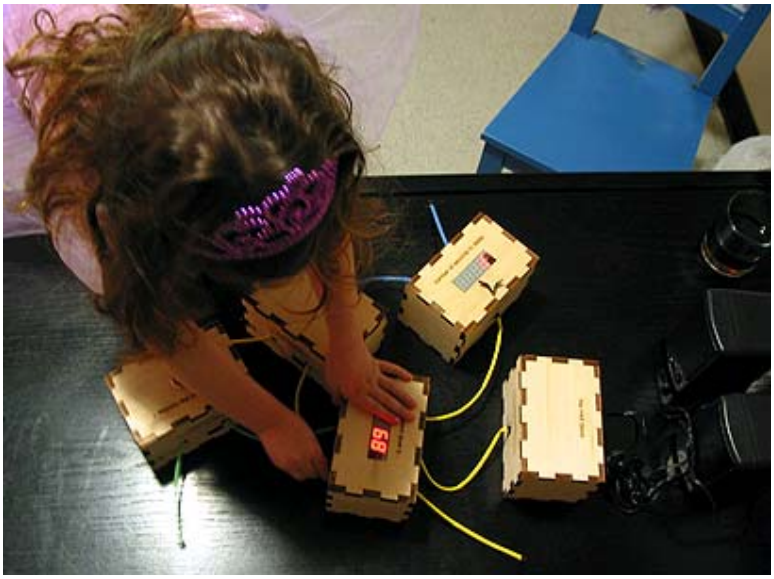

Figure 22: A four-year-old girl plays with Zuckerman's System Blocks
(image courtesy of Oren Zuckerman)

6.2 Future Scenarios

The current work described in the preceding section has demonstrated the breadth and potential impact of reflective systems modeling. However, there is much work remaining in evaluating the potential impact of using tangible toolkits in an organization and in developing sustainable methodologies for their use in an organization. The sketch of possible roles inside an organization actively engaged in reflective systems modeling which has been presented in this chapter has yet to be implemented in any organization. These toolkits and the roles surrounding their use will both affect and be affected by existing organizational structures and norms. The observations described in section 5.3 are only a limited view into these issues in the two contexts of our case studies. Obtaining an in-depth view into these issues in any organization would require a deep and

widespread use of such toolkits, allowing (as one USPS manager requested) individuals to keep such models on their desks to utilize at will. An in-depth study of these issues would doubtless require investigations in a variety of workplaces. The remainder of this section describes a number of other potential contexts where the practices of reflective modeling using tangible toolkits may have strong impact.

Both of the systems under study in this thesis's two case studies are primarily physical systems, though the actions and decisions of individuals play an important role in both systems, a point which was implicitly made by the USPS engineers' interest in finding rules for supervisors to follow when starting operations instead of setting schedules. Yet Forrester has demonstrated that dynamic system analysis may also be applied effectively to social systems, particularly business enterprises. In fact, Forrester suggests that a new role might emerge in corporations for an "enterprise designer." [9] While this type of analysis may have broad predictive power towards the behavior of complex social systems, it seems inevitable that if a specialized "enterprise designer" were to develop models and simulations regarding the future of a business, this data would still need to be acted upon by individuals who did not directly participate in the creation or manipulation of the model. This challenge, too, may benefit from a new means of interacting with dynamic simulations in order to build shared understandings among a management team about the characteristics of the social systems that they oversee.

There are also a variety of possible environments other than workplaces that could benefit from new means to build intuition of dynamic systems. One obvious example is formal education settings at a variety of age levels, from elementary schools to university education in business, engineering, science, economics, social science, and other fields. Business graduate schools already have a rich history of working with systems models to understand complex social and logistical systems, [6] and tangible toolkits such as those described in this paper may provide a more transparent means to build students and researchers' intuition and understanding of complex system models. Engineering and science programs also often focus on complex systems, often utilizing analytical approaches rather than simulations. However, tangible simulation toolkits could also prove to be a powerful means of interacting with dynamic systems in these fields.

Elementary and secondary education also present a broad range of challenges, and a number of researchers have worked to integrate systems modeling into these classrooms at all ages. [8] Oren Zuckerman has begun to address these issues for young children using portions of the modeling infrastructure created for this thesis, [39] but a great deal of work remains in building tangible interfaces appropriate for the classroom at a variety of levels. One particularly promising direction for classroom work may involve an approach deeply rooted in the direct experience of professionals engaged in reflective systems modeling in the local community. This could allow students to benefit from the situated knowledge of a professional's experience with his or her particular subject domain, providing both strong resources from that domain and a motivation to construct models which directly relate to real-world issues, and as they gain proficiency, the students may even shed light on new solutions to problems facing local businesses.

Other less formal educational settings also have the potential to benefit from reflective systems modeling. The work presented in this thesis includes one example of an informal learning setting, allowing youth to explore ideas about computer network architecture and network applications in the Computer Clubhouse. Other examples of such informal learning settings include museums, aquariums, after-school centers, and libraries. Such organizations often attempt to convey ideas about complex systems, and may be confronted with the same problems and misconceptions that we have seen in workplaces. Yet these settings may also have very different sets of individuals, values, and success metrics than learning in a workplace environment. More specifically tailored methodologies for designing and presenting such models may thus be required, and significant work remains in this area to develop these methodologies and evaluate their impact in a variety of informal settings.

Running through the challenges of building tangible tools for systems modeling and encouraging their use in reflective practice across a wide range of settings and problems are concerns of the interface presented to users of these models. Creating any common user interface will prove challenging due to the nature of the task of supporting practitioners building application-specific toolkits. Methods of presenting information, obtaining information from a user, connecting modules, and presenting cues that aid

understanding the system as both parts and a whole must be addressed, informed by studies of these systems' use in a variety of contexts. Already, work has been begun by Zuckerman [39] and Patten, et al [27] on designing interfaces for particular instances of traditional systems models for specific audiences. Much work remains to be done, however, on other alternative representations and interfaces, as well as interface toolkits for models in which the elements represent more complex objects, such as a Postal Service mail-processing operation. Though our use of the Tower system provided a flexible means of prototyping user interfaces, there is great potential for speeding the development and iteration of system models through the design and evaluation of hardware toolkits designed expressly for the purpose of building these systems models.

7 Conclusions

The work presented in this thesis has shown the feasibility of designing technical infrastructures to support building tangible models of dynamic systems, as well as two proof-of-concept case studies using such models to build understanding of dynamic systems, one in a workplace and another in an informal learning setting. These experiences have involved valuable themes and insights that are worth examining as we conclude these projects and consider the future direction of this work.

A central element of our approach to the problem of building environments for allowing novices to build models of dynamic systems has been the belief that tangible, concrete representations allow a new means of interacting with and relating to abstract ideas and results. This has been borne out by our case studies, and we have observed that this difference in the relationship between simulation objects and modelers is both pragmatic and affective. From a practical standpoint, we observed numerous advantages to the physical manifestations of the simulation objects. The simulations' scale makes it easy for a small group to gather around, and the distributed nature of the simulation presents information about each object locally, allowing different participants to examine different objects simultaneously and intuitively. These participants can also modify parameters or otherwise interact with different simulation object simultaneously, allowing different participants a sense of ownership of different physical objects in the simulation, such as different USPS supervisors being able to alter their own operations' parameters or different Computer Clubhouse members' ability to control their own model computer. Even the simple fact of having the simulation enter the participants' gestural space proved beneficial to individuals discussing a system's workings, such as the USPS visitors who promptly began heated discussions over our initial prototype, gesturing to indicate which mailflows which were moving too slowly, without needing to reach for a mouse or other interface mechanism. This also proved true in the Clubhouse members' discussions of how to arrange the model computer network. It seems clear in both cases that the physical instantiations of the simulation objects were helpful in the practical desires of individuals to discuss and interact with these models.

But even more remarkable was the affective dimension to the interactions between individuals and the physically realized simulation models. One USPS engineer's memorable expression, "we're running mail here," when leaning in to examine the mailflow model indicates an excitement and engagement with a simulation model that may prove difficult to replicate with an on-screen representation, even of a technically identical model. Similarly, the sudden engagement of USPS visitors with even our initial prototype, which didn't allow them to interact with the model at all, indicated a close connection to their work even before any explanation was given of the model's workings. The Computer Clubhouse members' long, excited, and possessive engagement with the computer network chat model certainly indicates a deep connection with their desire to communicate through novel mediums, and their anguish at having an undesirable color appear on their model computer may indicate even a closer connection to these small computers than these young people have to the much more sophisticated computers which they use daily. These affective reactions to the tangible models were, we believe, not merely due to the physical realization of the objects of these simulation models, but were especially prompted by the deep relationship between the individuals and the objects being modeled. A key advantage of the physical manifestations of these objects is to draw out and build upon these relationships in ways that may not be possible with traditional, abstract representations of on-screen simulations systems.

These deep relationships are also crucial to another central theme of our work in this thesis: the nature and use of situated knowledge. Though we have used the word "novice" throughout this thesis to refer to individuals who have not received any formal introduction to systems modeling, in reality many of the individuals in our studies have been very sophisticated in their understanding and discussions of the systems under study. The USPS engineers and managers were easily able to have extended, in-depth discussions of details of mailflow processes, and the Computer Clubhouse members were quick to give detailed descriptions of how they had used and created web sites. However, when these individuals encounter disagreement or uncertainty—whether about how to handle an unexpected influx of mail or how to explain the behavior of common network applications—they may lack the means to explore alternatives and communicate their conclusions convincingly to others. The best tools for these purposes will leverage both

participants' explicit experience and their tacit knowledge in order to help them revisit their ways of viewing the world. As important indications in this direction, in both of our case studies we have observed individuals using the physical simulation objects as both a means for exploration and a means of communicating their results to others transparently and conclusively.

Yet despite these promising results, a great deal of work remains to be done in these areas. Fully realizing the framework outlined in chapter 6 for the effective and long-term use of reflective systems modeling in an organization requires further work in developing the modeling infrastructure to enable the more rapid creation and iteration of robust, domain-specific toolkits as well as work to identify and evaluate successful organizational practices for learning and communicating using these tools in a variety of settings. However, it is equally clear that the potential benefits for success in understanding dynamic systems and communicating these insights are considerable, for both organizations and individuals.

8 References

[1] Cameron, B., Wijekumar, K. (2003). The effectiveness of simulation in a hybrid and on-line networking course. *SIGCSE Technical Symposium on Computer Science Education.* ACM Press: New York, pp 117-119.

[2] Colella, V. (1998). Participatory Simulations: Building Collaborative understanding through Immersive Dynamic Modeling. Master of Science Thesis, Media Arts and Sciences. Massachusetts Institute of Technology. Cambridge, MA.

[3] Computer Clubhouse Network web site. *http://www.computerclubhouse.org* Accessed May 17th, 2003.

[4] Cricket Software Download web page. *http://llk.media.mit.edu/projects/cricket/software/index.shtml* Accessed May 20th, 2003.

[5] Dekoli, M., Gorton, T. (2002) Modeling a Fault-Tolerant Routing Algorithm for a Modular Hypercube Packet Switch Fabric. MIT Class Paper. 6.829: Computer Networking. December, 2002. Included in appendices of this thesis.

[6] Forrester, J. (1969). Urban Dynamics. MIT Press: Cambridge, MA.

[7] Forrester, J. (1992) System Dynamics and Learner-Centered-Learning in Kindergarten through 12th Grade Education. Paper D-4337. System Dynamics Group, Sloan School of Management, Massachusetts Institute of Technology. Cambridge, MA.

[8] Forrester, J. (1994) Learning through System Dynamics as Preparation for the 21st Century. Paper D-4434. System Dynamics Group, Sloan School of Management, Massachusetts Institute of Technology. Cambridge, MA.

[9] Forrester, J. (1998) Designing the Future. Paper D-4726. System Dynamics Group, Sloan School of Management, Massachusetts Institute of Technology. Cambridge, MA.

[10] Freeman, J. US Postal Service. Presentation at MIT Media Lab Things That Think sponsor meeting. May 21st, 2002.

[11] Gorton, T., Mikhak, B., Paul, K. (2002) Tabletop Process Modeling Toolkit: A Case Study in Modeling US Postal Service Mailflow. Demonstration at CSCW 2002. November, 2002. Included in appendices of this thesis.

[12] High Performance Systems, Inc.™ web site. *http://www.hps-inc.com* Accessed May 18th, 2003.

[13] Horgen, T., Joroff, M., Porter, W., Schön, D. (1999). Excellence by Design: Transforming Workplace and Work Practice. John Wiley & Sons, Inc: New York, NY.

[14] Howstuffworks Internet Channel Home Page. Web site. *http://computer.howstuffworks.com/channel.htm?ch=computer&sub=sub-internet* Accessed May 17th, 2003.

