

# Softwear: A Flexible Design Framework for Electronic Textile Systems

Christopher M. Zeh

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

Masters of Science  
in  
Computer Engineering

Dr. Mark T. Jones, Chair  
Dr. Thomas L. Martin  
Dr. Robert P. Broadwater

April 10, 2006  
Blacksburg, Virginia

Keywords: Electronic Textiles, Simulation Environment, Quality of Service, Service Hierarchy

Copyright 2006, Christopher M. Zeh

# *Softwear*: A Flexible Design Framework for Electronic Textile Systems

Christopher M. Zeh

(ABSTRACT)

Because of their ubiquity and low cost fabrication techniques, electronic textiles (e-textiles) are an excellent platform for pervasive computing. Many e-textile applications are already available in the commercial, military, and academic domains, but most are very highly specialized and do not lend themselves easily to reuse or alteration. The purpose of this work is threefold: development of a methodology for building flexible and reusable applications that facilitates their use in the evolution of more complex systems, creation of a resource manager that realizes the methodology and enforces quality of service guarantees on tightly constrained textile resources, and construction of a simulation environment that allows for the rapid development and reconfiguration of systems to circumvent the need for the expensive physical prototyping process. This work discusses the effectiveness and appropriateness of the deployed event-driven hierarchical service model for application development. Additionally, this work explores the results of providing fault tolerance and quality of service guarantees in a textile environment that is particularly susceptible to faults. Further addressed by this work is the success of rapid prototyping and evaluation of applications in the simulation environment.

## Acknowledgments

“We are like dwarfs sitting on the shoulders of giants. We see more, and things that are more distant, than they did, not because our sight is superior or because we are taller than they, but because they raise us up, and by their great stature add to ours” — John of Salisburg (1115–1180).

This work would not have been possible without the help and support of my friends, family, and professors, to which I cannot do justice in such a brief section. I will endeavor, then, to briefly acknowledge the major contributors. I would like to thank Drs. Jones and Martin for bringing me in on the project, for the knowledge I have obtained along the way, and for their guidance in the creation of this framework and thesis. My final committee member, Dr. Broadwater, was instrumental in forming my view of flexible software creation, without which this work would also not have been possible. I never would have finished this work without the assistance in system administration, particularly early on, from David Lehn and Joshua Edminson. Also of tremendous help was Braden Sawyer, with whom the early stages of this work was codeveloped before he moved on to other projects. Last but not least, I would not be here, much less have been able to accomplish what I have, without the love and support of my parents, Van and Connie, and my fiancé Danette.

This material is based upon work supported by the National Science Foundation under Grant No. CCR-0219809.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Thesis Organization . . . . .	3
<b>2</b>	<b>Related Research</b>	<b>3</b>
2.1	Previous E-Textile Research . . . . .	3
2.2	Prior E-Textile Research At Virginia Tech . . . . .	4
2.3	Relevant Programming Methodology Work . . . . .	6
<b>3</b>	<b>Characteristics of Electronic Textile Environments</b>	<b>7</b>
3.1	Challenges of E-Textiles . . . . .	7
3.2	E-Textile Architecture . . . . .	8
<b>4</b>	<b>An Event-Driven Service Hierarchy</b>	<b>11</b>
4.1	Event-Driven Service Model . . . . .	11
4.2	Service Hierarchy . . . . .	11
4.3	Compatibility With Modern Low-Power Processors . . . . .	12
<b>5</b>	<b><i>Software</i> Application Description Methodology</b>	<b>13</b>
5.1	Achieving Flexibility and Reusability . . . . .	13
5.2	Supporting Publish/Subscribe . . . . .	13
5.3	Sample Application Class . . . . .	14
<b>6</b>	<b><i>Software</i> Resource Manager</b>	<b>15</b>
6.1	Discrete Allocation of Resources for Quality of Service Guarantees . . . . .	15
6.2	Fault Tolerance Mechanisms . . . . .	15
6.3	Hierarchy of <i>Software</i> Resource Managers . . . . .	16
6.4	Lower-level <i>Software</i> Resource Managers . . . . .	16
6.5	Upper-level <i>Software</i> Resource Managers . . . . .	18

<b>7</b>	<b>The Simulation Environment</b>	<b>19</b>
7.1	Ptolemy II Discrete Event Environment . . . . .	19
7.2	Wrapping Ptolemy For Portability . . . . .	20
7.3	Ptolemy Infrastructure Components . . . . .	21
7.3.1	Bus Arbiter . . . . .	21
7.3.2	Client-To-Bus Interface actors . . . . .	21
7.3.3	Routers . . . . .	23
7.4	Using Motion Capture Data . . . . .	23
7.5	Using Topology Files For Rapid Reconfigurability . . . . .	25
7.6	Simulation of Faults . . . . .	26
<b>8</b>	<b>Results</b>	<b>26</b>
8.1	Mapping Services Among Nodes . . . . .	26
8.2	Specifying An Application . . . . .	28
8.3	Rapid Prototyping And Reconfigurability . . . . .	29
8.4	Quality Of Service Guarantees . . . . .	30
8.4.1	Response To Non-Ideal Sensor Behavior . . . . .	30
8.4.2	Adaptation To Network Faults . . . . .	32
8.4.3	Energy Management . . . . .	32
8.4.4	Priority Inversion . . . . .	35
8.5	Overhead Of Implementing <i>Softwear</i> . . . . .	35
<b>9</b>	<b>Conclusions and Future Work</b>	<b>36</b>
	<b>Appendix A Running And Extending The System</b>	<b>44</b>

## List of Figures

1	E-Textile Prototypes At Virginia Tech . . . . .	5
2	Several e-TAGs Shown Relative To A Penny . . . . .	10
3	Example Service Hierarchy . . . . .	12
4	Interaction Model Of Service Assignments In <i>Softwear</i> Hierarchy . . . . .	17
5	Example Of Breaking A Broadcast Routing Cycle . . . . .	19
6	Illustration Of The 5 Function JNI Interface . . . . .	20
7	Ptolemy Screenshot Of Activity Recognition Simulation With Four Routers .	22
8	Distance Vector Routing Scheme . . . . .	24
9	Layout Of The Tables In The MySQL Database . . . . .	25
10	Sample Topology File For Activity Recognition Simulation With Four Routers	27
11	Example Of Routing Adapting To A Fault . . . . .	33
12	Various Energy Source Specifications That Translate Into Energy Vs. Power	34

## List of Tables

1	<i>Softwear</i> In Relation To Previous Research At Virginia Tech . . . . .	10
2	JNI 5 Function Model vs. Real Microprocessor Equivalent . . . . .	21
3	Description Of A Service . . . . .	28
4	Activity Recognition Performed With Various Sensor Sets . . . . .	30
5	System Performance With Non-Ideal Sensors In Various Configurations . . .	31
6	Time Spent Executing <i>Softwear</i> Functions In Simulation . . . . .	36

# 1 Introduction

In his now famous article to the *Scientific American*, Mark Weiser states that “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it” [1]. From Weiser’s vision of ubiquitous computing was born the emerging field of electronic textiles. Electronic textiles (or e-textiles for short) are textiles that have electrical components and interconnections woven directly into the fabric. For several different reasons, e-textiles make an ideal platform for embedding computing pervasively throughout our lives. First and foremost, textiles are already ubiquitous: they are naturally and permanently entwined with human culture, being integrated into every aspect of our daily life in form factors ranging from the clothing that we wear to the drapes and tapestries in our homes to the seats of our cars. Such familiarity and comfort with the interface will prove to be invaluable when adoption of a new technology is considered. Further, because precision techniques for the mass production of textiles are very mature and long-established, electronic textiles can yield both an inherently elastic and durable product with very low cost, particularly when compared to competing technologies.

E-textiles are already showing tremendous potential in commercial, medical, and military domains while presenting challenging research problems [2][3][4][5][6][7][8][9][10][11][12][13][14][15][16][17][18]. These various applications demonstrate a wide variety of uses and platforms for e-textiles that exhibit two essential facets of ubiquitous computing: personal devices for the individual and smart infrastructure devices for his environment. Examples of smart environment applications include smart boards and drapes that can locate a vehicle [1][19], while examples of some personal, wearable devices include a military uniform that determines the medical status of its wearer and garments that can sense the shape of a user and determine their activity [7][20][21][22]. The inherent wired communication running through an electronic textile has large power advantages over any competing wireless on-body network [23] and integrates itself unobtrusively in a way that other computing platforms like the circuit board and those with physically indiscrete connections between components simply cannot, making e-textiles less likely to hinder the user [20][24].

The purpose of this work is not the evolution of any single application or group of applications. Instead, this work aims to augment the infrastructure in which e-textiles are developed to facilitate the rapid creation of new or modified systems without the costly process of building and testing a physical prototype. Currently, there would be little support for evaluating the effect that trivial changes to a system would have on performance or quality of service guarantees. Examples of such unsupported alterations are moving computational or sensing components to different locations throughout the fabric or adapting an application to fit different form factors (such as a long-sleeve shirt versus a tank top). Additionally, there is a tendency to design a solution tailored to a particular problem and then essentially discard the work and start from scratch for the next project. Conversely, this work seeks to create an environment and attitude in which sensor code written months ago for a different application is seamlessly integrated into any other future application requiring similar sensor data

without having to modify or redesign the original work. By extending previous work instead of always beginning essentially from scratch, a new class of incrementally more complex and interacting application systems is possible.

In order to realize these goals, the *Software* Application Description Methodology was developed. This methodology forces applications to be constructed and operate in such a way that ensures both flexibility and reusability. To achieve this, applications must function in a fully event-driven fashion, acting only upon initialization, as response to received messages, and during timer interrupts. In addition to defining a structured way to craft applications, the event-driven approach lends itself to low-power idle states which are characteristic of systems such as e-textiles in which power is a meaningful constraint. Further, the proposed methodology compels applications to only send and receive data through the use of a publish-subscribe service model. Along with timer and message events, the service model provides a necessary abstraction away from the underlying hardware and textile layout so that applications can be developed independently. This allows a designer to focus on programming a textile as a whole with cooperating components instead of having to personally build each individual node with a specific higher-level purpose in mind.

Coupled with the *Software* Application Description Methodology was the development of the *Software* Resource Manager. The *Software* Resource Managers are first and foremost service brokers. The resource manager transparently maps published services and applications (services that require other services to function) to subscribers of those services using a well-defined message format. Before making any assignment of provider to consumer, the *Software* Resource Manager will make a determination as to whether the bandwidth and energy requirements can be met (using a priority-based system), denying admittance to those whose quality of service cannot be ensured. *Software* Resource Managers also fulfill an evolving set of upper-level functionality such as discovering faults occurring in the system and changing message routing.

Even good design practices and resource management techniques, however, are unable to achieve the rapid (non-physical) prototyping called for by this work. For this, a simulation environment was constructed that enables the testing of e-textile services and applications as well as serving as the testbed for the *Software* Application Description Methodology and Resource Manager. The simulation environment is constructed using the well-known Ptolemy II system developed at the University of California at Berkeley [25] and incorporates ideas and lessons from previous work conducted at Virginia Tech [15][19][20][26]. To better facilitate rapid reconfiguration, a simple Java program translates topology files into a form that the Ptolemy II simulation environment can execute. Simulation data is stored in MySQL databases [27] and is obtained from the Carnegie Mellon Graphics Laboratory [28], the Locomotion Research Lab at Virginia Tech, or from a motion analysis textile built at Virginia Tech [5]. Compared to physical prototypes, the simulation environment has tremendous advantages not only in reconfiguration and data variety, but also in simulating faults and

evaluating the overall system's response.<sup>1</sup>

## 1.1 Thesis Organization

The remainder of this thesis is organized as follows: Section 2 describes research related to this thesis in the e-textiles arena, detailing e-textile work conducted specifically at Virginia Tech, along with recounting selected programming methodology work that is relevant to the current task; Section 3 clarifies some characteristics of e-textile systems including their challenges and the proposed e-textile architecture; Section 4 gives an overview of the event-driven service hierarchy that is the backbone of this work, concluding with a brief note about the model's compatibility with modern processors; Section 5 delves into the *Softwear* Application Description Methodology and depicting a sample application class used in the development of the methodology; Section 6 relates the composition and function of the *Softwear* Resource Manager; Section 7 goes into detail about the Ptolemy-based simulation environment, its use, and infrastructure components in addition to how data feeding the simulation was obtained; Sections 8 and 9 elucidate the results of the thesis as well as the related conclusions and future work.

## 2 Related Research

### 2.1 Previous E-Textile Research

The field of electronic textiles has already generated a wide variety of applications throughout military, academic, and commercial domains. For instance, in the military sphere, Foster-Miller, Inc. has designed an e-textile system to enhance the situational awareness (and therefore survivability) of soldiers in the field while simultaneously monitoring their health [4]. Similarly, the wearable motherboard project and related work at Georgia Tech has created a 'smart' shirt that is capable of detecting bullet wounds and monitoring other vital signs of soldiers during combat [7][10]. Numerous interesting e-textile applications and environments exist in academia, including a textile computing environment suggested and simulated by a group at CMU that is both inexpensive and low in power usage [30]. Also, the MIT Media Laboratory has illustrated a wide variety of e-textile applications as well as methods to construct them [31]. The user of the 21st century demands interactivity, while also expecting an unobtrusive and easy to use system for processing information, leading the researchers at Georgia Tech to select e-textiles as the platform for developing their proposed personalized mobile information processing (PMIP) system [32][33]. Electronic textiles are beginning to

---

<sup>1</sup>It is worth noting that this work was developed somewhat exclusively with the I<sup>2</sup>C protocol as the communication network [29], although the *Softwear* Application Description Methodology, *Softwear* Resource Manager, and simulation environment are designed to be easily adaptable to other network architectures.

take off commercially as well. A sampling of these commercial products include work by Infineon, which has developed a wearable MP3 player that illustrates their method of attaching electrical components to a textile [34]. Additionally, by making use of conductive fibers that have been woven directly into fabric, Eleksen, Inc. has fabricated foldable textile keyboards [12][18]. Eleksen has further generalized this technology and concept, creating so-called ‘smart’ recliners and car seats [12]. Eleksen and many others have become involved with Apple’s iPod, which has spawned a surge in the creation of commercial e-textile and wearable products including iPod-compatible jackets, backpacks, and belts [12][17][18][35].

Recent work in electronic textiles, including work conducted at Virginia Tech, has demonstrated the vast potential that e-textiles possess in a broad range of health and medical applications. Of course, the work mentioned above by Foster-Miller, Inc. and Georgia Tech that monitors the health and well-being of soldiers [4][7][10] applies more generally in medical circles. Milior-Smartex has also created a vital sign monitoring system which is based on a single data acquisition point fed by various analog sensor transmissions [13]. Similar to this device is Vivometric’s ‘lifeshirt’, which is a vest that monitors health through various metrics like heart rate and blood pressure [6]. Recent work at UCLA proposes to regulate and monitor the administration of drugs using sensors attached to the fabric of an electronic textile [14]. Additionally, Philips has illustrated the operation of a garment that attempts to recognize and categorize various user movement with integrated sensor strips [9]. More specific in its intended purpose is SensaTex’s infant pajamas that check for Sudden Infant Death Syndrome; this work is largely based upon Georgia Tech’s wearable motherboard project [10]. ETH-Zurich examined woven conductive fibers at frequencies exceeding a gigahertz and characterized their performance [32]. The same group later extended their textile work: stimulating the muscles of stroke victims with an e-textiles system [3]. A broader survey of this research is presented in [2] and [32].

