2011

Decoupling User Interface Design Using Libraries of Reusable Components

Rajesh Sankaran
Louisiana State University and Agricultural and Mechanical College, rajeshxsankaran@gmail.com

DECOUPLING USER INTERFACE DESIGN USING LIBRARIES OF REUSABLE COMPONENTS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Rajesh Sankaran
B.E. Electronics and Communication Engineering,
Periyar University, 2004
August 2011

Acknowledgments

In ancient India, students left the comforts of their home to live and study in the ashram of the guru. At the ashram, the guru imparted knowledge, taught the students skills and prepared them for the life ahead. It was in some sense a wholesome education. In many ways, I believe that my time spent in LSU has provided me with a similar experience.

I learned from a wonderful and skillful craftsman and builder, Prof. Brygg Ullmer, who not only introduced me to the intriguing world of user interfaces research, but also to the simple nuances and aesthetics of design. I would like to thank him for the continuous support of my Ph.D study and research, for his patience, motivation, infectious enthusiasm, immense knowledge and for teaching me novel ways to look at old things. You gave me a new set of eye glasses to ponder and rediscover the world. Your guidance has helped me in all the time of research, in personal life and writing of this thesis. Thanks Prof. Ullmer.

My sincere thanks to my co-advisor Prof. Jaganathan Ramanujam for his encouragement, insightful comments, stimulating discussions, and hard and thought-provoking questions. Many thanks Dr. Ram, for lending your analytical mind and playing the devil's advocate. This thesis has enormously benefited from those brutal, but fitting and thoughtful questions and discussions. Your LaTeX wizardry saved my day, not just once. I am very much grateful for the knowledge you have imparted for the improvement of this work. Your insights and comments are very much appreciated.

I owe my deepest gratitude to Prof. Jerry Trahan. The successful completion of this manuscript would have been impossible without his invaluable contribution. He has made available his support in a number of ways over the years, and in the last year of this dissertation research, his support

and investment in me has transcended the traditional roles and responsibilities of a committee member by several orders of magnitude. I consider him a shadow third co-advisor and a mentor. Prof. Trahan, I cannot thank you enough for the thoroughness of your reading, for the innumerable hours you spent listening to my ideas, thoughts and plans, and finally for your thoughtful comments and gestures toward improving my writing skills.

Over the past six years, Prof. Ramachandran Vaidyanathan has had a profound influence on my scientific, education and personal outlook. His mentorship, guidance and support, both in and out of the classroom and in the scope of this dissertation has played a major role in my success both in graduate school and in my personal life. Under his tutelage, I have been motivated to strive for and achieve excellence, both personally and professionally. His guidance and the confidence placed by him in my abilities has shown me the right direction to follow and succeed in very tiring times in graduate school. Thanks Dr. Vaidy.

It is an honor for me to to have had Prof. Doris Carver as my minor professor and as a member of my dissertation committee. Through our limited interactions, I have learned immensely from her and benefited from her thought provoking comments. Dr. Carver, thanks for accepting to meet with me within a short time always (even amidst your busy schedule) and for taking the time to come to Baton Rouge for my defense, even amidst your many NSF related responsibilities.

Finally, I am grateful to Prof. Scott Baldridge, who beyond his responsibilities as the dean's representative, took the time to discuss my dissertation work and progress at length, several times. I would also like to thank him for the highly motivating discussion on the dissemination of the work underlying this thesis and providing me a new insight towards entrepreneurship.

I also thank the above committee members for being flexible in their schedules so that presentation times could be agreed upon, for finding time to meet with me during the course of this dissertation work and their overall support and investment in my success.

Besides my advisors and the rest of my thesis committee, I would like to sincerely thank Dr. Christopher Branton. Chris, I am deeply indebted to you for all the patience and support you have extended to me since day one. You have saved many bridges, calmed down many fears and listened to innumerable rants - technical and personal. Thanks for the support, technical discussions and advice on thesis writing.

I would like to thank my fellow students at the Tangviz lab, Toole, Ardaud, Setty, Liu, Jandhyala, Kallakuri, Allen, DeLatin, Dell, Reeser, Rogge, Washington, Wallace, Bradford, Carroll, Dever, Diabi, Douthut, Foley, Gavin, Hess, James, Laan, Losso, Morris, Oliver, Ramb, Seidel, Sun, Tregre, Wesley-Smith, Wiggings, Wiley and others for their unwavering support, fruitful collaborations on innumerable projects and technical discussions. I have learned and gained a lot of knowledge from our interactions over the years, and the papers we have co-authored are a testament to this synergistic and successful relationship. Special thanks to Cornelius on several fronts, you have helped me to pick myself up, dust myself and keep running. And Srikanth, for noticing the protocol directions of my work very early on, and making me realize its significance.

Likewise, I extend my sincere thanks to the graduate school folks, particularly the graduate editor Ms. Susanna Dixon, folks at the LSU ISO and Beth in the ECE graduate office. Thanks much for accepting my requests and processing my documents quickly. Summer Graduation would not have been possible without your support.

This research work was supported by funding from several sources including NSF MRI-0521559, NSF Creative-IT IIS-0856065, BoR LOT RII-0704191, LSU Foundation's George Reymond Scholarship, LSU Department of Electrical Engineering Conference Travel Award and LSU Graduate School Conference Travel Award. My sincere thanks to all the above funding avenues.

Folks at Argonne, Charlie, Rob, Pete, Kaz, William, Kamil, Paul, Dane, Pat, Tamra and several others, NYSE knows no risk takers better than you! Thanks for your belief in me and for giving

me yet another reason to wrap up my thesis and travel north to Chicago. Thanks for an exceptional summer filled with wonderful interactions when I interned with ANL.

Time takes the turbo route when having fun, time does not wait for anyone. And when time flies, life does not stand still either, it unwraps to create memories, and sometimes leads to work that lands one a doctoral degree. About half a dozen years of graduate school has culminated into this PhD dissertation, the completion of which has brought about a sigh of relief on many familiar faces. Since the early days of this thesis work when time was predominantly spent on reading research manuscripts and focusing on technicalities and implementation details supporting this thesis, to the later days when "Your Paper" or "That thing you are writing" as my friends referred to my thesis occupied much of the conversations relating to me, this thesis journey has been a life changing and eye-opening one.

Writer's block, "other things are more fun" and fun, this dissertation process has seen them all. Anyway, I have finished, but not just by my individual efforts, and I am glad. Srikanth, Cornelius, Karen, Ashlen, Kristen, Jen, Susie, Kim, Key-Young, Brittany, Bety, Lohit, Monika, Rajs, Selvams, Balachandran, Bhaskar, Amanda, Adam, Erin, Chris, Zack, Landon, Kexi, Ajays, Ana, Nicole, Cole, Santanu, Lavanya, Hari, Rajiv, Marcel, Sarkars, Guohui, Ashok, Pavani, Harsha, Lea Anne, Tony, Werner and several others at CCT and LSU, remember that all your efforts and support are recognized and appreciated. Thank you so much. Especially my CCT and RRL peeps, I could not have succeeded without the invaluable support of you all. I may not have gotten to where I am today, at least not sanely. A simple "thank you" cannot suffice to express my gratitude.

My appreciation likewise extends to my immediate and extended family, particularly mom, dad, Ramya, Aarvee, Khyathi and my fiancee Abhilasha. Thanks for the belief in me, for the unconditional support and for being there when I needed you all the most. I love you all, so much. This thesis would have been simply impossible without you.

Table of Contents

List of Figures

Glossary

ACPI The Advanced Configuration and Power Interface (ACPI) specification is an open standard for unified operating system-centric device configuration and power management. It defines platform-independent interfaces for hardware discovery, configuration, power management and monitoring.

API Application programming interface (API) is an interface in computer science that defines the ways by which an application program may request services from libraries and/or operating systems. An API determines the vocabulary and calling conventions the programmer should employ to use the services. It may include specifications for routines, data structures, object classes and protocols used to communicate between the requesting software and the library

ARM The ARM is a 32-bit reduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Limited.

ARP In computer networking, the Address Resolution Protocol (ARP) is the method for finding a host's link layer (hardware) address when only its Internet Layer (IP) or some other Network Layer address is known.

CAN bus Controller–area network (CAN or CAN-bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer.

Coreboard Motherboard

COTS Commercial, off-the-shelf (COTS) is a term for software or hardware, generally technology or computer products, that are ready-made and available for sale, lease, or license to the general public.

Daughterboard A daughterboard or daughtercard is a circuit board meant to be an extension of a motherboard (or 'mainboard'), or occasionally of another card.

DHCP Dynamic Host Configuration Protocol (DHCP) is a network application protocol used by devices (DHCP clients) to obtain configuration information for operation in an Internet Protocol network. This protocol reduces system administration workload, allowing networks to add devices with little or no manual intervention.

DSN Digital Serial Number (DSN) is a unique software-readable serial number hard-coded within Dallas Semiconductor 1-wire devices.

GUI Graphical User Interface

HCI Human–computer interaction

I^2C Inter-Integrated Circuit (I^2C) is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone. The name is pronounced eye-squared-see or eye-two-see.

I/O In computing, input/output, and I/O refer to the communication between an information processing system (such as a computer), and the outside world – possibly a human, or another information processing system.

LSB In computing, the least significant bit (lsb) is the bit position in a binary number having the lowest value.

Modality Modality refers to the path of communication between the human and the computer, such as vision or touch

Motherboard A motherboard is the central printed circuit board (PCB) in some complex electronic systems, such as modern personal computers, typically host to the central coordinating processor.

MSB In computing, the most significant bit (msb) is the bit position in a binary number having the greatest value.

PCB A printed circuit board (PCB) is used to mechanically support and electrically connect electronic components using conductive pathways, or traces, etched from copper sheets laminated onto a non-conductive substrate.

PIC PIC is a family of Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1640 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to Peripheral Interface Controller.

PRN A Pseudorandom number (PRN) is one of a sequence of numbers generated by a deterministic algorithm, where the sequence satisfies one or more of the standard tests for statistical randomness.

PRNG A Pseudorandom number generator (PRNG) is a deterministic algorithm that can generate a PRN.

RFID Radio-frequency identification (RFID) refers to the use of an object (typically referred to as an RFID tag) applied to or incorporated into a product, animal, or person for the purpose of identification and tracking using radio waves. Some tags can be read from several meters away and beyond the line of sight of the reader.

RS232 RS–232 (Recommended Standard 232) is a standard for serial binary data signals connecting between a DTE (Data Terminal Equipment) and a DCE (Data Circuit-terminating Equipment). It is commonly used in computer serial ports

RTL Round Trip Latency

TCP/IP The Internet Protocol Suite (commonly known as TCP/IP) is the set of communications protocols used for the Internet and other similar networks. Its name derives from two of its most important protocols: the Transmission Control Protocol (TCP) and the Internet Protocol (IP), which were the first two networking protocols defined in this standard.

TDMA Time division multiple access (TDMA) is a channel access method for shared medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots.

Through-hole A mounting scheme used for electronic components that involves the use of pins on the components that are inserted into holes (PTH - Plated Through-Hole) drilled in printed circuit boards (PCB) and soldered to pads on the opposite side

TUI Tangible User Interface

UI User Interface

USB Universal Serial Bus

Abstract

The integration of electronic and mechanical hardware, software and interaction design presents a challenging design space for researchers developing physical user interfaces and interactive artifacts. Currently in the academic research community, physical user interfaces and interactive artifacts are predominantly designed and prototyped either as one-off instances from the ground up, or using functionally rich hardware toolkits and prototyping systems. During this prototyping phase, undertaking an integral design of the interface or interactive artifact's electronic hardware is frequently constraining due to the tight couplings between the different design realms and the typical need for iterations as the design matures. Several current toolkit designs have consequently embraced component-sharing and component-swapping modular designs with a view to extending flexibility and improving researcher freedom by disentangling and softening the cause-effect couplings.

Encouraged by early successes of these toolkits, this research work strives to further enhance these freedoms by pursuing an alternative style and dimension of hardware modularity. Another motivation is our goal to facilitate the design and development of certain classes of interfaces and interactive artifacts for which current electronic design approaches are argued to be restrictively constraining (e.g., relating to scale and complexity). Unfortunately, this goal of a new platform architecture is met with conceptual and technical challenges on the embedded system networking front.

In response, this research investigates and extends a growing field of multi-module distributed embedded systems. We identify and characterize a sub-class of these systems, calling them embedded aggregates. We then outline and develop a framework for realizing the embedded aggregate class of systems. Toward this end, this thesis examines several architectures, topologies

and communication protocols, making the case for and substantial steps toward the development of a suite of networking protocols and control algorithms to support embedded aggregates. We define a set of protocols, mechanisms and communication packets that collectively form the underlying framework for the aggregates. Following the aggregates design, we develop blades and tiles to support user interface researchers.

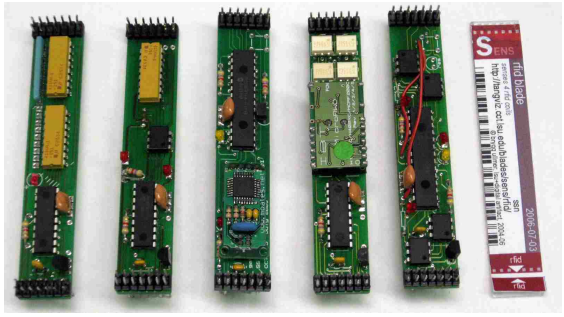
Chapter 1

Introduction and Motivation

> No river can return to its source, yet all rivers must have a beginning.
>
> Native American proverb

Evolving incarnations of computers and computational systems have occupied increasingly broader and impactful roles in society. This trajectory began with mechanical elements and counting machines like the abacus and has evolved into sophisticated electronic symbol processing entities. As a ubiquitous and indispensable part of the 21st century, these machines range from the embedded computers within hearing aids and pacemakers to large clusters of supercomputers that perform 10^{15} floating point instructions per second. Users interact with computers through user interfaces and interactive artifacts, which are expressed or embodied through ensembles of physical, graphical and potentially other mediations, and technically mediated by underlying electronics and software entities. This dissertation work makes contributions toward advancing the design and realization of new forms of interaction devices, some of them expressing discrete and continuous input to traditional variations on graphical displays and others realizing interactive systems of quite different nature.

In the academic research community, user interfaces (UIs) and artifacts are realized either as one-off implementations or by using hardware toolkits and prototyping platforms. Several of the current hardware toolkits and prototyping systems incorporate principles of modular design. In this thesis, we suggest an alternative modular electronic hardware approach toward supporting certain kinds of user interfaces, whose realization, we argue in § 1.2, is infeasible using current toolkit implementations. We propose blades and tiles (Figure 1.1)—a modular electronic system structured around bus modularity—for prototyping user interfaces and interactive artifacts. Our



(a) **Close-up of individual Blades**

(b) **Underside of a param-casket, a tangible user interface built using blades.**

FIGURE 1.1: Blades - embedded aggregates for HCI. Blades are functional modules designed specifically toward enabling human computer interaction researchers, enthusiasts and practitioners in the design, prototyping and realization of physical user interfaces and interactive artifacts. A networked collection of blades, as illustrated in subfigure (**b**), is an instance of an embedded aggregate. Subfigure (**a**) shows, from left, switch-LED, haptic rotor, USB-gateway, RFID, and intracomm blades, and a printed blade label affixed to the bottom of a blade. The bar-coded and color-coded blade-labels provide blade-specific details.

belief is that there is a gap in the embedded communication protocol spectrum that makes developing blades and tiles class of systems a formidable challenge. As part of this thesis work, we study this class of embedded systems[1] and develop a framework for realizing them for use in a variety of contexts. We identify and characterize this class of systems and develop the underlying communication fabric and infrastructure from the ground up by defining new protocols and developing communication mechanisms.

1.1 User Interfaces, Hardware Toolkits and Prototyping Platforms

The field of human-computer interaction (HCI) concerns itself with studying the interaction of users with computers and computationally-mediated systems. Hewett *et.al.* [HBC+05] define human-computer interaction as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them." Recent years have been a fertile time for new paradigms

[1] An embedded system is a microprocessor-based system that is built to control a function or a range of functions [Hea03].

in user interfaces and interaction. Examples include Attentive UI [MMC+00, Ver02, Ver03], Conversational UI [McT02, Blo04, McT05], Gesture UI [KK96, OCW+00, OC00], Tangible UI (TUI) [IU97, UI97, Ish09, Sha10] and Object oriented UI [Col94], to name a few. Several of these user interfaces (UIs) incorporate innovations involving hardware electronics that convert analog human inputs to digital computer-understandable electrical signals. In parallel, post-desktop models of HCI called Ubiquitous Computing and Pervasive Computing, and the larger umbrella of Physical Computing which encompasses ecologies of physical interactive objects surrounding humans in daily life [OI04, ID+04, Sta05], involve several such hardware and software entities embedded in ordinary and widespread objects. The design and evaluation of user interfaces and interactive artifacts has followed a set of three design steps [TT90, SRP07, Bor08]: (1) study of users and tasks, (2) empirical analysis of the designed interface, and (3) iterative refinement of the interface based on tests, results and empirical analysis [NB93, Bai93, BBLZ96]. This thesis concerns itself with a subset of these hardware interfaces and strives to advance the mid-to-later phase of the prototyping process involved. These phases of prototyping typically involve the refinement of the final shape, look and size of the interface and realization of a fully functional electronic hardware implementation, leading to the successful creation of a UI through an iterative series of prototypes [LM95, HM06a].

User interfaces and interactive artifacts research bring together researchers from the expertise domains of electronic hardware, software, mechanical and product design. From an implementation viewpoint, user interfaces and interactive artifacts are composed of several interaction modalities. The term modality[2] in this thesis refers to interaction elements or components including, for example switches, sliders, rotors, LCDs and LEDs. The functional implementations of these modalities include electronic hardware, firmware and software; and access to these modalities is extended by the physical form/tangible components (mechware) of

[2]We discuss our use of this term in detail in § 2.2.3 on page 38.

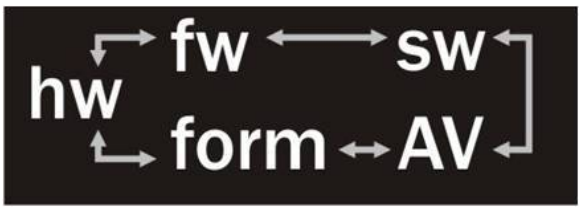


FIGURE 1.2: Coupling between the different design realms. UI design often involves iterations in its electronic hardware (hw), firmware (fw), system software (sw), audio and visual interaction channels (AV) and mechanical structure (form). Typical coupling between the different design realms in a UI prototyping process are shown by the arrows. For example, the change in the hardware implementation of the device often requires modification of the associated firmware.

the device. Currently, the electronic hardware realization during the prototyping process is done both as one-off designs and by using an emerging wave of UI hardware toolkits and prototyping platforms [GF01, VSG02, BRSB03, BG03, LAH+04, KA05, KLLL04, KEHO06, Jol06, HM06a, KTON06].

During the development of an interaction device, the various design domains impact the decisions and implementations in the other domains as shown in Figure 1.2. For example, the change in the size, shape and construction of an interaction device and orientation or spatial location of the modalities by a product designer often requires the redesign of the hardware. Simultaneously, hardware redesign can require software and firmware modifications, thus introducing a tight cause-effect coupling between the different design realms. Several of the current hardware toolkits and prototyping systems have accomplished a level of flexibility by adhering to principles of modular design. The hardware abstraction extended by the modular designs have also disentangled the strong linkages between the different design realms to a limited extent.

Modular designs have been adopted by HCI researchers for over 30 years. Perhaps the earliest use of modularity in HCI can be traced back to the 1980's when several building block type systems were developed for 3D geometric modeling [Ais79, AN84]. In product design, six variations are considered the widely employed typology of modularity [Mar07]. Ulrich and Tung [Ulr] introduce three types of modularity, namely, component-swapping modularity,

component-sharing modularity[3] and bus modularity. Pine and Davis [PD93] expand upon Ulrich and Tung's classifications with the addition of three more types, namely cut-to-fit modularity, mix modularity and sectional modularity.

Out of these six types, several of the generic user interfaces and interactive artifacts prototyping platforms and DIY systems including Phidgets, Bug Modules, Arduino and Microsoft .NET Gadgeteer[4] fall into the component-sharing[5] (see Figure 1.3-a) and component-swapping modularity (see Figure 1.3-b). Each of the systems consist of a variety of basic coreboards which can be paired with a variety of plug-in modules. The individual coreboards have the ability to support more than one kind of plug-in, and the plug-ins each can be paired with a variety of coreboards. This design can also be related to Dahmus *et al.'s* approach toward modularity [JDO01]. Phidgets, Arduino and Microsoft .NET Gadgeteer predominantly fall in their "fixed platform with various add-on modules" as illustrated in Figure 1.4-b. The coreboard + plugin modules approach has provided considerable flexibility to the designers of user interfaces and interactive artifacts, enabling them with the freedom to choose the modalities they wish to use and interface them with the coreboard.

By providing easier access to a wide variety of interaction modalities in a flexible fashion, the UI hardware toolkits and prototyping systems have made significant progress toward bootstrapping the initial development of user interfaces and interactive artifacts, especially from the perspective of software developers. The future of user interface and interactive artifact design begs for means to develop hardware that might be highly specialized in form and function, perhaps involving hundreds of sensors or displays in physically or logically descriptive designs. We believe

[3]Several researchers argue that the distinction between these two types is a matter of degree and they are complementary to each other. The difference between them lies in how the basic product and the components are defined [HK98, Mar07].

[4]The careful reader will have recognized that here we only consider platforms that are not context and interface specific unlike several of the cubical and other modular systems introduced and discussed in § 2.1. For convenience, in the following pages by prototyping platforms we refer to these generic platforms.

[5]In associating the toolkits to the component-sharing modularity and component-swapping modularity, we relax Ulrich and Tung's strict notion where only one component is connected to a coreboard at a given time, and interpret it as one or more.