[15] Ishii, H., Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people, bits and atoms. *Proceedings of CHI 1997*. ACM Press: New York, pp 234-241.

[16] Keshav, S. (1997). An Engineering Approach to Computer Networking. Addison-Wesley, Reading, MA.

[17] Klopfer, E. Thinking Tags in Participatory Simulations. Presentation on web site. *http://education.mit.edu/badges_files/frame.htm* Accessed May 19th, 2003.

[18] Kobayashi, K., Hirano, M., Narita, A., Ishii, H. (2003). A Tangible Interface for IP Network Simulation. *CHI 2003 Extended Abstracts*. ACM Press: New York, pp 800-801.

[19] Logo Computer Systems, Inc.™ Web site. *http://www.microworlds.com* Accessed May 17th, 2003.

[20] Lyon, C. (2003). Encouraging Innovation by Engineering the Learning Curve. Master of Engineering Thesis, Massachusetts Institute of Technology. Cambridge, MA.

[21] Martin, F., Mikhak, B., Silverman, B. (2000). MetaCricket: A designer's kit for making computational devices. *IBM Systems Journal* (Vol. 39, Nos. 3 & 4)

[22] McCoy, J., French, S. Abnous, R., Niccolai, M. (1981). A local computer network simulation. *SIGCSE Technical Symposium on Computer Science Education*. ACM Press: New York, pp 263-267.

[23] Millner, A. From Dragon Ball Z to MP3: Lower socio-economic status kids' conceptions of the Internet. Talk at MIT Media Lab. April 10th, 2003.

[24] MIT System Dynamics in Education Project. Road Maps Project. *http://sysdyn.mit.edu/road-maps/* Accessed May 18th, 2003.

[25] Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. Basic Books: New York, NY.

[26] Patten, J. (2001). Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. Master of Science Thesis, Media Arts and Sciences. Massachusetts Institute of Technology. Cambridge, MA.

[27] Patten, J., Ishii, H., Hines, J., Pangaro, G. (2001). Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. *Proceedings of CHI 2001*. ACM Press: New York, pp 253-260.

[28] Paul, K. US Postal Service. Private communication. September, 2002.

[29] Rabbit Semiconductor™ web site. *http://www.rabbitsemiconductor.com* Accessed May 21st, 2003.

[30] Resnick, M. (1990). MultiLogo: A Study of Children and Concurrent Programming. *Interactive Learning Environments*, vol. 1, no. 3.

[31] Resnick, M. (1994). Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds. MIT Press: Cambridge, MA.

[32] Schön, D. (1982). The Reflective Practitioner: How Professionals Think in Action. Basic Books: New York, NY.

[33] Schrage, M. (2000). Serious Play: How the World's Best Companies Simulate to Innovate. Harvard Business School Press: Boston, MA.

[34] Senge, P. (1990). The Fifth Discipline: The Art & Practice of the Learning Organization. Currency Doubleday: New York, NY.

[35] Starr, P. (1994). "Seductions of Sim: Policy as a Simulation Game," *The American Prospect,* no. 17 (Spring 1994). pp 19-29.

[36] Turkle, S., Papert, S. (1991). Epistemological Pluralism and the Revaluation of the Concrete. Constructionism. Eds I. Harel & S. Papert. Ablex Publishing Corporation: Norwood, NJ, pp. 161-191.

[37] UCLA Internet Report – "Surveying the Digital Future." (2003). UCLA Center for Communication Policy. January 2003.

[38] Verisim™ web site. *http://www.verisim.com* Accessed May 18th, 2003.

[39] Zuckerman, O., Resnick, M. (2003). A Physical Interface for System Dynamics Simulation. CHI 2003 Extended Abstracts. ACM Press: New York, pp 810-811.

9 Appendices

9.1 Rabbit Tower Foundation Hardware Implementation

The Rabbit Tower Foundation, shown in Figure 23, was created in order to provide a more powerful computational base for the Tower, in order to support much more memory, multitasking, and Ethernet and TCP/IP capabilities. The RCM2200 and RCM2300 modules from Rabbit Semiconductor™ [29] were selected for this Foundation. These modules contain the Rabbit 2000 processor and RAM and Flash memory. The modules are identical except that the RCM2200 adds the capability to plug directly into an Ethernet network. The two modules are pin-compatible, allowing the same Foundation board to serve both modules. The modules also use identical code, allowing the same virtual machine to run on both foundations, though the functions relating to controlling the Ethernet controller chip have no effect on the RCM2300.

Figure 23: The Rabbit Tower Foundation

The circuit board for any Tower Foundation must provide the following functionality:

- Tower connectors to stack layers above the Foundation
- 5V power connection: batteries or regulating a wall adapter
- Connecting the processor's input and output (I/O) pins to the Tower's pins
- Connector for an RS-232 serial adapter to download user programs
- LED to indicate power and whether a user program is running

- A button to start and stop user programs
- I2C support to communicate with Tower layers, preferably allowing future expansion to multimaster support

A diagram of the Foundation's circuit is shown in Figure 24. The only remarkable feature of the circuit is the presence of a PIC16F876 in the circuit, which is used primarily for its hardware support of the I2C bus (the Rabbit only supports this protocol in software) but also to use its hardware pulse width modulation unit to control the Foundation's blue LED. I used one of the Rabbit's serial ports to communicate with the PIC (also using its hardware serial port.) I devised a simple serial handshaking protocol to enable the Rabbit to control the PIC's communication on the I2C bus, as well as the rate at which it pulses the blue LED. One of the Rabbit's other serial ports is used to communicate with the PC to download user code, and a third serial port is connected to two of the Tower's pins.

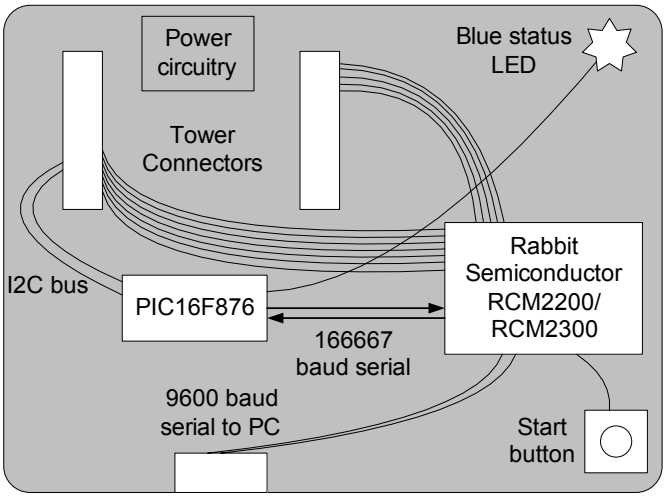

Figure 24: Rabbit Tower Foundation architecture

9.2 Rabbit Logo Virtual Machine Implementation

To allow users to program the Rabbit Tower Foundation in a transparent, easily accessible manner, I wrote a virtual machine (VM) in C that runs on the Rabbit module on the Foundation. This virtual machine acts as a bytecode interpreter for a variant of the Logo language. The design and implementation of this virtual machine are the result of a close collaboration with Brian Silverman, a colleague at the MIT Media Lab. The virtual machine shares the fundamental design of other embedded virtual machines for the Logo

language that Brian has created. [20][21] The basic features of this virtual machine are listed in Table 4 alongside those of the Tower's PIC virtual machine. Notable differences from the Cricket and Tower PIC virtual machines include the use of 32-bit integer and floating-point arithmetic, string and array manipulation, up to twenty threads of preemptive multitasking, and the addition of primitives to control the TCP/IP functionality of the RCM2200.

	PIC Virtual Machine	**Rabbit Virtual Machine**
Target platform	PIC16F876 and PIC16F877	RCM2200 and RCM2300
VM implementation language	PIC Assembly	Rabbit C
Processor clock speed	8 Mhz or 20 Mhz	22.1 Mhz
Processor data storage	384 bytes RAM 8 kilobytes EEPROM	128 kilobytes RAM 256 kilobytes EEPROM
Math	16-bit signed integers	32-bit signed integers and floating-point numbers
Timer	32-second timer	24-day timer & 128-year clock
Multitasking	2 processes	20 processes
Logo execution speed	~26,000 Logo instructions/sec (8 Mhz) ~65,000 Logo instructions/sec (20 Mhz)	~8,000 Logo instructions/sec
Variables	96 global variables (16-bit) Procedure arguments	128 global variables (32-bit) Procedure arguments Local variables inside procedures
I2C bus	Hardware support built-in	Uses external PIC16F876 on Foundation
On-board serial ports	One	Four
Other		String manipulation Arrays TCP/IP stack 10 Base-T Ethernet On-board flash storage

Table 4: Features of PIC and Rabbit virtual machines

The use of 32-bit integers and floating-point numbers was made possible through the use of the Rabbit's C language for the VM implementation, utilizing the language's support for these data types as the basis for the VM's data types. All variables and the stack are internally represented as 32-bit signed integers, though these are used for a variety of

purposes within the VM, including integers, floating-point numbers, pointers to strings and arrays in memory, pointers to global variables, pointers to Logo threads, pointers to blocks of executable bytecode, pointers to TCP/IP sockets, and pointers into an internal procedure table. In fact, relatively few of these are enforced by the compiler (really only the pointers to the procedure table, which are impossible to generate in the language), allowing the programmer the possibility of misusing a value of a particular type. In practice, however, this rarely occurs as novices program in these virtual machines, largely because they tend to use only integers as data, not maintaining pointers to strings, arrays, etc as user variables. Advanced users must exercise care in utilizing a variety of these datatypes simultaneously. The most obvious area of concern includes the addition of floating-point numbers, which require separate primitives for mathematical operations such as addition and multiplication, as well as explicit conversion to and from the integers more commonly used in user programs. Without adding type checking to the language—a substantial complication for the presently compact compiler—it is impossible for the compiler to identify such errors. The large, 32-bit values also allow the addressing of a much larger memory space, so much so that a high-order bit in such addresses is used as a flag to indicate whether the desired memory address is in RAM or non-volatile flash memory.

Preemptive multitasking is achieved by maintaining separate Logo stacks, instruction pointers, frame pointers, and stack pointers for each thread. Context switches are permitted after certain primitives, usually chosen to roughly correspond with the end of a line of Logo code. Thus primitives which return values, such as arithmetic operations and data accessors generally do not permit context switches, while primitives such as loops, waits, and data modifiers generally allow context switches. This is by no means a hard and fast rule, but rather is intended to permit the VM to execute a suitably substantial block of code without spending an unacceptably large fraction of its time context switching. With the increased capacity for multitasking, issues regarding race conditions are inevitable. To provide a structure for managing such concerns, I created the `no-multi` primitive, which executes a block of code while preventing context switching within the code. This allows code such as an I2C communication routine to ensure that another I2C communication sequence is not started during the first, and also allows complicated

modifications of shared variables to proceed without interference. Another obvious option for handling these issues is the creation of a system of locks, but these were considered unnecessarily cumbersome for this lightweight system, though their implementation remains an option for the future. The most extensive applications written thus far for this platform are the applications for the two case studies described in this thesis, each containing approximately 1500 lines of Logo code and compiling to approximately 10,000 bytes of compiled Logo bytecode; these applications make heavy use of multitasking, and the `no-multi` approach has proved effective thus far.

The string and array manipulation allowed in the Rabbit VM is notable mainly for its use of range checking and the challenges associated with automatically initializing the required length information in RAM. Strings and arrays are stored identically in memory, simply using different primitives to read and write either individual bytes or 32-bit words. In order to store length information for use in range-checking, two bytes of length information are prepended to the string data. The accessor and modifier primitives check that the specified index is within the string's bounds before performing their functions. A range check error halts the virtual machine and causes a sequences of flashes on the Foundation's indicator LED. A single byte (a zero) is also appended to the string data segment to ensure that the string remains zero-terminated for the benefit of the built-in C functions used to manipulate the strings. Strings and arrays are entirely allocated at compile-time, allowing constants to be used for their addresses. However, the lengths and trailing zeroes must be added to the strings in RAM after a program download and each time that the VM starts. Instead of creating a custom data structure to contain the data necessary for the initialization of these strings, I created two new Logo primitives to initialize strings in RAM. The compiler automatically generates sequences of these instructions to initialize RAM strings when the VM starts, as well as after a new program is downloaded to the Rabbit.