## 2.2 Prior E-Textile Research At Virginia Tech

Research conducted at Virginia Tech has identified key aspects and challenges to electronic textile environments [23][32]. Researchers at Virginia Tech have explored both the wearable and infrastructure areas of the e-textile design space, some prototypes of which can be seen in Figure 1. Virginia Tech’s emphasis on taking advantage of the economies of scale by adapting existing methods of low-cost and high-volume textile manufacturing distinguishes its work from much of the ongoing research in the e-textile field. Further differentiating work at Virginia Tech is a strong focus on the modeling and simulation of textiles to circumvent the expensive—in both time and money—process of physical prototyping. A NSF small ITR grant entitled, *Tailor-Made: Design of e-Textile Architectures for Wearable Computing*, represents Virginia Tech’s most recent work in e-textiles, of which the work presented in this thesis is largely an extension. The construction of the simulation and modeling environment alluded to earlier, coupled with the construction of physical prototypes that explore the textile design space and test the accuracy and usefulness of the environment, form the focus

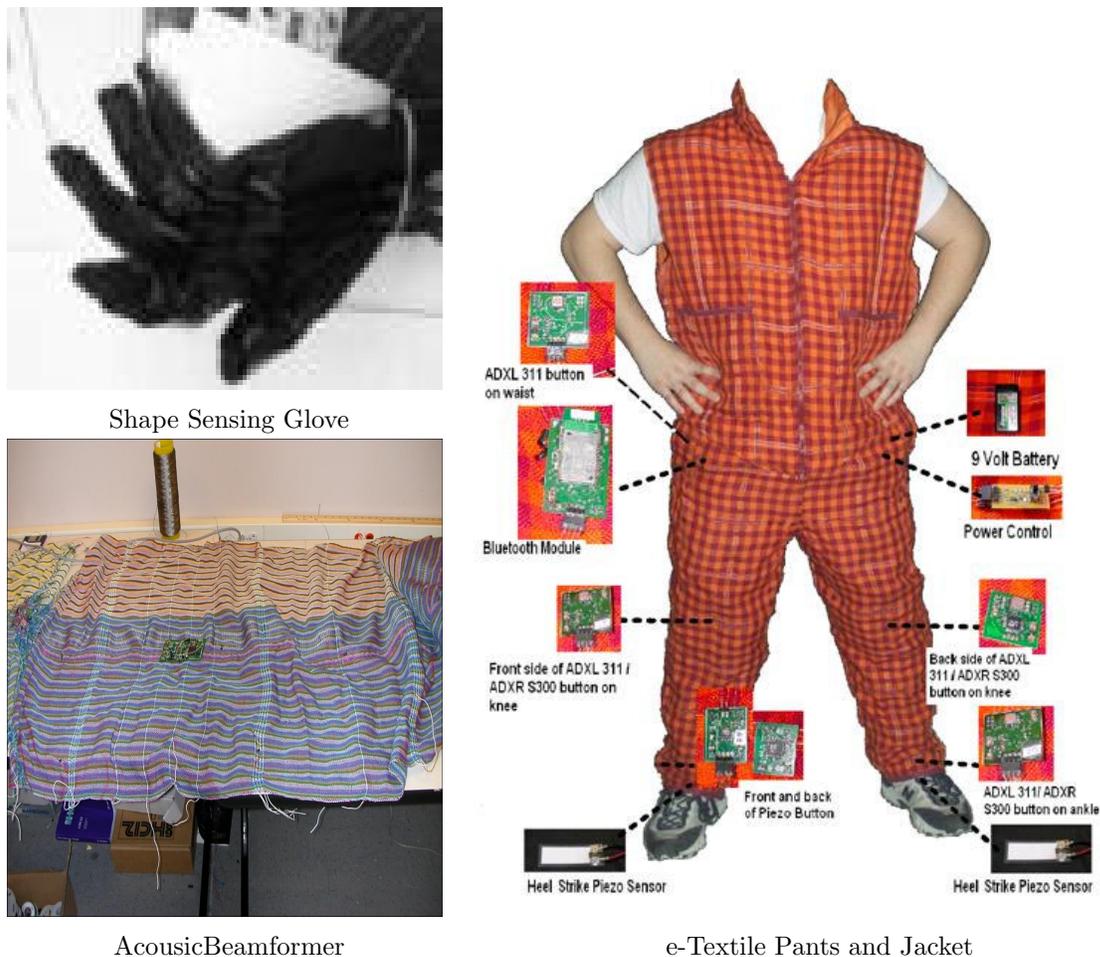


Figure 1: E-Textile Prototypes At Virginia Tech

of the Tailor-Made project. More specifically, Tailor-Made models both the layout of the textile (how sensors and computational elements are distributed in the fabric) and how the physical environment (e.g. human motion, folding) can influence the system [15][19][20][26].

The bounds on electronic textile systems became further detailed through the simulation on the case studies of acoustic beamforming and shape-sensing [20]. In previous Virginia Tech work supported by DARPA, the location of vehicles was pinpointed using large-scale (up to thirty feet long) acoustic-beamforming textiles [16][24][36]. The acoustic-beamforming prototype operated continuously over long periods on a single 9-Volt battery and made use of embedded communication between sensors and computational devices across a network woven directly into the fabric. In other previous work, the development of an e-textile that made use of e-Textile Attached Gadgets (e-TAGs) [37] research while extracting measures of gait from sensors (including angular velocity and acceleration at a joint as well as stride length and heel strikes) was facilitated by the Tailor-Made environment. This textile used those

measures to categorize the user's activity [20][22]. The design of this e-textile involved making design choices that would be relevant across a broad spectrum of users rather than custom-designed for any particular user, and because of this, Tailor-Made proved instrumental. The time-correlated sensor data fed into the simulation was generated by applying a database of body position data encompassing multiple users and activities [28] to models of the individual sensors. For instance, the electrical response of piezoelectric film was characterized and that model applied to determine what its reaction would be to excitation at a given joint [22]. Body position data was then used as input to all of these sensor models, and the resulting measures used to train a neural network that classified the simulated wearer's activity. After training the neural network with a sampling of simulated users, it was then used to determine the activities of actual users on the prototype e-textile; it did so with 94 percent accuracy [20]. The environment was later used to modify the existing prototype garment to perform functions related to clinical gait analysis [5].

### 2.3 Relevant Programming Methodology Work

Because the electronic textiles of the future will be executing multiple applications simultaneously, select research in the development of a high level programming methodology is relevant to the current work. Frequently, embedded systems, due to their sparsity of available resources and often extremely optimized code, will not employ any type of operating system environment. While this concept applies even more to an e-textile system, there are runtime environments designed specifically for low resource devices. One noteworthy example is the TinyOS project created by the University of California at Berkeley which applies a simple scheduler and basic application support to its component and event-based programming model [38][39]. This form of distributed program control makes use of the virtual machine concept, which involves environment code for the methodology to be ported onto various physical architectures using their own binary language. The University of California at Berkeley has additionally developed Maté, a virtual machine language that is built upon the TinyOS project and aims to further simplify sensor node programming [40]. It is important to note that a runtime environment such as the Java Virtual Machine, although it is moving closer with the development of the Java Card Platform [41], remains too large to be implemented on many sensor nodes, which typically have tens of kilobytes or less of instruction memory [40], although it may be feasible for larger microprocessing and external hosting nodes. Strict hardware implementations of the Java Virtual Machine, however, could still potentially serve as a platform of an e-textile node.

Many high level programming methodologies, particularly those relevant to e-textile work, use some variation of a service or agent based model. The Open Agent Architecture developed by SRI International allows for the cooperation of various distributed agents with one or more 'facilitators' aiding in the inter-communication of resources [42][43]. Jini offers a similar method for networking applications, especially those using grid-based communication schemes [44]. While these previously mentioned technologies present several approaches that

are useful and perhaps even scalable down to the level of embedded systems, they are not yet capable of the extreme scaling required for an electronic textile with its limited processing and memory. Related publish/subscribe techniques, however, hold great promise for textile systems. The publish/subscribe paradigm involves resources making their availability known through ‘publish’ messages and interested parties ‘subscribing’ to use those resources or be notified when certain events occur. A great deal of research has been conducted in this area and many enhancements and improvements have been suggested [45][46][47][48][49][50][51]. These optimizations include efficient ways to map events to subscribers [45][46], the introduction of content-based publish/subscribe middleware [47] that better enables routing [49] or reconfiguration [50], evaluation suites for testing an implementation’s effectiveness [48], and methods of mapping system tasks to nodes capable of executing them [51]. Many of these techniques apply tangentially to e-textile systems which will likely have orders of magnitude fewer subscriptions than their very larger network counterparts.

## 3 Characteristics of Electronic Textile Environments

### 3.1 Challenges of E-Textiles

Perhaps the primary challenge affecting electronic textile systems is the extreme limitation of resources. Design and programming methods for minimal resource systems such as electronic textiles are quite different from those of the larger, traditional systems. Whereas a traditional system can support various software algorithms and hardware configurations with sufficient memory, processing, power, and bandwidth, an e-textile must be inventive to manage its very tightly constrained resources. Compared to standard, off-the-shelf desktop systems, an e-textile will have capabilities many orders of magnitude lower: the single megahertz-clocked Atmel AVR microcontrollers [52], for instance, are used in our current shape sensing garment prototype [20]. Because the trends and mindset of electronic textiles demand increasingly invisible and integrated nodes, the individual nodes will tend to become smaller with equal or less processing capability, instead of larger or more powerful. E-textiles, often intended to be worn or deployed for extended periods of time before battery replacement, must also manage their limited power resources very carefully. Because power usage depends upon the current activity levels, and thus the current being drawn, across the textile, often-complex schemes are necessary to dynamically manage the power consumption of a textile [26][53].

Another difficult challenge for the implementation of e-textile systems is ensuring so-called ‘quality of service’. A non-significant segment of fulfilling quality of service (or QOS) demands is the careful management of power mentioned earlier (to ensure textile lifetime) coupled with the similarly stringent management of bandwidth and computing resources. Beyond ‘simple’ resource administration for QOS lies the related and difficult task of detecting and handling faults that occur in the system [24][26][30][32]. The low-cost and high-volume manufacturing techniques of the textile industry, even with its very high pre-

cision, will lead to relatively high defect rates when compared to more expensive processes for creating traditional networks [26][30]. Additionally, due to the deeply psychological ways in which fabrics will blend into their environment and not be seen as computing units, they will likely experience the same treatment and resulting wear and tear that standard fabric normally undergoes during its lifetime [30]. Not all faults are due to physical defects, although that makes them no less insidious to deal with. Because electronic textiles have to conserve their power, some portion of the individual nodes are likely to be in low-power idle or sleep modes, which are essentially transient faults as far as other dependent nodes are concerned [24][54]. Because e-textiles are likely to be in motion much more than conventional networks, motion artifacts and the effects of folding and shape changing further complicate the QOS picture [20]. Providing fault tolerance remains a continuing area of research that often involves dynamic adaptation, wide distribution of redundant resources, and location-based (as opposed to address-based) information schemes [20][24][26][30][32][54].

### 3.2 E-Textile Architecture

The construction of the electronic textile fabric is the very base level in the e-textile architecture. Prior research strongly suggests that a weave is a good structure for electronic textiles because of its flexibility, strength, and low manufacturing cost [24]. Physically weaving transistors or similar computing elements into a textile has been researched, but still remains a long way from practical consideration [24]. Accordingly, this work will focus on the current state of the art: incorporating into the fabric various specialized electrically conductive fibers, multiple power elements (including those in fiber form), and sensing and actuating elements, whose packaging is continually becoming more and more suited to deployment in a textile [31][34][55]. All of these would be directly woven into the fabric and augmented with the next layer of the textile architecture discretely attached to the fabric material: more powerful sensing or computation elements disguised in various textile form factors like buttons, tags, ribbons, zippers, and snaps [37]. This layer consists of both devices that are not yet available in weavable fiber form and those that need to maintain detachability for various other reasons like launderability<sup>2</sup> or cost. For the applications examined in this thesis, the communication and power lines will be woven through the textile itself, while the entire computation base will be strictly attached to the fabric [19][20][23][24].

An early design decision that permeates the work at Virginia Tech and is assumed to be true for the remainder of this work is that communication in the electronic textile systems will be strictly digital [23][24]. It is important to note that because most analog to digital converters have many channels, more conventional systems have, in the past, routed all analog communication to a single or relatively few central collection points, thereby eliminating the need to attach any digital components to the system. This approach, however, is unpractical

---

<sup>2</sup>More advanced packaging techniques for digital components will soon allow for the manufacturability of washable textiles [31][34].

for an electronic textile system for a number of reasons. Perhaps the two most compelling reasons why this approach fails is that there will be relatively little bandwidth along the critical paths for these A/D collection points and there is likely to be a high fault rate in the textile, which makes this type of centralization undesirable. Because of the need to convert many sensor and device outputs from analog to digital form, there must be a large number of processing nodes on the textile that, at the same time, must remain small enough to integrate seamlessly into the fabric so as not to compromise the textile's flexibility or physical appearance, and thus its ubiquitous nature.

The large number of processing elements required for digital communication coupled with their need to 'disappear' into the fabric leads naturally to the implementation of two tiers of computing nodes. The overwhelming majority of nodes will be very tiny nodes with responsibilities ranging from converting data from analog to digital, controlling simple sensors, and providing an interface to the communication backplane—relaying commands to their respective nodes and transmitting requested data onto the network. These smaller nodes will have the minimum functionality possible and, instead of growing more powerful as technology improves, will increasingly vanish into the fabric as newer technology makes that functionality available in more and more minute packages. Current research indicates that small PIC microcontrollers probably have the desired amount of functionality for this first level of processing [37]. Processors responsible for application level execution and overall system control and monitoring will be much fewer in number and make up the second tier of processing in the e-textiles. These second tier nodes will have sizes on the order of an ARM microprocessor [56], and unlike the first-level of nodes, will become more powerful (and remain constant in size) as technology improves. Multiple generations of e-TAGs [37] have been developed that currently realize the two-tier processor architecture described above. Figure 2 shows an assortment of e-TAGs previously developed at Virginia Tech.

Fault tolerance concerns dictate that all resources, not only sensors and communication paths, but also power providing components and system control, be both distributed and redundant. The large surface area offered by electronic textiles proves tremendously useful in this regard, allowing for multiple fibers of various types as well as multiple sensing/actuating and power generating elements to be scattered throughout the textile. If all of these resources are carefully managed, preliminary research suggests that system performance and lifetime can be greatly enhanced [26]. Managing these resources in such a dynamically changing and fault susceptible environment is a focus of much of the rest of this thesis. Table 1 gives an overview of the architecture proposed by this research.

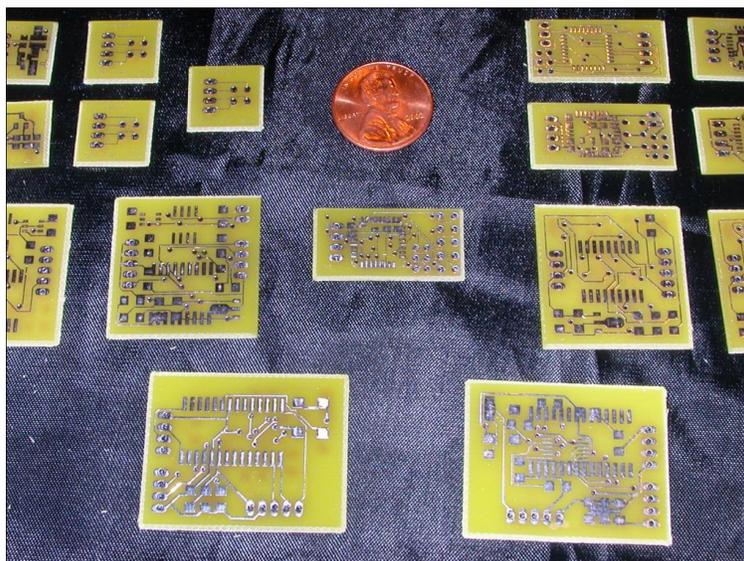


Figure 2: Several e-TAGs Shown Relative To A Penny

Generation	E-Textile Architecture	Programming Model	Application Functionality
State of the Art circa 2000	A PC or handheld device collecting analog data from a set of sensors	A single master program	Single application textile that cannot be reconfigured to support other applications
<i>Tailor-Made</i> : Capabilities and architecture demonstrated in 2002-2005	Multiple tiny sensing, computational, and actuating nodes communicating over a digital network	Each node executes a single program targeted specifically to that node	The textile performs one application at a time but can be reconfigured to execute new applications
<i>Softwear</i> : Capabilities provided as a result of this research	Two-tier architecture of nodes with many tiny sensing and actuating nodes and fewer application executing nodes that communicate digitally over a network	Applications described as a hierarchical composition of service requests that are transparently mapped to nodes distributed throughout the garment	The textile is capable of executing multiple applications simultaneously in a fault tolerant mode while maintaining QOS guarantees

Table 1: *Softwear* In Relation To Previous Research At Virginia Tech

## 4 An Event-Driven Service Hierarchy

### 4.1 Event-Driven Service Model

An event-driven approach is eminently well-suited for implementation in an electronic textile system. A primary reason that an event-driven approach is so ideal for e-textiles is that practically all meaningful work done by a textile occurs as a response to events: a node powering up, a sensing element obtaining a sample, a data-dependent node receiving a data message, or a fault occurring in the system. Further, considering the extreme low-power requirements of an e-textile, it is essential that nodes not be active unless necessary; the event-driven paradigm specifies that a node will respond to events as they occur and otherwise be idle, which is a very appropriate model for textile execution. At a higher level, research into object-oriented code shows that by forcing applications to be developed in this fashion, a programmer must, because of its very nature, create code that is structured and significantly more reusable and flexible to change than more seat-of-the-pants functional programming styles. Of course, application code developed in an event-driven environment is still susceptible to being very difficult to alter and not reusable if it becomes highly coupled, meaning that multiple components are developed together such that they depend upon each other.