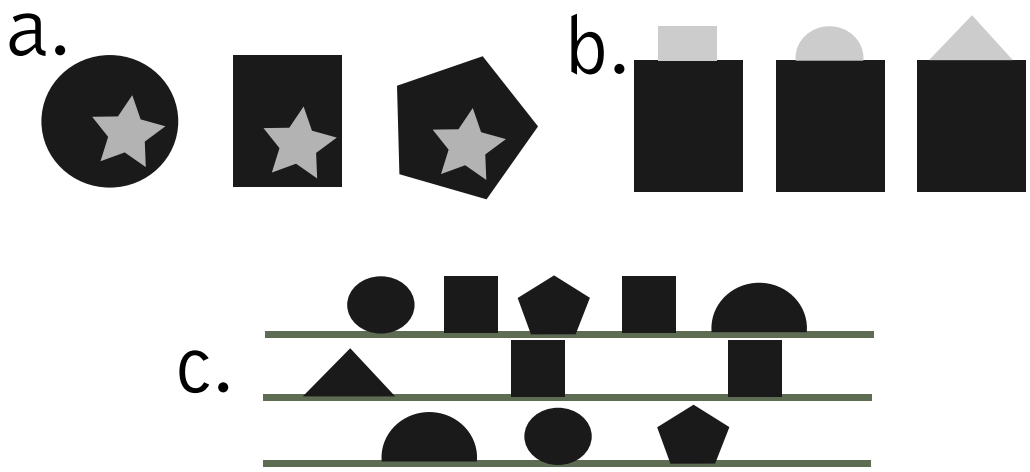


FIGURE 1.3: Modularity typology. a. Component-sharing modularity b. Component-swapping modularity and c. Bus modularity. [Ulr]

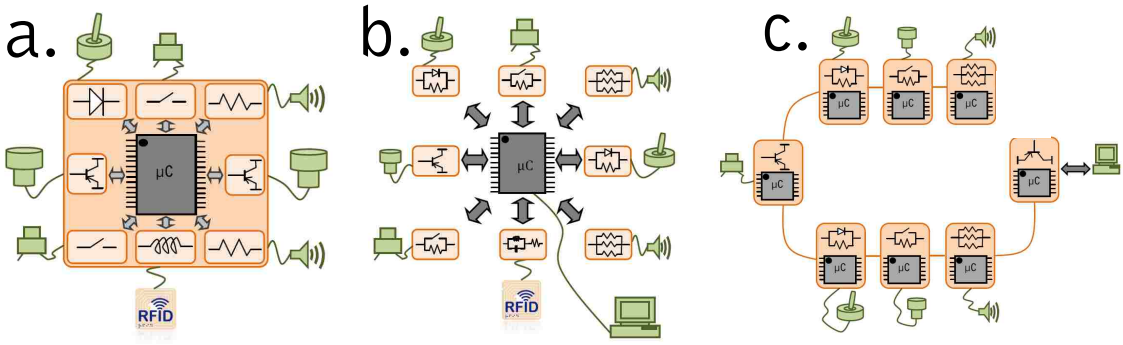


FIGURE 1.4: Electronic implementation with varying levels of modularity. a. Monolithic implementation commonly seen in final production version of UIs. b. Fixed platform with various add-on modules. c. A system realized using a collection of self-sufficient modules.

interaction devices of this kind should be possible to build, ideally by a small team with limited resources. However, for several reasons, we anticipate and will argue in the following section that systems of this scale may be difficult to realize with the available hardware toolkits.

1.2 Motivations

The coupling (see Figure 1.2) along with the fact that the various developers involved in the process speak unique design languages and use diverse sets of tools, introduces additional challenges in the prototyping process. To tackle the interface as first class citizens, the researchers need substantial

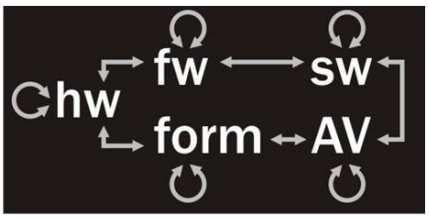


FIGURE 1.5: Decoupled iterative design process. The figure shows the prototyping experience as expected by designers, with the freedom to iterate independently in their design domains. Note the cycles on each design axis. These cycles represent the expectations of the designers to be able to iterate their designs, without forcing changes in other design axes.

freedom as illustrated in Figure 1.5 to iterate their designs, while collaborating on the overall form, function and usability of the interaction device.

To illustrate further, we imagine that the different collaborators participating in the design process have the following expectations:

I. hardware designers: ability to add interaction capabilities (i.e., sensing, display and actuation), without dependency on kind or number of modalities in use.

II. software designers: ability to access and support hardware capabilities through clearly modularized and abstracted libraries that represent and interact with hardware, while maintaining tolerance for varying network bandwidth, latency and reliability.

III. interaction designers: ability to add, remove, and migrate physical interaction elements (both at the level of physical interactors and supporting electronics) with confidence that the software, firmware and electronics will continue to function.

Supported by the hardware abstraction extended by the modular approaches, the current toolkits

and prototyping platforms have disentangled the strong linkages between the different design realms to a limited extent and satisfied the above expectations partially. However, the systems we envision quickly push the toolkits and the prototyping platforms to their functional limits as we describe next.

Toolkits are aimed at providing designers with the freedom to easily mix and match the modalities, both in terms of the kinds of modalities and quantities of each modality. The underlying modular hardware implementation of the current toolkits and prototyping platforms has typically involved a powerful embedded CPU as its main processor in the coreboard, with the various interaction modalities implemented as add-on peripherals (plug-in modules). The processing capabilities of the peripherals vary from dumb switches that are connected to digital or analog I/O pins of the core-processor, to sophisticated rotors providing haptic force-feedback that are interfaced through a high-level communication bus like USB.

The freedom and comfort enjoyed by the HCI researchers vanishes when component-sharing and component-substitution modular designs reach scalability limits due to a variety of reasons. From a hardware realization perspective, while this single coreboard based design offers a compact and simple solution, this design suffers from traditional scalability bottleneck due to limited input/output (I/O) pins (both number and kind) on the primary embedded processor of the coreboard[6]. The scalability bottleneck unfortunately introduces restrictions on the above mentioned freedom goal, by both constraining the number and types of modalities that can be used simultaneously.

Another challenge encountered is related to the processing power of the embedded CPU. Generally, the demand for data processing[7] on an individual modality basis can often be comfortably handled

[6]It can argued that these limits can be stretched by using expansion boards and multiplexers, however only to a limited extent.

[7]Other than manipulating digital data, there is often a need to convert data between physical and digital domains. The Nyquist-Shannon sampling limit [Jer77] constraints of analog (physical) to digital conversion introduce time (frequency) constraints on the capturing or generation and processing of stimulus and response.

by processors with modest capabilities like micro-controllers[8]. However, interaction sequences that include simultaneous human engagement with a combination of these time-sensitive modalities often quickly push the processing power requirements up. Often in such cases, while the coreboard can support the plugin modalities without interconnect constraints, the CPU of the board may be unable to support the computational requirements of the modalities. Since the toolkits and prototyping platforms provide researchers with a variety of modalities, it is only realistic to expect that researchers will often use them in groups and design interaction sequences that engage more than one of them simultaneously. A certain immunity from scale in this context can be provided by using powerful processors, including multi-core chips. However, the sophisticated program development environments, strong electronics skills and expensive instrumentation to develop interfaces with them are often largely unavailable in software and design oriented interaction research groups. Furthermore, increase in the complexity of using the toolkit or platform only hinders its uptake amongst the researchers.

Another issue is cost. As the researchers designing and developing the toolkits cannot predict (with any reasonable accuracy) how other researchers will utilize their system, engineering the system with the right kind of processors is impossible. Powerful processors are more expensive than their humble counterparts, and employing a powerful processor in all coreboards can be expensive and wasteful on the one hand. On the other hand, researchers tend to push systems to their limits toward achieving scientific breakthroughs, and embedded CPU restrictions only hinder the scientific process. Toolkit developers often overcome this problem to an extent by developing multiple coreboards with different core-processors (component-sharing modularity).

Beyond the above hardware concerns, the design also introduces firmware challenges. First, the modality agnosticism that toolkits aim to embrace requires the underlying firmware to support several modalities useful to designers, often with minimum firmware modifications for greater

[8]More powerful DSPs or ASIC processors may be required for handling even a single instance of more CPU demanding modalities.

usability. Secondly, with sequentially executed byte-code in single-core embedded-CPUs, the multiplicity and coincidence of modality-specific events requires concurrent servicing of each modality by time-slicing. Both the above requirements increase firmware complexity with larger number and types of modalities.

In order to support prototyping of UIs using several hundred interaction elements in a scalable and easily approachable fashion while providing modality support and features extendable by user developers and the community, the above challenges have to be overcome by the next generation of toolkits. Also, we believe that there is scope for further immunity from couplings between design realms. The challenges described here beg for better alternative approaches toward toolkit realization, perhaps by a design that can satisfy the processing needs by employing several simple processors and by exploiting the inherent parallelisms in the engagement with the different modalities. Consequently, this research aims to pursue the challenges in current modality scaling and coupling by introducing an alternative modular approach that is discussed in the following section.

1.3 Approach and Methodology

Modularization is often a goal of good design due to the functional independence it affords [PB96, GPA99]. The predominant approach amongst current toolkits is to divide the interface into easily replaceable modality-modules and perform the associated processing on a single embedded processor. We extend this modularity further and designate independent embedded processors to each modality. In our divide and conquer [HZ83] approach, a user interface or an interaction artifact is realized by interconnecting several modality modules each of which implement both the modality-specific components (transducers and other circuits) and an embedded processor for modality-related computation. We call our system blades and tiles (Figure 1.1). Our approach (Figure 1.4-c) aligns with Dahmus *et al.'s* preferred approach of "system is a collection of modules,

each of which can be one of several types" for architecting product families with heterogeneous users with heterogeneous demands [JDO01].

Of the four other types of modularity, our approach and the "pin and play" system [VVS+03] fit the bus modularity. According to Ulrich and Tung, bus modularity involves a standard structure with two or more interfaces that can attach any selection of components from a set of component types as shown in Figure 1.3-c. Extending Ulrich and Tung's [Ulr] arguments on component-swapping and component-sharing modularity, Pine and Davis [PD93] argue and suggest that while component-swapping, component-sharing and cut-to-fit modularity only allow variation in the type of component used, bus modularity extends flexibility in the choice, number and location[9] of components [Ulr, PD93]. This stand is immensely relevant to prototyping of user interfaces and interactive artifacts, and supports our design decision. During the prototyping phase, improved flexibility in changing the number of instances and location of an interaction modality are immensely beneficial to interaction and product designers as argued in Chapter 5.

Embedded systems that are collections of several electronic modules are relatively widespread including, for example,in automobile processor networks, hierarchical control systems, and networked control systems. In this vast ecosystem of multi-module systems, the class of embedded systems that model our blades and tiles distinguish themselves by the scale and distribution of their processing power. Our background research suggests that this class of systems is largely underexplored with potential applications including, for example, environmental sensing and robotics (see Chapter 3). We call them embedded aggregates.

A framework for embedded aggregates developed as part of this research, similar to a software framework [Pas02], is envisioned to provide a hardware library consisting of interoperable modules

[9]In the toolkit context, this strict difference is absent. Due to the malleable and flexible nature of interconnects that often connect the plug-in modules to the coreboard and the fact that each coreboard often supports more than one module simultaneously, some of these differences are overcome on a limited basis. For example, the placement of the modules can often be negotiated in a space around the coreboard as long as the interfacing protocol between the two supports the distance of separation.

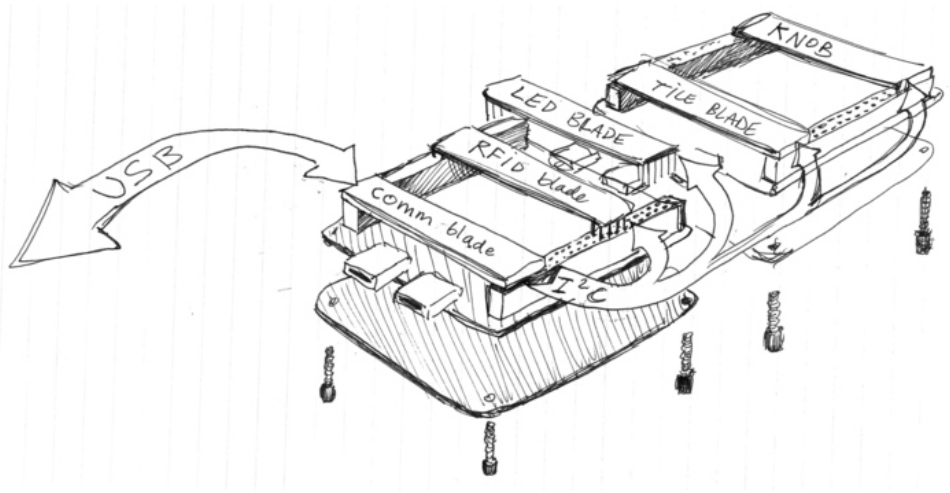


FIGURE 1.6: Artistic illustration of blades and tiles . An early illustration of the blades and tiles showing the communication and hardware mechanisms. Different blades are identified in the illustration. The large rectangular boards on which blades are affixed are called tiles. Interaction tiles used in our UI research are typically 10x10x1 cm in dimension. [Illustration by Laan, 2006]

and associated mechanisms to facilitate their operation. Towards this end, the framework outlines hardware, firmware, software and communication specifications. Using the embedded aggregate framework, we have designed and implemented blades and tiles (illustrated in Figure 1.6), a hardware prototyping platform for UI researchers. Blades and tiles inherit the embedded aggregate principles of scalability, re-usability, reliability and flexibility. Blades provide diverse modality capabilities in modular, interchangeable form. These capabilities include sensing, display, actuation, communication, computation, and power. Tiles serve as a compositional mechanism which typically couple ensembles of blades to ensembles of interactors (RFID coils, LEDs, etc.). Tiles typically contain few active components, mostly relating to power and user interaction, while delegating supporting electronics and computation to blades.

1.4 Research Contributions

User interface design can be informed and supported by both human factors and engineering perspectives, while simultaneously leveraging ideas from other research domains. By the interdisciplinary nature of this dissertation research cutting across HCI, computer networks and

embedded systems, several challenges have had to be addressed, with resulting contributions in both conceptual and implementation directions. This research revisits modular hardware approaches toward achieving further immunity from couplings between the various realms of UI design and prototyping (Figure 1.5). The primary objective of this dissertation research is to facilitate and enhance the decoupled design, development and realization of a class of user interfaces and interaction artifacts. The class of systems includes functional prototypes and evolving research deployments with several hundred interaction elements arranged over 3D spaces larger than ones attainable by current toolkits. The contributions of this dissertation research are as follows.

- We identify the challenges in the design and development of electronic user interfaces and interaction artifacts, and motivate an alternative modular hardware approach as means to further decrease the couplings between design realms, increase the design independence of the various experts and realize highly specialized user interfaces and interaction artifacts incorporating hundreds of interaction elements.
- We identify a class of multi-module embedded systems that share characteristics with our envisioned underlying modular electronics. We introduce the concept of embedded aggregates, and study their features and requirements.
- We present a holistic design of the embedded aggregate framework and develop a set of communication protocols, specifications and mechanisms that provide the foundations for realizing the aggregate class of systems.
- We develop an instance of the embedded aggregates called blade and tiles for HCI practitioners.

In short, this research has the following components.

- A specification for modular electronic hardware design and a library of such modules aimed toward UI design. The hardware specification includes physical, logical and electrical descriptions.
- Communication protocols for realizing such multi-module systems. This involves the design of the communication protocol suite with a holistic view including optimizations across the physical layer, MAC, and a compound datalink-transport layer.
- Algorithms and mechanisms toward realizing such embedded systems as open-ended designs with the core principles of scalability, flexibility and hardware-firmware reusability.
- A system software architecture for tethering electronic systems built using the multi-module framework to personal computers.

1.5 Dissertation Overview

The rest of thesis is organized as follows. Chapter 2 first presents a discussion on different research efforts that are related to the pursuits in this dissertation. Next, the remainder of Chapter 2 presents a discussion on a few scientific terms, clarifying our use of these terms for the purposes of this dissertation.

We develop and explore our concept of an embedded aggregate in Chapter 3. After a brief introduction, we motivate the need for aggregates using a variety of application contexts and highlight their core features. We then study the embedded aggregates juxtaposing them with systems and situations that share conceptual similarities to the aggregates. Following this study, we situate the aggregates in the vast ecology of embedded systems as a groundwork toward

understanding the underlying infrastructure needed for realizing them. The chapter closes with a discussion the embedded aggregate communication fabric and the envisioned framework.

With the insights gained from Chapter 3, a holistic design of the embedded aggregates is presented in Chapter 4. Beginning with the physical and logical topologies, this chapter explores and develops the communication fabric including packets, addressing methodologies, routing schemes, flow control and high level communication and control mechanisms. The chapter finally concludes with evaluation of some of the mechanisms toward their satisfiability in achieving the aggregates' design goals.

Next, Chapter 5 presents blades and tiles, the dissertation's core implementation realizing an embodiment of the embedded aggregates concepts and framework. We discuss the evolution of the firmware, software and hardware components and present an overview of the currently developed blades. A software architecture for using embedded aggregates in bladed form with a personal computer or other form of computational node is also discussed in this chapter.

Conclusions, suggestions for future work and discussions are presented in Chapter 6. The chapter gathers the results of the research work and elaborates envisioned future directions. Discussions on embedded aggregates in general and the blades and tiles in particular are also presented.

Chapter 2

Related Work and Terminology

> Various paths are tried; some are abandoned, others are pushed further. Before a solution is found, a great many paths of the maze may be explored.
>
> Herbert A. Simon

In the previous chapter we introduced the motivations behind this dissertation research and summarized the research contributions. Moving forward, this chapter initially explores existing scientific literature and presents a discussion of various research efforts that are related to the pursuits in this dissertation. Next, the remainder of the chapter presents a discussion on a few scientific terms, clarifying our use of these terms for the purposes of this dissertation.

2.1 Related Work

Over the last few decades, several modular hardware platforms/toolkits have been developed toward realizing a multitude of UIs and for supporting 3D and physical modeling. Additionally, hardware platforms designed for use in sensor networks research and systems supporting the electronic DIY[1] enthusiast markets are also increasingly being used by UI researchers. In the following pages, we will describe some of these efforts that hold relevance to this dissertation work. While existing platforms serve as the basis and motivation for the efforts in this thesis, many research pursuits in the field of robotics, tradition computer networking, spacecraft systems etc. have shaped our design and development process. Hence, following the descriptions of

[1] Do it yourself (or DIY) is a term used to describe building, modifying, or repairing of something without the aid of experts or professionals. The phrase "do it yourself" came into common usage in the 1950s in reference to home improvement projects that people might choose to complete independently.

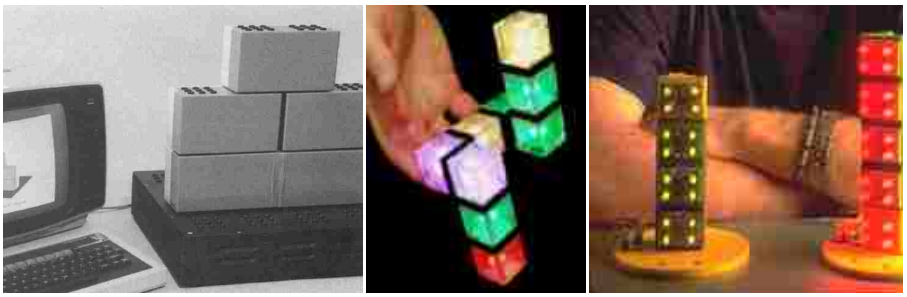


FIGURE 2.1: BBS, Peano and Stackables. (Left to Right) 1. The building blocks system by Aish and Noaks [Ais79, AN84], 2. Peano by Heaton [Hea00] and 3. Stackables by Kramer and Minar [KM97].

FIGURE 2.2: Frazer's modular systems [Fra95]. (Left to Right) 1. Universal Constructor, 2. Flexible Intelligent Modeling System and 3. Three - Dimensional Intelligent Modeling System.

the hardware platforms and toolkits, we introduce and discuss some of these relevant research directions.

Several of the earlier tangible interface designs were motivated by the building block nature of the LEGO^{TM} bricks, often leading to modular and electronically enhanced mechanisms to build physical world systems. The thought of a tangible 3D geometric modeling system incorporating building blocks with embedded computation has been researched by several groups over the past 30 years. In early 1980s, Aish and Noaks [Ais79, AN84] implemented a "building blocks system" (see Figure 2.1) aimed at modeling physical-world structures. This system was used to study both the geometric structure and thermal performance of a building being analyzed using it.

The mid 1980's also saw the development of various "intelligent modeling systems" and "machine readable models" by Frazer [FFF80, FFF81, FFF82]. Frazer developed "universal constructor" [Fra95], which was a collection of modular cubes that plugged into each other and

FIGURE 2.3: Navigational Blocks, Kramer's Tiles and AlgoBlocks. (Left to Right) 1. Camarata's Navigational Blocks [KCJ02a], 2. Tiles by Kramer [Kra98] and 3. AlgoBlocks by Suzuki and Kato [SK95].

helped build 3D reconfigurable models of cellular automata (see Figure 2.2). Extending this work, Kramer and Minar developed the Stackables [KM97] with distributed display capabilities (see Figure 2.1). Kramer also developed the Tiles [Kra98] which introduced mobile code (see Figure 2.3). Following the cubic structure of the Universal Constructor and Stackables, Heaton's Peano [Hea00] added touch sensing (see Figure 2.1). Expanding on prior work by Aish in late 70s, and subsequent research by Frazer in early 80s, and Anagnostou, Dewey and Patera's work on processors for manipulating the geometry of 3D models (Geometry Defining Processors) [DP88, ADP89] in late 80s, Anderson *et al.*'s Computational Building Blocks [AFM$^+$00] facilitate computer modeling with LEGO^{TM}-like blocks (see Figure 2.4).