The final notable feature of the Rabbit Tower VM, as compared to the Cricket and PIC Tower VM's, is the addition of primitives to create and use TCP/IP sockets using the RCM2200's Ethernet controller and the Rabbit's C language TCP/IP driver. This system has four basic functions: creating a socket to connect to a remote device, creating a socket

when a remote device connects to the Rabbit, sending data to an active socket, and receiving data from a socket. In order to work with multiple sockets, it quickly became clear that a socket data structure would be necessary. To represent these sockets internally, I used C pointers into an array of socket structs. Primitives and Logo functions allow programmers to modify the RCM2200's IP address or use DHCP, open and accept sockets, and read and write characters and strings to and from these sockets. One particularly clear example of their use is the Logo code for a simple web server, given in Figure 25, which ignores the web browser's request and simply sends a single web page to the incoming socket.

```
globals [hit-count]

to do-simple-web-server
 ignore launch [loop [tcp-tick]]
 sethit-count 0

 loop [
   let [sock sock-accept 80]
   sethit-count hit-count + 1
   sock-write :sock "|this is a test. <br><br> thank you for
visiting!<br>you are visitor number |
   num-to-str foo hit-count
   print-string foo
   sock-write :sock foo
   sock-close :sock
   cr print-string "done cr
 ]
end
```

Figure 25: Rabbit Logo code for a simple web server

9.3 Tower Development Environment Implementation

The Tower Development Environment is an integrated environment, written in Java and Logo, for compiling Logo code for a number of virtual machines, including the Tower and PIC virtual machines, as well as an assembler for PIC 16F87x microcontrollers using our own assembly mnemonic, in which we've done all of our PIC assembly development. The compilers and assemblers used by our research group, which have been directly based on Brian Silverman's work on embedded Logo virtual machines and assemblers, have long been written in Logo as a flexible environment for rapidly creating these compilers and assemblers. [21] The TDE contains a Logo interpreter written in Java by Brian Silverman and extended for this project. This interpreter is known as "JavaLogo."

Primitives were added to JavaLogo to control and obtain data from Java UI elements, as well as using the computer's serial port.

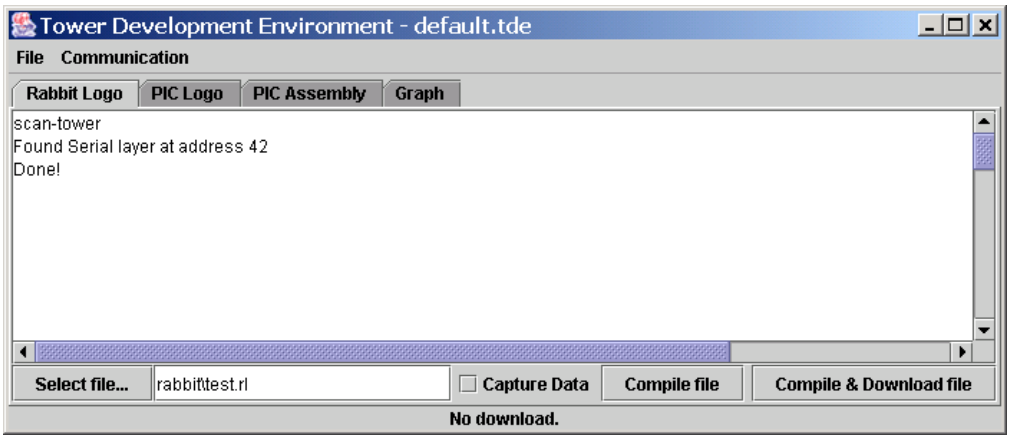

Figure 26: Tower Development Environment

The TDE began as an environment for programming the Rabbit virtual machine alone, but grew into a common platform for programming the full range of Tower virtual machines and assemblers, each on its own "tab" in the software, as shown in Figure 26. The JavaLogo interpreter is used to compile and assemble programs, script the user interface, and load and save files. This has allowed the compilers and user interface to be developed and modified rapidly as the various virtual machines and other functionality were improved. Our experience in developing a pure Java compiler for the Cricket logo language [21] has shown that this flexibility is not universal in implementations of compilers for these virtual machines.

Our compilers share many of the same basic features of the compiler described in [21], including a "command center" in which single lines of Logo code can be interactively executed, allowing the user to try small pieces of code while building and debugging an application. A user can also write procedures in external files and download them onto the appropriate platform to, in effect, add new primitives to the language for more complicated functionality. A number of functions for each platform have already been written, both for system commands and to interface with the variety of Tower layers. These can be seamlessly used in a file by adding an "include" line in the file.

One other important addition to the TDE is its graphing components, which allow a user to plot stored data or display data in real-time in a variety of formats. These graphing capabilities are based on the Logo Graph application, [4] a program that loads stored data from Crickets [21], which I developed as an MIT undergraduate working with the Media Lab's Lifelong Kindergarten research group under the direction of Bakhtiar Mikhak and Fred Martin. Though the graphing tools are intended to make it easy to graph commonly used data formats, much like Logo Graph, the TDE's graphing components can also be programmed in JavaLogo to create more complex representations, such as a histogram, an x-y plot of data, or decoding a more complicated representation. Data is collected from any other programming tab by clicking the "capture data" check box shown in Figure 26. When this is checked, the application records textual data sent back from the Tower (by using the print command, etc.) instead of displaying it in the command center. This allows a great deal of flexibility in obtaining data to be graphed, which has allowed the graphing components' use with all of our virtual machines without any additional work.

9.4 Serial Layer Hardware and Firmware Implementation

The Serial Layer for the Tower, shown in Figure 27, was created for this thesis work to provide the communication infrastructure for building distributed networks of Towers for systems modeling. Each layer contains four serial ports, each with 64-byte FIFO (first-in, first-out) buffers on its transmit and receive ports. Multiple layers can be used on the same Tower to increase the number of ports available. Three-pin connectors allow cables to easily connect Towers in a model. There is also a tri-color LED (red, green, and blue) next to each serial connector, providing the ability to "color-code" data flowing through the network under software control.

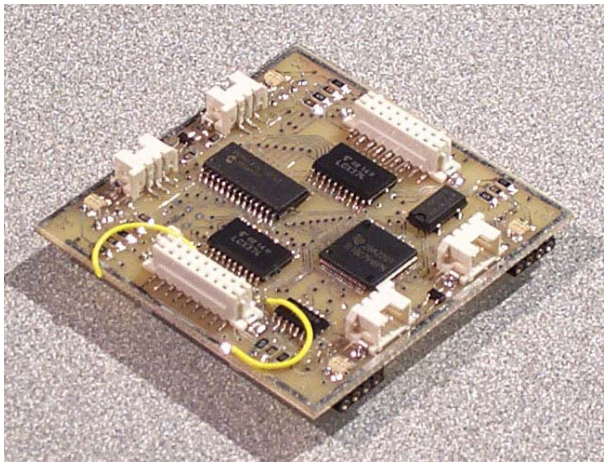

Figure 27: Serial layer

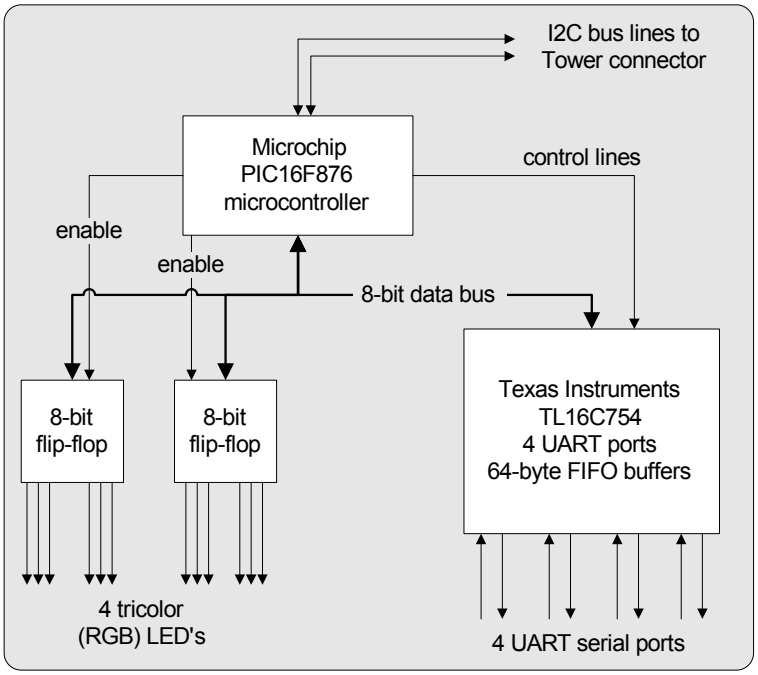

Figure 28: Serial Layer Hardware Architecture

The architecture of the layer is diagramed in Figure 28. The PIC microcontroller is used to communicate with the Foundation via the I2C bus. The PIC also controls the TI serial chip, which contains four buffered serial ports. The 8-bit data bus used for the TI chip is also used to store bits into the two flip flops, which are connected to the 12 LED's in the four tri-color LED's on the board. This allows the LED's to remain lit when the PIC is communication with the serial chip, without requiring a microcontroller with more input/output pins. I wrote firmware for the PIC in assembly language, starting with the

I2C communication framework developed for Tower layers by Chris Lyon. [20] The layer's software interface consists of the functions listed in Table 5. These allow the user to send and receive data from each port, as well as controlling the tri-color LED's on each corner of the board. The layer does not provide any direct support for packet parsing or multi-hop routing; the implementation of this functionality in the Rabbit Foundation is discussed in section 9.5.

Procedure and arguments	Description
write-left-flip-flop :value	Writes colors to the left two LED's
write-right-flip-flop :value	Writes colors to the right two LED's
set-uart-speed :port :num1 :num2	Sets the baud rate of a serial port using a special set of numbers related to the serial crystal speed (default is 9600 baud)
get-uart :port	Outputs a byte from a port's receive buffer
new-uart? :port	Outputs 1 if there is a byte waiting in the port's receive buffer, 0 otherwise
put-uart :port :val	Adds a byte to a port's transmit buffer
clear-tx-buf :port	Clears a port's transmit buffer
clear-rx-buf :port	Clears a port's receive buffer
get-up-to-n-bytes :port :n :buf :start	Obtains up to :n bytes from a port's receive buffer (as many as are available)

Table 5: Serial layer software interface

There are several areas in which the serial layer could be expanded to better support the systems modeling applications, as described in section 9.5.3, including vastly increasing the speed of the network created with these layers by perfoming packet-parsing and forwarding on the layer's PIC. The one purely technical challenge that has yet to be overcome is the problem of the serial layers leaking power to other Towers that are connected but not powered. This occurs because the default state of the serial transmit line is high, and the serial chip on the attached Tower can use this to obtain some power if that Tower is not powered. This partially powered state can leave the PIC on this layer and others in unusual states, causing the Tower to malfunction. This has been more of an annoyance than a serious challenge to using these boards with the Tower, and a set of optoisolators on the serial receive lines of the current revision of the board represent the latest failed attempt to resolve this issue.

9.5 Packet Parsing and Multi-hop Routing Algorithms

As described in section 9.4, the serial layer created to build networks of Towers only supports reading and writing bytes from its individual serial ports. However, a robust network communication protocol requires a notion of packets and a means of routing them between distant Towers, both so that Towers can communicate with non-adjacent Towers if this is required by the model and so that an external computer (a PC, laptop, or handheld PDA) can obtain information from all the Towers in the model. Furthermore, the configuration of this routing algorithm must be automatic in order to support rapid reconfiguration of the elements in a model. In the current iteration of the infrastructure described in this thesis, the packet parsing and routing algorithms are implemented in Logo on the Rabbit Foundation. The packet parsing and routing algorithms have also been ported to LCSI's Microworlds product [19], a Logo environment for desktop computers, in order to allow a desktop computer to act as another node in the Tower network. The remainder of this section describes the design and implementation of these algorithms and outlines an architecture for vastly improving the speed of the Tower network by implementing much of this functionality on the PIC on the serial layer.

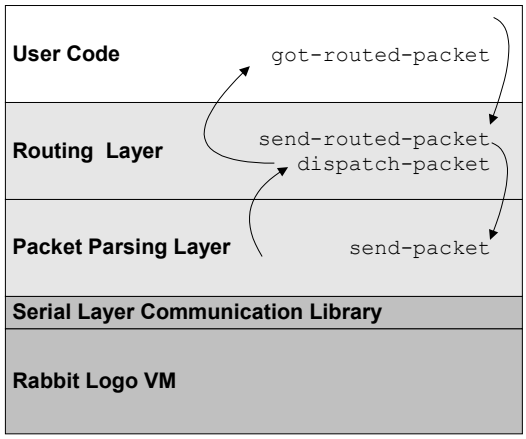

Figure 29: Packet and Routing Software Layers

9.5.1 Packet Parsing Algorithm and Implementation

The structure of a packet in this system is shown in Table 6. It contains length and checksum information in order for the Rabbit to identify and verify the integrity of a packet. The verification of packet identity has proved important in the face of rare errors

in byte reception by the serial layer, especially when removing or inserting cables between layers. The packet layer contains routines to create this packet structure given a data payload. A simple state machine written in Rabbit Logo assembles an incoming packet in a RAM buffer, finally checking the checksum to verify that the packet is intact. After receiving a packet, the packet layer calls the Logo function `dispatch-packet`, which the packet-parsing software does not define. This provides the needed linkage between the software layers, as shown in Figure 29.