In order to battle this coupling of devices and thus render the application code truly reusable and flexible, a publish/subscribe service model is introduced on top of the event-driven model. The publish/subscribe standard involves resources making their availability known through ‘publish’ messages and interested parties ‘subscribing’ to use those resources or be notified when certain events occur. The key here is that compelling subscribers to receive discrete services breaks the coupling between consumer and provider in that any number of providers may now be substituted so long as they provide the correct service. The service and event models together provide a necessary abstraction away from the underlying hardware and textile layout so that applications can be developed independently. This allows a designer to focus on programming a textile as a whole with cooperating components instead of having to personally build each individual node with a specific higher-level purpose in mind. This service model also fits well with the electronic textile architecture being proposed. Small nodes will have only the minimum required functionality while the larger nodes will receive data from these small nodes and perform higher level computation.

### 4.2 Service Hierarchy

When this thesis mentions a service hierarchy, it is referring to services that by their very nature require other services to operate: a simple but powerful idea. Applications are really, then, upper-level services that subscribe to various lower-level services and produce some meaningful output that can then be used by other services/applications that want the

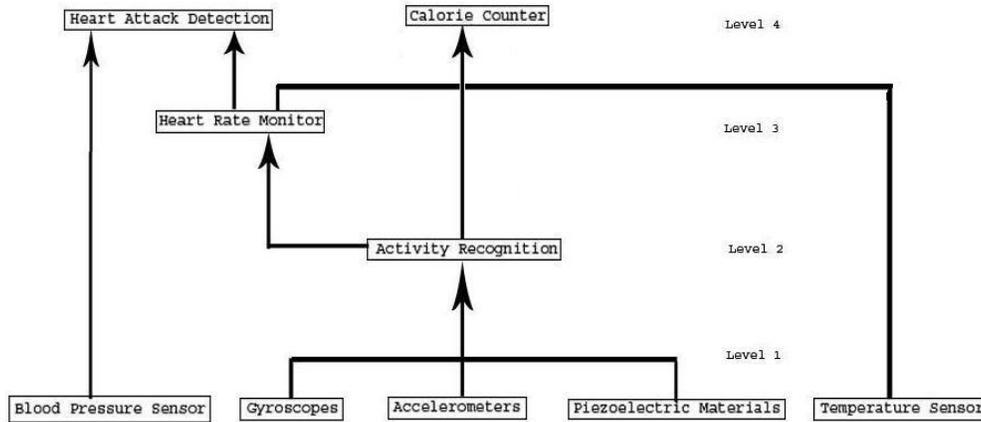


Figure 3: Example Service Hierarchy

output. This idea is contrary to the common tendency to design a solution tailored to a particular problem and then essentially discard the work and start from scratch for the next project. For instance, take the e-textile developed at Virginia Tech (described in more detail in Section 2.2), which takes the output of accelerometer, gyroscope, and piezoelectric sensors and attempts to classify a user’s activity [20][22]. By combining the already upper-level activity recognition service with a heart rate monitor and a blood pressure sensor, a system capable of detecting heart attacks can be created. Also, a temperature sensor could conceivably be added to the heart rate monitor and activity recognition services to determine calorie expenditure. This concept is depicted in Figure 3.

### 4.3 Compatibility With Modern Low-Power Processors

The *Software* Application Description Methodology and Runtime Systems that are described in great detail in Sections 5 and 6 will implement the event-driven, hierarchical service-based approach described above to manage the resources of an electronic textile system. The simulation environment detailed in Section 7 will similarly allow for the design of hierarchical, service-based applications that act only upon initialization/power-up, when they receive a message, or when a timer event (configured by the application) triggers it. This view of the nature of applications is consistent with that of the cutting-edge low-energy microprocessors currently under development, two prominent examples of which are the Sensor Network Asynchronous Processor (SNAP) developed by Cornell [57] and the Clever Dust 1 and 2 microprocessors developed by the University of California at Berkeley [58]. The SNAP system has a very low-power idle state with a small wake-up latency and is optimized to run sensor networks [57]. This is ideal for an environment in which applications are only active when servicing certain events and are otherwise in a low-power idle state. In addition,

SNAP's implementation of events using an event queue built directly into hardware with event scheduling support [57] is compatible with the current design that, by necessity, will implement events without managing operating system support. The Clever Dust processors are similarly event-driven with low idle power [58]. The Clever Dust system uses a timer-based system primarily to both wake up processing elements and check for inputs [58]. Sampled at lower clock rates [58] that would be consistent with e-textiles and lower-frequency sensor environments, Clever Dust is also an excellent match for realizing *Softwear's* event-driven approach to computing and application design.

## 5 *Softwear* Application Description Methodology

### 5.1 Achieving Flexibility and Reusability

The event-driven service hierarchy that is the subject of Section 4 is the foundation for the flexibility and reusability that the *Softwear* Application Description Methodology seeks to provide. The necessary limitation of execution to select events (namely start-up, receiving messages, and timer interrupts) coupled with the service-based model provides the abstraction required to accurately describe electronic textile components. For instance, a sensor cannot know about—much less perform—activity recognition or health monitoring or any other complex task, but can only monitor whatever it is recording at some sampling rate and make that data available to those who want it; this correlation with reality in terms of describing the behavior of the sensor means that the model is much more likely to be accurate and can be inserted whenever a similar sensor is needed in the future. This is sufficient to guarantee that any sensor will behave in exactly the same way (from the viewpoint of another node) regardless of its individual underlying hardware and software, thereby enabling the transparent binding and migration of services from one node to another should the need arise. Even if the more complex agent protocols (see [38][39][40]) could be supported on such tiny nodes, it is strictly extraneous in the context of these small electronic textile nodes and the system as a whole. That being said, however, applications must be able to robustly express what services and constraints on those services they can provide or will require before this methodology will be useful.

### 5.2 Supporting Publish/Subscribe

There are a couple of major challenges in the specification of services in the publish/subscribe model. First, a node needs to know various specifics related to its execution of a particular service including, for example, bandwidth and power usage. The assumption is that our existing modeling and simulation work will be able to largely assist, if not fully and automatically determine, many of these requirements. Preliminary work has shown that the

environment does indeed have the ability to generate an accurate accounting of the resource use of a given service/application [19][54], and so this problem will not be addressed further in this text. Secondly, services have to be stipulated in such a way that the model can still be implemented on the very simple nodes characteristic of the e-textile system while still facilitating the correct and efficient binding of published resources to subscriptions in the system. Critical to the realization of both goals is the concurrent development of the *Software* Runtime System described in Section 6, which acts as a service broker to match providers and consumers as well as manage system resources. An individual node needs only to provide a basic appraisal of its resource use in a pre-defined message format that includes fields such as a service/application identifier, priority, location of provided/requested resource, maximum/required data rate, and idle and active battery usage. This specification is not yet at the point where it is suitable for development by non-proficient programmers, but will instead serve as a design foundation which capable programmers can further extend to support a broader range of future system designers.

### 5.3 Sample Application Class

Along with the development of the *Software* Application Description Methodology was the simultaneous modeling of a garment for health monitoring. The goal is to eventually provide an electronic textile garment that not only monitors and analyzes several health-related physiological measures (such as heart rate, blood pressure, and temperature), but also uses activity recognition to place those measures in the appropriate context and look for causal relationships. This particular application was chosen for construction alongside *Software* for two reasons: first, it encompasses the critical design issues of hierarchical services, the need for operation longevity, and strict quality of service constraints, making it ideal for investigating critical aspects and alternatives for the specification of the methodology and the operation of the runtime system; second, it fills a void in the medical community for conducting physiological monitoring correlated with a patient's everyday lifestyle and activities [8] as opposed to in a physician's office, where measurements can often be incomplete, if not misleading. This diagnostic tool would be a tremendous augmentation for current laboratory measurements, which can capture very limited information because of the controlled and uncomfortable setting. For example, a single blood pressure measurement taken once monthly in the office of a doctor will not be nearly as indicative of a patient's true cardiac health as measurements spread out over weeks in an informal setting. To further the example, to allow for blood pressure readings to be properly interpreted, heart rate and a record of the user's activity should be known. Some sensors might require that a subject not be moving, again emphasizing the need for providing context—something that e-textiles are particularly adept at [20][21][22]—as a dependent service.

## 6 *Softwear Resource Manager*

### 6.1 Discrete Allocation of Resources for Quality of Service Guarantees

While the behavior of any particular electronic textile application can likely be characterized, the overall, system-wide behavior of multiple applications, particularly when they are based on environmental events, will not be known *a priori*. What can be known somewhat statically at a system level will be the wired communication paths and their capabilities (not whether they are fault-free) as well as the user-specified and rarely adjusted longevity that the system must maintain (viewed as a single user control). Therefore, any quality of service management scheme must incorporate significant dynamic monitoring of what applications and services are admitted into the system. A large body of research work has been conducted into ensuring quality of service dynamically, as an e-textile must, focusing primarily on monitoring resource usage with detection of QOS failures [59][60][61][62][63][64][65][66]. Some interesting approaches include a cooperative model as presented in [66], enabling applications to specify acceptable QOS trade-offs [61], and attempting to predict and correct when failures will occur [62]. While there is promise in the approach of [62], its reliance on duplicating services, like the requirements of the other approaches presented, does not scale practically to the level of the small electronic textile nodes that will permeate an e-textile system. The proposed e-textile system, making use of the *Softwear* Application Description Methodology, has a significant advantage in dynamic quality of service management as a result of the service model which characterizes what is happening in the system at a given time. System level knowledge of longevity and communication bandwidth is translated by the *Softwear* Resource Manager into a fixed amount of given resources to be discretely allocated on a priority basis. The aforementioned research in determining the resources that an application or service consumes [19][54], along with the discrete allocation of available resources, has in preliminary simulation shown to be sufficient for managing quality of service among applications that exhibit some average resource consumption. See Section 9 for thoughts as to potential future optimizations to this scheme.

### 6.2 Fault Tolerance Mechanisms

The *Softwear* Resource Managers incorporate a number of different mechanisms for fault tolerance. First, the resource managers are designed to not become a single point of failure. Instead, *Softwear* managers have two different modes of operation as described in detail in Sections 6.3, 6.4, and 6.5: upper and lower-level (referred to as masters and slaves for short). Typically, there will be many distributed slave *Softwears* which execute the service brokering responsibilities directly and can assume the workload of another if necessary. Any manager can assume the role of master and manage system level properties should the need arise,

though some system level information can be lost if a master were to catastrophically fail. Additionally, a master *Software Resource Manager* uses a polling system to ascertain the status of the slaves and the network. A slave uses passive timeout sensing to keep track of the services on its bus, allowing for the rebinding of services on failed nodes. Whenever it becomes necessary, routing paths can also be altered (using a distance vector based algorithm, see Section 7.3.3) to accommodate any network or other resource failure.

### 6.3 Hierarchy of *Software Resource Managers*

Early in the design process, the creation of both an upper and lower-level functionality for the *Software Resource Manager* was determined to be optimal for several reasons. First, in order to facilitate more able network monitoring and distribution/parallelization of the service brokering task, *Software* managers need to be heavily replicated and distributed throughout the system. Fault tolerance concerns also dictate that such locally intensive computation not be performed in a centralized manner. At the same time, however, a heavy communication and computation penalty (that could be intensified by communication or node faults) would be involved in trying to manage system level policies or attempt to discover and adapt to faults in a distributed manner. While seemingly antithetical, a higher-level resource manager was desirable for handling system level policies in a fault tolerant manner, with the understanding that the higher-level duties can be migrated to any lower-level manager in the event of a problem with the current master. Additionally, the division of duties more accurately reflects the service-based architecture that this work is developing, and accordingly, allows for the operation of each task, service brokering and system auditing, to be more flexible to change. Figure 4 illustrates the base case for interactions between the levels of the *Software* hierarchy in assigning services to applications.

### 6.4 Lower-level *Software Resource Managers*

The lower-level or slave *Software Resource Manager* serves as an agent for connecting clients advertising services with those requesting them. In order to achieve this, a simple matching algorithm determines which, if any, published services match inactive subscriptions using equal service identifiers, compatible locations, and sufficient data rate.<sup>3</sup> These matches are performed first on the lowest-resource consuming service among the inactive subscriptions with the highest priority. When no match is available, the slave will broadcast the subscribe request a single time to other networks in case there is a matching publisher in a different network. If the manager makes a determination that a subscription match has sufficient bandwidth on its network to run, the slave then formulates and sends a battery/power

---

<sup>3</sup>While the general assumption is that the number of services on any distributed slave will be many orders of magnitude smaller than those in larger, optimized publish/subscribe systems [45][46][47][48][49][50][51], future work may involve more complex schemes (see Section 9).

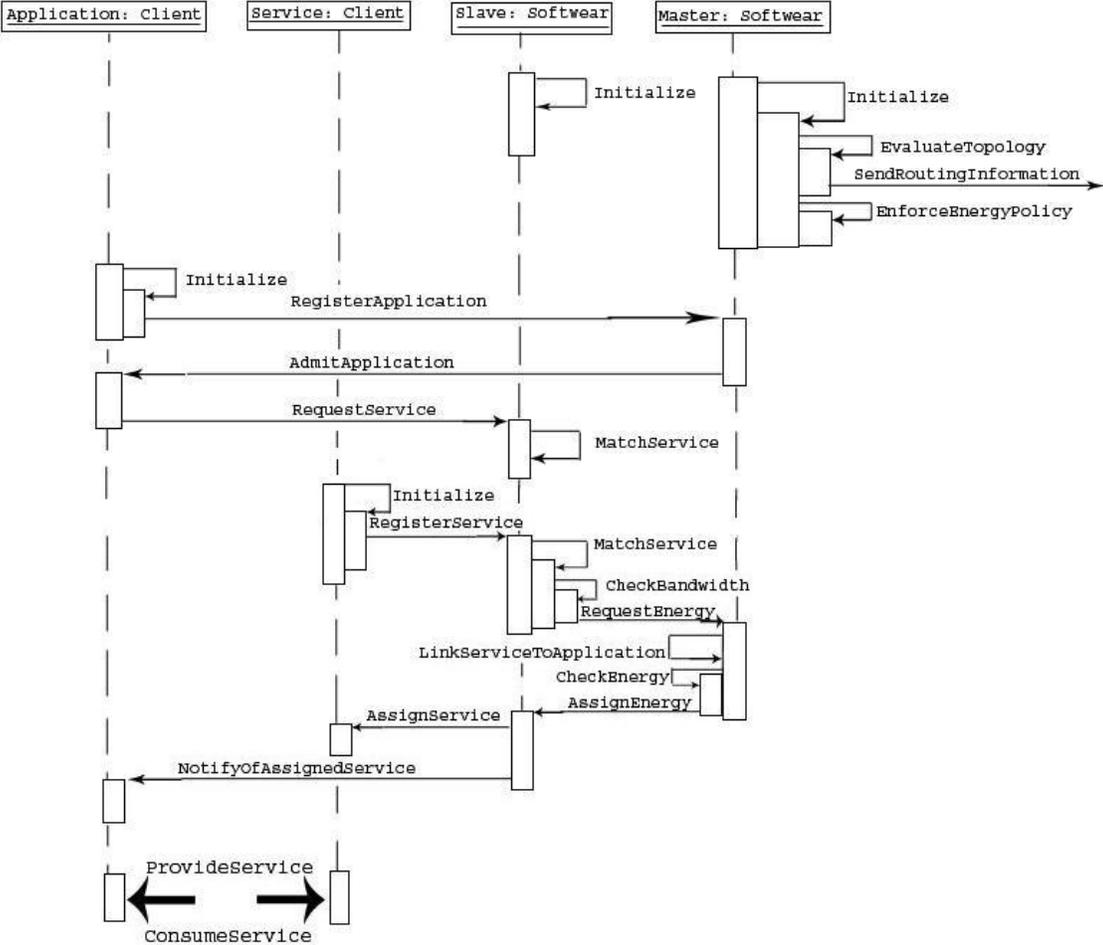


Figure 4: Interaction Model Of Service Assignments In *Softwear* Hierarchy

request to the upper-level manager. If the energy request is approved, the slave then sends a confirmation message to both the publisher and subscriber starting the service. When resources are not present, a slave *Softwear* will then preempt the operation of lower priority active services if possible.