Several modular systems were also developed toward research in tangible programming. Suzuki and Kato designed the AlgoBlocks [SK95], a collection of aluminum blocks that plugged into each other (see Figure 2.3). Each of these blocks represented a command, control (buttons to start and stop programs, knobs and levers to change parameters) or provided an output mechanism to help users physically program a LOGO-like language. Pursuing "user programming using physical objects" theme, McNerney developed the Programming Bricks [McN99], comprised of an ensemble of LEGO^{TM} bricks for functional programming.

The tangible programming concept has been extended toward hobby roboticists in a few systems developed. One such system is roBlocks [SG06] developed by Schweikardt and Gross. roBlocks

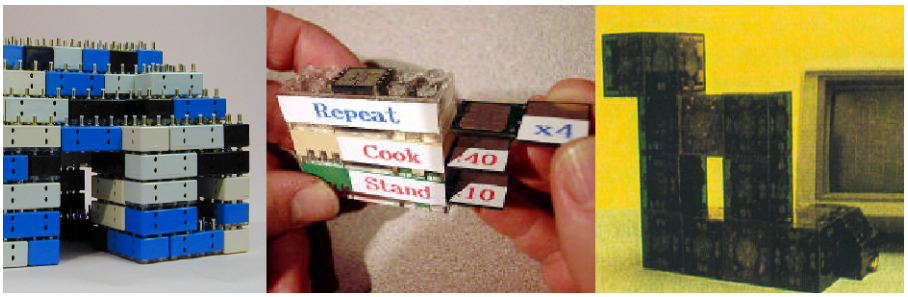


FIGURE 2.4: CBB, Programming Bricks and Geometry Defining Processors. (Left to Right) 1. Computational Building Blocks by Anderson *et al.* [AFM+00], 2. Programming Bricks by McNerny [McN99] and 3. Geometry Defining Processors by Dewey and Patera [ADP89].

is a construction kit that offers a tangible concurrent programming environment that encapsulates sensory, kinetic and computational behavior. These blocks snap together magnetically to construct robots. Schweikardt and Sitti extended their research into robotics and developed the StickyBricks [SG08], a modular system that comprised 30mm cubes with adhesive belts around their perimeter. This system was aimed toward fine grained modular self-reconfigurable robots and introduced adhesion to connect modules in place of traditional magnetic or mechanical means.

Active Cubes [KIMK00] incorporated a variety of sensing and actuation mechanisms and were snappable blocks that could be used to build responsive 3D structures (see Figure 2.5). The cubes could be uniquely identified and 3D spatial arrangement sensed in real-time. Active Cubes and Cognitive Cubes [SIW+02] were developed by Kitamura *et al.* In addition to the above cuboid modular systems, several other systems have been explored including Navigational Blocks [KCJ02a, KCJ02b], computational blocks for children [EEG+02], FlexM [ECDG06] and Computation and Construction Kits [EBE04] by Eisenberg, Buechley and Elumeze.

Modular hardware shaped in non-cuboidal forms has been used in several tangible interfaces. Oriented toward story telling and visual/tangible programming, systems have combined several elements to provide audio, visual and kinetic input and feedback. The Triangles system [GO97, GOI98] by Gorbet and Orth was intended to be used as a physical/interaction toolkit and consisted of small modular triangles (see Figure 2.5). The Triangles system was used for story telling,

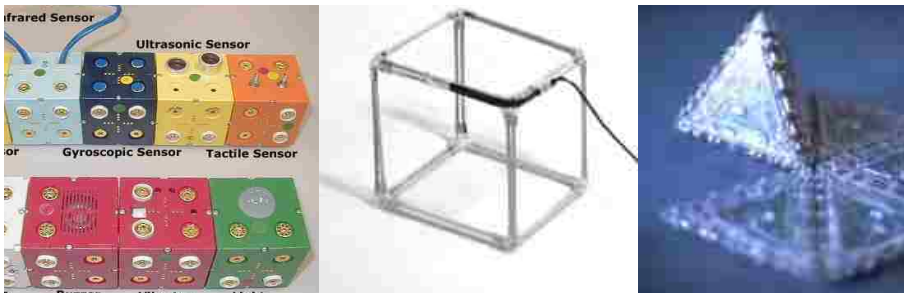


FIGURE 2.5: Active Cubes, FlexM and Triangles. (Left to Right) 1. Active Cubes by Kitamura *et al.* [KIMK00], 2. FlexM by Camarata *et al.* [ECDG06] and 3. Triangles by Gorbet and Orth [GO97, GOI98].

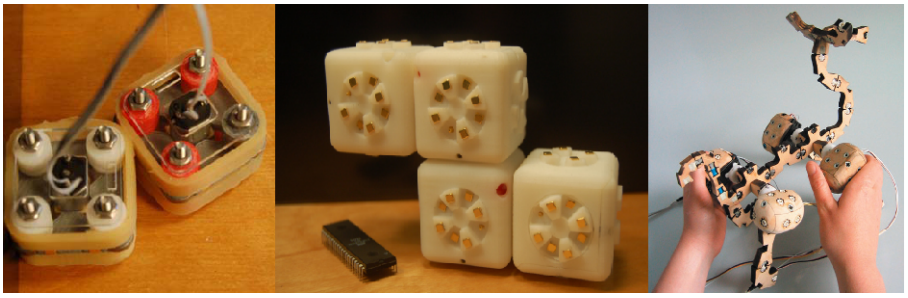


FIGURE 2.6: StickyBricks, roBlocks and Tobopo 3-D. (Left to Right) 1. StickyBricks by Schweikardt and Sitti [SG08], 2. roBlocks by Schweikardt and Gross [SG06] and 3. Topobo 3-D by Raffle *et al.* [RPI04].

for example in the "Cinderella 2000", which resembled a reactive audio comic book. Topobo 3-D [RPI04], a constructive assembly system, was developed by Raffle *et al.*, and uses many modules with built-in servo-motors and potentiometers to realize a user configurable system with kinetic memory (see Figure 2.6). The system uses a 4-channel 2-wire custom bit-banging protocol to transmit data between the kinetic modules and nodes.

Our work has been inspired in part by hardware prototyping toolkits such as Phidgets [GF01], Teleo [Mak09] and Arduino [Jol06]. Greenberg *et al.* developed the Phidgets "plug and play" building components that provide easy access to hardware sensors, actuation, and control via PCs. The Phidgets platform made device creation sufficiently simple that developers could concentrate on other issues such as form, use and design (see Figure 2.7).

Although not specifically intended for UI researchers, the Arduino hardware and software

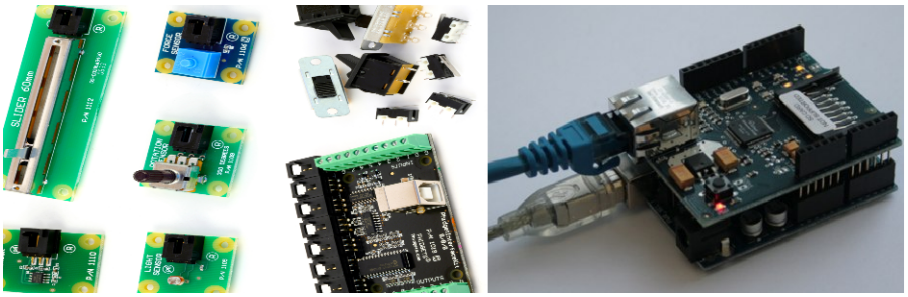


FIGURE 2.7: Phidgets and Arduino. (Left to Right) 1. Phidgets by Greenberg *et al.* [GF01] and 2. Arduno with an Ethernet shield [Jol06].

environment has been popular amongst artists and interaction designers alike. Arduinos are designed to be stand-alone or to communicate with software on a computer (e.g. Flash, Processing, MaxMSP). We see strong potential for integrating blades as supporting functional libraries in Arduino-based systems. Van Laerhoven *et al.* designed and presented a system with several interaction elements that can be rearranged in a fluid hardware space. Their "Pin and Play" system [VSG02], which was later extended to Voodoo-IO [VGRG07] by Villar *et al.*, uses a multi-drop communication network based on the 1-wire protocol with the underlying surface acting both as a physical shaping element and the medium for communication.

There have been a few systems developed primarily geared toward a ubiquitous computing context. The Smart-Its [BG03] developed collaboratively by several universities is one of such successful systems. It uses a I^2C bus[2] for connecting sensor/actuation boards. Similarly, Hartmann *et al.* use an I^2C bus for communication between their controller board and hot-pluggable input and output components in their d.tools system [HKB+06].

Currently, Bdeir *et al.* are developing littleBits [BEA+09], a library of discrete electronic components pre-assembled in tiny circuit boards. LittleBits are simple, intuitive, space-sensitive blocks aimed at making prototyping with sophisticated electronics easily accessible. The bits snap together with small magnets. Other such systems include the Bug Lab's BUGbase [Lab09] and

[2]The I^2C is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cell-phone.

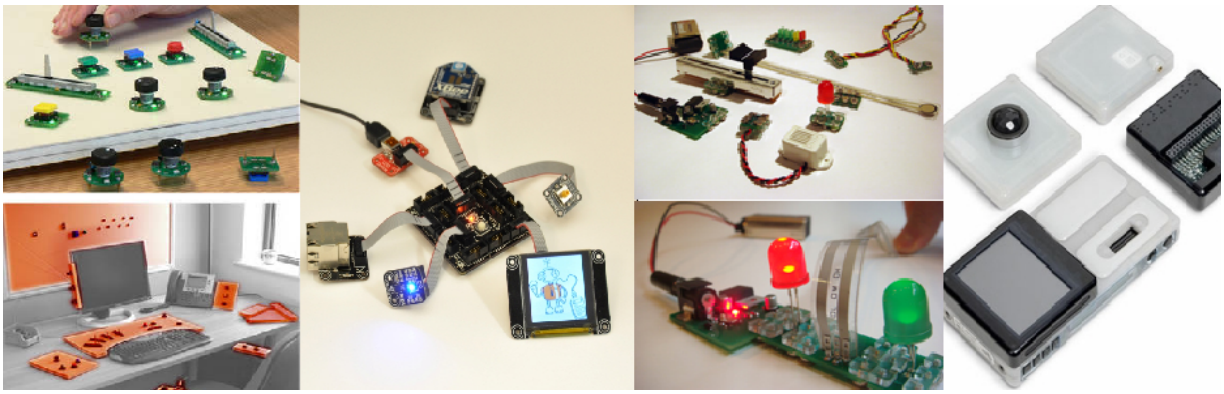


FIGURE 2.8: Voodoo IO, Gadgeteer, littleBits and Bugs. (Left to Right) 1. Voodoo IO by Villar *et al.* [VGRG07], 2. Gadgeteer by Hodges *et al.* [VSH11], 3. littleBits by Bdeir *et al.* [BEA+09] and 4. BUGbase and BUGmodules by Bug Lab [Lab09].

BUGmodules and I-BLOCK [NL03] by Nielsen and Lund. The I-BLOCKS are a collection of modular electronic building blocks embedded with various sensing and actuation mechanisms. The Microsoft .NET Gadgeteer [VSH11] is a rapid prototyping platform for small electronic gadgets and embedded hardware devices. Using a kit of hardware modules, it combines the advantages of object-oriented programming, solderless assembly of electronics and quick physical enclosure fabrication using computer-aided design (see Figure 2.8).

Many prototyping platforms for sensor networks have been developed over the years that enable researchers to realize systems with several sensing and actuation elements. Several of these systems adhere to a modular design and use wireless communication protocols for inter-module communication. Some of these platforms find substantial use in HCI, particularly in the pervasive and ubiquitous computing directions [BKM04, BLY05, OMSJ06, LCC+09], and are hence relevant to this dissertation work. For example, a system of ample relevance is the eBlocks [SCV04, SCH05] developed in early 2000's. eBlocks were developed by Vahid *et al.* toward use as electronic building blocks of basic sensor based systems. The eBlocks were aimed at enabling users with little or no programming or electronics experience to build basic but useful small-scale, low-power, sensor-based systems, including simple user interfaces.

The embedded aggregates architecture shares significant commonalities with various robotic

systems designed to embrace self-healing and self-configuring properties. The modular robotics platforms developed thus far have also allowed us to identify and tackle problems and application areas that exhibit massive parallelism and benefit from a distributed computing and control paradigm. These architectures are also currently gaining significance in many low cost satellite missions and space robotics. The following portions of the related work provide a brief description of several modular electronic systems that are being built for application in the fields of spacecraft design and robotics.

Several modular hardware approaches have been explored and implemented to realize large, complex systems. Yim *et al.* provide a detailed discussion of the current state of modular self-reconfiguring robotics in [YWMS+07]. They note that "efficient and scalable (asynchronous) communication among multiple units" is a prime planning and control challenge in the field. Hence, there are several concurrent efforts toward realizing mechanisms for modular hardware in robotics.

While several self-reconfiguring modular robotic systems have been and continue to be developed, we primarily consider Molecubes and PolyBot system because of their high conceptual and technical overlap with our design requirements. Molecubes developed by Zykov, Chan and Lipsonl, is a system of rugged manually reconfigurable cubes aimed toward study of self-reproducing machines and volunteer roboticists [ZCL07]. Molecubes use half-duplex RS-232 buses for internal and external communication, and the robots exhibit both self generated and manually created morphologies.

The PolyBots are a series of modular reconfigurable robots that can self-reconfigure and are being developed by PARC[3] for NASA's planetary explorations, with a design targeting manipulation in space and surface mobility [YDR00]. The three versions of robots G1, G2 and G3 employ

[3]PARC (Palo Alto Research Center Incorporated), formerly Xerox PARC, is a research and co-development company in Palo Alto, California, with a distinguished reputation for its contributions to information technology and hardware systems.

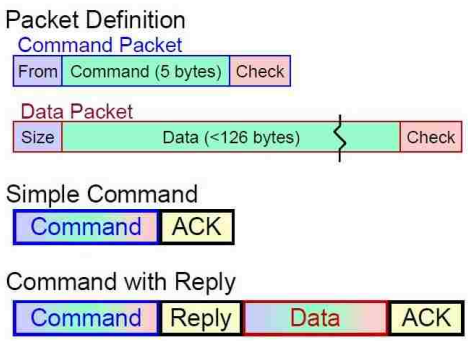


FIGURE 2.9: Simple I^2C based packet structure developed by Palmintier *et.al.*. The figure shows a simple I^2C based packet structure developed by Santa Clara University and Washington University for small satellite and multi-spacecraft missions [PKSS02].

a Massively Distributed Control Network (MDCN), implemented with a CAN bus [Bot00]. Restrictions of the CAN bus protocol, especially in terms of address space and data size, have been overcome with MDCN and a higher-level communication layer called the Attributes and Services Model (ASM). Decentralized control and computation, features common to these systems, are also being revisited toward assuring graceful degradation of services [Raw08].

Emerald, a small satellite, developed at Stanford and Santa Clara Universities, is built around a "Distributed Computational Architecture" [KPTT99]. The system involves an array of simple PIC microprocessors connected to a centralized processor via an I^2C serial bus. The researchers state that this architecture has proved useful in simplifying subsystem interfaces and parallel subsystem development, partly on account of the migration of subsystem-related software and hardware control functionality from the primary flight computer to the various subsystems [KPTT99].

Santa Clara University and Washington University developed a "Distributed Computing Architecture" for small satellite and multi-spacecraft missions [PKSS02]. Their design comprises a network of PIC micro-controllers linked together by an I^2C communication bus. The system also supports sensor and component integration via Dallas 1-wire and RS232 communication standards. A configuration control processor serves as the external gateway for communication to the ground control and other satellites in the network. The researchers share our opinion that the

I^2C standardizes many layers of the communication protocol, but does not provide sophisticated data transfer control mechanisms. In response, they have developed a simple packet structure and protocol for message transfer between subsystems as shown in Figure 2.9, and has been used in the QUEST [CHP03] small satellite mission.

Another effort geared toward space applications is the TU-Sat-1 [SKSL94] system developed at Taylor University. This spacecraft uses the I^2C bus for its interconnect framework similar to other systems discussed above. The spacecraft claims several innovations made possible through employment of a modular approach. These include distributed low power micro controllers and software; modular and integrated mechanical, thermal, and electrical design.

While modular designs are currently being developed and tested with small satellites in the class of micro and nano satellites, NASA has been considering proposals to build modular space systems [USA09], These are often composed of several lightweight modular satellites placed into orbit by light lift, then mated to a permanent support infrastructure in orbit called the motherboard. The NASA SPACECAST team argues that this evolutionary modular concept is both logical and cost-effective.

The common cross-cutting characteristic of all the systems discussed above and our concept of embedded aggregates is the design around a set of electrically + mechanically interconnecting modules that are swappable and reusable. These systems employ easy to use COTS [4] microcontrollers like PIC, Atmel controllers and ARM processors and interlinked using RS-232, I^2C, CAN bus or proprietary bit-banging protocols. We shall now briefly look at the current available COTS communication protocol stacks that can be used in the context of embedded aggregates.

[4]Commercially available Off-The-Shelf (COTS) is a Federal Acquisition Regulation (FAR) term defining a non-developmental item (NDI) of supply that is both commercial and sold in substantial quantities in the commercial marketplace, and that can be procured or utilized under government contract in the same precise form as available to the general public.

The Field-bus is a family of industrial computer network protocols [DS00, Tho05]. This umbrella specification incorporates the CAN and AS interfaces among many other standards. The CAN bus protocol [Bot00] and its extensions (such as CANopen) provide data control mechanisms and packet specifications for a multi-drop two wire full-duplex bus. The CAN bus also provides a four layer control.

A relevant extension to the CAN bus is the Massively Distributed Control Network (MDCN) [Bot00] developed by PARC for their robotics research. Compared to the existing widely used high-level CAN protocols, MDCN can address more communication nodes, has simpler APIs, is easier and more efficient to implement. Unfortunately the CAN protocol is heavily regulated by the automotive sector for which it was originally developed. The size of messages specified by the CAN interface is also a restriction.

The AS-Interface is an 'open' technology for connecting simple field I/O devices and is not sufficiently versatile for use in the embedded aggregates context. As we have discussed, we see the I^2C as a favorable system developed by Philips. However, the bus specification and implementation provides very rudimentary primitives for data transport between modules. In response, research teams have developed an elementary protocol over I^2C. Developed for the micro-satellites QUEST [CHP03] and Emerald [KPTT99], the extension provides the basic communication primitives and provides some high level communication mechanisms. The embedded communication protocols and their extensions are immensely relevant to this thesis work and a much detailed analysis and discussion on their applicability to the embedded aggregates is presented in detail in § 4.2.3.

Finally, while CAN and Field-BUS bus offer some transport layer services, and the I^2C is sufficiently open ended and flexible to allow extension by higher level protocols, there is currently no standardized mechanism like TCP/IP over Ethernet for use by these embedded devices. Such

a protocol and mechanism could potentially enable interoperability and reuse between all of these modular systems, thus facilitating the growth of scalable and flexible networks using these microprocessors.

Several communication protocols that form the ubiquitous Internet Protocol suite including TCP/IP, UDP, ARP etc. are relevant to this research work and we discuss them in detail in § 3.2.2 on page 51. Additionally, we also sought inspirations from the high level protocols and specifications developed for parallel super computers like the Beowulf clusters [SBS+95]. These mechanisms provide a similar support to distributed computing applications like our protocols for embedded aggregates. Parallel Virtual Machine (PVM) [GBD+94] was developed to facilitate networking and distributed computing over heterogeneous clusters consisting of Unix and MS Windows machines. PVM supports broadcast and multi-cast paradigms and provides C, C++ and Fortran libraries. PVM motivated the creation of Message Passing Interface (MPI) [SOW+95], a communication API and specification used to program parallel computers and super computers. Designed to exhibit high performance, scalability, and portability, MPI supports point to point communication between two nodes and collective communication calls for delegating and gathering data. MPI has evolved over the decade to its second revision MPI-2 that introduces four more basic functionalities, resulting in a total of eight supported concepts.

Open Multi-Processing (OpenMP) [DM98] is an API developed to support shared memory multiprocessing programming in multiple programming languages across several software and hardware platforms. OpenMP is a portable, scalable mode, developed through collaboration between major computer hardware and software vendors, and implements a multi-threading paradigm supporting both coarse-grained and fine-grained parallelism. POSIX Threads (Pthreads) [NB96] is a POSIX compliant threading standard, limited to the C programming language. Pthreads and OpenMP themselves do not support programming compute clusters with separate compute nodes that do not share memory, and are used along with the PMV and MPI

constructs. The protocols developed for embedded aggregates share commonalities with these protocols through the similarities in the distributed processing nature of the underlying hardware systems they support, and the dispersion and collection of messages they perform.

2.2 Terminology

In spanning several quite distinct academic disciplines – especially human-computer interaction and electrical engineering – this dissertation engages several common terms which are used with widely varying meanings. To clarify our uses of these terms for the purposes of this dissertation, we briefly consider our meanings for three key terms: prototyping, modularity and modalities.

2.2.1 Prototyping

Similar to words such as "design", the term "prototyping" is used across much of the spectrum of engineering disciplines. The New IEEE Standard Dictionary of Electrical and Electronics Terms [Rad97] defines prototyping as "a type of development in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process."

Building on this definition in a software engineering context, and relevant to this dissertation, Kordon and Luqi propose two axes for understanding prototypes with relation to the final system: "throw-away" and "evolutionary." [K$^+$02]. They consider the first as artifacts developed to be thrown away after producing some insight; and the latter as artifacts intended to progressively evolve into the ultimate product version.

From a product design perspective, Wall, Flowers and Ulrich argue that prototyping is multi-faceted [WUF92]. They write,

> "Technically, a prototype is the first thing of its kind. But "prototype" has come to mean many different things in the context of product development. For the developer of commercial satellites, the prototype may be the final product. At the other extreme, the development of a new ballpoint pen may involve more than 10 prototypes...Each is an example of a prototype; each emphasizes a different aspect of what a prototype does. In our definition of a prototype we include both electronic and physical representations of the part or product."

The authors suggest that different types of prototypes are used in a variety of ways to address several types of questions including functional testing, consumer perception testing, life testing and assembly planning. They also argue for prototypes serving as crystal balls to anticipate future problems and for comparing the evolving product with customer needs, requiring the prototypes to be matured fully functional entities.