Packet Component Description	Byte index	Length in bytes
Constant (hexadecimal 0xbe and 0xef)	0	2
Data length	2	1
Packet color (3 bits of red, green, blue)	3	1
Packet's data payload	4	(Data length)
Checksum	5 + (Data length)	1

Table 6: Packet Structure

The packet layer allows two serial layers to pass data structures between them, but does not provide for a way for Towers to communicate by passing data between intermediary nodes in a tabletop network. The routing layer provides a structure within a packet to indicate source and destination nodes, as well as an automatic routing algorithm to fill forwarding tables at each node. Like the packet-parsing algorithm, the routing algorithm and forwarding routines are written entirely in Rabbit Logo. When a node receives a packet destined for itself, it calls the Logo function `got-routed-packet`, which must be defined by the user code.

9.5.2 Routing Algorithms and Implementation

The structure of a routed packet is shown in Table 7. Functions in the routing layer create this structure given a data payload, and the layer unpacks the structure in order to provide the data payload to the user code via the function `got-routed-packet`. Note that the data structure described in Table 7 is placed within a packet by the packet parsing software layer, which handles the length and checksum of the packet. One notable feature of the routed packet structure is the presence of an opcode, easily allowing a variety of different

data types to be passed between nodes. This functionality must be built into the routing layer rather than implemented in the user code because the routing layer itself needs to communicate data between nodes in order to automatically build forwarding tables, and the opcode zero is reserved for this purpose. The packet-forwarding engine is quite straightforward, using a simple array of nodes indicating which port, if any, a packet destined for that node should be forwarded to.

Routed Packet Component Description	Byte index	Length in bytes
Opcode (0 for internal use, >0 for user code)	0	1
Destination node (255 = “anybody”)	1	1
Sending node	2	1
Data	3	(Routed data length)

Table 7: Routed packet structure

The algorithm to determine when nodes are added or removed and automatically build the forwarding table accordingly is by far the most complex part of this network infrastructure. There are three parts of this algorithm: link state maintenance, complete table reconfiguration, and periodic updates of table information. For the relatively small scale of the networks involved, the intent is to completely rebuild the routing table when a node is added or removed. Rare packet losses can allow this process to result in incomplete routing tables in some or all nodes, so periodic updates of the entire table are performed between neighbors in order to more slowly propagate any information that may have been lost during the last reconfiguration. In order to know when to rebuild the table at all, each node must maintain awareness of each of its neighbors. This is accomplished by having each node broadcast periodic “here I am” messages to its neighbors and each node maintaining a list of the last time it heard from each of its neighbors. If a new neighbor sends a “here I am” message or an old neighbor has not been heard from after some time, the node broadcasts a reset message and begins the process of rebuilding its table along with the other nodes. In the current implementation, “here I am” messages are sent every 4.5 seconds, and the timeout before a node decides its neighbor has disappeared is 15 seconds.

When one node broadcasts a reset message, every node rebroadcasts it to all of its other ports. A minimum time between such rebroadcasts prevents these packets from traveling forever through the network. Each node also sends a reply to the first node from which it hears a reset packet, and the originating node remembers this information, thereby building a tree inside the network, rooted at the node that sent the first reset packet. Note that every node (except the root) sends exactly one acknowledgement to its neighbor towards the root node, resulting in a loop-free tree. This tree may be used by applications to broadcast messages to every node without concern that packets might loop forever.

After rebroadcasting a reset packet and clearing its own forwarding table, each node sends a route announcement packet, including the target node and the number of hops towards that destination. Nodes hearing these announcements compare the included hop count to the entry for the target node in their forwarding table. If the entry does not exist or the newly announced route is shorter than the previously saved route, the table is updated and the node sends its own route announcement to its other ports, advertising the newly shortened route. Since routes are only rebroadcast when they are shortened, the algorithm rapidly converges to an optimal routing table in each node, containing a shortest possible route to each of the other nodes in the network.

However, this is only true in the absence of packet loss. We have observed that even a low loss rate can result in some nodes being unreachable from some other nodes. To enable the network to reach stability even in the face of packet losses, the nodes also periodically broadcast their entire routing tables to their neighbors, who compare each entry and update their tables if necessary. However, route announcements are not automatically sent at this point; other nodes will discover the new route slowly via these same periodic broadcasts of nodes' forwarding tables. In our current implementation, the time between forwarding table broadcasts is 15 seconds.

9.5.3 An Improved Architecture for Embedded Network Infrastructure

Because the design of networks and routing protocols was not the focus of this thesis research, no rigorous study of the performance of this network infrastructure has been performed. However, there is clearly room for considerable improvement in speed as well

as reducing the workload on the Foundation processor by performing packet parsing and forwarding directly on the Serial Layer's PIC microcontroller. The reduced workload on the Foundation could also allow the implementation of a TCP/IP-like system of reliably bytestreams between nodes. The only obvious limitation to the PIC microcontrollers currently in use for this system is the amount of RAM that would be required to store packets before they were sent over another port or to the Foundation. However, the newest microcontrollers provide up to several kilobytes of RAM, which would be sufficient for the packet-parsing and forwarding needs. No other change to the serial layer hardware would be necessary.

The architecture of this proposed solution may best be described by focusing on the use of RAM on the Serial Layer, as shown in Figure 30. Each port would require a buffer to assemble a single incoming packet before it is transferred to an internal buffer for dispatch. Each port would also require at least two bytes in addition to the buffer to save the current state of the packet-parsing state machine and the buffer's current length. After a packet is assembled and verified by a state machine, a packet forwarding routine would search the forwarding table for the destination port. If no such entry exists or if the destination field indicates that the packet is intended for this node, the packet is flagged for delivery to the Foundation processor. The packet and its destination are deposited into one of a number of internal packet buffers awaiting dispatch. The packet dispatch routine would transfer a packet to an available destination port and mark the buffer as empty. The Foundation would query the layer to obtain any new packets intended for this node. It would also deposit packets to be sent directly into an outgoing buffer along with a destination port number. The Foundation would continue to build routing information during the reset process and send forwarding table entries to the layer. Some strategy would be necessary for dropping packets as the internal buffers fill, and several well-known strategies are available in literature on computer networking. [16]

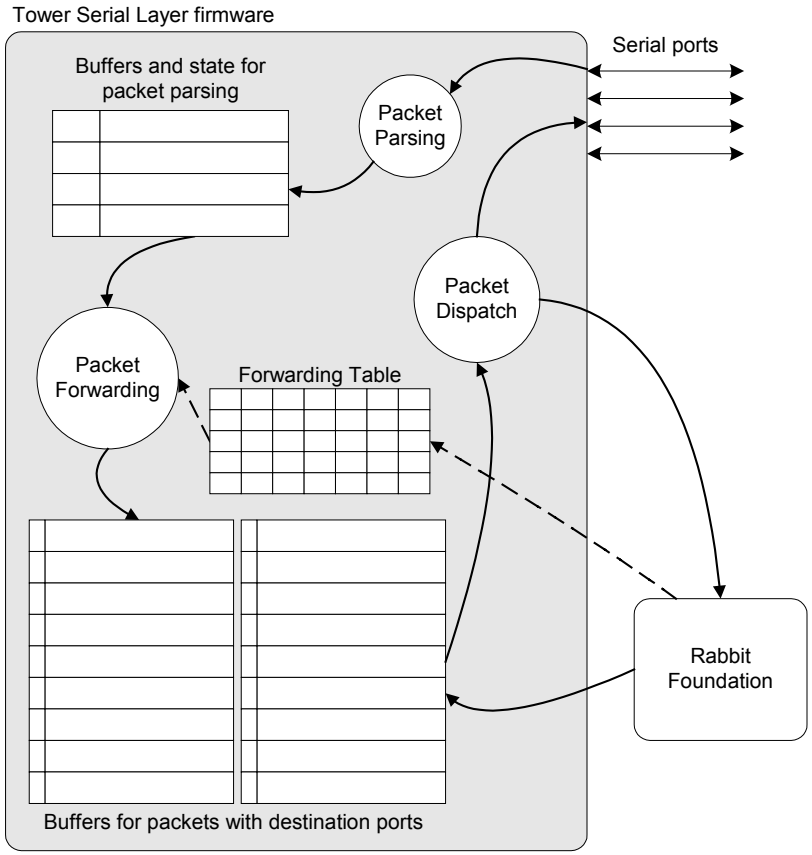

Figure 30: Proposed Serial Layer Packet Forwarding Architecture

Moving the packet-parsing and forwarding onto the serial layer would create a substantial increase in the speed of a packet moving through nodes, possibly as large as two orders of magnitude. This can be accomplished on a relatively inexpensive microcontroller such as the 18F series of PIC microcontrollers. The primary requirement of this architecture is memory, and two kilobytes of RAM would provide enough space for a fifty-entry forwarding table and forty buffers for thirty-byte packets, along with the other necessary variables and buffers. The split between the layer and the Foundation also closely mirrors the architecture of modern routers which have very fast mechanisms for packet forwarding but much slower mechanisms for building routing tables and handling some special types of control packets. In fact, modern routers often use an embedded PC for these "slow-path" packets, much like our use of the Rabbit Foundation for this functionality. [16]

9.6 Gorton, Mikhak & Paul CSCW Paper

The following paper was presented as a demonstration at CSCW 2002 in New Orleans. Because it was not actually published in the CSCW proceedings, it is included here for reference. More information about this work's relation to this thesis may be found in Section 3.1.

Tabletop Process Modeling Toolkit: A Case Study in Modeling US Postal Service Mailflow

Tim Gorton and Bakhtiar Mikhak
Grassroots Invention Group
MIT Media Lab
20 Ames St., Room E15-020A
Cambridge, MA 02139 USA
{tgorton, mikhak}@media.mit.edu

Ken Paul
United States Postal Service
MIT Media Lab Liaison
20 Ames St., Room E15-384A
Cambridge, MA 02139 USA
kpaul@email.usps.gov

ABSTRACT
Powerful simulation software packages often produce results that are counter-intuitive. These results are therefore hard to internalize for the target audience who needs to use these results in making nontrivial choices in setting up and managing complex systems with intricate real-time dynamics. We are developing a tangible modeling toolkit to allow novices to set up physical representations of the process that they are interested in modeling and collaboratively interact with it in real time. We will demonstrate our current implementation of this toolkit and the simulation tool that we, in collaboration with the United States Postal Service, developed to help postal managers figure out how to balance work hours to workload and increase throughput of time-sensitive mails through hands-on manipulation of various operations modeled in the simulation.

Keywords
System dynamics, tangible user interface, simulation model, workplace research, mailflow analysis

INTRODUCTION
The field of system dynamics has provided valuable methods for understanding the structure and behavior of complex systems through the use of computer models. [1] This has provided a framework for understanding and improving the way that organizations adapt to changing conditions [11] and a means of analysis for data collected by workplace observation studies. [3] Yet developing an understanding of these models and simulations remains a significant challenge both for learning researchers and toolbuilders who attempt to address this area. [2, 9, 10, 12]

Our work to develop a tabletop system for building models of dynamic systems has evolved as a close collaboration between the Grassroots Invention Group at the MIT Media Lab and the United States Postal Service (USPS), a major Media Lab sponsor. This demonstration will present our work thus far with the Postal Service while suggesting more general applications of this system. This collaboration has allowed us to further develop our tool based on insights provided by interactions in this real world context. A closer look at the problems in this context help show how this approach can provide solutions to challenges that are inherently associated with how we learn, think, and communicate using simulation tools rather than with technical limitations of the tools themselves.