In addition to the service brokering task, the slave *Softwear* managers also play a role in the fault tolerance of the system as a whole. First, any slave (as mentioned in previous sections) may assume upper-level duties by flipping its master flag and subsequently managing the system. Also, the slaves are responsible for answering the polling of the current master, a duty which plays a part in keeping the master *Softwear* manager informed about which communication paths have faults. The low-level manager also snoops on its respective bus not only to detect and deal with services that have timed-out, but also to passively sense and manage non-local services that are using its network resources. The operation of the low-level *Softwear* Resource Managers is designed to be replicated and redundant, so each low-level manager listens to a shared address and make their decisions in a deterministic fashion; such messages will not be routed across busses unless there are no fault-free *Softwear* slaves on the current bus.

## 6.5 Upper-level *Softwear* Resource Managers

The upper-level managers perform an entirely different piece of the resource managing scheme than their lower-level counterparts. The upper-level managers are not, however, completely divorced from the service managing schemes, though their part is smaller. The master *Softwear* handles the admission of applications (services that depend upon other services) to the system. The master does *not* know at registration time what services an application will request nor the sum total of the resources that the application would require, because this knowledge cannot be known by an application in this model. Instead, the master *Softwear* manager is informed only of the resources that an application—and not its dependent services—will use. As the system continues to execute and resource requests come in to the master that are associated with a given application, only then are dependent services linked with that application. This relationship largely addresses the priority inversion problem that can plague service-based systems.<sup>4</sup>

The other responsibilities of the upper-level *Softwear* Resource Managers are to enforce system-level energy requirements and distribute routing information. The energy management scheme involves dividing into discrete, assignable units the amount of energy that the system can assign and distributing the energy units on a priority basis when requests come in from slaves. The amount of energy that the system has available at any given time is generated with an algorithm similar to a multi-level leaky bucket policy (adapted from

---

<sup>4</sup>Priority inversion occurs when an higher-level and higher priority service needs a lower priority service to operate and is unable to obtain it because the priority discrepancy means the lower priority service is not introduced into the system.

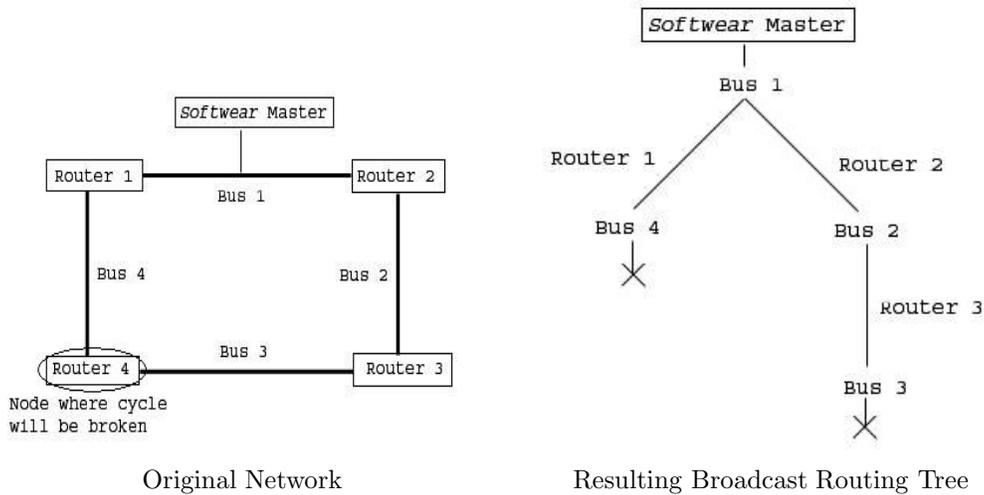


Figure 5: Example Of Breaking A Broadcast Routing Cycle

network traffic policing schemes [67][68]) with the peak rate, average rate, and burst rates determined by system level policies for longevity. More specifically, a source’s energy versus power (or lifetime versus current) profile is used to determine the total amount of energy available over the required system lifetime, and that total amount of energy is then averaged over the system lifetime. *Softwear* masters play a critical role also in the initial setup of system routing information. First, because the system must support broadcast messages, the master uses initial network topology information to determine the locations of any network cycles and break them to prevent perpetual repetition of looping broadcast messages. This principle is demonstrated in Figure 5. Following that, the master sends a message detailing the cost of using each bus (for use in a modified distance vector scheme, see Section 7.3.3) and the addresses that can be found on that bus.<sup>5</sup> Also, in response to its polling or timeout information, a master can change the cost of using a bus or change the layout of clients across busses to force the routers to work around network faults or unpreferred networks.

## 7 The Simulation Environment

### 7.1 Ptolemy II Discrete Event Environment

The environment in which the *Softwear* Resource Manager will operate needs to be purely event-driven and must support both message and timing events. A thoroughly tested software framework that fulfills these requirements is Ptolemy II [25], which was developed at

<sup>5</sup>This mapping allows routers to reduce the size of their routing tables and routing messages to include only busses and not individual clients.

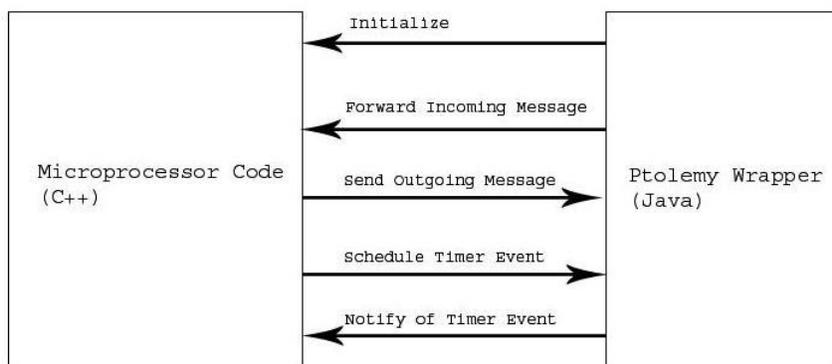


Figure 6: Illustration Of The 5 Function JNI Interface

the University of California at Berkeley and makes use of Vergil as its visual interface. Specifically, the simulation environment is constructed using Ptolemy II's Discrete Event Director. This director creates a platform where an entity is only active when it is directly servicing an event and implements an event queue that offers support for simultaneous and future events throughout the system. As described in Section 4, events in this e-textile system are limited, after component initialization, to either a message received at an input port or a pre-scheduled timing event. This simulation model developed in Ptolemy is fully consistent with the type of environment in which the *Softwear* Resource Manager would be deployed.

## 7.2 Wrapping Ptolemy For Portability

Because the sample applications are being developed in the Ptolemy II simulation environment, special care was taken to ensure that application code would be portable to the physical electronic textile prototype. This task required providing a link between the Java code in which Ptolemy actors are written, and in which very few e-textile nodes could run, and the C++ code which is much more likely to be supported on smaller microprocessor nodes. To accomplish this, the Java Native Interface (JNI) [69] was introduced to run native C++ functions in Ptolemy's Java environment. To remain consistent with our programming model and to further guarantee that only the Java methods and not the C++ code would need to be rewritten, a common skeleton Ptolemy program and C++ communicate through 5 predetermined events: Java calls the C++ initialization; Java passes a bus message to C++; Java informs C++ that its timer event has occurred; C++ sends a bus message to Java; and C++ sends a timer event to Java. As a result of this partitioning of tasks, the C++ code that will run on the e-textile nodes needs only be rewritten for functions which will be largely architecture dependent. Figure 6 and Table 2 further illustrate this model and its tight correlation with real microprocessors.

JNI Function	Purpose In Simulation	Real Microprocessor Equivalent
Initialize	Configuration and initialization of system and state variables as well as application/service registration	System power-up with same functionality
ForwardIncomingMessage	Response to messages from the network	Incoming network message interrupt
SendOutgoingMessage	Sending messages to the network	Negotiating and sending of an outgoing network message
ScheduleTimerEvent	Arranging a future time to execute some functionality	Configuring a timer to interrupt the microprocessor
NotifyOfTimerEvent	Execute some functionality independant of a received message	Handling a timer interrupt

Table 2: JNI 5 Function Model vs. Real Microprocessor Equivalent

### 7.3 Ptolemy Infrastructure Components

Three primary infrastructure components are designed to incorporate e-textile clients into the Ptolemy environment that is being simulated: Bus Arbiters, Client-To-Bus Interfaces, and Routers. Figure 7 shows a sample Ptolemy layout for the four router case of the activity recognition simulation.

#### 7.3.1 Bus Arbiter

The Bus Arbiter component in the simulation environment models, at a high-level, the behavior of a bus.<sup>6</sup> The primary difference between the model that the environment employs and an actual component is that the environment does not process information at a bit level. For instance, differing from the methods of many specifications describing how to obtain control of a communication bus, the simulation component issues a busy signal that notifies clients that the bus is busy until the next message is received. Also, since standard bus networks support a multi-master (more than one element sending to the bus) mode, the simulation analogue performs an arbitration on fully received and not bit-level perspective messages, outputting the winner. Similarly, instead of delivering bits one at a time and modeling single bit propagation times, an entire message is delivered in simulation after the delay that the message would incur as a whole. The result of the higher-level modeling is that the same behavior of many standard busses is preserved while dramatically reducing the time and computing power necessary to represent the network's operation.

#### 7.3.2 Client-To-Bus Interface actors

The Client-To-Bus Interface Actors serve as the link between the simulated busses and the individual e-textile nodes. These interfaces, of course, support the representative network scheme described in Section 7.3.1. The client interface actors queue messages coming from the e-textile client and attempt to send those messages onto the bus whenever the network is not

---

<sup>6</sup>The I<sup>2</sup>C protocol [29] serves as a starting point for the model

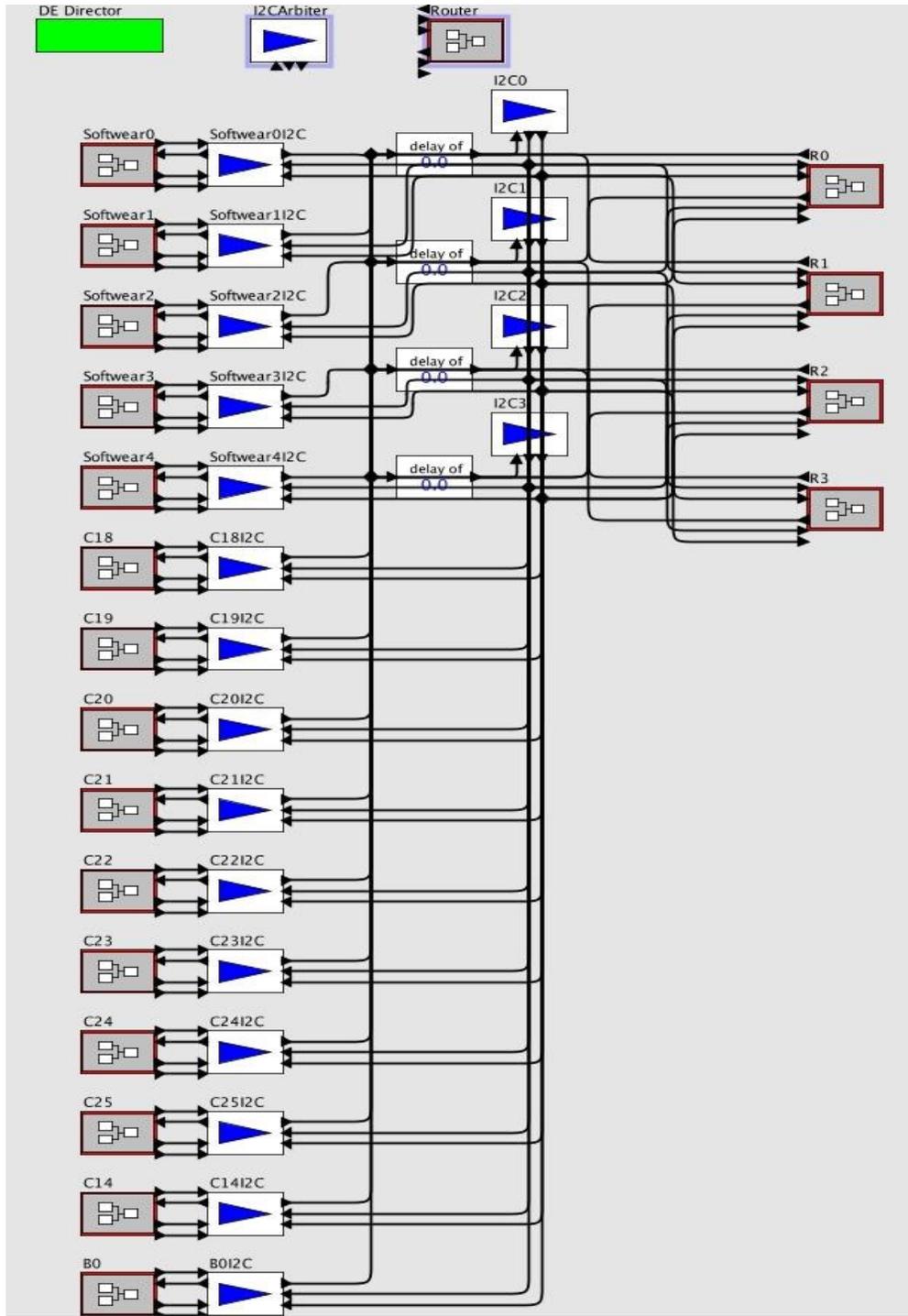


Figure 7: Ptolemy Screenshot Of Activity Recognition Simulation With Four Routers  
 Actors are from left to right: clients, client-to-bus interfaces, busses, and routers

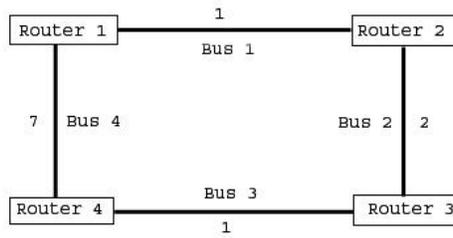
busy. The interface actors check the message the bus actually delivered against its own to see if it won arbitration. These Client-To-Bus Interface Actors compare an incoming message's destination address against a configurable list of addresses and deliver those matching to its client. This ability to listen to multiple addresses on a bus enables both bus snooping and multicasting. The interface also supports the complete removal and reinstatement of a client into the system, which is used to model system faults and unavailable nodes.