Rapid prototyping has evolved as a growing field, progressing from earlier technologies like stereo-lithography and selective laser sintering. Wall, Flowers and Ulrich describe "rapid prototyping technologies" as a phrase used to describe "processes that have shortened the design-to-first-part cycle". In [D$^+$94] the term "rapid prototyping" is discussed in the context of physical fabrication – e.g., sometimes by computer-controlled milling or hand carving, but especially through processes like 3D printing, laser fabrication amongst others.

In the context of human-computer interaction, and specifically the space of "physical computing," "tangible interfaces," and other "reality-based interfaces" (RBIs), these perspectives offer several useful insights and distinctions. We argue that the most common approach of the last decade can be seen as following the "throw-away" approach. In this vein, emphasis is frequently upon realizing some artifact in the shortest possible period of time (e.g., one to a few hours) in which certain computationally-mediated behaviors are expressed. Such systems often employ physical materials

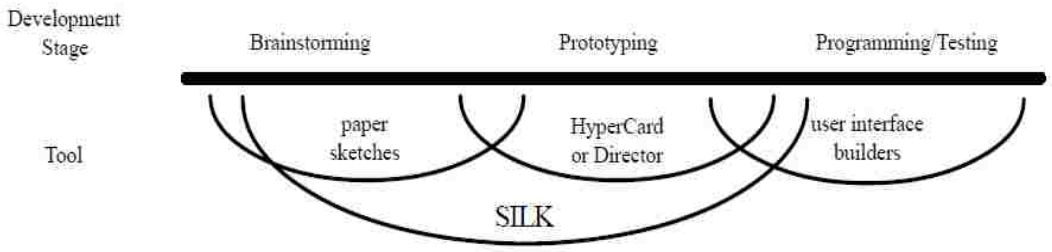


FIGURE 2.10: GUI prototyping by Landay. Three phases of prototyping - brainstorming, prototyping and programming/testing [LM95].

like cardboard, clay, or foamcore combined with pre-existing electronic modules that allow a small number (e.g., 5-10) of switches, LEDs, and other electronic elements to be realized in the earliest stages/phases of product conception [HM06b].

For physical prototyping of RBIs, an "evolutionary" approach could be conceived as (following [K$^+$02]) artifacts intended to progressively evolve – e.g., over the timescales of months and years – into the "ultimate product version." In his doctoral thesis, Landay breaks the development stage of a graphical user interface into three phases - brainstorming, prototyping and programming/testing [LM95]. This view of UI development (Figure 2.10) is relevant to this dissertation, and Landay's view of prototyping considerably overlaps with our interpretation of the word. From the various uses of the word prototyping, the question "what is a prototype?" seems to be approached by researchers in a variety of ways. In the context of this research work, by prototyping we refer to producing functional designs of the UIs with active electronics and approaching final fully functional instances. We extend this notion of prototyping, and by a prototype we refer to systems in the later stages of design, implementation and realization, possibly past the initial foam-core and cardboard modeling stages, and also complete initial pilot-deployments.

2.2.2 Modularity

Theories of modularity have been developed and have flourished through both theoretical and empirical studies. The concept of modularity has been approached and treated by researchers in several fields including mathematics, languages, biology, psychology, technology, management studies, systems and engineering. In the various fields of study, the use of modularity manifests itself in a variety of consistent themes like domain specificity, innately specified, hierarchically nesting, encapsulation, levels of decomposability, recombinability and extensibility.

Towards achieving the decoupling goal, this thesis argues for the use of modular design in the realization of the hardware and software. The terms "modular" and "modularity" are used broadly in science and engineering, with often different notions, implications and approaches toward measuring them in a system. Hence, initially we will disambiguate the terms in the context of this thesis, and present to the reader our understanding and treatment of the word in this document.

A number of different academic disciplines have bearing upon the definition of modularity. Several of these disciplines have relevance to this thesis including industrial design, product design, biology, and software systems architecture. Perhaps the earliest written record of modularity as a method to solve problems may be traced back to the work of Descartes in the 1630s. In his "Discourse on Method and Meditations on First Philosophy" [DS68], Descartes, a dominant influence among French philosophers, thinkers and scientist, proposes to divide a problem into smaller modules for tractability. He writes,

> "...to divide each of the difficulties under examination into as many parts as possible, and as might be necessary for its adequate solution. To conduct my thought in such order that ... as it were, step by step, to the knowledge of the more complex..."

A systematic study of complexity was most notably pursued by Alexander and Simon in the

1960s. In his seminal paper titled “The Architecture of Complexity” [Sim62], Simon argues that hierarchy is one of the central structural schemes that the architect of complexity uses. In his arguments, Simon assumes a much broader notion of the word hierarchy, larger than its etymological suggestion of subordination by an authority. Simon uses hierarchy to refer to,

> “...all complex systems analyzable into successive sets of subsystems...”

This notion of hierarchy has important implications for this thesis work and will become evident in the following chapters. Moving further, Simon develops the notions of decomposable and nearly decomposable systems, concluding that hierarchic systems are often nearly decomposable and are usually composed of only a few different kinds of subsystems in various combinations and arrangements. Although Simon does not make any explicit connections to modularity, the theme can be sensed through the arguments. From a management perspective based on Simon’s above mentioned decomposable systems concept [CC10], Campagnolo and Camuffo extend their notion of modularity and say,

> “So conceived, modularity elaborates on Simon’s intuition that complex systems perform better if they have hierarchical and ‘near decomposable’ structures.”

Developing their notion of modularity, Campagnolo and Camuffo suggest that modularity is a continuum, noting:

> “Every system is modular to some extent: very few systems are composed of parts that interact and affect each other so tightly that there is no opportunity to mix-and-match the subsystems they are made of. A system’s degree of modularity will be higher if the gains achievable through a modular structure are larger than those achievable through an integral one.”

They also offer an important tempering remark to their assessment of modular design:

> "The flexibility of a modular architecture stems from its ability to substitute different modules without having to redesign other components. "

Researchers in the field of biology have studied modularity in connection with anatomy, metabolism, organs and developmental units. Biologists Dassow and Munro in their work on EvoDevo [VM99] state,

> "...a module has a characteristic intrinsic behavior in the absence of any specific, persistent, exogenous influences on its components, and it may be triggered to express this behavior through a small number of (one or a few) generic inputs... a developmental module is a collection of elements whose intrinsic behaviors and functional interactions yield a mechanistic explanation of an identifiable developmental process or transformation."

From a cognition and pragmatic point of view, Sandra, Östman and Verschueren use modularity to describe the compartmentalization of knowledge in mind [SÖV09]. Furthermore, they describe a module as a "specialized, differentiated and encapsulated mental organ". Similarly, Elman describes a module as [Elm97] "a specialized, encapsulated mental organ that has evolved to handle specific information types of particular relevance to the species." From a cognitive science perspective, Fodor in his book titled "The Modularity of Mind" [Fod83], defines modules as cognitive systems that meet nice specific criteria including encapsulation, unconsciousness, speed, shallow outputs and obligatory firing. In his discussions on Fodor's work, Elman argues for domain specificity as the most important criterion.

Several researchers view modularity as a systems concept and a design decision. Developing a general theory of modular systems drawing from Simon's works, Schilling presents an arching interpretation of modularity [Sch00] and writes,

> "Modularity is a general systems concept: it is a continuum describing the degree to which a system's components can be separated and recombined, and it refers both to the tightness of coupling between components and the degree to which the "rules" of the system architecture enable (or prohibit) the mixing and matching of components."

Diving deeper into modular system theory, Schilling in the context of coupling and recombination of modular systems further describes modularity as a the degree to which a system's components can be separated and recombined. Schilling introduces the concept of 'synergistic specificity' as,

> "The degree to which a system achieves greater functionality by its components being specific to one another can be termed its synergistic specificity... Systems with a high degree of synergistic specificity might be able to accomplish things that more modular systems cannot; they do so, however, by forfeiting a degree of recombinability."

Summing up her discussions on modularity, Schilling makes an interesting observation on the positioning of a system on the modularity spectrum noting,

> "...the trajectory of systems (with regard to modularity) is bidirectional: as the environment changes (causing demands or inputs to become more or less heterogeneous) or the separability of the system changes, the system might migrate up or down a trajectory, toward or away from increasing modularity."

In his seminal paper titled "On the Criteria to be used in Decomposing Systems into Modules" [Par72], Parnas argues for modularization as a mechanism for improving the flexibility

and comprehensibility of a system (software), while reducing its development time. Parnas identifies and presents several ways of decomposing a software system, suggesting that the effectiveness of modularization lies in the criteria used in breaking the system. He adds,

> "To achieve an efficient implementation we must abandon the assumption that a module is one or more subroutines, and instead allow subroutines and programs to be assembled collections of code from various modules. "

Parnas suggests that modules are components of a system that hide decisions that change, and concludes that hierarchical structure and "clean" decomposition are two desirable but independent properties of a system structure. These views are perhaps some of the earliest views of modularity from a software system perspective. In another treatment of modularity in the software domain, VanHilst and Notkin perceive modularity as a means for separation of concerns [VN96]. Through their arguments, VanHilst and Notkin delineate the effects of modularization boundaries on the basis of number of decisions they encode. Holding modularization as an art, they argue for systems to be decomposed around smaller design decisions, ideally encoding one decision per module. These comments support the design decisions in this thesis and are aligned to our understanding of modularity, decomposition and modules.

The doctoral dissertation of Wang [Wan06] deals with modularity from an information-theoretic perspective and Wang defines modularity and modules as follows,

> "Modularity of a Decomposed System is an attribute describing the degree of overall relative coupling among the parts of the decomposition at different levels in different dimensions. A module is a unit of a system which is nearly independent of the context and interacts with other units by interfaces."

Exploring the use of the term modularity, Wang summarizes modularity as having the characteristics of hierarchy, globality, multi-dimensionality, relativity and universality. Wang also argues that functionality may not be a good criterion for decomposing a system into modules on the basis that there is no function category which is complete, standard and formal. This view sits orthogonal to product design researchers who argue for functional decomposition. For example, discussing modularity from a functional perspective, Ulrich states [Ulr95] that a modular product or subassembly has

> "...a one-to-one mapping from functional elements in the function structure to the physical components of the product."

and suggests that all interfaces between the components of different modules are decoupled. A broad variety of relevant literature exists in the areas of product design. From a product life cycle perspective, Gershenson, Prasad and Allamneni explore incorporation of modularization into mechanical design [GPA99] and suggest that modularization, owing to the functional independence created by its use, is a goal of good design. They observe,

> "Modularity allows for the reduction of service costs by grouping components so those less reliable components are easily accessed. In addition, grouping components into modules by how they are recycled can greatly reduce product retirement costs."

This property of modular design resonates with our envisioned use of modular hardware for developing research prototypes as discussed in Chapter 3. Perhaps the most differentiating theme of our approach is how we use modularity from ground up. In the product design literature, which has great relevance to HCI, several approaches to architecting systems have been explored. Several researchers have investigated and argued for the advantages of incorporating modular product designs, including product configuration options available for both the producers and the

consumers. Considering three perspectives of modularity in product design, namely functional, life-cycle and mixed, the authors suggest the presence of an 'optimal' level of product modularity. A similar view of modularity has also been used by Langlois [Lan92] and Sanchez [San95] toward explaining the degree of modularity of a system. We find maximum resonance with views of Dahmus, Gonzalez-Zugasti and Otto on modular product design. Exploring modular product design with an example use case in electric power tools [JDO01], they suggest an approach where the basic platform is made of several modules. They write,

> "Rather than a fixed product platform upon which derivative products are created through substitution of various add-on modules, the approach here permits the platform itself to be one of several possible sizes or types. Thus, the system is a collection of modules, each of which can be one of several types...An ideal architecture is one that partitions the product into practical and useful modules. Some successfully designed modules can be easily updated on regular time cycles, some can be made in multiple levels to offer wide market variety, some can be easily removed as they wear, and some can be easily swapped to gain added functionality."

From another product and systems standpoint, Huang and Kusiak use modularity to refer to the "decomposition of the architecture of a product family into distinct building blocks (modules) used to meet various functions of products" [HK98]. Huang and Kusiak also present several ideas relating to functional decomposition of a system which are greatly aligned and is discussed later in this dissertation. Considering technological complexity in modular design, Tsai and Wang distill modules as components and individual parts of a system that are mutually orthogonal in functionality (aiding in decreasing complexity as they are designed) [TW99].

To sum up, modularity is defined, used and developed by practitioners of the various fields of study in a variety of ways. Development, evolution and morphology form the basis in biological

sciences, and in engineering, physical-form, spatial arrangement, function, life-cycle and other hybrid aspects push modularity into multi-dimensions. Several of these dimensions are relevant to this dissertation and have strongly influenced our interpretation of the term modularity.

2.2.3 Modalities

Modality, as a term, has several prevailing interpretations. The term modality is defined and interpreted differently by practitioners of various sciences and humanities including education, humanities, linguistics, medicine, logic, transportation and human-computer interaction. Of the several dozen broad definitions, the treatment of the word in semiotics, medicine and HCI hold maximum relevance to this thesis work. The definition of the term in semiotics is aligned with the works of Linguists Peirce (particularly "The Logic of Relatives" [Pei97]), and a modality is interpreted as a way in which information is encoded to be presented to humans, including a channel by which signs are transmitted like image and music. In medicine, the term modality is used in both the study of human sensory system and in medical imaging. While in the former a modality is used to refer physical phenomenon such as temperature, pressure, taste, light etc. [Fre74], in the latter modality is used to refer to equipment, probes and transducers used to obtain measurements and parameters from human subjects [Beu00, Dha03].

From an HCI perspective, Nigay and Coutaz in their discussions on multimodal systems [NC93] state their interpretation of modality as,

> "Modality refers to the type of communication channel used to convey or acquire information. It also covers the way an idea is expressed or perceived, or the manner an action is performed."

This interpretation of modality may be used to describe the collection of both senses through which humans can perceive information and also the underlying transducers (sensors and actuators). Our

interpretation of modality is based on the notions in [NC93] and medical imaging. By modality, in this thesis we hence refer to interaction elements or components including, for example switches, sliders, rotors, LCDs and LEDs, that may be used to present information to humans or sense their inputs.

2.3 Chapter Summary

The direction of research and implementation performed in this thesis work is strongly guided and influenced by prior pursuits spanning directions including prior UIs, UI toolkits, robotics, spacecrafts, tradition computer networks and DIY hobbyist electronics. Several of these motivating, guiding and complementary systems and research explorations were presented, with many others discussed through the following chapters. With an understanding of the research space of this dissertation and guided by the notions of a few key terms shaping this thesis work, we proceed toward developing the embedded aggregates foundation. The following chapter presents the notion of an embedded aggregate, distinguishes aggregates from existing embedded systems, argues for their utility and identifies advancements in key aspects of embedded system communication that are required toward realizing the aggregate class.

Chapter 3

Embedded Aggregates

> Most people are in favor of progress, it's the changes they don't like.
>
> Author Unknown

Embedded systems taking the form of collections of several electronic modules are relatively widespread, including, for example, automobile processor networks (vehicle bus), hierarchical control systems, and networked control systems. In this diverse ecology of multi-module systems, a class of embedded systems distinguishes itself by the relative scale and distribution of processing power and control architecture. We call these systems embedded aggregates. Modules in an embedded aggregate typically have mutually comparable processing power, communicate with one another and hold a peer status in the system. In the simplest case, each embedded aggregate module is built around a microprocessor, often without dedicated communication co-processors. The aggregate modules invoke each other's functionality and collaborate using module-specific messages and commands through the communications fabric that holds them together.

An embedded aggregate is a collection of several interconnected and synergistic electronic modules deployed and organized within a few feet of one another. The modules making an aggregate have heterogeneous configurations. First, the CPU speeds of the processors of the modules may range from a few MHz to several hundred MHz. Second, the module level RAM may extend from a few hundred bytes to several MB and finally, the modules may run a real OS like Linux, may be based on RTOS [DMT00] or may implement no formal OS. Each aggregate module performs an ensemble of functions and is envisioned to be a cost-effective [1] drop-in building block of the

[1] We approach this from the overall life-cycle cost of the system [GHSR97].

system. Embedded aggregates resemble large and elaborate embedded systems (like air traffic control and manufacturing assembly lines) in architecture, but are comparable to single-board devices in size. There are a variety of motivations for building systems in this multi-module fashion, including improved fault tolerance, graceful degradation of performance, open-endedness, extensibility, scalability, reusability, decoupled development of the embedded system functionality and continued augmentation through specific function oriented modules.

In this chapter, after a brief introduction to current embedded system design trends and their implications, we describe certain application contexts that have motivated and shaped the idea of embedded aggregates. Next, we present a discussion that situates the aggregates class of systems in embedded design space and highlight open research avenues in communications and protocols for their realization.

Varying levels of hardware and code (software and firmware) integration have been pursued by engineers and designers to meet the functional and design requirements of an embedded system. For example, common design objectives for embedded systems include energy efficiency, low cost and high performance. To realize these combined objectives (sometimes described as non-functional requirements), often a tightly-coupled consolidated design approach is followed. Enabled by parallel advances in VLSI technology, in this approach, devices are today frequently built around a few system-on-chip (SOC) processors and only a handful of additional components, enormously reducing overall costs. In practice, these technological innovations have scaled well in unison and devices have tended to become progressively smaller, integrated and complex entities across a broad diversity of end applications [TP06]. Deviations from this design trend are sometimes found in situations that demand the placement of several hardware and software entities over a large spans of physical space, for example in air traffic controls and manufacturing assembly lines. Even in such cases, the individual subsystems are generally compact and dense modules.

With improvements in density of packaging, current embedded design approaches collocate the large majority of system resources on a few components, often realizing a very dense implementation. For a growing number of application areas, however, while this approach is feasible, it is functionally and objectively constraining. In § 3.1, we argue that application areas other than our HCI pursuits introduced in Chapter 1 are bound to emerge in the future as embedded aggregates are more widely realized and employed. Beyond this dissertation, we believe the embedded aggregates class of systems has largely been unexplored, and represent a departure from the current predominant trends in hardware miniaturization and integration.

Implementing the protocols and the encompassing communication fabric toward the realization of embedded aggregates remains a formidable challenge. The influence of embedded system design trends upon communication technologies has left a collection of protocols that are intractable for the aggregate class of systems. Protocol development efforts have tended to concentrate on mechanisms for communication at the scale of a single circuit board or for linking components across a large physical space (several meters) through electrically noisy environments.

For realizing the emerging embedded aggregate class of systems, however, we will argue existing protocols are either inadequate or infeasible and limiting due to complexity and overheads (discussed in § 3.3, § 3.4 & § 3.5). In the few efforts to realize multi-module embedded systems that are currently underway, the protocol challenges, to a large part, have lead to one-off designs of communication and networking frameworks [KPTT99, Bot00, CHP03]. These communication frameworks have tended to be both system and situation specific, and hence there is a gap in the embedded communication spectrum and this gap poses challenges in realizing aggregates.

In the following sections we argue in support of the embedded aggregates class of systems, motivate the need for new protocols and mechanisms, and study the aggregate space toward developing a broader communication framework. We begin with a discussion on a few applications

that shaped our concept of embedded aggregates and their design goals in the following section.

3.1 Motivations for Embedded Aggregates

From a pragmatic perspective, one can argue that the top level design and performance goal of an embedded system is to accomplish the desired tasks within the given time constraints. This goal primarily positions an embedded system as a function-specific implementation, rather than a generic computing device. Yet, in a broad class of emerging applications, embedded architectures that are open-ended, scalable and flexible are in demand. These features are orthogonal to an embedded design approach that gravitates toward unifying and co-locating system resources. We feel that these application areas demand and benefit from system architectures that:

- are open-ended to support system building, extension, maintenance and replacement in a progressive fashion as resources and finances allow;
- can alleviate integration and realization obstacles due to the physical rigidity of the underlying PCBs (mechanical flexibility);
- can grow to distribute several electronic modules over a physical space and allow rapid rearrangement of modules (mix & match and move about in physical space);
- can tolerate a degree of uncertainty in the integrity of the system arising due to the nature of its operating environment (harsh environments); and
- can extend service in the case of component failure unlike monolithic design that often leads to complete loss of all functionality with limited scope for recovery.

We describe a few motivating application areas below that make some of our above comments more concrete.

3.1.1 Physical Environment Sensing

Environmental sensing provides several embedded systems opportunities and challenges. With the rapid growth in the number of MEMS and other miniature sensors, the number of physical parameters and events sensed in such systems is on the increase. These sensing systems are assuming a cyber-physical dimension of the larger computing infrastructure.

The use of embedded computing and sensing entities differs substantially from one use case to another, depending on the environment sensed and manipulated. In some cases, sensors are deployed in close proximity, often in a 3D space of a few cubic feet, to study the space for humidity, temperature, air velocity and particle density [Rus01, ZSS05, SBL+07]. At times, sensors may be deployed in corrosive environments and are exposed to the elements, so an exhaustive shielding of the sensing system from the environment may be impractical. In such cases, it might often be beneficial to deploy sensing systems that can perform despite sensor faults and partial system failures. Finally, there is often a requirement to add, remove or rapidly interchange sensors in an ad hoc fashion toward sensing a different set of parameters or replacing broken sensors [MXD+07]. The above requirements and challenges demand a sensing system that can flexibly incorporate several sensing nodes and is capable of performing its sensing responsibilities, with graceful degradation, when individual nodes become unresponsive.

Wireless networks have been a predominant means to interconnect sensors [Pot00, Rag06], and several efforts in wireless sensor networks have produced very efficient networking and organization protocols [Zig, ALM05, YHE02, WH02]. Yet, some sensor networks may need to be deployed in settings that are incapable of sustaining wireless communication or will benefit from simpler and more cost-effective communications due to expected faults and failures. These settings benefit from a low-cost wired network.