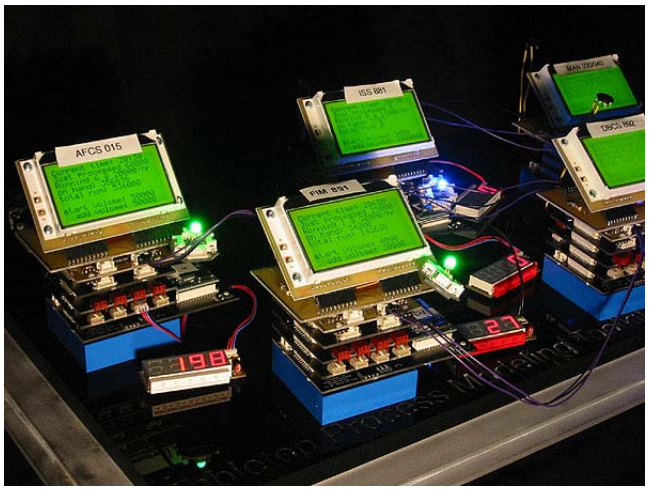

Figure 1: Second-generation prototype of Tabletop Process Modeling Toolkit model of a USPS mailflow

CHALLENGES FACING THE POSTAL SERVICE
The Postal Service manages 38,000 post office locations, 700,000 employees, and dozens of processing facilities. Each facility contains as many as a hundred mail-processing machines. Systems modeling plays a major role in the management of the postal service on every level; however, the effectiveness of these simulations is limited due to the challenges in communicating their results to the managers who need to utilize these results in real-time decision-making. There is a great deal of anecdotal evidence from engineers and managers within USPS that managers find the results of these simulations counter-intuitive and untrustworthy, and, as a result, often ignore them entirely.

CSCW'02, November 16–20, 2002, New Orleans, Louisiana, USA.

It is important to note that the problem is not with the existing simulation tools' technical merits or fidelity in capturing the subtleties of the situations they are used to model but rather the accessibility of their results to decision makers. At the root of the problem, the interaction between the engineering team and the management team often consists largely of opaque written reports because the simulations models used are too abstract and inaccessible for non-experts to understand or manipulate.

OUR APPROACH

In order to overcome the difficulties in allowing non-experts to develop their intuition of such complex simulations, we are guided by the constructionist theory, which asserts that individuals learn most effectively when constructing artifacts in the domain under study. [7] Thus, we have focused on developing a toolkit to allow managers to be an integral part of designing and manipulating these dynamic simulations. This has the potential to overcome the disconnect between the abstract specifications on which engineers base these simulations and the concrete experiences and tacit knowledge of the managers for whom the results of these simulations are intended.

Another dimension of the problem is that most of the current simulation packages are based on advanced mathematical representations and are therefore inaccessible to most non-engineers. We overcome this limitation by developing programming languages and design environments that are based on different ways of thinking about simulations. We aim to take advantage of intuitive representations that are grounded in the practices of the managers in their daily work. This involves a direct mapping of the computational elements in the simulations to objects in the process under study.

It is important to keep in mind that supporting alternative representational choices does not guarantee that non-experts will be able to transparently understand and manipulate the relevant parameters in the simulations. To address this issue, we insist that these simulations not be "black-boxed" and can be opened up so that their inner mechanisms can be examined and tinkered with in easily accessible interfaces and programming languages.

Another drawback of many existing simulation packages is that they are not well suited to collaboration among a group of engineers and/or managers. The use of a single computer screen, keyboard and mouse makes it difficult for multiple users to engage with an onscreen model. Our approach to overcoming this issue is to provide physical representations of the objects in a simulation so that a group of people can gather around a tabletop to collaboratively construct and manipulate dynamic simulations.

A CONCRETE CASE: *Modeling the flow of mail at a United States Postal Service Facility*

Individuals at various levels of the Postal Service have suggested a variety of potential uses for this approach in their organizations. In order to ground our work in a specific scenario, we have chosen to focus on the problem of modeling the flow of mail at a USPS processing facility, such as the one at Fort Point Channel in Boston. This has allowed us to tap a wealth of local expertise and interact directly with those facing this problem on a daily basis.

The Problem Statement

Analyzing the flow of mail in the Postal Service is very challenging due to the varying ways that mail moves through a processing plant. Different types of mail (e.g. advertising mail, first class mail, parcels, periodicals, etc.) have different delivery standards, and thus these different types of mail have separate streams within a facility.

All of these mailflows, which move through automation and mechanization within a postal facility, create a complicated web of physical and information flows.

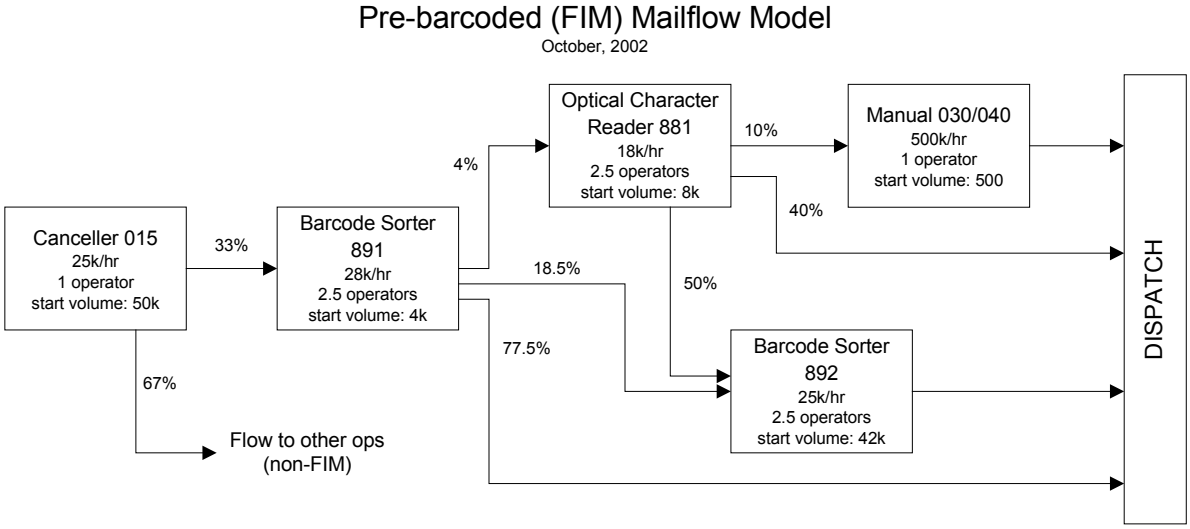

Figure 2: United States Postal Service Mailflow Model

The Existing USPS Simulation Model

The existing simulation model used in the Postal Service requires a week of training to learn to use and requires someone who already has a very good understanding of mailflow. The user enters volumes of each mail type, operation numbers, arrival profiles, maintenance schedules, percentages of mail that flow from one operation to another, and so on. The simulation produces equipment requirements in order to process the associated volumes of mail. The user must trust the simulator since it produces only the final results in table format, without any way for a user to build his intuition about the underlying model. Additionally the tool does not calculate people requirements (staffing), let alone attempt to optimize staffing according to varying workloads.

The Prototype

The flowchart in Figure 2 was the basis of our prototype that only dealt with one mail type, pre-barcoded letters. This mail type is referred to as FIM (Facing Identification Mark). An actual volume arrival profile for FIM mail was used based on a high volume Monday in the Boston Processing and Distribution Center in South Station.

Figure 3: USPS Engineer Benny Penta (seated) discusses the model with a colleague at the Fort Point Channel USPS Facility in Boston.

Each operation that FIM mail might flow to, depending upon the letter's address, was designated as a node in the system and the associated characteristics were assigned to it. A separate embedded computer system, called a Tower, represented each node, and these nodes were connected with wires for communication between them. Each Tower transferred the mail flowing from one operation to another and statistics were stored at each node. The characteristics were treated as parameters (operational throughput, operational staffing, threshold on-hand volumes etc.) that could be varied by the user in a hands-on manner while the simulation was running.

The result of each simulation run produces the work-hours and start/end times of each operation. This will allow the managers of a particular operation to staff each operation.

Our evaluation of this prototype has been largely informal to date. We have demonstrated this system to many engineers and managers from the United States Postal Service and other sponsoring companies of the MIT Media Lab. Many of these visitors quickly pointed out that the simulation capabilities of our prototype were limited compared to the software packages used in their organizations, but many were also extremely enthusiastic about this approach's possibilities for improving internal communication and involving non-technical members of their organization in designing and exploring simulations used in their fields. Their feedback has been invaluable for developing our ideas and focusing our work.

TECHNOLOGICAL INFRASTRUCTURE

Over the past year, we have developed two iterations of our prototype for the Tabletop Process Modeling Toolkit.

Initial Prototype: *The Cricket System*

The initial prototype of the Tabletop Process Modeling System, seen in Figure 4, was constructed using the Cricket system, utilizing several Cricket bus devices, including the tricolor LED devices and LCD display devices, to display relevant information about the simulation's state. [4] We achieved communication between Crickets using pairs of infrared communication bus devices, though this method severely limited the amount of data that we could reliably send between Crickets in the system. This model was based on a similar FIM model as the current system, but did not allow the user to interact with the nodes in the system to adjust parameters. Despite its limitations, this prototype served well as a means of obtaining feedback on our approach from a variety of visitors to the Media Lab.

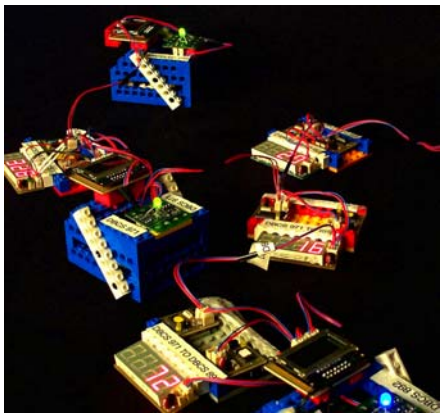

Figure 4: Initial Prototype of Tabletop Process Modeling Toolkit

Second-generation Prototype: *The Tower, a Modular Computation System*

Our current prototype of the Tabletop Process Modeling System, shown in Figure 1, was constructed by extending a modular computational prototyping system developed by our research group at the MIT Media Lab called the Tower. [5] The Tower consists of a set of Foundations with increasing computational capabilities, the first of which was based on a Microchip PIC microcontroller. Connectors on each Foundation provide the physical and electrical architecture for stacking layers to add functionality to the system, including data storage, infrared and RF communication, sensors, displays, and other layers. For this project we developed both a new, more powerful Foundation based on the Rabbit Semiconductor RCM2300 processor module and a new layer that uses serial ports and cables to communicate with other Towers in the simulation.

Each Foundation runs a compact interpreter for the Logo language, which enables novices to quickly write programs to control the capabilities of the Foundation and attached layers. These Logo programs, written on a desktop computer, are downloaded to the foundation via a serial cable. We have developed Logo libraries to handle packet communication and routing between Towers in the tabletop network, providing a robust communication infrastructure upon which users can rapidly build their simulations.

Our growing collection of Tower layers enables us to easily add a variety of capabilities to these tabletop models. We use the sensor layer to add buttons, sliders, and dials to control the simulation's parameters. LCD display layers and tri-color LED boards allow us to display relevant information about the state of the simulation on each Tower, providing both detailed information and important visual cues about the simulation as a whole.

Related Work

Approaches to enabling novices to build intuition of dynamic systems have been the subject of intense work. Forrester has stressed the importance of immersive explorations as a means to developing understanding such systems [2], and his System Dynamics in Education project at MIT has produced *Road Maps*, a series of exercises using STELLA software to build understanding of this field. [6] Resnick developed *StarLogo* for creating and exploring complex systems, using a massively parallel variant of the Logo language to program vast numbers of individual "turtles" and their shared environment. [10] Patten et al. have developed *Sensetable,* upon which they projected an on-screen representation of a dynamic simulation and manipulated parameters of the model by moving physical objects. [8]

DEMONSTRATION

At CSCW 2002, we will demonstrate and reflect on the application developed in collaboration with the USPS. We will discuss the architecture of the prototyping toolkit that we developed to construct this demonstration and a much wider range of applications that this system can support, including workflow modeling, tabletop network modeling, and system dynamics in general.

ACKNOWLEDGMENTS

We thank Chris Lyon and Brian Silverman for their continued contribution to the Tower System. We also thank the management staff of the USPS facility at Fort Point Channel, Boston for their valuable feedback, and especially USPS Engineers Benny Penta and Greg Love for devoting their time to guiding the further development of this toolkit.