### 7.3.3 Routers

The routers used in this system are realized with multiple Client-To-Bus Interfaces (one for each bus) and a custom Router Loop Control actor. While the client interfaces handle the responsibility of putting messages onto and receiving messages from their respective busses, the Router Loop Control handles all major routing functions and will be referred to as 'the router' for simplicity during the rest of this discussion. The router receives at least one message from the master *Softwear* Resource Manager (see Section 6.5) assigning addresses to busses and giving a weighted cost to the use of each bus. Routers will then execute a modified distance vector scheme amongst themselves (a detailed description of the basic algorithm can be found in [67][68]). As mentioned in the discussion of the upper-level *Softwear* Managers in Section 6.5, the routers only store a mapping of addresses to busses and accordingly greatly reduce the length of their routing tables and distance vector advertisements (the cost to reach each bus in the network). A router, upon receiving a distance vector announcement from its neighbors, will take note of which port it arrived on and add the corresponding network's cost to the advertised values. Then, if the computed cost of using that neighbor is lower than the best known current cost, the routing table is correspondingly updated. The neighbor advertising the best route is recorded along with the port to send messages out to, so that a router knows when a route's cost is no longer accurate—i.e., when the neighbor with the previously-advertised lowest cost raises its estimate. Figure 8 shows an example of the algorithm.

## 7.4 Using Motion Capture Data

Because accelerometers, gyroscopes, and piezoelectric material models will not be able to collect meaningful data in the simulation environment, precollected data is stored in MySQL databases [27] and is read in by the various sensors like any sample collected physically. The simulated data is accessed first by location (allowing interpolation if necessary) and then by time and device type. Figure 9 illustrates the layout of the MySQL database. This data will often be completely prefetched at the beginning of simulation to prevent the repeated set-up for MySQL function calls which increases the overall simulation time. The data used to populate the MySQL database is obtained from one of three places: the Carnegie Mellon Graphics Laboratory [28], with dependent measures extracted from a collection of C3D (body position) files; the Locomotion Research Lab at Virginia Tech, where important measures



Weighted Network

Timestep 0

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	$\infty$	$\infty$	0

Router 1

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
2	0	0	$\infty$	$\infty$

Router 2

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
3	$\infty$	0	0	$\infty$

Router 3

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
4	$\infty$	$\infty$	0	0

Router 4

Timestep 1

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1/2	7/4	0
2	0	0	$\infty$	$\infty$
4	$\infty$	$\infty$	0	0

Router 1

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	0	$\infty$	0
2	0	$\infty$	2/3	1/1
3	$\infty$	0	0	$\infty$

Router 2

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
2	0	0	$\infty$	$\infty$
3	2/2	0	0	1/4
4	$\infty$	$\infty$	0	0

Router 3

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	$\infty$	$\infty$	0
3	$\infty$	0	0	$\infty$
4	7/1	1/3	0	0

Router 4

Timestep 2

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1/2	3/2	0
2	0	0	2	1
4	7	1	0	0

Router 1

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1	7	0
2	0	0	2/3	1/1
3	2	0	0	1

Router 2

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
2	0	0	2	1
3	2/2	0	0	1/4
4	7	1	0	0

Router 3

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1	7	0
3	2	0	0	1
4	3/3	1/3	0	0

Router 4

Timestep 3

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1/2	3/2	0
2	0	0	2	1
4	3	1	0	0

Router 1

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1	3	0
2	0	0	2/3	1/1
3	2	0	0	1

Router 2

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
2	0	0	2	1
3	2/2	0	0	1/4
4	3	1	0	0

Router 3

Router	Bus 1	Cost/Predecessor		
		Bus 2	Bus 3	Bus 4
1	0	1	3	0
3	2	0	0	1
4	3/3	1/3	0	0

Router 4

Figure 8: Distance Vector Routing Scheme

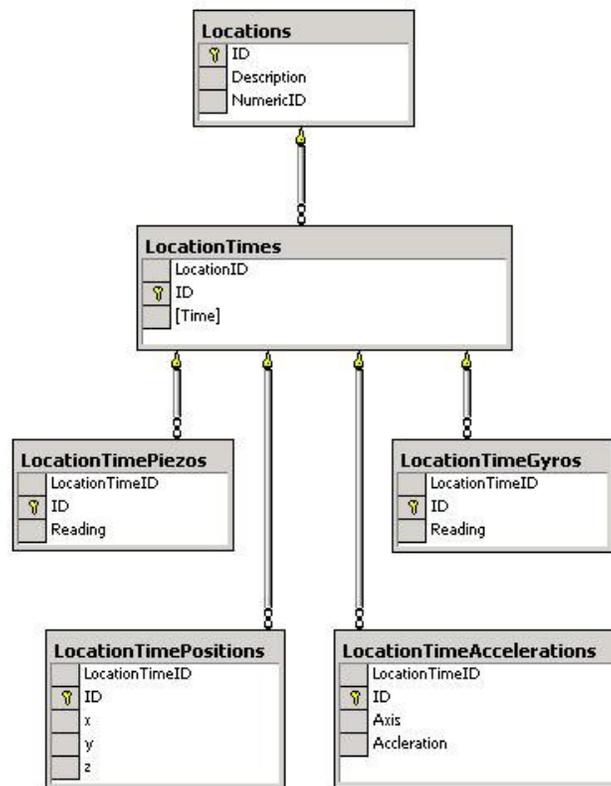


Figure 9: Layout Of The Tables In The MySQL Database

are similarly extracted from body position data; or directly from a motion analysis textile built at Virginia Tech [5].

## 7.5 Using Topology Files For Rapid Reconfigurability

Rather than attempt to physically place, set-up, and duplicate all of the necessary Ptolemy actors when simulating a given e-textile system configuration, a simple Java program translates topology files into a form that the Ptolemy II simulation environment can execute. These topology files take essential parameters—such as a client’s bus address, a bus’s speed, and a router’s network list—along with any number of optional parameters for a given textile client and generate an xml file output with all of the actors’ parameters properly set and the actors systematically placed. This translator makes use of basic rules along with an xml header file that provides classes (which common objects are instances of) to create the Ptolemy compatible file. These topology files allow the system designer to rapidly recalibrate the system: adding, deleting, or altering elements in the system essentially at will with little

required effort. Topology files simplify the testing of asking questions such as ‘How well would the system perform if there were three sensors instead of six?’ or ‘What would happen if this sensor was moved to here or sampled at this rate?’ Many aspects of the system, from network and client behavior to routing schemes, would have been significantly more difficult to verify without the rapid reconfiguration capabilities provided by this approach. The topology file that was used to generate Figure 7 is shown in Figure 10.

## 7.6 Simulation of Faults

A major advantage of the simulation environment work is the ability to test how the overall electronic textile system would respond when faced with a variety of faults to the system. Creating faults in a physical prototype is incredibly difficult, and sometimes permanent, whereas in simulation, nothing more should have to be done than setting a signal. This is the approach taken by this work: every component in the system can instantly be removed or reinstated into the system by sending a signal into the corresponding Client-To-Bus Interface Actor (see Section 7.3.2). These kill/resurrect events are introduced by a custom configurable event generator that reads the event times from a file. Using files opened at runtime gives the additional advantage of being able to simply edit the files offline between runs to change the fault profile instead of having to re-generate the Ptolemy file before the changes will take effect. Note that because all system routers (see Section 7.3.3) use internal Client-To-Bus Interface Actors for sending and receiving messages to the corresponding bus, any outgoing port or an entire router itself can be similarly disabled. During runtime, the master *Softwear* Resource Manager can also adjust the cost of any bus to the infinity value, essentially creating (though more often reacting to) a network fault which all routers in the system must recognize and respond to.

# 8 Results

## 8.1 Mapping Services Among Nodes

The specification of services for the service-based model has proved highly successful and flexible under the current model. Every service implemented in the simulation environment thus far has been easily translated into and completely specified by the current *Softwear*-type service messages. Table 3 gives a description of these *Softwear*-type service messages. Under this framework, for instance, an accelerometer halfway between the left knee and left ankle can adequately distinguish itself from providers of other services, accelerometers at different locations, and accelerometers at the same location but with different addresses on the bus. Further, this accelerometer service can be characterized in such a way that provides *Softwear* Resource Managers with the information necessary to make informed resource and quality of

```

#comment lines begin with #
#parser assumes syntax of commands is correct
#add I2C networks: name,speed,cost
add,i2c,I2C0,400000,1
add,i2c,I2C1,400000,2
add,i2c,I2C2,400000,2
add,i2c,I2C3,400000,4
#add Softwears: name,address,className,faultFile,network
add,client,Software0,9,Software,NULL,I2C2,Locations,Left knee,Left knee,Location1v2,100,master,true,
myAddress,9,topology,'../braden_temp_work/autogen/topoTestRouting.txt'
add,client,Software1,10,Software,NULL,I2C0,Locations,Left ankle,Left ankle,Location1v2,100,master,false,
myAddress,10,topology,'../braden_temp_work/autogen/topoTestRouting.txt'
add,client,Software2,11,Software,NULL,I2C0,Locations,Right ankle,Right ankle,Location1v2,100,
master,false,myAddress,11,topology,'../braden_temp_work/autogen/topoTestRouting.txt'
add,client,Software3,12,Software,NULL,I2C2,Locations,Left knee,Left knee,Location1v2,100,master,false,
myAddress,12,topology,'../braden_temp_work/autogen/topoTestRouting.txt'
add,client,Software4,13,Software,NULL,I2C3,Locations,Right knee,Right knee,Location1v2,100,master,false,
myAddress,13,topology,'../braden_temp_work/autogen/topoTestRouting.txt'
#add clients: name,address,className,faultFile,network
add,client,C18,18,SimSummary,NULL,I2C0,Service,'XYG',Locations,Left ankle,Left ankle,Location1v2,100,
DataRateMax,1000,DataPoints,3
add,client,C19,19,SimSummary,NULL,I2C1,Service,'XYG',Locations,Right ankle,Right knee,Location1v2,100,
DataRateMax,1000,DataPoints,3
add,client,C20,20,SimSummary,NULL,I2C2,Service,'XYG',Locations,Left knee,Left ankle,Location1v2,100,
DataRateMax,1000,DataPoints,3
add,client,C21,21,SimSummary,NULL,I2C3,Service,'XYG',Locations,Right knee,Right knee,Location1v2,100,
DataRateMax,1000,DataPoints,3
add,client,C22,22,SimSummary,NULL,I2C2,Service,'XYA',Locations,Left front waist,Left ankle,Location1v2,100,
DataRateMax,1000,DataPoints,2
add,client,C23,23,SimSummary,NULL,I2C3,Service,'XYA',Locations,Right front waist,Right knee,Location1v2,100,
DataRateMax,1000,DataPoints,2
add,client,C24,24,SimSummary,NULL,I2C0,Service,'Pie',Locations,Left heel,Left ankle,Location1v2,100,
DataRateMax,10000,DataPoints,1
add,client,C25,25,SimSummary,NULL,I2C1,Service,'Pie',Locations,Right heel,Right knee,Location1v2,100,
DataRateMax,10000,DataPoints,1
add,client,C14,14,SimActivity,NULL,I2C0,Locations,Right knee,Right knee,Location1v2,100,Service,'Pan',
ConfigFile,'../work/activityFiles/Configuration.dat'
#add energy components: name,address,className,faultFile,network
add,client,B0,15,Battery,NULL,I2C0,voltage,9.0,regVoltage,5.0,mAHConstant,1135.6,mAHExponent,-1.1922,
Locations,Left ankle,Left ankle,Location1v2,100
#add routers: name,network1,network2,routerID,faultFile
add,router,R0,I2C0,I2C1,0,null
add,router,R1,I2C0,I2C3,1,null
add,router,R2,I2C2,I2C3,2,null
add,router,R3,I2C1,I2C2,3,null
#end by writing to a file
end

```

Figure 10: Sample Topology File For Activity Recognition Simulation With Four Routers

Field	Description
Address	The bus address of the node
Status Byte	Byte telling whether this is a consumer or provider, an application or service, its priority, whether it is being assigned by <i>Softwear</i> , and if it is seeking admission into the system or removal from it
Service ID	Three byte unique identifier of a service
Locations 1 and 2	Two on-body reference points between where a given service lies
Location 1 vs. 2	Percentage detailing how far between locations 1 and 2 a provider is or how far between locations 1 and 2 a consumer can tolerate
Data Points	Number of points of data exchanged by the service
Data Rate	The maximum data rate a provider is capable of or the data rate required by the consumer
Idle Power Usage (providers only)	The amount of power used by the service when it is idle
Active Power Usage (providers only)	The amount of power used by the service when it is active
Application ID (consumer only)	The application ID (like service ID) for which the requested service will be used

Table 3: Description Of A Service

service decisions without greatly burdening what is assumed to be a very small and ‘stupid’ lower-level node. Consumers of a given service can similarly stipulate exactly what service they require to facilitate *Softwear’s* effective mapping of publisher to subscriber.

This pairing of services is completely transparent to the user and does not involve any *a priori* knowledge by nodes of any other nodes in the system. The lack of *a priori* information removed potential coupling in the system that would greatly reduce the reusability of the designed applications and services. The system searches for candidate matches in lists ordered by resource requirements and selects the first publisher that meets the subscriber’s criteria. This ‘greedy’ algorithm does not necessarily provide the most optimum configuration from a system perspective, but does yield good results. The association of consumer to provider involves arithmetic comparison for most parameters used in the matching process. Location, however, proved to be somewhat non-trivial and is broken down into three cases in the current system: the subscriber does not care about the location; the subscriber’s and publisher’s locations correspond (forward or reversed) and the publisher is within the tolerance; and the publisher is exactly located on a particular body part that falls within the subscriber’s tolerance. While this scheme adequately addresses adjacent body locations, a more robust and linear description of on-body location than the one currently implemented (based upon C3D data file markers) is needed to handle matches like ‘no more than halfway between the head and ankles’.

## 8.2 Specifying An Application

The *Softwear* Application Description Methodology has demonstrated itself to be an accurate and appropriate model for the applications and services that have currently been implemented. The JNI 5 function model discussed in Section 7.2 and illustrated further in Table 2 provides a useful abstraction that is key to describing applications outside of Ptolemy

simulation as well. The behavior of applications in this system is accurately depicted by the methodology: all state initialization and registration of/for services is handled in the initialization (often finished by received responses to registration messages or times scheduled to take future steps); periodic sampling/evaluation tasks naturally fit into timer interrupt sequences; processing of received services in addition to tasks finishing initialization processes likewise naturally correspond to responses to received messages.

By examining the translation of the existing off-line activity recognition application into a methodology-compatible, real-time application, the use of the model is highlighted for proof of concept. Without going into too much detail, the application recognition application currently under development is an extension of previous work referred to in [20][22] and now uses singular value decomposition in conjunction with principal component analysis to project high-dimensional data onto a space of reduced dimensionality. Unknown activities are classified using cosines corresponding to the likeliness that a user is performing a certain activity. It does this off-line, evaluating a window of time (on the order of a second) against constants generated from previously training the system. The design of the application began by specifying for the initialization the sensor configurations desired, the training file containing the necessary constants, and the time window size, all factors which make up the state of the application. The data that the application used was summary statistics (e.g. maximums, minimums, averages) from raw sensor data computed by the application. The required summary statistics—not raw data to minimize communication—in the converted system was provided to the application by services from previously created sensors for which it registers. In the receiving messages function, the application takes the data from the services as it arrives and places it in the proper location, performing the activity recognition when a window’s worth of data arrives. Lastly, the application registers its activity recognition as a service available to others, a function beyond the scope of the original application. This translation was accomplished in an afternoon and generates identical<sup>7</sup> data as the original that was not described by the *Softwear* methodology.