3.1.2 Human-Computer Interaction Devices

Human-computer interaction devices range from the ubiquitous keyboard and mouse to application-specific custom interfaces. Interaction devices have been our primary application area for embedded aggregates. In addition to the applicability of blades and tiles at the prototyping stage, we are motivated toward a modular approach for several other reasons.

First, we had to develop and deploy several dozen sets of 3-5 different interaction devices at several distant locations. These devices were experimental interfaces, with a number of hardware, mechware, firmware, and software design iterations spanning a period of several years from the time of the initial deployment planned. During this gradual process, we aimed to avoid lock-in and obsolescence, maximize the level of hardware and firmware reuse, and minimize costs and waste as the system evolved.

Next, we desired a toolset that could enable the construction of much larger interaction devices that are highly specialized in form and function, perhaps involving hundreds of sensors or displays in physically or logically descriptive designs (e.g., with geographical or diagrammatic 2D or 3D layout) [UKK+01]. We wanted to leverage the same toolsets developed for our work on the smaller interaction devices, applying and extending these same hardware, software and firmware investments toward our furure plans for large-scale highly-customized devices. We therefore aimed for forward compatibility in our hardware and firmware development efforts through embedded architectures that could scale, allowing reuse of our current implementations of interaction modalities.

Finally, we felt that such elaborate interaction systems should be realizable, ideally by small teams with limited resources (time, expertise and finances) and also through collaborations between several groups of HCI practitioners. Following the kinds of user interface toolkit approaches common in the realm of traditional graphical interfaces and inspired by hardware toolkits and

prototyping platforms, we envisioned a toolkit that is composed of a series of functional modules. These functional modules would ideally be developed as libraries of interaction modalities from which the appropriate modalities may be chosen toward realizing a new interface.

3.1.3 Fractionated Systems

Tightly integrated, monolithic electronic systems can lead to large amounts of electronic wastage due to failure or obsolescence of a few components in the system. In contrast, a fractionated approach spreads functions from a central monolithic unit to distributed heterogeneous modules. Compared to the all-in-one approach, it increases the number of components and the initial cost of production, while prospectively extending the life of a system and its uptime, as a fractionated construction is conducive to progressive upgrades, partial repairs and replacements [TFCG06, HN, O'N09]. It has been argued that developing systems in a fractionated format has cost benefits over the total life of the system [MW05, Bro06b]. Fractionated design finds use in a variety of contexts, on Earth and beyond in outer space. We will now discuss two of these application areas.

3.1.3.1 Robotics

At present, several research groups are pursuing development of robotic systems with self-healing and self-configuring properties [NF00, RBKV02, KM07, GSRRU07]. These robotic systems require the underlying hardware to be both flexible and robust, and a modular system that can exhibit such traits is subsequently often considered [YDR00, SMS06]. These modular robotic systems are particularly aimed for use in the hostile environments of outer space and rugged terrain. A decoupled modular approach can support self-reconfiguration by shuffling (mechanical detachment, relocation and reattachment) modules, self-healing by replacement of non-functional modules, and energy savings by detaching faulty modules. Finally, the modular approach enables distributed sensing and decision making capabilities, along with multi-point locomotion and flexibility, which are important features required for maneuvering through difficult terrain.

3.1.3.2 Spacecraft

Satellites operate in very harsh, radiation-prone and increasingly debris-ridden outer space. Above the material and production cost of a satellite, the expenditure to put one in space and maintain it in orbit contribute enormously to the overall cost of a mission. Often these two costs are comparable. With rocket launches that carry very heavy satellites tending to be both expensive and risk prone, many huge spacecraft are being realized as constellations and formations of several smaller satellites [SBM01, Lai01, KSBY02]. However, the individual satellites have remained complex monolithic systems, often experiencing little upgrades or repairs, reducing their lifetime. To this end, several researchers have proposed a fractionated approach toward satellite building [BEH02, MGGR02, TKYM04, Bro06a, col09, BEC09].

These proposals cite advantages in terms of increased flexibility and robustness, during design, procurement and mission operations. Under this fractionated approach, individual spacecraft are built using mass-produced electronic blocks, with standardized functionality and interconnects. The individual satellite blocks can be enhanced, upgraded or extended as finances allow and requirements arise.

Current proposals push predominantly toward wireless connectivity amongst the modules forming a satellite [SB08, Bro06a]. A two-tier communication system, involving wireless and wired networks, would find use in such fractionated spacecraft, particularly in low-cost fractionated nano, pico and mid-sized satellites. These hybrid wired-wireless systems would be analogous to the WDM optical networks with copper control-plane [PG05]. The satellite could use a low-speed wired network for house-keeping activities and choose to set up high-speed wireless communication links on-demand. This approach reduces energy costs (wireless radios consume more power) and wireless networking component costs.

All the applications discussed above demand or benefit from the features that can be extended

to them by building their embedded-hardware core in a multi-module fashion. This outlook is strongly shared by the various research fields and several multi-module architectures have since been created by researchers [YDR00, SMS06, ZCL07, SG08]. In the field of robotics, "efficient and scalable (asynchronous) communication among multiple units" is stated as a prime planning and control challenge [YWMS+07], a goal that cements the position of the modular approach.

Our embedded aggregate notion builds from these core ideas, requirements and challenges. The above motivating contexts suggest a current need to implement the embedded aggregate class of systems and develop the underlying design principles and mechanisms. While modularity in hardware design is a well-known and tried concept, architectural decisions regarding where and how modularity is expressed can have major implications for reconfigurability, reuse, extensibility, and scalability. These design decisions together with ideas on federation of processing power and delegation of functionalities distinguish embedded aggregates from other modular embedded systems.

Having described some embedded aggregate use cases, we next venture toward developing the design principles of embedded aggregates.

3.2 Networking and Communication Conceptual Parallels

In the previous section, we introduced our motivating contexts for the embedded aggregate class of systems. To realize the overall functional objectives of the embedded device, the modules require communication mechanisms and protocols to connect, organize and exchange data. Several embedded communication protocols have been developed over the years. In this section, we identify the suitable protocols for use in the embedded aggregates. We begin with an exercise of juxtaposing and comparing current embedded communications, with the mechanisms and protocols in a familiar non-embedded communication settings in §2.3.1. We next identify the

kinds of facilities required of the aggregates' communication infrastructure in §2.3.2 and 2.3.3, and finally conclude with a discussion of the suitability of the currently available protocols in realizing the aggregate communication infrastructure.

3.2.1 Similarities with Operating Systems and Inter-Process Communication

An embedded framework consists of hardware interface specifications, protocols, mechanisms and communication channels that together enable and guide the development of a class of embedded systems. To gain an appreciation of the current embedded communication protocols, we take up an exercise of juxtaposing and comparing an embedded framework with a computer operating system (OS). An OS is physiologically similar to an embedded framework in several ways and this study provides very useful insights. We begin this exercise by illustrating similarities and parallels between a OS and an embedded framework, and then situate the currently available protocols in the embedded communication spectrum.

An OS facilitates the execution of various application software. The OS manages access to the hardware, applications and data through a set of inter-process communication (IPC) mechanisms. Additionally, IPCs also enable communication among the several processes and threads (sub-programs) that make an application and the various components of the OS itself. The choice of IPC may vary based on the type of data, the required level of communication abstraction and bandwidth and latency of communication between threads.

The entities (i.e., protocols and specifications) of an embedded framework are analogous to the facilities provided by an OS to the software applications it sustains. The various electronic components making an embedded device are logical counterparts of the software applications. The IPCs provided by the OS enable applications to share data and communicate with one another. Similarly, in the embedded front, the data-bus, communication channels and protocols allow

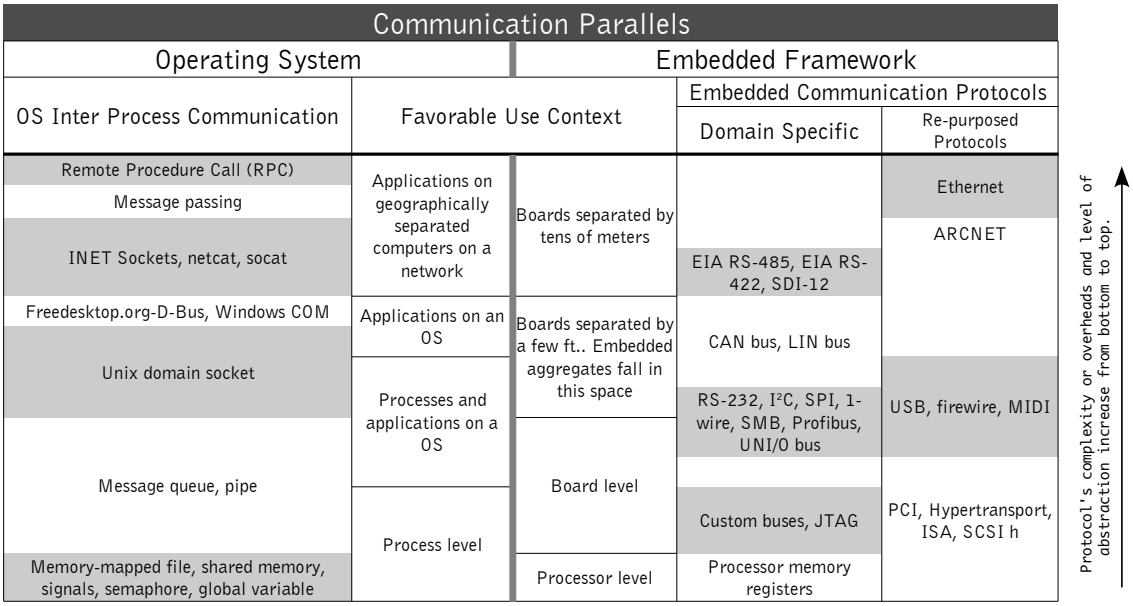


FIGURE 3.1: Parallels between IPC mechanisms and embedded communication protocols. The table juxtaposes the communication mechanisms in an OS and in an embedded system. Protocol boxes that overlap two use contexts signify their applicability in both cases. A few protocols are suited for the physical scale of the embedded aggregates. However, these protocols are not sufficiently feature rich to support the aggregates class of systems.

components to exchange data and work with each other in a system. Hence, the communication mechanisms provided in both OSes and embedded device platforms perform the same function - they enable and dictate the architecture of the overall system.

To illustrate and clarify the analogies, Figure 3.1 outlines the parallels. Both an embedded framework and a mainstream OS can support their constituent entities through a variety of communication mechanisms. In the embedded context, low-level communication mechanisms have received considerable research focus and sufficed the resulting board-level communications needs. For the cases where communication across geographically separated embedded boards is to be achieved, wired mechanisms, with signaling and protocols resistant to errors, have also been developed. Sometimes, mainstream networking protocols like TCP/IP and UDP over Ethernet and ARCNET are used to link boards to achieve modularity, scalability and ad hoc operation.

On the outset, systems of the physical scale of the embedded aggregates are substantially supported

by several communication protocols. However, in addition to the protocol's ability to support board-to-board communication over short distances, the aggregates require protocols that provide a rich set of features that can support their dynamism and open-endedness. While CAN Bus and LIN Bus provide many features toward realizing readily-composable multi-module systems, they still lack several required features, as explained in §2.3.2 and 2.3.3. Hence, in their current state, none of the currently available protocols can support aggregates as a drop-in communication mechanism.

This exercise has helped us place the different embedded protocols in relation to the physical dimensions to which they cater and obtain a simplified picture of the protocol space. Furthermore, it has also helped us to situate embedded aggregates in the same physical spectrum. Next, we will gather a basic set of features that are required in a communication protocol toward realizing the embedded aggregate class of systems. Computer networks and supercomputing communication mechanisms have ideas and insights to offer in this regard, due to their appreciable congruences with aggregates.

3.2.2 Similarities with Other Computer Networks

A computer network [PD07] is made of several independent machines that often exchange commands and data with other computers. The network enables the computers to interdependently accomplish their overall objective by facilitating the collaboration among them. As such, this organization and functioning is similar to that of embedded aggregates. Mechanisms to discover newly added modules, uniquely identify them, and provide the means to communicate with other existing modules are essential. A suitable communication method should also allow modules to send and receive data independently with negligible communication overhead to other modules. This objective requires either dedicated one-to-one connections or efficient use of shared communication buses.

If a communication resource sharing approach is followed, then unique ways to address modules

in a communication and built-in low-overhead techniques to recover self-addressed traffic on the bus are essential. These require protocols and mechanisms to dynamically or statically assign addresses for communication. Towards this end, network protocols like Internet Protocol (IP) [P+81, Hui98], Dynamic Host Control Protocol (DHCP) [DBV+03] and Address Resolution Protocol (ARP) [Plu82] provide valuable knowledge and insights. No communication can be 100 percent reliable, particularly in the case of embedded aggregates, as one of the required features is survivability in environmentally hostile conditions. Hence, methods for recovering from faults and sustaining gracefully degraded communication are required. Several of these ideas can be found in communication protocols like Transmission Control Protocol (TCP) [CK74, SW95] and Internet Control Message Protocol (ICMP) [P+81, Ark01].

3.2.3 Similarities with Message Passing Interface (MPI)

A general computer network and the underlying protocols are designed and implemented assuming a generic communication pattern between the computing nodes. Consequently, several basic communication primitives are provided by the protocols, but often these primitives are modified or enhanced for providing additional services or toward achieving very specific network implementations. Networking architectures for supercomputing clusters with special protocols like the Message Passing Interface (MPI) [Sni98] are such special instances. Several similarities between a supercomputing cluster and a multi-module embedded system stem out of the very goal-oriented optimizations of both systems.

Regular computer networks that span large physical spaces suffer from system dynamics and are more susceptible to faults that are a consequence of the geographical separation between nodes [BP95, PFTK98, JD02]. In contrast, many of these network effects are minimized in clustered computers due to close physical spacing between nodes [PD03, Com08]. The smaller physical space also reduces demands on the communication mechanisms in terms of physical

signalling and error correction. Since embedded aggregates have physical dimensions on the order of a few feet, they lack these distance effects of general networks, so we can simplify the communication protocol, an advantage in the CPU-constrained aggregates.

Finally, communication mechanisms, such as collective communication calls, methods for virtual topology and synchronization as seen in MPI are beneficial in the aggregates context. For example, embedded systems would benefit from having point-to-point and collective communication methods offered as basic primitives by the underlying protocol. Together with a simplified and holistic communication design, these can empower embedded devices with the required high-performance, scalable, and portable network.

Our study of computer networks and MPI has revealed mechanisms and optimizations that are essential for a suitable protocol. The current set of available embedded communication protocols do not support these capabilities, and we require a new suite of communication protocols and mechanisms for embedded aggregates.

3.3 System Setting

The previous sections motivated embedded aggregates and made the case for developing fitting communication protocols. This communication infrastructure presents optimization challenges, as aggregates have tight resource constraints and limited processing power, requiring analysis from the ground up, with specifically tailored simplifications and functionalities. Hence, before we embark on the design of a new communication infrastructure, we discuss the architecture, physical scale, logical scale, hardware capabilities and temporal scale of the envisioned aggregate systems toward understanding the design space.

Architecture, Physical and Logical Scale: The success of the IBM-PC architecture cemented the position of the "CPU with co-processors" design as the staple architecture for computing and

embedded systems alike. Embedded aggregates deviate from this asymmetry in decision making to a more homogeneous setup. Aggregates are anatomically and architecturally similar to present-day computer networks, where each computer is dedicated to a particular task and communicates with other computers toward achieving an end objective. In physical scale, the aggregates are comparable to routing switches, i.e., in the order of a few feet, and in logical scale, the embedded aggregates approach several hundred modules. In a few cases, systems with several thousand processors are conceivable, but we have focused on situations where systems typically will be composed of several tens of modules.

Module Hardware Capabilities: From the perspective of hardware capabilities, embedded aggregate modules find themselves straddling between computing nodes that formed the Internet of the 1980s and present day networking switches like [Fen81, DS87, DT01, DYN03]. On the lower end, the hardware capabilities, including the CPU clock-speed in the range of few tens of MHz, internal bus speed, a few hundred bytes of RAM and ROM, are comparable to the earlier Internet compute nodes. The Internet nodes had co-processors aiding the primary CPU in accomplishing the networking, memory management and other tasks. On the other hand, embedded aggregates often have modules built around a single CPU responsible for the activities of the node and satisfying its networking requirements. This constrains the CPU cycles that nodes have for communication.

Temporal Scale: Embedded aggregates fall predominantly in the soft-real time embedded system classification. Simultaneously, they are also suited for hard-real time applications. As Figure 3.2 highlights, an embedded aggregate features time scales across several orders of magnitude. Individual modules with CPU speeds in several tens of MHz can individually accomplish tasks in a hard real-time fashion. However, when communication with other modules is involved, often only soft real-time performance is achievable. Inter-module communication is expected to take several tens of milliseconds due to interlink latency and prioritization. Finally, communication with external entities and systems may take several hundred milliseconds to a few seconds depending

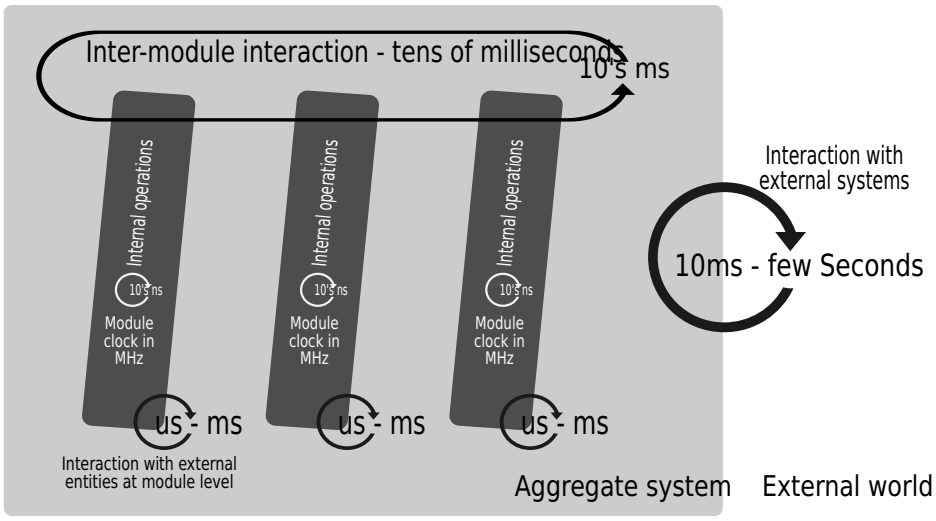


FIGURE 3.2: Embedded aggregates time constants. The compartmentalization and modularization of processing power in embedded aggregates leads to a variety of temporal responsivenesses. As shown in the figure, tasks may be completed in as low as a few micro-seconds individually by the modules or can take as high as a few seconds when involving communication with external systems.

on types of external communication protocols used and their timing implications. Therefore, the distribution of associated functionality and the implementation of modules plays an important role in the underlying time constants achievable by an aggregate. By appropriately designing and equipping the individual modules, systems developers can make the aggregates applicable in both the hard and soft real-time contexts.

In summary, the embedded aggregates present us with a set of constraints and communication infrastructure design requirements. A reasonable aggregate communication fabric must provide and support several functionalities while being sensitive to the hardware constraints and end-application requirements. Other than our conceptual considerations for embedded aggregates, we also faced additional constraints in the implementation of blades for use in user interfaces and interaction artifacts design research. To sum up, our overall design considerations for embedded aggregate protocols + mechanisms and blade hardware included:

- Blade form-factor considerations [2].
- Lower cost per module.
- Heterogeneous embedded processors with:
  - CPU speed : few MHz to several hundred MHz.
  - RAM : few hundred bytes to several MB.
  - OS : some without a formal OS, RTOS based and real OS like Linux.
- Wired communication towards lowering cost.
- Additional implementation constraints including availability of processors with only one I^2C port in the required pin-count and packaging type, and several processors with only I^2C slave mode functionality.
- Through-hole component based hardware design.

3.4 Chapter Summary

This chapter identified a class of embedded systems with distinct characteristics and motivated the notion of an embedded aggregate. Motivated by their applications and strengths, we studied and developed the aggregate concepts, design criteria and requirements by juxtaposing the aggregates with comparable familiar system settings. The study following the comparative analysis, undertaken to place the embedded aggregates amidst current systems, enabled us to identify and pursue related literature discussed earlier in Chapter 2. Supported by these assignments and an understanding of the embedded aggregates' physical, logical and temporal scales and the design considerations highlightened above, we proceed to the next chapter where we develop a favorable architecture for the aggregates and a supporting communication infrastructure.

[2] Discussed further in § 5.4.1.1.

Chapter 4

Architecture, Protocols and Mechanisms for Aggregates

> The idea is to try to give all the information to help others to judge the value of your contribution; not just the information that leads to judgment in one particular direction or another.
>
> Richard P. Feynman

Given the strong similarities between mainstream protocols and the requirements of the aggregates, we begin this chapter with the question: what prevents us from using a tailored version of them in the embedded aggregates context?

Supported by growth in CPU speeds and raising to the demands of systems and applications built using such CPUs, the protocols that link everyday computing nodes have grown from their modest beginnings in the early 70s to immensely capable and subsequently complex networking stacks. While the growth of the networking stack has been commensurate with the development of the computing systems they support, the current protocols would be found cumbersome and impractical if an exercise to implement them in the computers of 1970-80s is carried out.

As noted earlier, the embedded aggregate modules are even less powerful the than computing machines of the 70s. Therefore, how flexible, elaborate, powerful and complex should the protocol for the aggregates be?

Let us look at a routing switch. A switch is made of several switching elements that are interconnected to accomplish data transfer between the different ports of the switch. The network traffic exchanged in a switch is several orders of magnitude greater than that to which individual computing systems on the network are exposed, and amongst the highest in the Internet as a whole.

The simplicity of the communication protocols in the bowels of a switch and protocol-specific design of the switching elements enables high-speed exchange of data bytes and high reliability under extreme traffic loads.

Given their resemblances to the Internet as whole and its underlying switching elements, embedded aggregates would benefit from protocols that embrace the simplicity of the switch routing protocols, while also providing some of the powerful features that enable realizing a heterogeneous network like the Internet. With this direction, in the next section we begin to devise a holistic design approach in developing the communication infrastructure.