REFERENCES

1. Forrester, J. (1961). Industrial Dynamics. MIT Press. Cambridge, MA.
2. Forrester, J. (1994). Learning Through System Dynamics as Preparation for the 21st Century. <http://sysdyn.mit.edu/sdep/papers/D-4434-1.pdf>
3. Horgen, T., Joroff, M., Porter, W., Schön, D. (1999). Excellence by Design. New York, NY: John Wiley & Sons, Inc.
4. Martin, F., Mikhak, B., Silverman, B. (2000). MetaCricket: A designer's kit for making computational devices. *IBM Systems Journal* (Vol. 39, Nos. 3 & 4)
5. Mikhak, B., Lyon, C., Gorton, T. (2002). The Tower System: a Toolkit for Prototyping Tangible User Interfaces. Submitted as a long paper in CHI 2003.
6. MIT System Dynamics In Education Project. Road Maps Project. <http://sysdyn.mit.edu/road-maps/>
7. Papert, S. (1991). Situating constructionism. In Papert & Harel, (eds.), Constructionism. Cambridge, MA: MIT Press and other papers therein.
8. Patten, J., Ishii, H., Hines, J., Pangaro, G. (2001). Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. Proceedings of CHI '01. ACM Press, pp 253-260.
9. Perkins, D., Grotzer, T. Models and Moves: Focusing on Dimensions of Causal Complexity to Achieve Deeper Scientific Understanding. AERA Symposium on Complex Causality and Conceptual Change, April, 2000.
10. Resnick, M. (1994). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. Cambridge, MA: MIT Press.
11. Senge, P. (1990). The Fifth Discipline: The Art and Practice of the Learning Organization. New York, NY: Currency/Doubleday.
12. Wilensky, U. & Resnick, M. (1999). Thinking in Levels: A Dynamic Systems Perspective to Making Sense of the World. Journal of Science Education and Technology. Vol. 8 No. 1.

9.7 Dekoli & Gorton 6.829 Paper

Tim Gorton and Margarita Dekoli submitted the following paper as the final project in the fall 2002 term of MIT course 6.829, "Computer Networking." More information about this work and its relation to this thesis may be found in Section 6.1.

Modeling a Fault-Tolerant Routing Algorithm for a Modular Hypercube Packet Switch Fabric

Margarita Dekoli, Tim Gorton

{dekoli, tgorton}@mit.edu

Abstract

This paper presents the design of a modular packet switch fabric in a hypercube topology. We have constructed a working model of this switch fabric using embedded electronic systems and used this model to measure the design's performance and resilience to link and node failures.

Keywords: Hypercube topology, fault-tolerant routing, packet switch, switch fabric.

Introduction

This paper presents the design of a modular network packet switch fabric based on a hypercube topology, similar to the topology described in [9], and a fault tolerant routing strategy within the hypercube switching fabric. We have built a slower version of this switching fabric using an embedded prototyping system called the Tower [10,] and we have implemented and evaluated strategies for fault tolerance, accommodating failure of links and nodes and automatically recovering as they are restored.

Hypercube topologies have been used in massively parallel processing machines for interprocessor communication [14, 5] because of their scaling properties and the ease of implementing self-routing by viewing each node as a binary number such that nodes connected by an edge differ in exactly one bit of their addresses. Routing can be then accomplished easily on each intermediary node by forwarding a message along any link that will fix one bit of the difference of the current node's address and the target node. Variations on this scheme have been proposed for interprocessor routing to provide various degrees of fault tolerance. [5, 12] Variants of the hypercube topology have also been proposed to increase connectivity and decrease their radius in order to provide more efficient routing between nodes. [12]

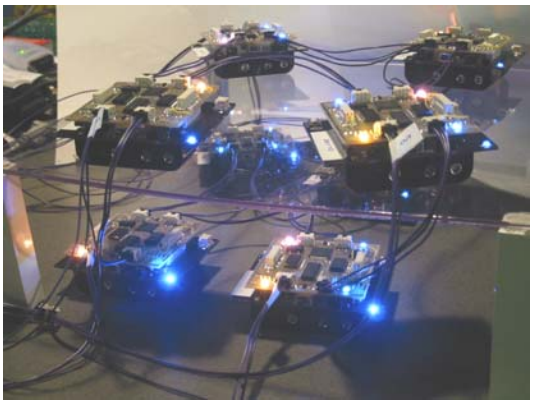

Figure 1: The 8-node hypercube packet switch fabric model using embedded electronics

These properties have also been employed to build ATM switches based on a hypercube topology. [9] and [11] provide different examples of connecting inputs and outputs to the hypercube as well as different ideas about using buffering in the internal nodes. [12] builds on these examples to design three different ATM switches based on variants of a modified hypercube called a folded hypercube.

Our model of this switch fabric, shown in Figure 1, is a functional version of the switch, if downsized in terms of speed, memory and number of nodes. This allows us to explore fault-tolerant routing behaviors while allowing our results to scale to real-life network switches. This approach to modeling could also allow future work in exploring different topologies and buffering strategies inside the fabric.

Related work

Since our model is comprised of hardware components that can be connected in arbitrary topologies, a great deal of consideration is due to the existing designs for a switch fabric. Our approach focuses on space-division switching fabrics, the simplest of which is the crossbar, in which all the inputs to the switch are mapped to the outputs through a grid of connections, forming crosspoints at each intersection. There

are a number of disadvantages in this configuration, most importantly the number of high number of crosspoints for a large switch, and also the fact that if one crosspoint fails, the connection between an input and an output becomes permanently isolated. [8]

These issues have led to the creation of more complex configurations of fabric switches with multiple stages of nodes that route traffic along the fabric in a more efficient way. Early work in the telephone industry enabled the design of larger non-blocking crossbar-like fabrics from smaller crossbars. [1, 3]

Another interesting class of switch fabrics are those which use small elements to enable "self-routing" of packets to destination ports. In this scheme, the switch marks packets as they enter the switch with the destination port, and the intermediate fabric elements route the packet to the destination according to that address. The classic self-routing fabric is the Bayan switch fabric, which consists of simple four-port elements that simply pass packets arriving at their inputs to one of their outputs depending on a particular bit of the destination address. [8] It is interesting to note that in the Bayan network the topology is critically important in order to ensure correct routing.

Unfortunately, this network has serious issues with blocking if two packets destined for different ports have to cross the same internal link. Switches using Bayan networks typically require a sorting network and other infrastructure to reorder packet arrivals in order to avoid blocking. Another approach is to introduce buffering at internal nodes, though this only partially solves the utilization problem and can dramatically increase the size and cost of a complex switch. [8] Depending on small amounts of internal buffering can also increase the likelihood of unnecessarily dropping packets in the case of traffic bursts.

In addition to the issue of buffering, we must also consider the topology of the nodes that make up the switching fabric. This project primarily focuses on the topologies and strategies for fault-tolerance in such a switch, characteristics mostly of hypercube topologies, which is the topology that we have chosen for this project. The hypercube has a number of useful properties that will assist in the design of fault-tolerant routing algorithms. [6] These properties include its symmetry, distributed and regular structure, and relatively small diameter.

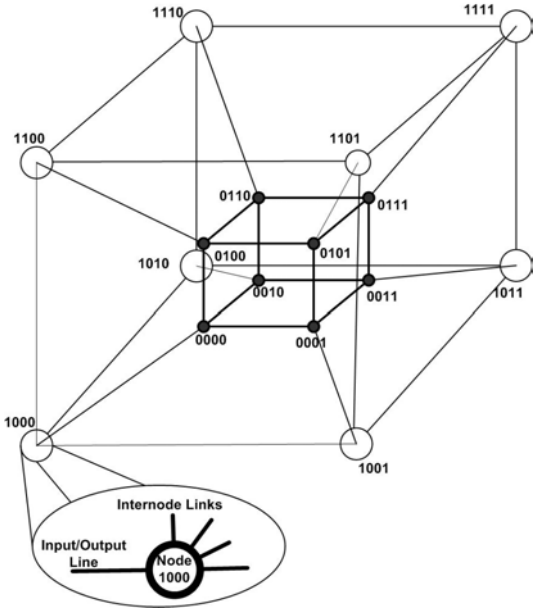

Figure 2: The binary addresses of nodes in the hypercube

To build upon the self-routing properties of the hypercube, research has been conducted to improve the fault-tolerance of hypercube networks [6], work that is tied directly to the strategies implemented in our model.

One of the features of this topology that helps in the presence of errors is the redundancy that exists in the connections between nodes, i.e. if a node fails the shortest path to a target node may not be valid but a longer path could be found from the surrounding nodes. In order for the routing protocol to do so, each node would need to store some information about its neighbors and refresh it periodically. [6]

The classic hypercube topology can only accommodate an increase (or decrease) of nodes in the system by a factor of 2, which has an impact on the scaling of the network both in terms of number of nodes to add (or remove) and also on the complexity of the resulting topologies and connections between nodes. Future work on different topologies using our model might be informed by variants of the hypercube topology that have other factors of scaling [5] For our present purposes, we will limit ourselves to the classic hypercube.

Another noteworthy variant of the hypercube architecture is the division between the data path and the control information path along the same hypercube topology. [9] This can allow multiple data rates and link technologies to serve the data

packets and the much smaller control information. A related strategy which we will explore uses rapidly-moving control packets to find and reserve a path from the input node to the output without wasting resources on moving the packet itself down fruitless paths. [8]

For measuring the efficiency of our model switch, we refer to some of the metrics that are proposed for switch fabrics including the actual size of the switch, the latency and the throughput [2] but are essentially focusing on the aims of fast recovery from a fault either in a node or a connection in the network and the correctness of the routing algorithm during such failures.

Technology

Our model of the hypercube switching fabric consists of a number of small computer modules, each containing four serial ports for internal connections to other nodes. A microcontroller will perform the routing functionality of each node. The remainder of this section describes the prototyping system that we are using to construct our model, the Tower.

The Tower

The Tower is a modular electronics design system created by the Grassroots Invention Group at the MIT Media Lab to allow easy snap-together design of complex electronics systems. [10] The Tower is comprised of a Foundation module with the core processor and other boards that stack on top of it (Figure 3), providing a wide range of functionality including sensing, actuation, data storage, and communication. Power is provided by 4 AA-batteries or a wall adaptor.

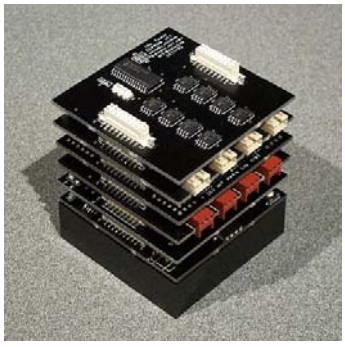

Figure 3: The Tower System

The PIC microcontroller at the core of the Tower Foundation may be programmed in assembly language or by using a compact virtual machine, which executes a variant of the Logo language.

The Serial Layer

For this project, we are using the serial board for the Tower on a PIC Foundation. This layer was created to form a network by connecting a number of Towers together on a tabletop in arbitrary topologies to allow them to exchange information.

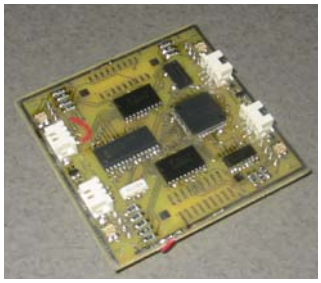

Figure 4: The Serial Layer

Each serial layer contains 4 buffered serial ports. The serial layers on different Towers are connected together in arbitrary topologies using three-wire cables carrying transmit, receive and ground signals. The layer also contains a "tricolor" LED next to each port that flashes when data is transmitted through the port to help visualize when data flows out trough the port.

A PIC microcontroller on the layer uses a parallel interface to control the Texas Instruments chip containing the four serial ports and the tricolor LEDs. The Foundation controls the serial layer through an industry-standard interprocessor bus protocol.

Architecture

Based on the hypercube topology that is illustrated in Figure 2, we implemented a 3-dimensional hypercube topology for our switch fabric using 8 Tower systems as nodes in a cube, as shown in Figure 5.

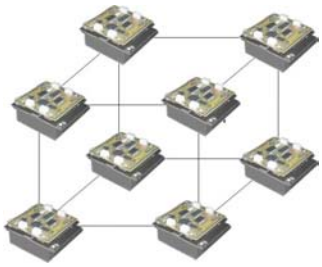

Figure 5: Cube implemented with Towers

Each node was implemented as one Tower system with one of the serial boards that we describe above. The connections of the cube were made by connecting the serial ports of each

node with its neighbors, as illustrated in Figure 5. Specifically, each node has 3 ports that are used for internode communication, and one is used for I/O communication, as shown in Figure 6. The one remaining serial port can be used in an implementation of the fabric switch in a 4-D hypercube topology where four serial ports are needed for the internode communication.

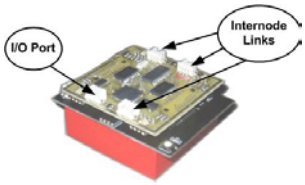

Figure 6: Hypercube node and its serial ports: 3 used for internode communication and 1 for I/O

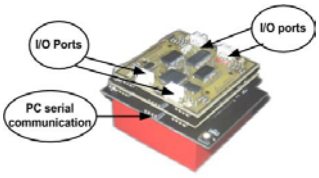

Figure 7: Traffic generator serial ports: 8 used for I/O hypercube communication and 1 for serial communication with the computer

For our simulations we used a similar configuration of the Tower system as a traffic generator, shown in Figure 7, to create the traffic for our switch fabric. We added another serial layer to this Tower system in order to have 8 serial ports for each of the nodes in the hypercube and used the extra one for the serial communication with the PC. The link between the PC and the traffic generator is used to allow our PC software to specify the packets to be generated and to receive the results.