### 8.3 Rapid Prototyping And Reconfigurability

The activity recognition example from the previous section also showcases the potential of the system to enable rapid prototyping and reconfiguration. Using topology files (see Section 7.5) to test the current training and activity sets under various sensor configurations, Table 4 was created. As can be seen in the table, when compared against data from a sitting subject, the activity recognition correctly identified the user as sitting in four of the five cases. The activity recognition cosines are not as widely separated between the top candidate activities as the algorithm is capable of achieving, which suggests that the training set is not adequately separating the activities in the k-dimensional space chosen. This point is further emphasized

---

<sup>7</sup>The two application models differ by the start times of the windows collecting data, thus giving negligent differences in output.

Simulation Data Taken From Sitting Subject					
Sensor Set	Likelihood of Given Activity				
	Lying Back	Running	Sitting	Standing	Walking
LANK	.893	.194	<b>.990</b>	.570	.155
LANK, LFWT	.855	.235	<b>.994</b>	.579	.735
RANK, LFWT	.996	.019	.765	<b>.997</b>	.215
LANK, LFWT, LKNE	.870	.201	<b>.986</b>	.613	.018
LANK, LFWT, LKNE, LHEE, RANK, RFWT, RKNE, RHEE	.904	.041	<b>.965</b>	.740	.208

Simulation Data Taken From Standing Subject					
Sensor Set	Likelihood of Given Activity				
	Lying Back	Running	Sitting	Standing	Walking
LANK	.901	.184	<b>.987</b>	.588	.169
LANK, LFWT	.885	.206	<b>.986</b>	.637	.127
RANK, LFWT	<b>.990</b>	.003	.718	.989	.288
LANK, LFWT, LKNE	.885	.171	<b>.980</b>	.630	.013
LANK, LFWT, LKNE, LHEE, RANK, RFWT, RKNE, RHEE	.911	.035	<b>.962</b>	.750	.194

\*Service Providers are: Piezoelectric Sensors on left heel (LHEE) and right heel (RHEE),

ADXL x,y accelerometers on left hip (LFWT) and right hip (RFWT),

4 xy accelerometer/gyroscopes on left knee (LKNE), right knee (RKNE), left ankle (LANK), and right ankle (RANK)

\*Data taken over 1 sec windows

Table 4: Activity Recognition Performed With Various Sensor Sets

when examining the standing subject results that did not classify the user as standing once for the five sample sensor configurations. Different activity sets, training parameters, or time window size, for instance, will likely produce better results. Claims such as the previous ones, as well as an endless amount of others, are testable with the simulation environment quickly and without the cost of continuous physical prototyping. While the evaluation of different activity sets, sensor configurations, and training parameters for optimality is still not a trivial problem, it is greatly facilitated by the simulation environment which allows for rapid reconfiguration. This ability affords the designer the opportunity to spot trends among various configurations and make alterations (like converting a vest and pants system into a tank-top and shorts system) when necessary.

## 8.4 Quality Of Service Guarantees

### 8.4.1 Response To Non-Ideal Sensor Behavior

In order to evaluate quality of service guarantees and more specifically the implied timely delivery of sent messages, non-ideal behavior that will be exhibited outside of simulation must be taken into account. Table 5 shows a somewhat extensive test of average and maximum message latencies under high allotted bandwidth (90, 95, and 99 percent usage) to ensure that allocated bandwidth is actually delivered in the presence of non-regulated system level and service registration messages. This is particularly important when considering that services will also display non-ideal behavior. The sampling of latencies using sensors that experience gaussian-distributed, purely random, and bursty data rates is very promising when service providers vary around the published average rate. As can be seen from the table, the only scenario in which a message seems to be delayed indefinitely is when every sensor

Single Bus, 90 % Bandwidth Usage (359.6k)		
Scenario	Average Message Latency (s)	Maximum Message Latency (s)
All normal sensors	.0003138	.04226
LHEE gaussian, all others normal	.0003327	.03904
LHEE unbiased random, all others normal	.001508	.001660
LHEE bursty, all others normal	.001509	.01452
All gaussian	.0003912	.01078
All unbiased random	.007756	.03674
All bursty	.008000	.04360
LKNE and LFWT normal, LHEE and RANK bursty, RFWT and RHEE unbiased random, RKNE and LANK gaussian	.003327	.02774

Single Bus, 95 % Bandwidth Usage (380k)		
Scenario	Average Message Latency (s)	Maximum Message Latency (s)
All normal sensors	.0005273	9.987
LHEE gaussian, all others normal	.00035439	.1194
LHEE unbiased random, all others normal	.001606	.009040
LHEE bursty, all others normal	.001606	.01562
All gaussian	.0005337	.03272
All unbiased random	.009961	.04360
All bursty	.009962	.04360
LKNE and LFWT normal, LHEE and RANK bursty, RFWT and RHEE unbiased random, RKNE and LANK gaussian	.003642	.02778

Single Bus, 99 % Bandwidth Usage (396k)		
Scenario	Average Message Latency (s)	Maximum Message Latency (s)
All normal sensors	.0004994	9.998
LHEE gaussian, all others normal	.0003552	.9083
LHEE unbiased random, all others normal	.001642	.01518
LHEE bursty, all others normal	.001642	.01518
All gaussian	.0006344	.01224
All unbiased random	.01067	.04282
All bursty	.01067	.04282
LKNE and LFWT normal, LHEE and RANK bursty, RFWT and RHEE unbiased random, RKNE and LANK gaussian	.004538	.02929

Four Bus, 90 % Bandwidth Usage (359.6k)		
Scenario	Average Message Latency (s)	Maximum Message Latency (s)
All normal sensors	.0004200	.008720
LHEE gaussian, all others normal	.0007098	.01824
LHEE unbiased random, all others normal	.001617	.01288
LHEE bursty, all others normal	.001617	.01288
All gaussian	.001300	.01524
All unbiased random	.001806	.02191
All bursty	.01149	.004470
LKNE and LFWT normal, LHEE and RANK bursty, RFWT and RHEE unbiased random, RKNE and LANK gaussian	.003380	.001980

Four Bus, 95 % Bandwidth Usage (380k)		
Scenario	Average Message Latency (s)	Maximum Message Latency (s)
All normal sensors	.0004360	.0111
LHEE gaussian, all others normal	.0004759	.01824
LHEE unbiased random, all others normal	.001665	.01350
LHEE bursty, all others normal	.001940	.02306
All gaussian	.07000	.3444
All unbiased random	.05777	.2702
All bursty	.01018	.004450
LKNE and LFWT normal, LHEE and RANK bursty, RFWT and RHEE unbiased random, RKNE and LANK gaussian	.04803	.2047

Four Bus, 99 % Bandwidth Usage (396k)		
Scenario	Average Message Latency (s)	Maximum Message Latency (s)
All normal sensors	.0008688	.01650
LHEE gaussian, all others normal	.0007511	.01482
LHEE unbiased random, all others normal	.001700	.01384
LHEE bursty, all others normal	.001684	.008720
All gaussian	.1131	.4837
All unbiased random	.2161	.8908
All bursty	.2415	1.017
LKNE and LFWT normal, LHEE and RANK bursty, RFWT and RHEE unbiased random, RKNE and LANK gaussian	.1851	.7872

Single Consumer, 8 Service Providers, 7 Seconds of Data Before Services Timeout, 400k Bus Speed

\*Service Providers are: piezoelectric sensors on left heel (LHEE) and right heel (RHEE),

ADXL x,y accelerometers on left hip (LFWT) and right hip (RFWT),

4 xy accelerometer/gyroscopes on left knee (LKNE), right knee (RKNE), left ankle (LANK), and right ankle (RANK)

\*Non-ideal behavior includes: Gaussian variation about average rate, sending k samples in bursts at 1/k data rate, unbiased random variation around average rate (0\*-2\*)

Table 5: System Performance With Non-Ideal Sensors In Various Configurations

behaves perfectly ideally on a single bus (the delayed message was a non-critical confirmation message that repeatedly lost bus arbitration). Non-ideal behavior of services does not have a significant negative effect on the system as a whole under large loads. Multiple bus cases with routers do only slightly worse at high usage, and even help to smooth away some of the non-ideal behavior and obtain better results at lower usage levels.

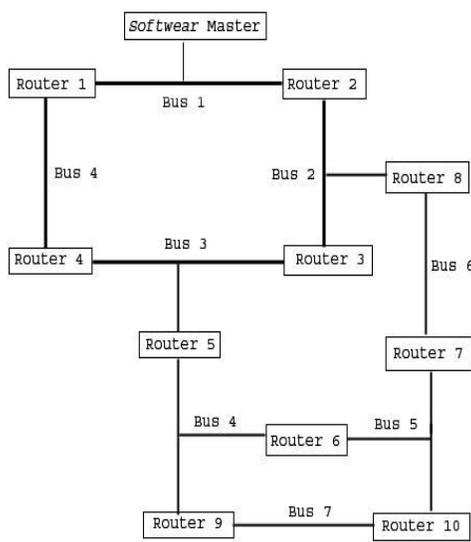
#### 8.4.2 Adaptation To Network Faults

The detection of network faults is based primarily on a polling system initiated and evaluated by the master *Softwear* Resource Manager. Because polling, for fault tolerance reasons, is a broadcast message, an occurring fault cascades throughout the entire routing tree as shown in Figure 11. Looking at the figure, router 3 undergoes a fault that *Softwear* initially assumes to be on bus 3. *Softwear* then disables all routers connected to the suspected faulty bus (routers 3, 4, and 5) and reestablishes its routing tree for the next polling cycle. In the example, this adjustment fixes the fault, so further action is not needed, but if it were a fault that emerged on bus 2, the next step would route through router 4 and fix the fault. The method used in establishing new routes is deterministic and involves disabling all routers associated with a candidate bad network, choosing sequentially a single router to interface to the suspected bad bus, and reordering the list so that a different router will be chosen to interface to the faulty bus next time if the problem persists. The method tends to find new routes quickly and cycles through every router being active around a faulty bus before looping back due to the reordering employed. While admittedly one can envision this algorithm performing quite poorly in the face of a large number of faults occurring close together in time (not unheard of in a textile environment), much of the problem is due to the concept of broadcast trees, which could prove costly to be rid of in bandwidth usage (e.g. message ID's, hop counts) and the complexity of routers and routing.

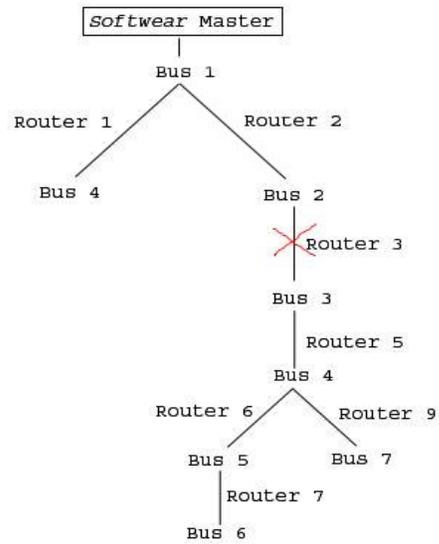
#### 8.4.3 Energy Management

Energy for an electronic textile is as difficult to model as it is tightly constrained. Batteries tend to have a non-fixed amount of energy depending on the power usage of the source across its lifetime [70]. Figure 12 demonstrates some common views of this energy vs. power relationship assuming that power usage is constant over the source's lifetime. This work uses a combination of first and second order techniques to determine the total amount of energy that will be available from a source over its lifetime and partitions that resource out in discrete increments to satisfy the required life of the system. For instance, from a typical datasheet for a common 9V battery, the Energizer X22 [71], a relationship between hours of battery life before the voltage drops to 5V and current drawn can be derived as approximately:

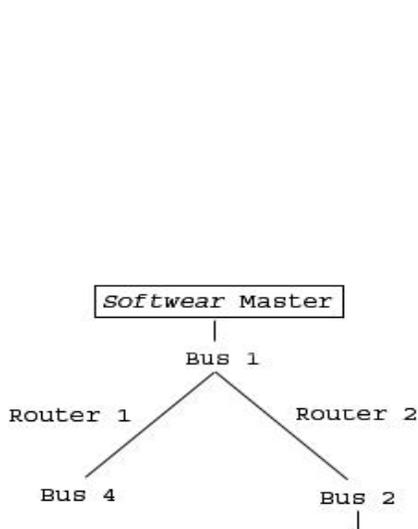
$$\text{hours of battery life} = 1135.59 * (\text{current in mA})^{-1.1922016}$$



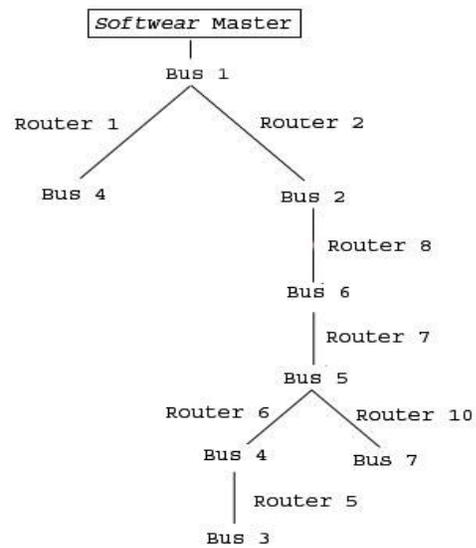
Original Network



Resulting Broadcast Routing Tree With A Fault



Software's View Of The Faulty Network



New Broadcast Routing Tree Around Fault

Figure 11: Example Of Routing Adapting To A Fault

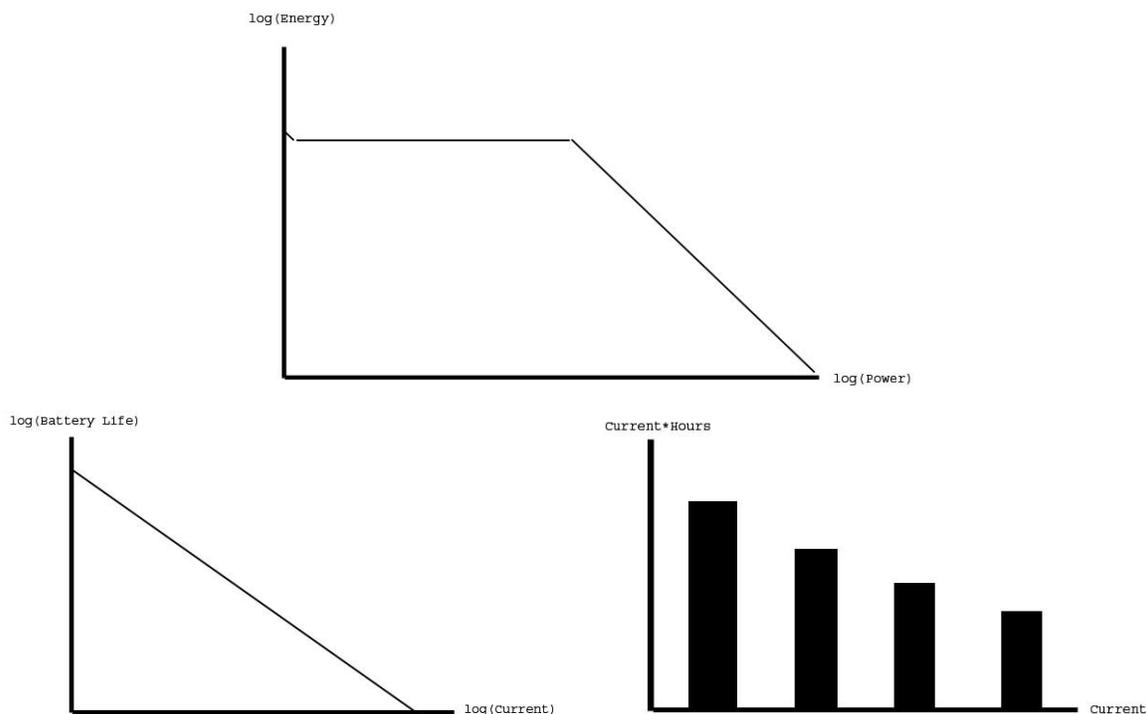


Figure 12: Various Energy Source Specifications That Translate Into Energy Vs. Power

For example, then, if 25 hours of battery life was necessary, a constant 24.553 mA of current can be drawn. So the total amount of energy available from that source is approximately:

$$\begin{aligned}
 \text{total energy}(J) &= \text{time}(s) * \text{current}(A) * \text{approximate average voltage}(V) \\
 &= 3600 * 25 * .024553 * \frac{9 - 5}{2} = 15468.293J
 \end{aligned}$$

This total amount of energy available is then averaged out over the expected lifetime of the system. This scheme provides an accurate first order model of energy consumption that is enforced by the *Softwear* master. *Softwear*'s leaky bucket strategy restricts the average, peak, and burst energy usage, which would tend to keep the source's lifetime from being noticeably *worse* than modeled. The strategy works perfectly in the simulation environment but is based largely upon services knowing their respective active and idle power consumption (see Section 8.1), a practice that will need substantial testing in actual textile system to be completely trusted. Additionally, individual nodes are responsible for entering low-power sleep modes, which is probably the best practice but represents a significant modeling challenge for managing power in future work.