4.1 Holistic Design Rationale

The relationship between networking architecture and performance is an extremely challenging one. In an effort to tackle this challenge, the Open Systems Interconnection (OSI) model was developed to guide and standardize communication system design [Zim80]. This model aims for "separation of concerns" and divides communication systems into seven layers for tractability, where each layer is a collection of conceptually similar functions provided by a set of protocols. Unfortunately, this layering approach often hides the information that lower layers require for optimizing their performance, leading to the replication of functions and loss of efficiency, making the OSI model less favorable in certain cases. For example, the time-tested Internet Protocol Suite, also commonly known as the TCP/IP Suite, though constructed as a set of four layers, is ironically not intended to be OSI-compliant [BM02, AT]. OSI ordering constraints conflict with efficient implementation of its data manipulation functions. The OSI model's loss of efficiency for gains in structuring and abstraction is also unfavorable in the embedded aggregates context. The lack of dedicated communication hardware in the aggregates further magnifies the computational side-effects of layering. Therefore, we take a holistic approach toward the design of communication infrastructure for embedded aggregates.

The communication infrastructure comprises a collection of protocols and specifications extending from the electrical logic and topology of the physical wires to protocols for exchanging data between the modules. In our holistic design, we chose communication topologies, protocols and their functionalities such that they complement each other, providing the required overall functionality in a synergistic fashion.

In this section, we present the various required components of the communication system, discuss available techniques and present the evolution of our specification and implementation. Following a bottom-up approach, we begin with the networking topology and progressively refine our choices of topologies and associated mechanisms as we progress through the discussion. We will discuss the applicability of the various available embedded communication protocols and locate protocols that best suit our design goals and topology choices. Features in the currently available protocols do not meet our requirements. However, in lieu of designing another physical layer protocol, we lean toward an existing physical layer implementation and associated data-link layer specifications that, through extensions, can support embedded aggregates. Through this holistic approach, we maximize the properties and features offered by the communication infrastructure as a whole by leveraging and creating complementary functionality in the different parts.

4.2 Network Topology

Modules in an embedded aggregate are primarily interconnected using wired communication. This reduces individual module cost and RF front-end overheads and increases the selection of micro-controllers available for realizing individual modules. The first step in implementing a wired network is the network topology. The network topology presents itself in two realms, the physical and the logical spaces.

4.2.1 Physical and Signal Topology

The choice of the inter-connection among embedded aggregate modules requires consideration of several attributes. The ability to add modules and easily reshape, expand or reduce the system is an important design criterion. Hot-plugging of modules and providing the ability to reconfigure the system at run-time is also a valuable feature. Furthermore, modules require means to mutually communicate with every other module in a duplex fashion. We will now consider the applicability of several network topologies in the context of embedded aggregates.

Point-to-Point: Point-to-point is the simplest of all topologies, with a dedicated physical channel between any two modules in an embedded aggregate. The physical channel can be established permanently, or dynamically, using circuit switching. Unfortunately, the point-to-point topology requires setting up (or removing) several newer (existing) data paths, with the addition (removal) of every module. This greatly increases network wiring costs and imposes non-scalable requirements, as either dedicated pins are needed for each channel or an increasing number of channels must multiplex on the same pins. Furthermore, firmware complexity increases owing to the number of communication interrupts. The ensuing cost and time penalties therefore make it unsuitable for embedded aggregates.

Mesh: A mesh topology, due to large number of interconnects and challenges in setting up and tearing down connections during system modifications, is also infeasible in the embedded aggregates context.

Star: The star has a point-to-point link between each module and a central hub module. The star topology admits simple addition and removal of modules. On reaching pin limitations on the hub module, a larger network may be built using several interconnected hubs, each with a star arrangement of attached modules. While this is feasible and practical, as seen from its extensive use in hubbed Ethernet, the number of data paths is far too numerous for embedded aggregates. Star

topology can quickly lead to a substantial amount of wiring that is physically comparable to the size of the modules themselves and is often tangibly constraining to the placement and reorganization of modules. Subsequently, we find the star topology impractical for embedded aggregates.

Ring and Line: A ring connects modules in a circle where the modules themselves are part of the data path. Nodes are opaque to the transmission medium, and when presented with data, individual modules are responsible for receiving, parsing and retransmitting the data appropriately, immaterial of the intended recipient. This additional communication overhead can drastically reduce the performance of individual modules. Furthermore, when an embedded aggregate module is detached, the ring is broken and the communication between other modules is disrupted until the ring is re-established. Hardware level reconfiguration of data paths (bridging the path across a removed module) and the associated transients add to the complexity and instability of the process. An open ring, called a Line topology, also presents similar concerns. The ring and line are therefore unusable for embedded aggregates.

Bus: A bus is centered around a single data path to which modules connect. Unlike the ring and line, the modules themselves are not a part of the data path, but share the data path to communicate in a half-duplex fashion. The bus topology greatly reduces the number of data paths required, however, the cost of managing the network is higher than the star and mesh topologies. This greatly increases the communication functionalities required of independent embedded aggregate modules. Nevertheless, the implicit flexibility and the wiring simplicity of a multi-drop bus of sufficient bandwidth can outweigh the addressing and management overheads. Owing to this advantage, the bus topology is a feasible candidate.

Tree: A tree topology mitigates some of the module addition and removal concerns with ring and line topologies, though, it requires more data paths than a bus. Furthermore, as a hierarchical topology, a tree admits several simultaneous data transfers, increasing throughput. Unlike the bus,

the modules in a tree are part of the data path. However, unlike the ring and line, the data transfer on a tree demands varying levels of participation from the modules, depending on their position in the tree, and offers more graceful degradation of performance from module faults. Similarly, the extent of disturbances to data traffic due to the addition and removal of modules depends on the location of the module and is significantly less than ring and line topologies. Finally, while the bus topology has a single point of failure, the tree topology provides some level of immunity from faults and instabilities, due to the hierarchical segregation of communication. Hence, the tree topology is a feasible candidate, extending significant data throughput and fault-tolerance advantages.

In the above discussion, both the bus and tree topologies emerge as favorable candidates for use in embedded aggregates. The physical simplicity of the bus is unrivaled, and the performance benefits extended by the tree, due to implicit hierarchy and isolation, are immensely useful in the embedded aggregates context. Subsequently, we lean toward a hybrid bus topology, namely the hierarchical-bus arrangement. Unlike the linear bus, the hierarchical bus has multiple endpoints. A heirarchical bus has a tree structure, with a bus at each tree node. Multiple modules connect to each bus. A module connecting to two buses realizes a connection between the two corresponding tree buses.

4.2.2 Logical Topology

The choice of physical topology was guided by the flexibility it offered at the hardware level, its ability to support embedded aggregates design criteria, and scalability, cost and complexity of the physical implementation. The logical arrangement of the modules guides the data transfer amongst them and is strongly linked to the Media Access Control (MAC) protocol adopted. In our discussions of the logical topology, we will present and discuss the MAC protocols and choose a logical topology that is compatible and complementary to the chosen hierarchical-bus physical topology.

Media Access Control (MAC): In examining the physical topology, we leaned toward limiting

the number of data paths required for communication. This optimization results in topologies that require several modules to communicate over the same data-path. The sharing of the data path benefits from moderation and is accomplished through a MAC mechanism. Several MAC protocols have been developed and studied in wired and wireless communication settings. We will now present and discuss the suitability of a few of these mechanisms in the embedded aggregates context.

Polling: In polling, one module acts as the physical data-path arbitrator and periodically polls other modules connected to the data-path. The fetch-push polling mechanism prevents modules from transmitting on the data-path at their discretion and can be constraining for hard real-time embedded applications. By carefully planning the frequency of polling and controlling the data-path occupancy based on the number of sharing modules, however, soft real-time targets can often be comfortably met. Furthermore, polling can be implemented utilizing modest CPU resources. Hence, the inherent simplicity and determinism in polling make it suitable for use in the embedded aggregates context.

Time Division Multiple Access (TDMA): TDMA is a data-path sharing mechanism where communication on a channel is performed in distinct time slots. TDMA is deterministic, can guarantee a contention-free transmission slot for each module and is extensible to dynamic contexts. In TDMA, modules transmit data on the data-path in their alloted slot, which is calculated using a local clock maintained in-sync with a global clock. This synchronization is achieved through special synchronization messages transmitted by a module that acts as the communication monitor. Yet, modules transmitting within the bounds of their alloted slots is an ideal situation. Often clock skew and synchronization discrepancies require guard bands between successive transmissions, reducing throughput.

TDMA, unlike polling, enables modules to exchange data without the need for a store and forward

mediator. This feature introduces irregularity in the arrival of data, however, and the associated processing time is detrimental to module operation. Context switching penalties owing to switching from their primary operation to communication-related activities quickly negate this advantage of TDMA. In comparison to polling, TDMA offers little advantage with the extra overheads and is unfavorable in the embedded aggregates context.

Token Ring & Token Bus: The token ring and bus protocols involve passing a contention-free time slot for data transmission on the bus from one module to another using a data packet called a token. A module possessing the token may transmit data and is required to be aware of and pass the token to the next module on completion of its transmission activities. Hence, the reformation of the ring on the entry or exit of modules is required. Token protocols have found extensive use in many computer networks, however, they demand intensive software and hardware mechanisms for robust functioning. While their determinism and starvation prevention capabilities make them suitable for embedded aggregates use, the software and hardware overheads make them less favorable than polling and unsuitable for embedded aggregates.

Carrier Sense Multiple Access (CSMA): The CSMA scheme and derivative CSMA-Collision Detect (CD) entrust modules with the responsibility to scan the data-path for any ongoing communication before transmitting their own data. The CSMA-CD protocol enhances CSMA, with modules performing a predefined operation on detecting collision resulting from two or more modules beginning transmission simultaneously on an idle channel.

On collision, modules stop transmitting, and the communication channel is returned to idle by a jamming signal transmitted by the communication arbitrator. A truncated exponential backoff is used to space out transmission retransmissions. However, often the bit dominance properties of the transmission channel are utilized to support bitwise-arbitration and achieve throughput improvements. Nevertheless, the flexibility offered by CSMA aligns well with module

independence, communication requirements and hot-plugging features envisioned in embedded aggregates. CSMA is therefore a favorable protocol for use in embedded aggregates.

In conclusion, both polling and CSMA present various trade-offs and are favorable for use in the embedded aggregates context. However, owing to their inherent differences and incompatibilities, choosing one mechanism precludes the other. We shall hence defer the MAC selection and reevaluate the choices in the light of suitable embedded communication protocols. Consequently, we will also defer the selection of the logical topology. Going forward, we will consider and discuss the various available embedded protocols next, following which we will progressively refine our choice of MAC mechanism and logical topology.

4.2.3 Embedded Communication Protocols

As presented in Figure 3.1, several embedded communication protocols have been developed and are built into present day micro-controllers. In the selection of communication protocols, we exclude custom buses as they often lack hardware level support enjoyed by other established protocols. Previously, in the conceptual parallels section, we discussed and situated the protocols and mechanisms for embedded aggregates in the context of other ubiquitous standards. Following our earlier assertions, it is clear that protocols and standards developed for long distance embedded communication, like RS-485, and other industrial serial buses like Fieldbus, are too elaborate for use in embedded aggregates with needless hardware, signaling and wiring overheads. Consequently, we focus on protocols we classify as for on-board and closely-spaced board-to-board buses.

RS-232 and SPI: In our discussion of physical topologies, the star topology was found to be restrictive to our use case. Following this assessment, we drop both RS-232 and SPI buses, insofar that RS-232 requires a star topology for interconnecting several modules due to lack of native multi-drop capabilities. SPI, though a multi-drop bus, requires an enable line that demands a star

topology centered around the bus arbitrator.

UNI/O and 1-Wire bus: The UNI/O bus and 1-Wire bus are both multi-drop capable, and therefore, are favorable toward realizing the bus topology. However, both these buses support and are designed for lower datarates than other available choices and so are less favorable.

CAN and LIN: The CAN and LIN buses from the automotive world bring in very interesting features. They are multi-drop buses, implemented in a large set of commercially available micro-controllers, well specified, and designed for electrically and thermally noisy environments. The LIN bus is a small and slow network system protocol that is complementary to, and often used as a sub-network of, the larger CAN network. The LIN network limits the network to 16 modules and hence is restrictive in the embedded aggregates application.

The CAN bus, on the other hand, can address 128 unique devices, supports data rates up to 1Mbit/s, provides a simple bus arbitration scheme based on CSMA with bitwise arbitration (CSMA/BA) and so shows more promise. The CAN protocol restricts the size of message data to 8 bytes, however; this can be insufficient in the embedded aggregates context. Furthermore, the CAN bus specification, due to its proprietary nature, licenses and the strong control enforced by the patent holders, is considerably stifling and inflexible for application in embedded aggregates.

I^2C, SMBus and PMBus: The I^2C bus protocol and its derivatives SMBus and PMBus are a collection of multi-drop, multi-master, single-ended, two-wire serial buses. I^2C protocol neither specifies message packets nor does it provide built-in mechanisms to form and support a network of modules. The specification includes only electrical and physical wiring guidelines and identifies messaging primitives.

The SMBus and PMBus extend the I^2C specifications, with a particular eye toward use on a computer motherboard. They add support for dynamically addressing modules, error checking

mechanisms, and introduce rigid voltage and data-rate specifications. As such, the SMBus and PMBus specify and offer mechanisms and several features envisioned in the communication infrastructure for embedded aggregates. However, SMBus is implemented in a limited number of digital devices, and PMBus, still in its infancy, is yet to gain wide scale adoption.

The basic I^2C emerges as a better alternative due to its widespread presence in micro-controllers and largely unspecified nature, which provides room for customization and optimization oriented toward embedded aggregates. In conclusion, of all available embedded communication protocols, we find the I^2C specification as the best suitable physical layer protocol for embedded aggregates.

Refining Topologies, MAC and Protocol

In our discussions in this section, we have chosen a hybrid hierarchical-bus as our physical topology with I^2C as the physical layer protocol, and we have two candidates for MAC: polling and CSMA. To choose one of the MAC mechanisms, let us situate and discuss interactions between the MAC mechanisms and I^2C. The devices on an I^2C network are of two types, master and slave. In the master-slave configuration, an I^2C master uses polling to send and receive data from a slave. Natively, the I^2C specification allows multiple masters on the network with each capable of communicating with the slaves individually. In such a multi-master situation, I^2C resolves any bus contention through CSMA/BA. Hence, the choice of I^2C protocol, on the outset, does not determine the choice of MAC mechanism. This calls for further comparison between polling and CSMA in terms of flexibility and overhead.

First, the best effort performance of CSMA, ensuing nondeterminism, and increased module-side operations toward channel monitoring and back-off weigh against its flexibility and channel utilization advantages in comparison to polling. Second, CSMA performance drops significantly as channel congestion increases due to repeated collisions, and CSMA is susceptible to starvation under bit-arbitration [Abr, TB80]. Increasing the bit-rate of the communication medium to alleviate

some of these issues, while plausible, begs for faster CPUs. Bit-rate improvement finds challenges in the I^2C protocol. I^2C implementations in most chips are limited to 400 Kb/s, though the protocol specification does allow 3.4 Mb/s.

In an I^2C network, two modules in the master mode are unable to exchange data. A master losing contention is, consequently, unable to receive data from the winning master. This creates obstacles in a multi-master mode supported by CSMA-BA. Similarly, two modules in the slave mode are also unable to communicate directly between each other. The implicit store and forward capabilities of the polling master help to overcome the slave mode issue, albeit with bandwidth overhead. Finally, in a bandwidth-restricted scenario, polling offers advantages over CSMA including better performance in terms of number of modules sharing the bus and determinism [Lam80]. The above tradeoffs motivate polling as a better fitting MAC mechanism.

In the beginning of the section, we set out with a goal of putting together parts of the communication infrastructure from already available protocols. We aimed to develop a system architecture and define the basic mode of communication among the modules. In conclusion, as discussed earlier, a communication architecture that enables the mutual communication between any two or more modules at their own discretion is undoubtedly best suited for embedded aggregates. Other optimizations, such as infrastructure setup costs, processing costs and ubiquity of physical interconnect protocols, amongst others, however, are found to be orthogonal to the realization of such an infrastructure. We conclude that a logical star topology, using polling on each bus of a hierarchical I^2C network, is best suited in the embedded aggregates context, providing reasonable flexibility through limited compromise on communication independence. Next, we discuss the algorithms and mechanisms that complete the communication infrastructure.

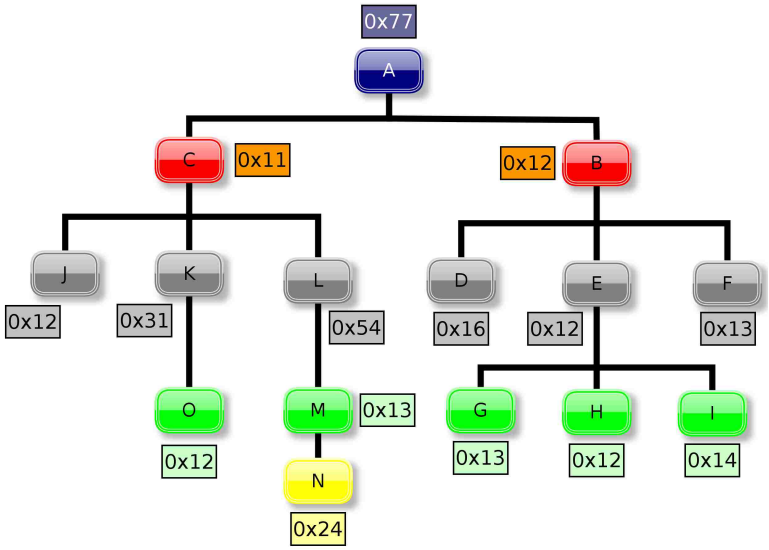


FIGURE 4.1: A simple embedded aggregate. The illustration here shows an embedded aggregate composed of 15 modules and 5 levels of logical hierarchy. Module A in blue is called the root communicator and is the root of the tree structure. The various colors identify the modules in each level of hierarchy. The 7-bit I^2C addresses of the modules are denoted by their side. In the hexadecimal representation of the address, if a module has the I^2C address 111 0000, we write its address as $0x70$.

4.3 Embedded Aggregates I^2C Network

Our embedded aggregates communication architecture follows a hierarchical tree structure built upon many single-master I^2C buses. Some of these masters also connect to another I^2C bus in slave mode and act as bridges between the buses. This bus straddling by the modules leads to the tree structure and hierarchies. The bus where the module serves as a master is logically one level of hierarchy below the bus where it acts as a slave, and every I^2C bus connects several slave modules of one level and a master module from the level above it. For simplicity, the bus is considered to be at the same level of hierarchy as the slave modules attached to it. Figure 4.1 shows an example embedded aggregate. In this example, modules A, B and C are connected to an I^2C bus, where module A acts as the bus master. Modules B and C and the bus connecting them are at level two of the hierarchy. Module B, while a slave on this bus, acts as a master on the bus at level three that connects it to D, E and F. From our interpretation of buses and hierarchies, no bus is at level one, which contains only one module, designated the root communicator.

Modules that straddle two buses either operate on both the buses simultaneously in the appropriate I^2C mode or switch between the master and slave modes and buses. The choice of implementation (switching or simultaneous presence) is left to the hardware designers, however, modules are required to switch rapidly enough to allow the embedded aggregate to meet its temporal goals. In the following section, we shall describe our design of communication packets for transfer of data through these buses.

4.4 Communication Packets

In a connectionless packet switching system, the design of a communication packet plays a pivotal role in system performance. In general, protocol designers gravitate toward lowering overheads and improving transmission efficiency. The effects of packet overheads and sizes on efficiency are further compounded in the embedded aggregates context, due to hard constraints on the physical layer data-rate. For example, while large packets provide a good balance between useful-data size and control-information size, they also lead to longer occupancy of channel and subsequently affect the polling periodicity and conflict with the soft real-time constraints of embedded aggregates. Consequently, balancing functionality and simplicity toward minimizing computation overhead is an important goal in our communication design.

In our packet design efforts, we derive motivation and ideas from the Internet protocols V4 [P+81] and V6 [SD+98], CAN-bus [Bos91], SMBus [For03], and also protocols used to route flits of data in the interconnection networks [DS87, DYN03, DT04] inside a router. Nonetheless, in defining the packet structure, we follow some of the end-to-end principles [SRC84, Moo02, BC01, Pax06, CDC98] introduced by Saltzer *et al.*, and improvised by several others over the decades, and have defined a structure that deviates from traditional Internet packets.

Our design embeds a packet, specifically designed for embedded aggregates, into a basic I^2C

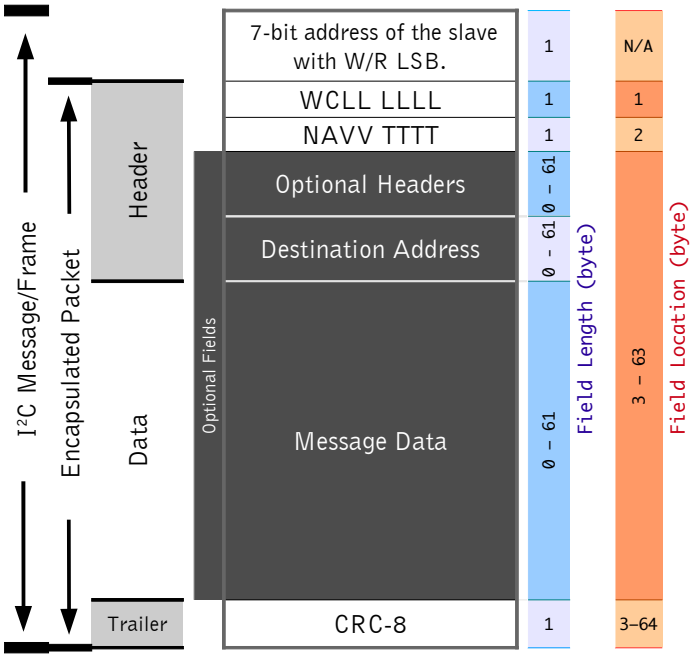


FIGURE 4.2: Embedded aggregates packet structure. When viewed on the bus sans the start and stop bits, a message appears in the above format. The field location is counted from the start of the enclosed packet.

communication frame. A packet in our implementation, as shown in Figure 4.2, is restricted to a size of 64 bytes. This restriction stems from the expected uses of embedded aggregates and traffic patterns. Embedded aggregates are suitable in situations where a task can be accomplished by segmenting it into several sub-tasks that can be processed and accomplished in parallel. Sub-tasks are expected to be executed autonomously, using short communications among processors. Hence, by restricting packet length, we prevent modules from blocking channels with long messages, while permitting frequent exchanges of short messages.