The hypercube is a distributed system where all the nodes are equivalent in terms of functionality and the code they run. The only difference between them is their individual node address.

The hardware, as described in previous sections, has limited capabilities since it is based on a PIC microprocessor and has no buffer available to hold even a single ethernet packet. Thus each node acts as a crossbar to connect its ingresses and egresses once a routing path is established using small control packets under the microcontroller's control. This architecture is shown in Figure 9.

The core model developed here is the switching fabric in the form of a cube. If these hypercube nodes were connected to buffers on the ingress and egress sides, that pair would constitute the equivalent of a line card in a router, as seen in Figure 8.

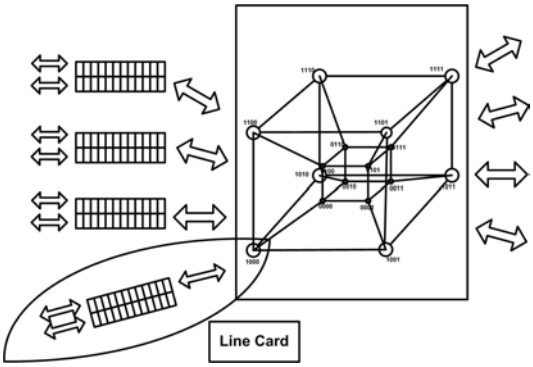

Figure 8: Router line card composed of a node and buffers

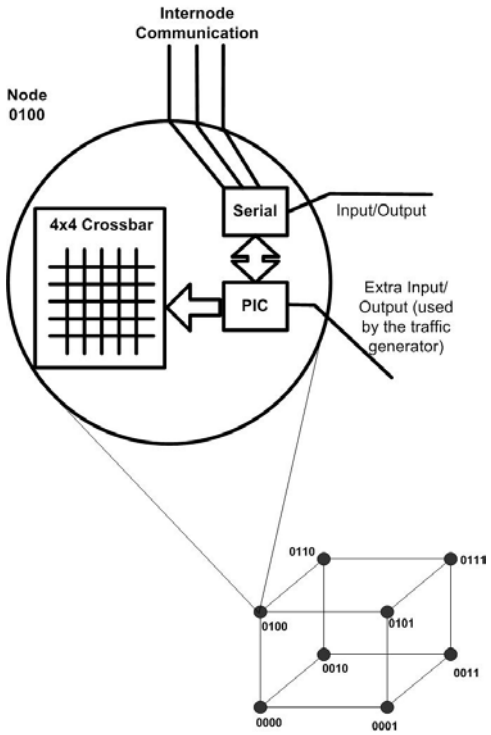

Figure 9: Example node 0100: Each node implements a 4x4 crossbar on the microprocessor that will forward the bytes it receives from the ingress to the egress as quickly as possible.

Routing protocol implementation

Since our nodes cannot buffer packets in transit, our routing protocol is using a path-claiming phase to reserve a route for the packet to follow. The path-claiming phase is performed by control packets that are propagated through the switch fabric to reach the destination node. If all the nodes in between can successfully reserve their ports for this communication to take place, then the destination replies with a packet signaling a successful path reservation. If the reply is a

failure, or no reply at all, the sender may assume that the route was not reserved for the data packet. The control packets are routed the same way as the flow of data through the fabric switch. That is, each link traversed corrects one bit of the address, and the bit to correct is randomly chosen at each node.

Each node is able to determine where it can forward packets since it knows the addresses of its neighboring nodes. This information is regularly broadcasted from every node to all its neighbors through KEEP_ALIVE messages.

All the nodes keep a small amount of state in order to keep track of the mapping between the ingress and egress ports when a path has already been established. With that information they are able to emulate the 4x4 crossbar for forwarding the packets. We model this crossbar using the microcontroller to copy bytes from the ingress to the egress, but a real crossbar implementation would be limited only by the speed of the internal links and external packet buffers. Thus, our model deals mainly with the path-claiming phase since the fabric throughput is limited by the link utilization that results from the path-claiming phase.

Our case study will be to investigate the fault tolerance of our switch fabric during the path-claiming phase while individual connections and nodes are failing. If a failure happens before a given path-claiming phase then the switch fabric should be able to route around the failure. If the failure happens during or after the path-claiming phase, packets in flight will be dropped. During our simulations and analysis, we will observe the packets that are being sent successfully through our switch fabric and the percentage of successful attempts for a path reservation.

The implementation of our model is shown in Figure 10. The 8 nodes implement the switch fabric and the tower on the left is the traffic generator. The blue lights on each node designate KEEP_ALIVE messages sent periodically to their neighbors and the yellow lights designate reserved node egresses.

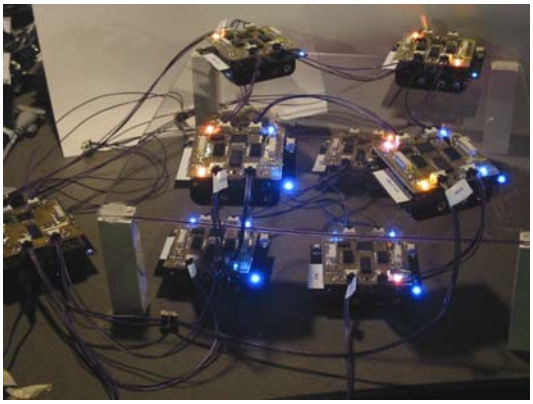

Figure 10: The 3-D hypercube packet switch fabric model & traffic generator

Benefits and challenges of this modeling approach

Since software models are widely used to study the behavior of new routing protocols, it is worth explaining our motivations for building a physical model and the challenges that this entailed. Our approach is based on the fact that this is not a simplified model of a real switch fabric; rather it is the equivalent of a real fabric in a very different scale of speed and size. This may provide a unique environment for rapidly evaluating new ideas and protocols.

The implementation of this protocol in the same sort of distributed fashion as a real system may provide for a higher fidelity of simulation as well. For example, the difference in the time for control packets to travel links and be processed by nodes may lead different paths to be reserved. Without data from a software simulation and a full-scale hardware switch, it is difficult to determine whether these issues are significant.

We also hope that this model can provide a more transparent way for laymen and system designers alike to develop intuitions about how routing protocols function in this sort of distributed system. Using tangible models and interfaces as a means to understanding dynamic systems in this way is an active research topic in the computer-human interaction research community. [13,4]

The challenges in creating such a model are quite different from creating a computer-based simulation. Instead of being able to allocate “infinite” resources to a simulation this physical model has very limited memory, actual serial ports and requires the implementer to handle the

system's physical resources on a more low-level, closer to the hardware of a switch.

Testing conditions and Results

We evaluate the performance of our model by collecting data about the results of the path-claiming phase. We evaluate the fault tolerance of our switch fabric by inducing various failures during our simulations, including node and internode link failures. During these tests, a PC issued commands to the traffic generator Tower, creating packets with a random destination. If the path reservation failed, that destination was retried twice before the packet was abandoned.

Normal operation

We present the normal operation of our model through a number of graphs that show aggregated data over 200 iterations of the routing protocol at each traffic load. Specifically, we are evaluating the packets that each node attempted to send and the number of packets that were actually sent successfully. We also evaluate the number of successful path reservations and the overall number of path reservation attempts. As mentioned earlier, we collect our data with an increasing percentage of probability of a packet arriving at each node's ingress, varying from 20% up to 100%.

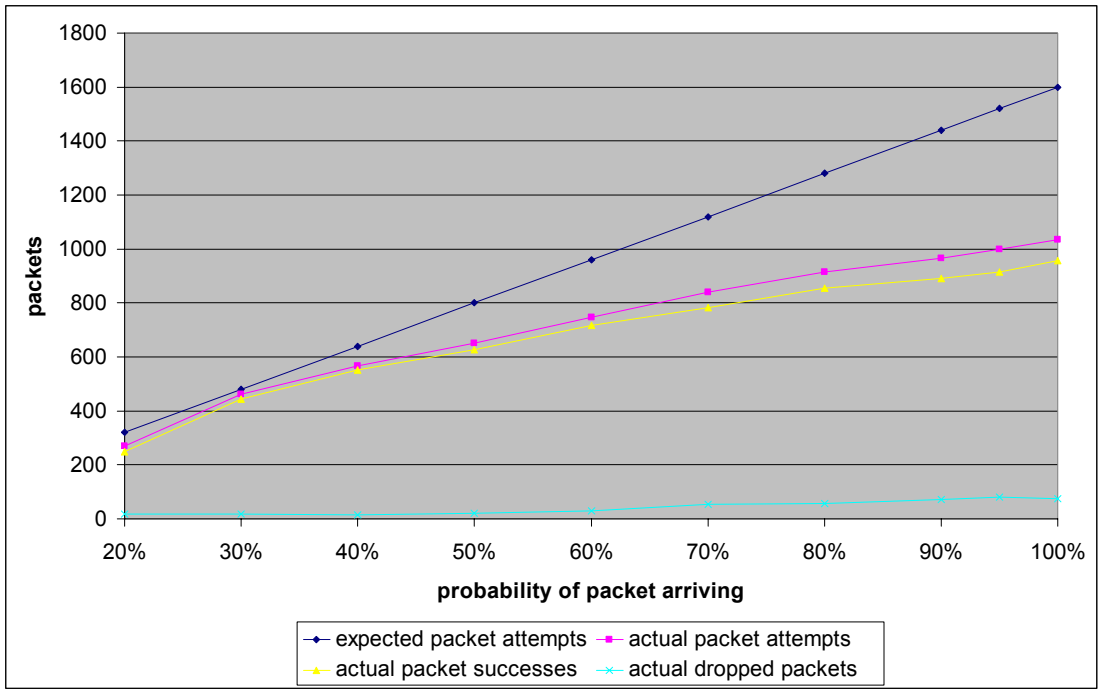

Figure 11:Combined view of our 4 measurements for increasing traffic in the switch fabric

Our data regarding the normal operation of the fabric will be aggregated over 200 path reservation tries for different sets of packet arriving probabilities per node. That is, the percentages represent the probability of a new packet arriving at an idle ingress link. It is difficult to isolate the performance of the switch fabric from the buffering strategy used, and this metric does not assume any particular buffering strategy but may differ from actual operation as a result. This issue remains to be explored, as will be discussed later.

In Figure 11, we observed the difference between the expected number of attempts to send packets through the switch fabric (dark blue line) and the number of actual packet attempts (in pink), which follows the first one more closely when the overall traffic through the switch fabric is lower (20-30% probability of packet arrival). As the switch fabric becomes more loaded, the slope of the attempted packets decreases as a result of retries at the ingress node. The packet successes (yellow line) follow closely the numbers of the attempted packets, but as the traffic increases a few more packets are being dropped. Overall it appears that a speedup of two would be more than sufficient to ensure that all packets are delivered to their egresses at line

speed, since the actual traffic at continuous arrivals over 900 packets is more than half the optimal value of 1600 for this trial.

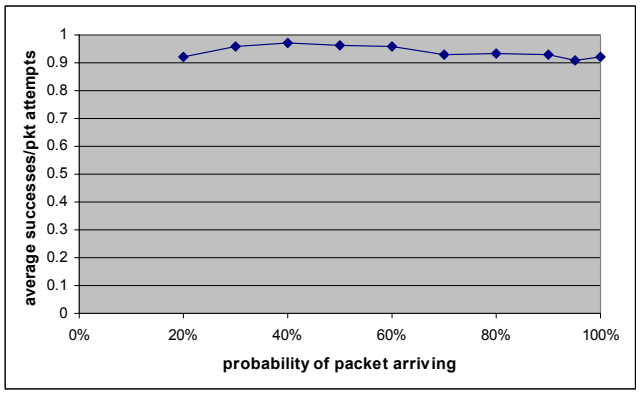

Figure 12: **Packets successfully transferred per packet attempted**

As seen in Figure 12, the average path reservation success per packet attempted remains roughly the same. The success rate is better when the traffic through the switch runs at 30-60% of its capacity and it has a slight drop at 70-100%.