#### 8.4.4 Priority Inversion

Priority inversion occurs when a higher-level and higher priority service needs a lower priority service to operate and is unable to obtain it because the priority discrepancy means the lower priority service is not introduced into the system. In an electronic textile with very limited resources, priority inversion has the potential to drastically reduce resource utilization. This system largely eliminates the priority inversion problem through informed service brokering in the master *Softwear* managers. The list of energy requests is deterministically ordered and each service that requests energy is linked with its associated application that should have registered itself before being admitted into the system. This process is recursive in that an application that is providing a service to a higher-level application will be grouped (along with its dependent services) with the higher-level application when the lower-level application has a lower priority. Thus, priorities are passed from an application to its dependent services, eliminating priority inversion in the master *Softwear*. There remain two cases, however, in which priority inversion does become possible in the lower level *Softwear* managers, although neither case represents an error in the system, being a clear decision of the afflicted application. The first case is when an application fails to register itself for admittance into the system, potentially losing the protection afforded applications. The second case is when an application elects to request a service at a lower priority than itself: the system must allow this practice to support the likely scenario that an additional service will increase the quality of an application's service but is not strictly necessary for it to run.

### 8.5 Overhead Of Implementing *Softwear*

Implementing the *Softwear* Application Description Methodology and deploying the *Softwear* Resource Managers will impose some overhead onto a textile system with already limited resources. The *Softwear* manager program is currently 1972 lines long, with approximately thirty percent of that being documentation or empty lines. The near sixty-nine kilobytes of *Softwear* source code, after being compiled for a specific microprocessor, will require a level of instruction memory not available on the smallest tier of microprocessors. The processor usage statistics are better. Table 6 shows the total time spent executing various upper and lower level *Softwear* functionality in the simulation environment on a single gigahertz processor. While a gigahertz processor is far more powerful than a textile component will be, the table still demonstrates the low processing burden imposed upon a microprocessor that is running a *Softwear* Resource Manager. The service brokering and other tasks of the resource managers incur little bandwidth overhead. For instance, in the single application, eight service, four bus case described in Table 6, less than seven kilobits of total bandwidth is devoted to initialization and assignment of services, while approximately half that amount is used to remove the services later. Unlike a single configuration, non-adaptable system (one in which data messages could conceivably be transmitted without system level information), service messages in the textile system have a five byte 'header' consisting of a status byte

Less Than 2 % Bandwidth Usage						
<i>Software</i> Type	Initialization (ms)	Forwarding Messages (ms)		Timer Firing (ms)		Total Time (ms)
		Average	Total	Average	Total	
Master	2815	.1207	7.000	3.000	3.000	2815
Slave 1	4.000	.06494	35.00	.7143	5.000	44.00
Slave 2	3.000	.04878	14.00	.5714	4.000	21.00
Slave 3	3.000	.05195	8.000	.5714	4.000	15.00
Slave 4	4.000	.04724	12.00	.5714	4.000	20.00

95 % Bandwidth Usage						
<i>Software</i> Type	Initialization (ms)	Forwarding Messages (ms)		Timer Firing (ms)		Total Time (ms)
		Average	Total	Average	Total	
Master	2836	.1220	10.00	.5000	1.000	2848
Slave 1	3.000	.04346	1017	.4000	2.000	1022
Slave 2	3.000	.04533	747.0	.4000	2.000	752.0
Slave 3	3.000	.02392	890.0	.4000	2.000	895.0
Slave 4	3.000	.03117	230.0	.6000	3.000	236.0

Single Consumer, 8 Service Providers, Four 400k Busses, Four Routers

7 Seconds of Data Before Services Timeout, 15 Seconds of System Run Time, 1 GHz processor

Table 6: Time Spent Executing *Software* Functions In Simulation

and ID's of both the sender and service being provided. For services that send relatively little data per message, the five bytes can potentially double the bandwidth usage of that service. Even in that most extreme case, however, the reusability, ability to adapt to faults, and overall system flexibility gained through providing that basic information justifies the bandwidth usage.

## 9 Conclusions and Future Work

The *Software* Application Description Methodology coupled with an event-driven service hierarchy form a flexible and robust model for electronic textile systems. The model does not inhibit what an application can do, but instead imposes a sort of discipline upon the design process. Resulting applications not only tend to exhibit a greater deal of reusability than application code produced purely by intuition, but also are well suited to an electronic textile environment and modern low-power processors. The service model can still be refined to allow greater flexibility in the specification of services and variations of more complex publish and subscribe techniques can be adapted to the system, but the current system has been able to completely and uniquely identify all services and their resource requirements up to this point. The publish/subscribe paradigm provides an independent and detached method of developing components that would allow a designer to leverage future work off of previous work to create a new class of incrementally more complex and interacting application

systems.

The *Softwear* Resource Managers represent another key component in this electronic textile framework by functioning as both service brokers and enforcers of quality of service guarantees. The service brokering task has been highly successful in assigning sufficient services to subscribers without falling victim to priority inversion, although more complex matching algorithms may provide slightly better results, particularly for more nuanced requests of location or specifying options about what service is needed. Enforcing quality of service has proven more problematic because of its complexity and limited knowledge that *Softwear* managers will, by necessity, have. Network bandwidth is well managed even in cases of non-ideal service providers. Further, network faults are detected and routed around if possible. Energy management by the system, conversely, can potentially be greatly improved by enhancing current management schemes. The three major energy management areas that can be addressed in future work are better (second or higher-order) schemes of determining available energy, a system-wide support for nodes entering low-power sleep states, and re-searching whether a better model exists for modeling energy available from multiple sources. Also, the *Softwear* managers largely rely on the truthfulness of applications and services in depicting their resource usage and requirements, while a scheme that provides some sort of enforcement may be desirable when less trusted applications are present in the system.

The Ptolemy simulation environment provides a useful vehicle for more fully and quickly exploring the design space of electronic textiles. The use of topology files that are translated into Ptolemy compatible files greatly speeds up the configuration of a system, and permits the user to rapidly test various configurations and application parameters. The environment was able to generate intricate fault patterns and timing that would have been difficult or impossible (not to mention expensive) to do on a physical prototype. The JNI function model and the Ptolemy infrastructure components largely mask away architecture dependent functionality, which leads to portable application code. The future of the project lies in the eventual movement of these applications and *Softwear* Resource Managers out of this simulation environment and onto physical prototypes to test the effectiveness of the design framework and resource management. Despite the success of the simulation environment, a true deployment of the system will be needed to test the operation of the system and make refinements to present schemes as problems emerge. Work is also being done on programming devices through boot loading on physical textiles to allow the e-textile system to be fully configured on the fly, even the application code that runs on the individual nodes.

## References

- [1] M. Weiser, “The computer for the 21st century,” *Scientific American*, pp. 94–104, September 1991.
- [2] R. Service, “Electronic textiles charge ahead,” *Science*, vol. 301, pp. 909–911, August 2003.
- [3] T. Kirstein, M. Lawrence, and G. Troester, “Functional Electrical Stimulation (FES) with smart textile electrodes,” *International Workshop on a new generation of wearable systems for e-health*, December 2003.
- [4] P. Wilson, C. Carey, B. Farrell, J. Gassner, P. Haugsjaa, M. Pan, J. Teverovsky, C. Winterhalter, and J. Fairney, “Electro-optic fabrics for the warrior of the 21st century,” Technical Report TR-99/030L, Soldier Systems Center-Natick, July 1999.
- [5] J. Edmison, M. Jones, T. Lockhart, and T. Martin, “An e-textile system for motion analysis,” in *International Workshop on New Generation of Wearable Systems for eHealth*, pp. 215–223., December 2003.
- [6] P. Grossman, “The lifeshirt: A multifunctional ambulatory system that monitors health, disease, and medical intervention in the real world,” in *International Workshop on New Generation of Wearable Systems for eHealth*, pp. 73–80, December 2003.
- [7] E. Lind, R. Eisler, G. Burghart, S. Jayaraman, R. Rajamanickam, and T. McKee, “A sensate liner for personnel monitoring applications,” in *Digest of Papers of the First International Symposium on Wearable Computing 1997*, pp. 98–105, 1997.
- [8] J. Healey, “Sensing physiology in the unconstrained ambulatory environment,” in *International Workshop on New Generation of Wearable Systems for eHealth*, pp. 194–200, December 2003.
- [9] J. Farrington, A. Moore, N. Tilbury, J. J. Church, and P. Biemond, “Wearable sensor badge and sensor jacket for context awareness,” in *Proceedings of the Third International Symposium on Wearable Computing 1999*, pp. 107–113, October 1999.
- [10] S. Park, C. Gopalsamy, R. Rajamanickam, and S. Jayaraman, “The wearable motherboard: An information infrastructure or sensate liner for medical applications,” *Studies in Health Technology and Informatics*, vol. 62, pp. 252–258, 1999.
- [11] M. Goldstein, O. Baez, and P. Danielsson, “Employing electrical field sensing for detecting static thumb position using the finger-joint gesture keypad input paradigm,” in *International Symposium on Wearable Computers*, pp. 173–174, October 2000.
- [12] Eleksen, Ltd., “Elektex© wireless fabric keyboard.” Online, 2004. Available: <http://www.eleksen.com>.

- [13] R. Paradiso, G. Loriga, N. Taccini, M. Pacelli, and R. Orselli, “Wearable system for vital signs monitoring,” in *International Workshop on New Generation of Wearable Systems for eHealth*, pp. 161–168, December 2003.
- [14] R. Jafari, F. Dabiri, and M. Sarrafzadeh, “Reconfigurable fabric vest for fatal heart disease prevention,” in *UbiHealth 2004: 3rd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications*, September 2004.
- [15] R. Shenoy, “Design of e-textiles for acoustic applications,” Master’s thesis, Bradley Department of Electrical and Computing Engineering, Virginia Tech, 2003.
- [16] R. Parker, R. Riley, M. Jones, D. Leo, L. Beex, and T. Milson, “Stretch – an e-textile for large-scale sensor systems,” 2002. p. 59.
- [17] MobileMag, “Burton and apple deliver the burton amp jacket.” Online, January 2003. Available: <http://mobilemag.com/content/100/102/C1371/>.
- [18] C. Barylick, “Wearable iPod gear at Macworld,” *United Press International*, January 2006. Available: [http://tech.monstersandcritics.com/news/article\\_1075202.php/-Wearable\\_iPod\\_gear\\_at\\_Macworld](http://tech.monstersandcritics.com/news/article_1075202.php/-Wearable_iPod_gear_at_Macworld).
- [19] M. Jones, T. Martin, Z. Nakad, R. Shenoy, T. Sheikh, D. Lehn, and J. Edmison, “Analyzing the use of e-textiles to improve application performance,” in *IEEE Vehicular Technology Conference 2003*, pp. 2875–2880, October 2003. Symposium on Wireless Ad hoc, Sensor, and Wearable Networks.
- [20] T. Martin, M. Jones, J. Edmison, and R. Shenoy, “Towards a design framework for wearable electronic textiles,” in *Seventh International Symposium on Wearable Computing 2003*, pp. 190–199, October 2003.
- [21] B. Clarkson, K. Mase, and A. Pentland, “Recognizing user context via wearable sensors,” in *Proceedings of the Fourth International Symposium on Wearable Computing 2000*, pp. 69–75, October 2000.
- [22] J. Edmison, M. Jones, Z. Nakad, and T. Martin, “Using piezoelectric materials for wearable electronic textiles,” in *Proceedings of the Sixth International Symposium on Wearable Computing 2002*, pp. 41–48, 2002.
- [23] Z. Nakad, M. Jones, and T. Martin, “Communications in electronic textile systems,” in *The International Conference on Communications in Computing 2003*, pp. 37–43, June 2003.
- [24] Z. S. Nakad, *Architectures for e-textiles*. PhD thesis, Bradley Department of Electrical and Computing Engineering, Virginia Tech, 2003.
- [25] University of California at Berkeley, “Ptolemy II.” Online, 2005. Available: <http://ptolemy.eecs.berkeley.edu/ptolemyII/>.

- [26] T. Sheikh, “Modeling of power consumption and fault tolerance for electronic textiles,” Master’s thesis, Bradley Department of Electrical and Computing Engineering, Virginia Tech, 2003.
- [27] MySQL, “MySQL: The world’s most popular open source database.” Online, 2005. Available: <http://www.mysql.com/>.
- [28] CMU Graphics Laboratory, “CMU Graphics Lab Motion Capture Database.” Online, 2003. Available: <http://mocap.cs.cmu.edu/search.html>.
- [29] Phillips Semiconductors, *The I<sup>2</sup>C Bus Specification*, 2.1 ed., January 2000. Available: [http://www.semiconductors.philips.com/acrobat\\_download/literature/9398/-39340011.pdf#search='i2c%20specification'](http://www.semiconductors.philips.com/acrobat_download/literature/9398/-39340011.pdf#search='i2c%20specification').
- [30] P. Stanley-Marbell, D. Marculescu, R. Marculescu, and P. Khosla, “Modeling, analysis and self management of electronic textiles,” *IEEE Transactions on Computers*, vol. 52, pp. 996–1010, August 2003.
- [31] E. Post, M. Orth, P. Russo, and N. Gershenfeld, “E-broidery design and fabrication of textile-based computing,” *IBM Systems Journal*, vol. 39, pp. 840–860, 2000.
- [32] D. Marculescu, R. Marculescu, N. Zamora, P. Khosla, S. Park, P. Stanley-Marbell, S. Jayaraman, S. Jung, C. Lauterbach, W. Weber, T. Kirstein, D. Cottet, J. Grzyb, G. Troester, M. Jones, T. Martin, and Z. Nakad, “Electronic textiles: A platform for pervasive computing,” *Proceedings of the IEEE*, vol. 91, pp. 1995–2018, 2003.
- [33] S. Park, K. Mackenzie, and S. Jayaraman, “The wearable motherboard- a framework for Personalized Mobile Information Processing (PMIP),” in *39th conference on Design automation*, pp. 170–174, ACM Press, June 2002.
- [34] S. Jung, C. Lauterbach, and W. Weber, “Integrated microelectronics for smart textiles,” in *Workshop on Modeling, Analysis, and Middleware Support for Electronic Textiles 2002*, pp. 3–8, October 2002.
- [35] J. Horwitz, “Wearable iPod video displays, compared.” Online, January 2006. Available: <http://www.ilounge.com/index.php/articles/comments/wearable-ipod-video-displays-compared/>.
- [36] M. Jones, T. Martin, and Z. Nakad, “A service backplane for e-textile applications,” in *Workshop on Modeling, Analysis, and Middleware Support for Electronic Textiles 2002*, pp. 15–22, October 2002.
- [37] D. Lehn, C. Neely, K. Schoonover, T. Martin, and M. Jones, “e-TAGs: e-Textile Attached Gadgets,” in *Communication Networks and Distributed Systems: Modeling and Simulation*, January 2004.