An embedded aggregate packet has a fixed two-byte header, data segment and a one-byte trailer. As shown in the figure, the data segment, extended header and destination address fields are optional, leading to packets of variable lengths between 3 and 64 bytes. We shall now briefly discuss the anatomy of a packet. We begin with details on encapsulating the packet in an I^2C frame and follow with detailed discussions on the design objectives and composition of the three segments.

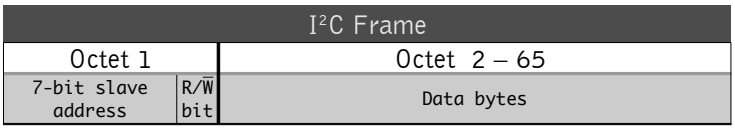


FIGURE 4.3: Embedded aggregates I^{2}C frame. The data bytes in the frame contain our custom designed packet. As shown in the illustration, only 7-bit slave addresses are used in our implementation.

4.4.1 Packet Encapsulation

The I^{2}C bus forms the communication backbone in embedded aggregates. Unlike other physical layer protocols, like the Ethernet [MB76] and CAN bus, the I^{2}C protocol does not specify any packet structure. However, it supports single-message and combined-message transmissions, and we use the single message transmissions that permit the exchange of several bytes of data in a transmission.

Every data transmission on the I^{2}C bus commences with a 7-bit or 10-bit slave address followed by a bit to signify a read or write operation. This is followed by a stream of bytes that are clocked in or out of a slave device by an I^{2}C bus master. This bare-bones approach of the I^{2}C protocol, while minimalistic in its offerings, is flexible and advantageous. We embed our custom designed packet into the stream of data bytes following the slave address. Figure 4.3 shows the complete embedded aggregate I^{2}C frame with the encapsulated packet of data.

4.4.2 Header

Headers and trailers contain the information for packet processing and routing. Generally, message packet headers include fields to describe the packet length, next hop, protocol data, source and destination address, routing information and sequence number. In our design we deviate from this traditional approach and change the composition of these segments toward achieving better performance in the unique situations witnessed in the embedded aggregates context.

Header Fields	
No.	Field significance
1	W- Message destination bit.
	0 The header does not contain any address information. The packet is destined for the modules that receive it.
	1 Destination address follows the extended header field.
2	C- Continuous read/write.
	0 No more messages to be sent in the current polling cycle.
	1 Slave Module: Requesting one more read by the Master module. Master Module: One message to be delivered in the current polling cycle. Next message will follow after predefined guard band.
3	LL LLLL – Packet Length.
	Six bits dedicated for representing the length of the packet. The length of the packet includes the header, data and the trailer segments.
4	N – extended header presence indicator.
	0 Extended header field is empty.
	1 Extended header field non-empty, header bytes follow.
5	A – Request branch level acknowledgment
	Acknowledge CRC validity and that correct packet was received.
	0 No Acknowledgment requested.
	1 Acknowledgment requested.
6	VV – Protocol version number.
	Two bit version number for the protocol. The current version in 00.
7	TTTT – Packet type.
	4-bits to represent 16 different types of packets. 14 different packets have been defined and two are reserved for future use.

FIGURE 4.4: Embedded aggregate packet header parameters. The header is composed of several mandatory and optional fields. These fields assist in message routing and flow control amongst others.

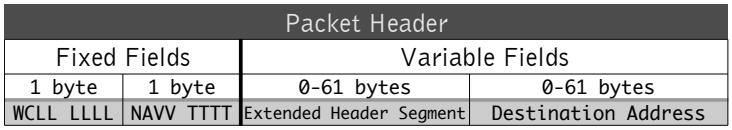


FIGURE 4.5: Header of an embedded aggregate packet. The packet header is of variable length with a two byte fixed sub-segment and two optional sub-segments. The header of the packet follows the read/write bit of the I^2C frame.

Our packet design uses a variable length header, as shown in Figure 4.5. The length of the header depends on the optional extended header and destination address segments. Figure 4.4 describes the various header fields. Our design excludes header fields for holding the length of the header and the offsets for destination address and data segments. We overcome this requirement through extended header and destination address segments that are conducive to recovery (by the modules), without any offset information.

Two common fields of a packet header are the source and destination addresses. In our design, we include the destination address, unless the recipient of the message is the destination as indicated by the W bit set to 0. On the other hand, we find no compelling reason to include the sender address as part of the header and have chosen to exclude it. The reasons for this complete lack of source information are two-fold. The obvious first reason, and advantage, is the decrease in packet size. The second reason stems from the kinds of responsibilities we do not place on the communication infrastructure. In our design, we do not hold the communication framework responsible for providing packet delivery acknowledgments or extending any protection against attack by adversaries. In order to appreciate our stand and rationale behind these decisions, let us look at the implications of a design that lacks the source address in a packet.

To begin with, the sender cannot be provided packet delivery acknowledgments or notified if the receiver is unreachable. This is challenging in a situation where tasks are primarily accomplished through collaborations between the modules. Hence, we defer the message delivery confirmation responsibility to the receiving module following an end-to-end acknowledgment model. Instead

of acknowledging the receipt of the message, the recipient module may selectively confirm the completion of the actions relating to the task as required.

At this size and scale, the chances of modules in a embedded aggregate becoming intermittently unreachable are low. Therefore, we do not strongly require unreachable destination notifications. Nonetheless, we define special messages that can be used to probe the presence of a module. It is expected that a module will use the appropriate message to locate unreachable modules after two attempts of data transmission and lack of end-to-end acknowledgment.

On the destination end, the lack of source address creates two challenges. To begin with, the destination receives the data from an anonymous source, introducing security and trust issues. Generally, we assume implied trust between the modules, and do not consider the lack of source address an issue in this context. However, if trust is lacking, the modularization, unencrypted communication and open-ended design make them more susceptible to attacks by the introduction of adversaries. One can moderate the introduction of modules and enforce security through encrypted communication, though this is expensive, both financially and computationally. We have implemented certain hardware-level features and components, in our blades & tiles system, for fault tolerance and communication that can also be used to enforce limited security functions.

The second challenge to the destination module is responding to a received packet. Packets in embedded aggregates predominantly contain data that is to be consumed by the destination, with no expected response to the sender. A software parallel to this is a function-thread that can terminate when finished as it has no return data. In cases where a response is expected, the sender's address may be included in the data segment. We do not specify where the address is accommodated in the data segment and leave it to the modules to specify it through their API. This approach is comparable to passing the return location as part of a software function invocation.

In addition to the end-to-end confirmation, a limited acknowledgement is incorporated through

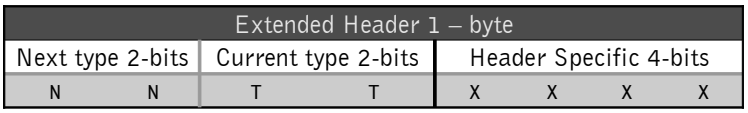


FIGURE 4.6: Extended header. The one byte long extended header is an optional inclusion into the main header. Envisioned uses include communication congestion and status updates.

the use of an acknowledgement field in the header. This field is used to guarantee the successful exchange of a packet across a hop of the tree through retransmissions. When the acknowledgement bit is set to 1, the receiving module confirms to the sender the integrity of the packet and the packet is retransmitted by the sender until correctly received. The wider goals of faithful transmission and task completion are accomplished through branch-level acknowledgements and end-to-end module task-level acknowledgements. While all our required functionalities are realized through the fields provided by the fixed part of the header, we expect several more requirements to arise in the future. Along with the two fixed header bytes, several more fields are envisioned to be implemented through the use of the extended header segment. We shall next discuss the design of our extended header segment.

4.4.2.1 Extended Header Segment

The extended header segment, if present, immediately follows the two fixed header bytes. These headers are currently unspecified; however, we have an initial design for the extended headers that is conducive to modules recovering the header segment, without the requirement of offsets. In our suggested approach, the inclusion of the extended header segment is signified by the extended header presence bit. As shown in Figure 4.6, each extended header is one byte long and has four bits of valid header-specific data and four bits dedicated to signifying the current and next header types. The two-bit header descriptor allows four unique extended header types.

To accommodate an extended header segment of arbitrary size, we use the next header field to signify the end of the header sequence. The next header field must be different than the current

Message Types	
Packet Type	Type of message contained in the packet
0000	Acknowledgment message.
0001	Module inquiry, identification, authorization and current functional status message.
0010	Proxy address configuration message.
0011	System configuration message.
0100	Communication parameters message.
0101	Broadcast message.
0110	Leafcast message.
0111	Module and resource advertisement message.
1000	Network configuration and management message.
1001	Reserved for future use.
1010	Reserved for future use.
1011	Network and transmission error message.
1100	Request for fetching packet from module.
1101	Generic message.
1110	Ping, status and heartbeat message.
1111	Compound message.

FIGURE 4.7: Message types. The data segment of a packet carries sequences of bytes of data whose order and significance are defined by the message type field in the header.

header field, except for the last header of the extended header sequence in which the two fields are the same, indicating the end of the sequence.

4.4.3 Data Segment

The data segment follows the header and carries the data intended for the destination. The type of data contained is qualified by the message packet type, as mentioned in the header description. With four bits to specify the type, our implementation defines sixteen kinds of messages, as shown in Figure 4.7. Each message type specifies the associated order and significance of message bytes. However, in a few types, we make no effort to specify these details and leave it to the hardware implementer to define them, as they are specific to the module. The data segment is restricted to a maximum size of 61 bytes per packet.

4.4.4 Trailer

Although we expect high levels of successful transmissions in a typical setting, the reliability goal necessitates checking for errors. The 1-byte trailer of a packet holds a CRC-8 checksum and follows the optional data segment (Figure 4.2). A CRC-8 for the whole packet carries a large overhead in time. At several way-points, our routing techniques modify the packet header, requiring recalculation of the checksum at each way-point if the checksum covers the entire packet. Instead, our current design computes the CRC-8 only for the data segment of the packet, avoiding the intermediate recalculations. This limited checksum can lead to propagation of incorrect header fields unchecked and, therefore, is a compromise between performance and faithful transmission.

Our use of CRC, given limited resources, emphasizes avoiding the interpretation of an inconsistent packet by the destination over the goal of faithful routing and delivery of the packet. An inconsistent packet, for example, may be generated either due to transmission medium errors or improper parsing and interpretation of the packet by the modules en-route to the destination. Specifically, in our design, an erroneously set W bit and the lack of offsets can lead to a part of the data segment to be interpreted as address information. In such cases, the data may be routed to an unintended recipient. With the partial CRC-8 in place, the packet on reaching the wrong destination will contain an invalid data-segment checksum, leading to the data being discarded by the receiving module.

4.5 Module Address

In our design, the destination address of the packet is an optional part of the header and follows the extended header segment. In this section, we will describe our module addressing scheme and discuss how it lends itself to our visions of sculpting embedded aggregates composed of several hundred modules. All our source and destination addresses are uniformly specified in our packets

using this addressing scheme.

Connectionless packet switching requires nodes to be uniquely addressed for data routing. Addressing is an inherent feature of the I^2C protocol, with each slave identified using a 10-bit or 7-bit address. The 10-bit mode provides a larger pool of addresses over the 7-bit, but requires two bytes of data to communicate the address. Consequently, in a system design where the modules are assigned predefined noninterchangeable addresses, the 10-bit mode supports a larger library of modules, some or all of which may be used to form an embedded aggregate. While this advantage, when considered in isolation, casts 10-bit addressing as a better choice, the advantages quickly diminish when considered in the light of other challenges and requirements.

The 7-bit addressing can support fewer modules than 10-bit addressing, but available address space is not the only factor that bounds the number of modules on the bus and, consequently, in a system. The electrical capacitance of a bus also plays a restrictive role and varies based on the length of the bus and characteristics of the attached modules. Often, the bus capacitance leads to communication issues when the module population nears a few dozen. This module limit can be more economically achieved using the 7-bit addressing, which supports as many as 128 unique addresses, and also reduces the size of the messages by one byte. Hence, we implement a hierarchical bus using statically or dynamically assigned 7-bit addresses and support address reuse. A hierarchical bus overcomes the module population limitations due to bus capacitance and can implement systems with several hundred modules. Additionally, it also provides ways for easy isolation of faulty branches and nodes and extends capabilities for achieving the reliability design goal.

4.5.1 Addressing Policy

The 7-bit I^2C addressing mode provides 128 unique addresses; however, not all of them are available for use. The I^2C specification classifies 16 of these addresses as reserved for specific

Address Space Segmentation	
I²C Address	Description
0x00 to 0x07 and 0x78 to 0x7f	Reserved by I²C specification
0x77	Root communicator
0x76 and 0x74	Default addresses of new modules
0x75	Gateway port to external systems
0x08 to 0x0f	Reserved for future use in embedded aggregates.
0x10 to 0x73	Address available for use in embedded aggregates.

FIGURE 4.8: I^{2}C address space segmentation in embedded aggregates. Of the 128 addresses available in the 7-bit addressing scheme, 28 are reserved addresses for specific purposes and 100 addresses are available for use by the modules.

purposes. One such address is the Global Call address, 0x00. In addition to the above reserved addresses, we restrict the use of another 12 addresses. The remaining 100 addresses are available for use in addressing modules, both statically and dynamically. With two levels of hierarchy, these 100 addresses can support as many as 10000 modules. In our addressing method, as shown in Figure 4.8, we have reserved addresses for special situations and for future use. These include new modules awaiting address assignment, external communication channels and special modules, like the root of the tree structure, as discussed below.

4.5.1.1 Root Communicator

One address reserved is for the root communicator. The root communicator performs the leading role in system structuring, control and maintenance, and communication. In our hierarchical I^{2}C communication infrastructure, the root of the tree structure is the I^{2}C bus arbitrator and the master of the topmost hierarchy. In our example network in Figure 4.1, module A colored blue is the root of the tree, that is, the root communicator. On powering up, this module always assumes the static address 0x77 and performs its communication and organizational responsibilities.

4.5.1.2 Default Address of New Modules

Modules in our system may have a static I^2C address or may have an address dynamically assigned to them. Nevertheless, when a module first joins the bus, it assumes an uninitialized slave mode with I^2C address $0x76$. Following this, the module participates in address assignment rounds toward acquiring an address and initializing.

4.5.1.3 Gateway Modules

Our networking infrastructure provides communication capabilities inside an embedded aggregate. Simultaneously, there is a requirement for communication with external entities, such as computers, generic electronic systems and other embedded aggregates, and communication with them is infeasible using this internal communication infrastructure. Therefore, we have designed several modules called gateways aimed at facilitating external communication.

Gateways are essentially protocol translators that can communicate with an external entity using a different protocol and form a bridge between the two systems. For example, we have implemented a gateway that can interface with an external computer using USB or Bluetooth protocols. We have reserved address $0x75$ to denote the external communication channel of a gateway module. This address supplements and is different from the protocol address assigned to the gateway module. An aggregate may contain several gateways, and § 4.6.2 supplies details on sending and receiving packets through the gateways.

4.5.2 Embedded Aggregates Protocol Address Representation

In our design, the 7-bit I^2C addresses are reused, and the same address is often assigned to two or more modules on different I^2C buses. Consequently, a packet cannot be uniquely routed using only a 7-bit module address. A similar situation is often encountered in traditional Internet due to the

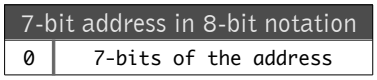


FIGURE 4.9: Representation of the 7-bit I^{2}C address in 8-bit format. This representation forms the basis for the multi-byte protocol address and adds a single bit field to qualify the next hop as a way-point or the destination.

use of a private network IP address, where it is resolved using Network Address Translation (NAT). A NAT-like mechanism is cumbersome in the embedded aggregates context, however. Instead, we introduce a new address notation called the protocol address that uniquely identifies each module and also provides routing information for switching a packet through the hierarchy of modules. A protocol address is one or more bytes long and depends on the module's location in the hierarchy.

The protocol address derives from the I^{2}C addresses of the module and its ancestors in the tree. The multi-byte protocol address of a module is created by prefixing its address with addresses of its ancestors in the decreasing order of ancestry. As the root communicator is the ancestor of every module, we uniformly omit it from all module protocol addresses. For example, all modules in level two of the hierarchy have a one byte protocol address. The modules at level three that are children of the above modules have a two-byte protocol address, and the size of the protocol address extends by a byte for each successive level of hierarchy. A gateway port on the root communicator is an exception to the rule of omitting the root communicator address, and the protocol address of the port is two bytes long and includes the address of the root communicator (Figure 4.11).

Similar to the extended headers, a protocol address and, consequently, the destination address segment is designed to be recovered without length and offset information. A protocol address uses an 8-bit representation of the 7-bit I^{2}C address and modifies the extra bit to signify way-points and hops. The byte order of the I^{2}C addresses of the module and its ancestors together with the hop bit enable us to create a sequence that can be extracted deterministically from the message packet as follows. Firstly, as shown in Figure 4.9, we represent the 7-bit address of any module in a 8-bit form, with the leading bit set to 0.

Full address of a module					
8-bit address of module at Level 2		Levels 3 to N-1	8-bit address of module at level N	Levels N+1 to parent of module.	8-bit address of the module
1	7-bit I²C address	. . .	1 7-bit I²C address	. . .	0 7-bit I²C address

FIGURE 4.10: Multi-byte embedded aggregates protocol address. This address uniquely identifies each module in the system and its length depends on the location of the module in the hierarchical communication framework.

Module Protocol Address	
Module	Address in Hexadecimal notation
A	0x77
C	0x11
L	0x9154
M	0x91D413
N	0x91D49324
H	0x929212
A's gateway port	0xf775
Default gateway	0x75

FIGURE 4.11: Protocol addresses. Protocol addresses of modules in Figure 4.1.

Next, we construct its multi-byte protocol address by combining the 8-bit representations of the I^2C addresses of its ancestors in the decreasing order of ancestry and setting the hop bits of the ancestors to 1. Finally, as illustrated in Figure 4.10, the 8-bit representation of the I^2C address of the module with the hop-bit set to 0 is added as the last byte in the sequence.

The end of the destination address segment is signified by the first byte following the extended headers that has its hop-bit set to 0. For the purpose of illustration, let us consider the configuration of modules in the hierarchy letter shown in Figure 4.1. In the figure, the I^2C addresses of the modules in 8-bit notation are denoted along with a letter identifying the module. Figure 4.11 shows the protocol addresses of a few modules. In I^2C communication, only slave modes have addresses associated with them. A module is addressless in the master mode. In Figure 4.1, for example, modules B and E have the same 7-bit I^2C address in their slave modes, however, on the bus connecting them E has the address $0x12$; whereas, B is the master and addressless. Hence, a parent and a child can have the same 7-bit I^2C address.

Slaves on the same bus are assigned distinct I^2C addresses and, consequently, different protocol addresses. The I^2C address of a module can, however, be reused in a different bus at the same level of hierarchy. The uniqueness of their protocol addresses in this case is a consequence of their different ancestors. In the next section, we shall discuss how this distinctive protocol address aids in the routing of packets.

4.6 Routing

In a typical packet switching network, each module faces the task of either consuming a packet or forwarding it to the next hop. A module will either receive the address of the next hop as part of the packet or is held responsible for deciding on a suitable next hop. Our design uses the former by virtue of the protocol address, which is conducive to source routing. The source module may have the destination's protocol address or procure it dynamically through address request messages and resource advertisement messages supported by our protocol.

The transfer of a packet between any two modules takes place through a series of pulls and pushes. A packet is pulled upwards through the hierarchies to the lowest common ancestor of the source and destination, and then pushed down the tree to the destination module. A pull is accomplished by a master module clocking data out of a slave module on the I^2C bus, and the push by clocking the data into a slave. To illustrate, in Figure 4.1, to perform a packet transfer from H to D, first E pulls a packet from H, then B pulls the packet from E, and finally B pushes the packet to D.

Processors must possess some buffer space to receive and process messages. We do not make any assumptions on the size of the available buffer, and our specification provides special messages to communicate buffer issues and negotiate packet lengths. Creators of an embedded aggregate should choose a minimum transmission unit (MTU), i.e., packet size, based on their holistic view of the system. In systems of modules with no previous knowledge on the MTU, however, a trial

and error method of sending packets, beginning with a conservative size and resizing packets based on error messages, is prescribed.

Our routing mechanism has been intentionally kept simple and disallows splitting of packets by routing nodes, and a module drops messages exceeding its buffer size. On the other hand, a routing module can bundle several messages into a single message packet, when plausible. In any transmission, it is the responsibility of the source to package the message in appropriately sized packets, taking into account the buffer sizes of the intermediate nodes and the destination. In the following discussions, we shall describe the routing algorithms and the role of the protocol address. We begin with a discussion on the communication inside a single aggregate then examine external communication through the gateway modules.

4.6.1 Intra-Aggregate Communication

Routing in embedded aggregates is strongly tied to the intricacies of the underlying I^2C layer and, as discussed earlier, involves several upward pulls and downward pushes. The pulling and pushing mechanisms involve a coordinated effort between the master and slave modules. We employ different strategies in pushing and pulling packets in an effort to reduce packet sizes, channel occupancy, and load on routing nodes and increase the throughput. We accomplish this by modifying the address and destination indicator field, 'W', in the header and the destination address.