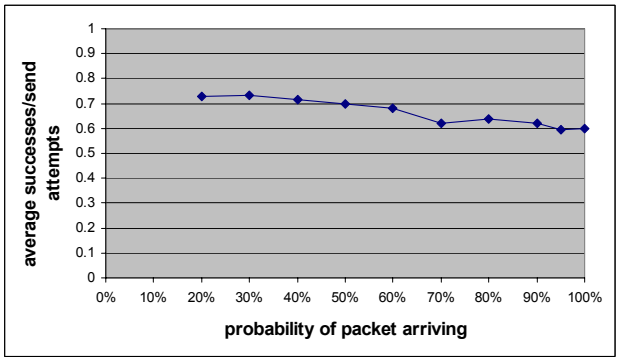

Figure 13: Successful path reservations per reservation attempt

The graph in Figure 13 shows the same success rate over the overall attempts to send packets to the destination. The difference observed here is due to the fact that it might take multiple attempts to successfully send a packet to the destination (up to three attempts per packet). The average number of tries per packet varies between 1.3 and 1.6 as shown in Figure 14.

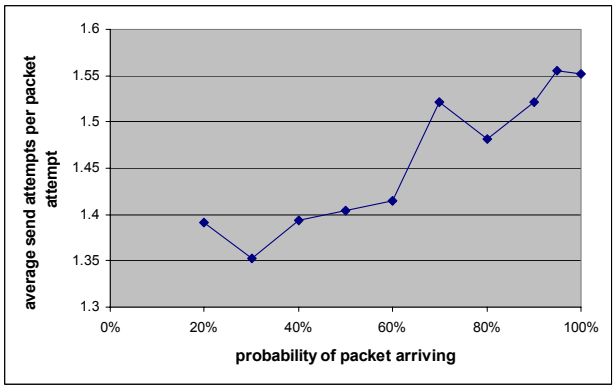

Figure 14: Average number of retries to send or abandon a packet

In the graphs presented thus far it is difficult to understand why the success rate doesn't drop significantly as the traffic through the switch fabric increases to heavy load. In fact, we observe a slight increase in the successful path reservations from 95% to 100%. To understand this behavior of our model better, we present a different analysis of our data and we observe the successful path reservations per packet attempt for destinations that are one, two, and three hops away from the sender (Figure 15).

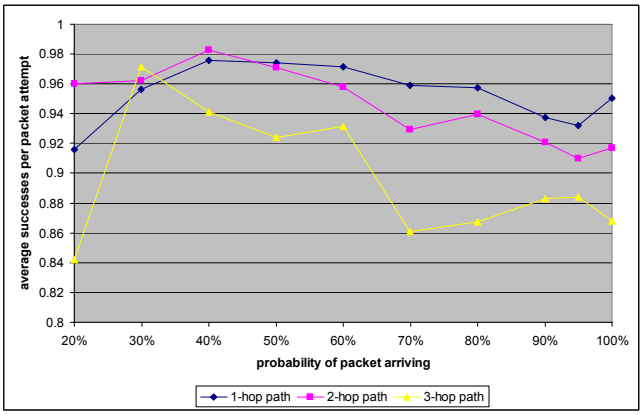

Figure 15: Successful path reservations to targets one, two and three hops away from the destination

This shows that as the traffic through the switch fabric increases, it is more successful in reserving paths one hop away, and the success rate decreases when the destination is two hops away and even more during a 3-hop distance. This behavior is probably the reason for the overall success rate increase at 100% probability, since the longer routes that fail leave links to be used by the shorter route requests. The data displayed of the left of the 50% case represents only a few tens of packets being exchanged

through the switch fabric per trial for longer paths and therefore shows a lot of variance.

Failure conditions

To test the behavior of the switch fabric in the presence of failures, we increased the total cycles to 1000 (rather than 200) in order to arrive to more statistically valid conclusions. We hold the arrival probability per node stable at 80% and we try four different cases:

1) the normal operation of the switch fabric where all the nodes are functional and connected properly to each other,
2) the experiment under the same conditions except a broken link between nodes 4 and 5,
3) the same experiment as the first one, except node number 5 has failed, and
4) the same experiment as the first one, except nodes 4 and 5 have failed.

- The first (blue) column is the overall successful path reservations under normal conditions.
- The second (dark red) column represents the successfully reserved routes during normal operation between pairs of nodes that were reachable during the failure. For example, the difference between this column and the blue one in the first column grouping represents all the successful path reservations between all the nodes except nodes 4 and 5.
- The third column (yellow) represents the number of successfully reserved routes during failure. Therefore, the difference between the second and third column represents the losses of our switch fabric during the various failures.

The crucial information from this data is the number of packets, which could have been delivered but were not during the failure,

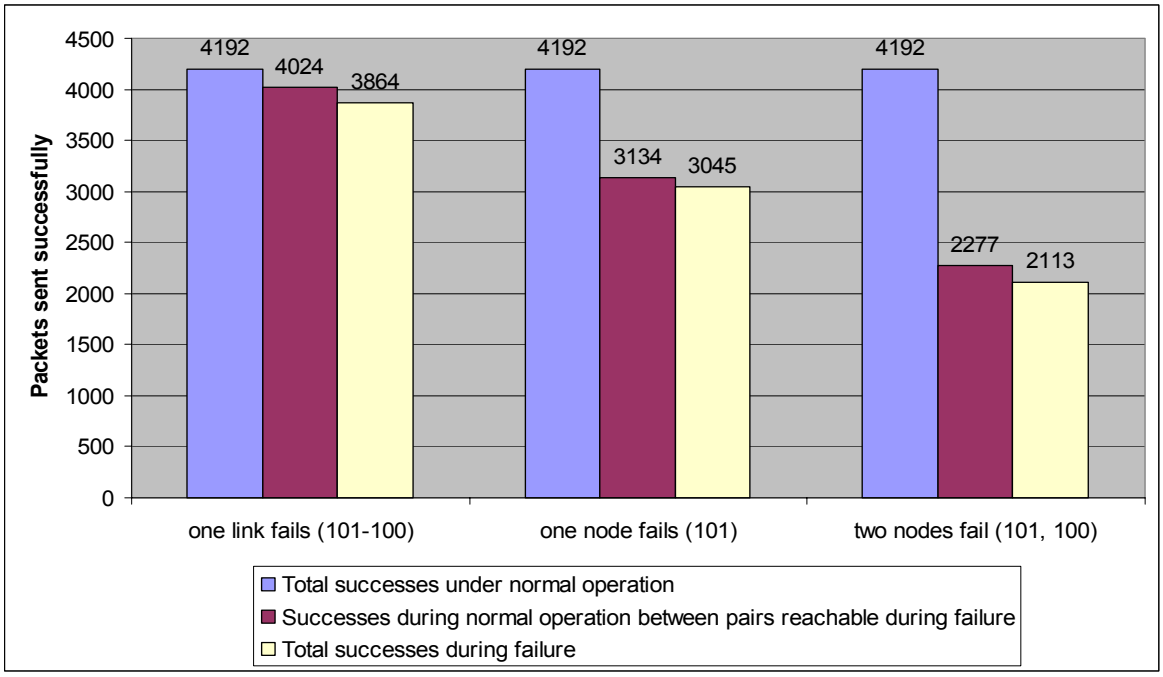

Figure 15: Performance during failure

The comparative results of the switch fabric's performance in the three types of failures are illustrated in Figure 16. The three sets of columns represent the failure conditions (i.e., one link fails, one node fails, 2 nodes fail). The three individual columns in each grouping represent the following:

compared to those input-output pairs' performance under normal operation. This is basically the difference between columns two and three of the Figure 16, since that is the amount of successful results under normal conditions that cannot be repeated under failures. The paths reservation percentage has decreased in the case of the failing link (no. 2) by 4%, in

the case of one failing node (no. 3) by 3%, and in the case of two failing nodes (no. 4) by 7%. These percentages show the very small drop in performance of our switch fabric under failures. Though the fabric experiences some added contention for links when links or nodes fail, our graphs also indicate a decreased overall traffic flow during node failures, which would cause lower overall utilization, providing extra resources to aid in routing despite the failure. Our data indicates that this allows the fabric to continue routing almost as efficiently as before the failure.

Future Work

A great deal of future work remains to be done in evaluating routing algorithms for modular hypercube packet switches using our modeling technique. Our current algorithm can be further explored by providing bursty traffic or traffic that does not use all of the inputs and outputs uniformly, to more accurately simulate a variety of network configurations and uses. Different buffering schemes should also be studied to examine their effect on the efficiency of the fabric. Also, as described earlier, software simulations and data from a full-scale hardware implementation would provide valuable validations for our model.

Other explorations might be to experiment with different topologies by expanding the model to four dimensions and changing the way in which inputs and outputs are connected to the fabric. For example, [11] uses a topology in which half of the nodes are connected to ingresses and the other half are connected to egresses.

Our model has a minimal amount of buffering on each node that is used by the routing protocol for claiming paths. By adding substantial buffering at these internal nodes different routing strategies could be explored and evaluated against our results.

Other routing strategies could also be explored, including performing multiple retries during the path-claiming phase to increase the success rate or allowing route reservation packets to be redirected when no path exists at a node to lead the packet towards the target. Packets to different destinations could also be tried instead when a path reservation fails in an attempt to better utilize the fabric when a path to an output is blocked. Alternatively the routing could be performed without a path-claiming phase at all, simply pushing packets through the fabric. This could provide interesting opportunities to explore large, clockless configurations.

Conclusion

Our model has demonstrated the correctness of our routing protocol, as well as provided a platform for us to test its performance and fault tolerance. The tests we performed show that our routing algorithm performs reasonably well under uniform load, requiring a 2x speedup to perform at line speed. These tests have also demonstrated the resilience of the algorithm to link and node failures, showing that only a minor decrease in other traffic is caused by the failure of a link or node. Furthermore, because the link utilization is likely to be the most important factor in fabric throughput, these results are expected to scale to more realistic link and processor speeds. A wide range of work remains to be done, however, to fully evaluate our algorithm and analyze other possibilities using our modeling approach.

Acknowledgements

We would like to acknowledge Hari Balakrishnan for his guidance during this project. We also thank Chris Lyon and Bakhtiar Mikhak of the MIT Media Lab's Grassroots Invention Group for their continuing work on the Tower system.

References

[1] Cantor, D.G. (1971) "On Non-blocking Switching Networks", Networks, 1 367-377.

[2] Cisco, (2002). White Paper on: "Catalyst 8500 CSR Architecture", <http://www.cisco.com> referenced on 11/14/02.

[3] Clos, C. (1953). 'A study of non-blocking switching networks', Bell System Technical Journal, March 1953, pp. 406-424.

[4] Gorton, T., Mikhak, B. Paul, K. (2002). Tabletop Process Modeling Toolkit: A Case Study in Modeling USPS Mailflow. Demonstration at CSCW '02. <http:// gig.media.mit.edu/publications.html> Accessed 12/16/02.

[5] Hameenanttila, T., Guan, X.L., Carothers, J. D., Chen, J.X. (1996). The Flexible Hypercube: A New Fault-Tolerant Architecture for Parallel Computing. Journal

of Parallel and Distributed Computing 37, No 0122, pp. 213-220.

[6] Kaneko, K., Ito, H. (1999). Fault-Tolerant Routing Algorithms for Hypercube Networks. International Parallel and Distributed Processing Symposium, 1999.

[7] Kavipurapu, G.N., Nourani, M. (2001). Switch Fabric Design and Performance Evaluation: Metrics and Pitfalls. <http://www.iris-technologies.net> referenced on 11/14/02.

[8] Keshav, Srinivasan (1997). An Engineering Approach to Computer Networking, Chapter 8. Addison-Wesley Pub Co, New York.

[9] Matsunaga, T. (1993). Sorting-Based Routing Algorithms of a Photonic ATM Cell Switch: HiPower. IEEE Transactions on Communications, Vol 41, No.9, Sept. 1993.

[10] Mikhak, B., Lyon, C., Gorton, T. (2002). The Tower System: a Toolkit for Prototyping Tangible User Interfaces. Submitted to CHI 2003. <http:// gig.media.mit.edu/publications.html> Accessed 11/13/2002.

[11] Pao, D.C.W.; Chau, W.N. (1995). Design of ATM switch using hypercube with distributed shared input buffers and dedicated output buffers. IEEE 1995 International Conference on Network Protocols.

[12] Park, J.S. (2001). The Folded Hypercube ATM Switches. Ph.D. Thesis, Virginia Polytechnic Institute. September 18, 2001.

[13] Patten, J., Ishii, H., Hines, J., Pangaro, G. (2001). Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. Proceedings of CHI '01. ACM Press, pp 253-260.

[14] Squire, J.; Palais, S. M. (1963). Programming and design considerations of a highly parallel computer. Proc. AFIPS Spring Joint Computer Conference, Mar. 1963, pp. 395-440.

[15] Sullivan, H.; Bashkow, T.R. (1977). A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I. Proceedings of the 4th Annual Symposium on Computer Architecture, March 1977: pp 105-117.