- [38] J. Hill, “A software architecture supporting networked sensors,” Master’s thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 2000.
- [39] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, “System architecture directions for networked sensors,” in *Architectural Support for Programming Languages and Operating Systems*, pp. 93–104, 2000.
- [40] P. Levis and D. Culler, “Maté: a virtual machine for tiny networked sensors,” in *Architectural Support for Programming Languages and Operating Systems*, pp. 85–95, December 2002.
- [41] Sun Microsystems, *Java Card Technology*, 2006. Available: <http://java.sun.com/products/javacard/>.
- [42] P. R. Cohen, A. J. Cheyer, M. Wang, and S. C. Baeg, “An open agent architecture,” in *AAAI Spring Symposium*, pp. 1–8, March 1994.
- [43] A. Cheyer and D. Martin, “The open agent architecture,” *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4, pp. 143–148, March 2001.
- [44] Waldo, *Jini architecture overview*. SUNLABS, July 1998. Available: <http://java.sun.com/products/jini/whitepapers/architectureoverview.pdf>.
- [45] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, “Matching events in a content-based subscription system,” in *Symposium on Principles of Distributed Computing*, pp. 53–61, 1999.
- [46] A. Campailla, S. Chaki, E. M. Clarke, S. Jha, and H. Veith, “Efficient filtering in publish-subscribe systems using binary decision,” in *International Conference on Software Engineering*, pp. 443–452, 2001.
- [47] A. Carzaniga and A. Wolf, “Content-based networking: A new communication infrastructure,” in *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, October 2001.
- [48] A. Carzaniga and A. L. Wolf, “A benchmark suite for distributed publish/subscribe systems,” Tech. Rep. CU-CS-927-02, Department of Computer Science, University of Colorado, April 2002.
- [49] S. Courtenage, “Specifying and detecting composite events in content-based publish/subscribe systems,” Tech. Rep. CSCS/2002/01, Cavendish School of Computer Science, University of Westminster, 2002.
- [50] G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco, “Bringing dynamic reconfiguration into publish-subscribe systems.” Online. Available: [citeseer.nj.nec.com/499280.html](http://citeseer.nj.nec.com/499280.html).

- [51] B. P. Gerkey and M. J. Mataric, “Murdoch: Publish/subscribe task allocation for heterogeneous agents,” in *Fourth International Conference on Autonomous Agents* (C. Sierra, M. Gini, and J. S. Rosenschein, eds.), (Barcelona, Catalonia, Spain), pp. 203–204, ACM Press, 2000. Available: [citeseer.nj.nec.com/310284.html](http://citeseer.nj.nec.com/310284.html).
- [52] Atmel Corporation, “AVR 8-Bit RISC products.” Online, 2004. Available: <http://www.atmel.com/products/AVR/>.
- [53] T. Martin, D. P. Siewiorek, A. Smailagic, M. Bosworth, M. Ettus, and J. Warren, “A case study of a system-level approach to power-aware computing,” *ACM Transactions on Embedded Computing Systems*, vol. 2, pp. 255–276, August 2003.
- [54] Z. Nakad, M. Jones, and T. Martin, “Fault tolerant networks for electronic textiles,” in *The 2004 International Conference on Communications in Computing*, pp. 100–106, June 2004.
- [55] G. Spinks, B. Kim, G. Wallace, and R. John, “Electroformation of conductive polymers in a hydrogel support matrix,” *Polymer*, vol. 41, pp. 1783–1790, 2000.
- [56] D. Seal, *ARM Architecture Reference Manual, 2/E*. Boston, MA, 2001.
- [57] V. Ekanayake, C. K. IV, and R. Manohar, “An ultra-low-power processor for sensor networks,” in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cornell University, October 2004.
- [58] B. A. Warneke, *Ultra-Low Energy Architecture and Circuits for Cubic Millimeter Distributed Wireless Sensor Networks*. PhD thesis, University of California at Berkeley, 2003.
- [59] L. Welch, B. Shirazi, B. Ravindran, and F. Kamangar, “Instrumentation, modeling, and analysis of dynamic, distributed real-time systems,” tech. rep., Department of Computer Science and Engineering, University of Texas at Arlington, 1998.
- [60] L. R. Welch, P. V. Werme, L. A. Fontenot, M. W. Masters, B. Shirazi, B. Ravindran, and D. W. Mills, “Adaptive QOS and resource management using a posteriori workload characterizations,” in *IEEE Real Time Technology and Applications Symposium*, pp. 266–275, 1999.
- [61] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, “On quality of service optimization with discrete QOS options,” in *The IEEE Real-Time Technology and Applications Symposium*, pp. 276–286, 1999.
- [62] B. Ravindran and T. Hegazy, “A predictive algorithm for adaptive resource management of periodic tasks in asynchronous real-time distributed systems,” in *International Parallel and Distributed Processing Symposium*, p. 30, 2001.

- [63] B. Ravindran, L. R. Welch, C. Bruggerman, B. Shirazi, and C. Cavanaugh, “A resource management model for dynamic, scalable, dependable, real-time systems,” in *IPPS/SPDP Workshops*, pp. 931–936, 1998.
- [64] B. Ravindran, “Quality of service management in distributed asynchronous real-time systems,” in *European Conference on Parallel Processing*, pp. 489–496, 1999.
- [65] L. R. Welch, B. Ravindran, B. Shirazi, and C. Bruggeman, “Specification and modeling of dynamic, distributed real-time systems,” in *RTSS*, pp. 72–81, 1998.
- [66] S. Brandt, G. Nutt, T. Berk, and J. Mankovich, “A dynamic quality of service middleware agent for mediating application resource usage,” in *The IEEE Real-Time Systems Symposium*, pp. 307–317, 1998.
- [67] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring The Internet*. Addison-Wesley, 3rd ed., 2005.
- [68] A. Kumar, D. Manjunath, and J. Kuri, *Communication Networking: An Analytical Approach*. Morgan Kaufmann, 2004.
- [69] Sun Microsystems, *Java Native Interface: Programmer’s Guide And Specification*, 2006. Available: <http://java.sun.com/docs/books/jni/>.
- [70] D. Linden, *Handbook of Batteries*. McGraw-Hill, 3rd ed., 2002.
- [71] Energizer, *Energizer X22*. Available: <http://data.energizer.com/PDFs/X22.pdf>.

## Appendix A Running And Extending The System

This appendix is provided for the benefit of my colleagues at Virginia Tech, present and future, who wish to use or extend my work.

### Creating An Application

1. Determine all parameters necessary to configure the application or specify its state
2. Determine all periodic functions that the application might take, e.g. a sensor samples its signal
3. Determine what services the application depends on and will provide, e.g. a service can provide activity recognition and needs accelerometer data at the left and right knees
4. Creating the Ptolemy Java file
  - (a) Determine where inside of the primary Ptolemy directory to place the file
    - \* My main directory is `home/chrisz1/ptolemy`
    - \* My actors are under `home/chrisz1/ptolemy/ptolemy/actor/custom`Include the corresponding package name into the Java file, e.g. `package ptolemy.actor.custom;`
  - (b) Import the required libraries from Ptolemy, e.g.

```
import ptolemy.actor.TypedAtomicActor;
import ptolemy.actor.TypedIOPort;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.NameDuplicationException;
import ptolemy.actor.Director;
import ptolemy.actor.util.Time;
import ptolemy.data.expr.Parameter;
import ptolemy.data.Token;
import ptolemy.data.ArrayToken;
import ptolemy.data.ScalarToken;
import ptolemy.data.IntToken;
import ptolemy.data.*Token;
```

\* where \* represents any additional data types needed by parameters
  - (c) Create the class and save in file `ClassName.java` in chosen file location, e.g.



```

////////////////////////////////////
////          Methods for Native C Code to call          ////
////////////////////////////////////

//Allows C to send messages to the bus
public void sendBusMessage(int data[])
{
    try
    {
        String tmp = "(" + data[0];
        for (int c=1; c<data.length; c=c+1) tmp+= "," + data[c];
        tmp+=")";
        ArrayToken message = new ArrayToken(tmp);
        messageOut.send(0,message);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

//Allows C to schedule events
public void scheduleFire(double time)
{
    try
    {
        Director dir = getDirector();
        Time t = new Time(dir,time+dir.getModelTime().getDoubleValue());
        dir.fireAt(this,t);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

//Allows C to adjust the bus addresses it will listen to
public void adjustAddresses(int data[])
{
    try
    {
        String tmp = "(" + data[0];
        for (int c=1; c<data.length; c=c+1) tmp+= "," + data[c];
        tmp+=")";
        ArrayToken message = new ArrayToken(tmp);
        addresses.send(0,message);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

(d) Compile the java program

Under the primary Ptolemy directory, create a .h file, e.g.

```
javah -jni -d ptolemy/actor/custom ptolemy.actor.custom.ClassName
```

This creates a .h file under the specified directory, e.g.

```
ptolemy/actor/custom/ptolemy_actor_custom_ClassName.h
```

(e) Create the corresponding .c file to implement the native functions initializeC, forwardBusMessage, and fireCalled, e.g.

```

#include "ptolemy_actor_custom_ClassName.h"
#include other necessary header files

////////////////////////////////////
////          user defined data types          ////
////////////////////////////////////

*any user defined data types

////////////////////////////////////
////          function declarations          ////
////////////////////////////////////

*any declarations for additional functions that simplify readability or provide common functionality

////////////////////////////////////
////          global variables          ////
////////////////////////////////////

jmethodID idSendMessage;
jmethodID idScheduleFutureFire;
jmethodID idAdjustAddresses;
*any other global variables
*NOTE: a shared library in this context will share data among its instances, so if there is going to be more than one client
*implementing this shared library, making global variables arrays and passing a unique index for each client will save
*a lot of headaches, and because the data is shared, it will not increase overall space usage!

////////////////////////////////////
////          JNI native methods          ////
////////////////////////////////////

//Handles the case where ClassName has scheduled a timer event
JNIEXPORT void JNICALL Java_ptolemy_actor_custom_ClassName_fireCalled(JNIEnv * env, jobject obj)
{
    *any periodic functionality like a sensor sampling and providing data as a service
    *SimBurty and SimSummary c files provide excellent examples of retrieving data from mysql through parameterized procedures
}

//Handles the case where Software received a message on the I2C bus
JNIEXPORT void JNICALL Java_ptolemy_actor_custom_ClassName_forwardMessageI2C(JNIEnv * env, jobject obj, jintArray message)
{
    jint *dArray = env->GetIntArrayElements(message, 0);
    *respond to the message, which will be responses to your service/application registration messages or data provided to you
    *send output data or schedule a time to do so if the data is dependant upon received service messages
    env->ReleaseIntArrayElements(message, dArray, 0);
}

//Initializes Software to prepare it for future operations
JNIEXPORT void JNICALL Java_ptolemy_actor_custom_Software_initializeC(JNIEnv * env, jobject obj, configuration parameters)
{
    //Get JNI Method ID's
    jclass cls = env->GetObjectClass(obj);
    idSendMessage = env->GetMethodID(cls, "sendMessage", "([I)V");
    idScheduleFutureFire = env->GetMethodID(cls, "scheduleFire", "(D)V");
    idAdjustAddresses = env->GetMethodID(cls, "adjustAddresses", "([I)V");

    *initialize all global variables
    *retrieve all information from configuration parameters
    *to obtain string variables,
    jboolean throwAway;
    const char *str = env->GetStringUTFChars(stringParameterName, &throwAway);
    stringName=str;
    env->ReleaseStringUTFChars(stringParameterName, str);
    *register your application or service with Software, or schedule timer interrupt to do so later
    *to use java functions, env->CallVoidMethod(obj,JNI Method ID,parameters);
    *configure bus snooping if desired using method idAdjustAddresses
}

```

- (f) Compile the c code with
- ```

g++ -shared -I/project/software/Java/jdk1.5.0_03/include/
-I/project/software/Java/jdk1.5.0_03/include/linux
ptolemy_dir/ptolemy/actors/custom/ptolemy_actor_customClassName.c
-o ptolemy_dir/ptolemy/actors/custom/libLibraryName.so

```
- \* use -I/usr/include/mysql/ -lmysqlclient to include mysql database access
- (g) Incorporate the application into a Ptolemy topology file for use, e.g.
- ```

add,client,uniqueName,busAddress,ClassName,FaultFile,bus,
otherParameterName,otherParameter,nextParameterName,nextParameter,

```

## Configuring And Running A Ptolemy XML File

1. Create a topology file
  - (a) Identify what clients (and their parameters) you wish to run
  - (b) Determine network topology

- (c) Include all busses on the system, e.g.  
`add,i2c,busName,busSpeed,busCost`
  - (d) Include a Software Master, e.g.  
`add,client,uniqueName,busAddress,Software,FaultFile,bus,master,true,  
myAddress,busAddress,topology,'nameOfTopologyFile',  
Locations,Left knee,Left knee,Location1v2,100`
    - \* Note: Locations is a special parameter type which looks up the numerical equivalent of a text location in the database
    - \* To use, it requires that classpath be adjusted, e.g.  
`export CLASSPATH=/home/chrisz1/mysql-connector-  
java-3.1.10:$CLASSPATH`
  - (e) Include at least one Software slave on each bus, e.g.  
`add,client,uniqueName,busAddress,ClassName,FaultFile,bus,master,false,  
myAddress,busAddress,topology,'nameOfTopologyFile',  
Locations,Left ankle,Left ankle,Location1v2,100`
  - (f) Include all other clients, e.g.  
`add,client,uniqueName,busAddress,ClassName,FaultFile,busName,  
otherParameterName,otherParameter,nextParameterName,nextParameter,`
  - (g) Include all energy components as clients implementing some energy Class-Name
  - (h) Include all routers, e.g.  
`add,router,uniqueName,bus1,bus2,routerAddress,FaultFile`
  - (i) End the topology file with the command “end”
2. Convert the topology file into Ptolemy xml file using Generate function, e.g.  
`cd chrisz1/braden_temp_work/autogen (directory where Generate is located)  
java Generate topologyFile somePtolemyFile.xml`
  3. Opening and running the Ptolemy file
    - (a) cd into primary Ptolemy directory
    - (b) If using mysql databases, adjust the classpath, e.g.  
`#export CLASSPATH= /mysql-connector-java-3.1.10/mysql-connector-java-  
3.1.10-bin-g.jar:$CLASSPATH`
    - (c) open the file with vergil, e.g.  
`java -XmsLowestMemoryBound -XmxUpperMemoryBound  
ptolemy.vergil.VergilApplication somePtolemyFile.xml`
    - (d) Include Displays or File Writers to observe bus outputs if desired
    - (e) Click the run button in Ptolemy toolbar

## Editing The Conversion Process From Topology Files To XML

1. Editing the Generate.java file
  - (a) Edit the Generate.java source (currently in subversion's autogen folder)
  - (b) Recompile with javac Generate.java
2. Editing the skeleton file used for Ptolemy
  - (a) Start from the existing skeleton or create a new one in Ptolemy, e.g.

```
java ptolemy.vergil.VergilApplication
/home/chrisz1/braden_temp_work/autogen/Skeleton5.xml
```
  - (b) After creating your desired skeleton file, view the xml source
  - (c) Copy the entire file except the final closing bracket and save this as header.txt in the same directory as the Generate.java file

## Setting Up The MySQL Database With Data

1. Copy folder mysqlProcs into home directory (or update procedures accordingly)
2. Log onto mysql, e.g. `mysql -h hostComputer -u UserName -p`
3. Provide password
4. Select database by executing: `use SimData;`
  - \* If a new table is desired, edit `sp_EmptyDatabase` as required using existing code as guide
5. Change delimiter to `$` by executing the line: `delimiter $`
6. Completely empty the database by executing the line:  
`source mysqlProcs/CreateProcedures$`
7. Add data into database by executing the `./read` executable under `readReal` or `readC3D` folders in subversion
  - \* If `Locations` table is empty, execute the line: `call sp_Locations()$`
  - \* Otherwise if the `Locations` table has empty descriptions, execute the line: `call sp_LocationDescriptions()$`
  - \* If using `readReal`, execute the line: `call sp_LocationMapping()$`
  - \* If a different address to location mapping is desired, edit the `sp_LocationMapping` and execute the line: `source sp_LocationMapping$`