4.6.1.1 Pulling from I^2C Slave to Master

The creators of the I^2C protocol envisioned its use predominantly in the context of interfacing memory modules and less intelligent components to microprocessors. Consequently, the protocol does not provide mechanisms that allow the master and slave to negotiate the size of the exchanged data. For example, when reading from a slave, the master clocks out data of size decided by it

with no input from the slave. This is particularly constraining in the aggregates context, as the modules operate mutually independently of one another, unaware of the data each module may have available for sending. Fixing the size of the packet to solve the issue is an unfeasible solution, and to overcome this challenge, our packets include the size of the packet in the first byte. The introduction of the packet size field leads to an inversion of control, wherein the slave can specify the size of packet it has available for sending. A master module polling a slave pulls (reads) the appropriate number of bytes, as mentioned in the packet size field. As the master module polls the slave at periodic intervals, a slave with no data for transfer sets the packet size field to zero.

In a packet transfer from a slave to a master, the slave module holds the responsibility of creating valid data packets in a timely fashion and the master module is required to read the packet from the slave. On I^2C address match with read flag, the slave module places the packet on the bus in accordance with the I^2C signaling and timing specifications, starting from the header byte to the trailer byte. If the master module reads a zero for the packet length, it ends the transmission by reading one more byte, followed by a stop bit, else, it continues to read the full packet of specified size. On the completion of the packet transfer, the master module either consumes the packet or passes it to the next hop.

As described in Algorithm 1, the packet is consumed by the master if either the destination field 'W' is set to zero or the destination address matches its protocol address. In the events requiring further routing of the packet, the next hop of the packet depends on the location of the destination in the tree, with respect to the master module. If the master is an ancestor of the destination, the master strips the part of the address prefix that corresponds to its address and pushes the message to its appropriate child that is an ancestor of the destination. Otherwise, the packet is left unmodified and is similarly pulled by a module one level of hierarchy above the current master.

To illustrate our routing process, let us consider an example packet transfer from E to M, in

Algorithm 1: Algorithm for packet handling on pull from child.

```
1  foreach Packet do
2      if Wbit = 0 then // packet does not need to be passed on, the receiver is
       the destination
3          Consume Packet;
4      else
5          Extract Destination_Address;
6          if MSByte(Destination_Address) is a Reserved_Address then
7              ...// routines for other reserved addresses
8          else
9              if Destination_Address is equal to Self_Address then // packet addressed
               to self
10                 Consume Packet;
11             else if Self_Address is equal to 0x77 then // self is the root
               communicator
12                 Set Next_Hop_Address = MSByte(Destination_Address);
13                 Strip Next_Hop_Address from Destination_Address;
14                 Decrement Packet_Size by 1;
15                 Set Wbit = MSBit(Next_Hop_Address);
16                 Write Packet to Next_Hop_Address << 1;
17             else if Self_Address is an ancestor of Destination_Address then // packet
               addressed to descendant
18                 Strip Self_Address from Destination_Address;
19                 Set Next_Hop_Address = MSByte(Destination_Address);
20                 Strip Next_Hop_Address from Destination_Address;
21                 Decrement Packet_Size by length(Self_Address)+1;
22                 Set Wbit = MSBit(Next_Hop_Address);
23                 Write Packet to Next_Hop_Address << 1;
24             else
                   // packet addressed neither to self nor descendant
                   Send Packet to the module one level above;
```

Figure 4.1. Initially, a packet intended for M is created by E, as shown in Figure 4.12, with the W bit set to 1 and destination protocol address $0x91D413$. Module E presents this packet when polled by module B, its parent, and on receiving, module B extracts the destination address as the W bit is set to 1. Next, module B looks for matches between its address, $0x12$, and the destination address, $0x91D413$, and the lack of a partial match between the addresses implies that module B is not an ancestor of the destination M. Consequently, module B passes the packet without modification to module A.

If module B is an ancestor of M, then the protocol address of M will begin with the protocol address of B, with leading bit of the last byte of B's address set to 1. This follows from our protocol address notation discussed earlier and the fact that an ancestor is a way-point to the destination. For example, let us consider module L with the protocol address 0x9154. Here, the address of L is contained in M's address with the leading bit of the last byte, 0x54, set to 1. L is an ancestor of M, more precisely its parent.

Due to our addressing notation, the root communicator is an exception and its protocol address is not a part of any other module's address though it is a parent or ancestor of every one of them. As the address of A is included in the protocol address of the gateway port implemented in A, this gateway port will then take the address $0xF775$. When communicating with one's parent, the address field may be avoided. For example, module E can create a packet intended for module B with the W bit set to '0' and drop the address field. The setting of the W bit indicates to module B that the packet is intended for it and lacks a destination address.

4.6.1.2 Pushing from I^2C Master to Slave

In packet pulling, a module forwards a packet, with no modification, to its parent when it is not an ancestor of the destination. However, in the push routing, we modify the address field toward reducing packet sizes. The algorithm for pushing packets is described in Algorithm 2. Let us follow

Packet Flow from E to M				
Module	Packet Size	W Bit	Dest. Address	Description
E	S	1	0x91D413	Packet destined for M created and available for transfer to B.
B	S	1	0x91D413	Packet pulled from E.
	S	1	0x91D413	Destination not a descendant and packet available for transfer to A.
A	S	1	0x91D413	Packet pulled from B.
	S-1	1	0xD413	Packet sent to C with I^2C address 0x11.
C	S-1	1	0xD413	Packet pushed from A.
	S-2	1	0x13	Packet pushed to L with I^2C address 0x54.
L	S-2	1	0x13	Packed pushed from C.
	S-3	0	None	Packet pushed to M with I^2C address 0x13.
M	S-3	0	None	Packet received and consumed.

FIGURE 4.12: Packet flow between two modules in the same aggregate. The table shows the flow and route of the packet from module E to M. The green line identifies the hop where the pull mechanism gives way to the push mechanism as the packet has reached the lowest common ancestor. The decreasing packet size and change in address information past the lowest common ancestor is a consequence of the push mechanism.

our earlier example and trace the path of the packet further, as illustrated in Figure 4.12. As C and B cannot communicate directly as siblings, the packet is from B by A. As the packet is not addressed to the gateway port, A sends the packet to the module with the I^2C address implied by the first address byte.

However, a few modifications are made to the packet before the push. First, the address of the next hop, C, is removed from the destination protocol address and the value of the MSBit of the extracted byte is placed in the 'W' field. Hence, the packet when sent to C has the 'W' bit set to 1 and destination address $0xD413$. At C, the packet is similarly pushed to L, after modifications to the 'W' bit, destination address and size fields. Finally, the packet is sent from L to M; however, now the destination address field is removed, and the W bit is set to 0. M, on receiving the packet, consumes it as it is the intended destination.

A packet from E to M travels through the root-communicator and requires five hops. Generally, packets can be diverted at lowest common ancestor without passing through all the hierarchies. In

Algorithm 2: Algorithm for packet handling on push by parent.

```
1  foreach Packet do
2      if Wbit = 0 then // packet does not need to be passed on, the receiver is
       the destination
3          Consume Packet;
4      else
5          Extract Destination_Address;
6          if MSByte(Destination_Address) is a Reserved_Address then
7              ...// routines for other reserved addresses
8          else
9              if Destination_Address is equal to Self_Address then // packet addressed
               to self
10                 Consume Packet;
11             else
12                 Set Next_Hop_Address = MSByte(Destination_Address);
13                 Strip Next_Hop_Address from Destination_Address;
14                 Decrement Packet_Size by 1;
15                 Set Wbit = MSBit(Next_Hop_Address);
16                 Write Packet to Next_Hop_Address << 1;
```

the case of E and M, however, A is the lowest common ancestor. For illustration, let us consider a packet transfer from D to G. This process commences with D creating a packet addressed to G at $0x929213$. As dictated by our routing scheme, on receiving the packet, B recognizes that it is an ancestor of G. Subsequently, B removes the most significant bytes from the address of G, leaving only information that sufficiently describes the route to G from B. The modified packet now has destination address $0x9213$ and W bit set to 1, and is similar to a packet pushed from A to B. B now further modifies the packet destination address field to $0x13$ and sends the packet to node E, with the W bit set to 1. The next push is the last transmission of the packet and it reaches G in three hops.

4.6.2 External Communication

The push and pull mechanisms form the basis of routing internal data in embedded aggregates. Uniformity in addressing and communicating with internal and external entities alike is an important goal. We next describe the addressing of external entities and illustrate how push and pull schemes are used in communication through gateways.

4.6.2.1 External Communication through Unified Addressing and Routing

A gateway module has one or more communication ports in addition to the I^2C communication used for internal networking. These communication ports offer links to external systems through protocols, including Bluetooth, 802.11 networks, USB, RS-232 and Ethernet. The packet structure for internal communication was based on the internal dynamics of embedded aggregates. Similarly, we have defined specific packet structures for external communication with generic digital systems that exploit the capabilities of the communication channel and the external entity. The details of these different packets are outside the scope of this paper. However, in situations where two embedded aggregates are connected using an external link, the repackaging of data is avoided, and

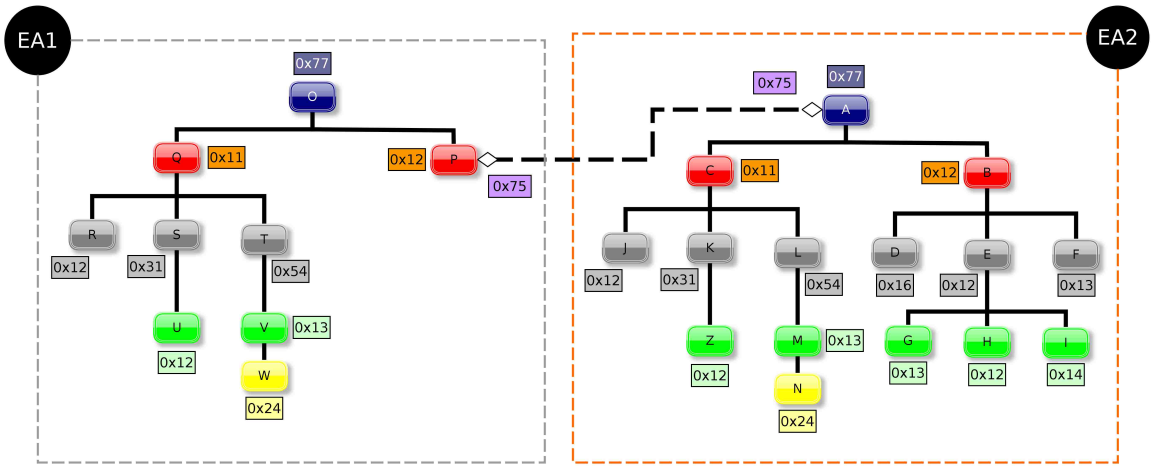


FIGURE 4.13: An embedded aggregate pair. The two aggregates EA1 and EA2 are interconnected through an external link between P and A. An important distinction between the two aggregates is the consolidation of internal and external communication abilities. In EA1, the two functionalities are split between O and P, while in EA2, module A acts as both the root communicator and the gateway module.

the internal packets are delivered unmodified and encapsulated in the external protocol's packets or frames.

Our design allows extending the internal protocol addressing and routing for enabling communication between modules across different connected embedded aggregates. Let us consider the two networks in Figure 4.13 for the purpose of illustration. EA1 and EA2 are connected through an external communication link between module P in EA1 and module A in EA2.

Consider a simple packet transfer from module R in EA1 to module J in EA2. Similar to in-system packet transfer, R creates a packet with the address of J. The address of J in EA2 address space is $0x9112$, but this address is insufficient for routing a packet from R to J. However, if the packet is introduced into EA2 with the address $0x9112$, it will be delivered to J. This can be accomplished by passing the packet through module P in R's network. Module P can be reached by address $0x12$. The external port of P is associated with the reserved address $0x75$. This addresses port is treated like a virtual module under P so its protocol address is $0x9275$. Module P holds the responsibility of sending the packet through the external link, as described in Algorithm 3, and the other modules

are oblivious to the communication intricacies.

Algorithm 3: Algorithm for routing packets at the gateway.

```
1 foreach Packet do
2     if (MSByte(Destination_Address)) & 0x7F is equal to 0x75 then
3         Set Next_Hop_Address = MSByte(Destination_Address);
4         Strip Next_Hop_Address from Destination_Address;
5         Decrement Packet_Size by 1;
6         Set Wbit = MSBit(Next_Hop_Address);
7         Send Packet to over the appropriate communication
          channel to external entities.;
```

A packet addressed to $0x9275$ in EA1 will be delivered to A by P with no address field and the W bit set to 0. Hence, $0x9275$ is essentially a proxy for A in EA1. Now, building the full address of J is much simplified. Since the packet is not destined for A, we set the leading bit of the second byte of the address field to 1, making it $0x92F5$. The address of J, $0x9112$, is now concatenated with it to create the complete address $0x92F59112$. Thus a packet introduced with address $0x92F59112$ in EA1 will be delivered to J in EA2, as shown in Figure 4.14.

4.6.2.2 Default Gateway

Towards further simplifying external communication, we introduce the abstract notion of a default external communication channel or gateway. Our implementation allows the default gateway channel to be reached with the address $0x75$. In systems with a single gateway, the message will be delivered to the gateway by the communication modules by appropriately modifying the packet destination address. Not every communication module needs be capable of modifying the address. The root-communicator is ultimately responsible as the packet is bound to reach the root communicator. For example, in our packet from R to J in Figure 4.14, the packet may be addressed to $0xF59112$ by module R. This approach is also valuable in situations where the modules are unaware of the address of the gateway and have resource constraints that prevent them from acquiring the address.

Packet Flow from R to J				
Module	Packet Size	W Bit	Dest. Address	Description
R	S	1	0x92F59112	Packet destined for J created and available for transfer to Q.
Q	S	1	0x92F59112	Packet pulled from R.
	S	1	0x92F59112	Destination not a descendant and packet available for transfer to O.
O	S	1	0x92F59112	Packet pulled from Q.
	S-1	1	0xF59112	Packet sent to P with I^2C address 0x12.
P	S-1	1	0xF59112	Packet pushed from O.
	S-2	1	0x9112	Packet intended for external device. Packet sent through communication port.
A	S-2	1	0x9112	Packet received from external device.
	S-3	1	0x12	Packet pushed to C with I^2C address 0x11.
C	S-3	1	0x12	Packet pushed from A.
	S-4	0	None	Packet pushed to J with I^2C address 0x12.
J	S-4	0	None	Packet received and consumed.

FIGURE 4.14: Packet flow between two modules on different embedded aggregates. The table shows the packet flow from module R to J. The blue line in the table shows the aggregate boundary. The packet flows through an external communication link on this hop. Another point to note is presence of only one green line signifying the change from pull to push. This may however not be the case always. Depending on the location of the two gateway modules, the packet may be pulled and pushed in both the systems. In this situation the packet is only pushed in EA2 as it is introduced by the root communicator.

In the current world of multiple wired and wireless communication technologies, it is only logical to expect several communication links to external entities. The links may uniquely connect to different external systems or a collection of them may connect an aggregate with a single external device. In situations where multiples links exist, the system may benefit from choosing the links on a case by case basis. The default gateway approach is beneficial here, as the root communicator, and other communication oriented modules, can modify the address of the packet based on factors including current traffic patterns and load on a particular channel. Furthermore, in the multiple unique links scenario, one channel may be designated as the default gateway and can use the shortened address notation. For example, the channel with the maximum traffic may be associated with the shorter address for overall reduction in data transfer and channel occupancy.

4.6.2.3 Virtual Gateway

In implementing gateways, several external communication links can be unified and consolidated in a single module, creating compound-gateways. For example, a single module may provide wireless communication capabilities, like Bluetooth and 802.11b/g modems. Additionally, using higher communication protocols like TCP/IP, gateways may use a single communication resource, like Ethernet or 802.11b/g, to communicate with several external entities. Both the above cases introduce challenges due to the plurality of links and only one reserved address to specify them. Furthermore, as we expect to support an open-ended system where modules may be replaced on the fly, we also want to support situations where several gateway modules may be replaced with a single compound-gateway, with reduced interruption of communication and readdressing. Towards overcoming the above challenges, our design outlines virtual gateways.

Under the virtual gateway addressing method, a compound-gateway module may be associated with several I^2C addresses and consequently, protocol addresses. While the compound-gateway module accepts and responds to only one I^2C address, the translation between the virtual address

and the real address is performed by the root communicator using lookup tables. The compound-gateway similarly uses lookup tables toward choosing the appropriate communication channels, as notified by the packet address. This approach abstracts the gateway implementation, allowing modules to address resources and exchange packets, without addressing changes between the unified and distinct gateway instances. However, this advantage comes at the cost of increase in root communicator and compound-gateway routing complexity and should not be expected to be supported by all systems.

4.7 Flow Control

Our vision for embedded aggregates includes a heterogeneous system consisting of modules with different processing speeds and I^2C communication capabilities. The I^2C specification outlines several data rates ranging from 10kbits/s to 3.4 Mbits/s, however, these data rates are not supported in all implementations of I^2C. Consequently, flow control plays an important role in both controlling data transfer and negotiating the bus clock-rate. Flow-control in embedded aggregates is achieved by using a combination of I^2C clock-stretching, transmission buffers in the sender and receiver, and the continuous read/write field 'C' in the header.

Clock-stretching, an inbuilt feature in I^2C, extends a closed-loop flow control. Aggregates generally use data rates between the 100 kbit/s standard mode and 400 kbit/s fast mode, with real-time speed-up and slow-down. In our implementation, the majority of the transmissions are at around 350 kbits/s. However, in cases where clock-stretching is infeasible, the communication defaults to 100 kbits/s. On the buffer front, modules generally implement a 64 byte circular buffer, into which an incoming packet is placed. Packets are predominantly manipulated and assimilated in-situ, however, in certain cases, alternative secondary buffers are used based on available storage space.

Finally, the 'C' field in the header provides a means for the master or the slave to negotiate another packet transfer in the same polling cycle. Polling leads to a discretization and windowing of time, during which the communication between any two modules is carried out. The frequency of polling is expected to be in the order of a few tens of milliseconds, consistent with the performance and timing requirements of the embedded aggregate. While in the basic case each module is polled by its master once every polling cycle for an out-bound packet, modules may have more than one packet to send in a given polling cycle. In such cases setting the 'C' field to 1 requests another transfer. When set by a master, it notifies the slave of another available packet that will be delivered after the guard-band. On the other hand, the slave may set the same bit on an outgoing packet to request another pull from the master immediately after a guard-band.

Together, these three flow control mechanisms provide substantial flexibility in implementing embedded aggregates involving a heterogeneous set of modules. In the following sections, we will develop the mechanisms toward realizing a embedded aggregate from the functional modules.

4.8 Heuristics and Algorithms for Module Discovery and Initialization

An embedded aggregate is envisioned to be open-ended like an ad hoc wireless network. While a wireless network between a set of modules can be created in an ad hoc fashion, wired networks, and embedded aggregates as consequence of using them, require a data-path skeleton to grow and evolve. An embedded aggregate is grown with the root communicator as the seed and shaped using the modules that bridge two or more I^2C communication buses. The communication hierarchies thus created provide the basic framework into which other modules can be introduced toward realizing an aggregate. We have developed algorithms toward organizing the modules and creating the final aggregate using the framework, and they are discussed below.

The module initialization process involves identifying a new module, obtaining information

about it and assigning it a protocol address. In the aggregates, the initializations may begin with the root-communicator discovering the modules in level 2 of the hierarchy. The underlying I^2C protocol does not provide means to perform such an initialization, so we have developed initialization techniques. Each module in the embedded aggregates is distinguished using a unique identification number (ID), which plays an important role in the module initialization process. In our implementations, the modules use a Dallas Semiconductor 1-wire Digital Serial Number (DSN) IC (DS2401) [Sem] that provides a 7-byte unique ID. We will now discuss our module discovery and address assignment process.

4.8.1 New Module Discovery

The primary goal of the module discovery process is to obtain the unique IDs of the non-initialized modules. A module communicates on the I^2C bus either in the master or the slave mode. The I^2C specification does not require the I^2C implementation in a processor to be both master and slave capable. In fact, a majority of the processors function only in the slave mode, while others can operate both in the master and slave mode. In order to support aggregates with a wide variety of processors, our design allows both kinds of implementations. We have hence developed different mechanisms for discovering the master-capable and slave-only modules, and they are described below. All the techniques require a module to be slave capable. The new module discovery step requires a co-ordinated effort between the uninitialized modules and module enlisted with the responsibility of discovering them - the discoverer. Both the discoverer and the uninitialized modules have to be connected to the same physical I^2C bus.

Towards discovering new modules, and subsequently assigning addresses to them dynamically, we researched existing algorithms and implementations. As noted earlier, the I^2C specification does not outline any methods, whereas the SMBus outlines an address resolution protocol (SMBus-ARP) [For03]. The SMBus specification is derived from I^2C and defines master and slave modes,

voltage levels and timings more strictly. The SMBus-ARP hinges upon the modules being able to perform collision sensing and back-off while transmitting in slave mode. For the ARP to work, the modules are required to abort transmission when they lose the bus arbitration. However, this requirement is not enforced by the I^2C protocol, and therefore, is not implemented in most chips, making the SMBus-ARP not viable in the aggregates context.

Due to the lack of mandatory collision sensing and back-off, a feasible approach, in order to facilitate the communication of the uninitialized module details to the discoverer, has to restrict the use of the I^2C bus to one module at a time. An alternative way would require means for the discoverer to obtain module ID and initialization parameters through alternative communication channels. We have studied and implemented four different mechanisms, namely, slotted approach with participation in master mode, slotted approach with participation in slave mode, bus access mediation through power control and module ID retrieval through additional 1-wire infrastructure [AS97].

The two slotted approaches are less efficient than the algorithm followed by SMBus, but support a broader set of processors. The first slotted approach is applicable only for master-capable modules, whereas the next slotted approach supports all modules. The power control method accomplishes bus exclusivity by restricting the number of uninitialized modules on the bus to one, by selectively turning the modules on, one at a time. The discoverer can power on the modules successively and obtain the IDs by reading from address $0x76$. The final ID retrieval method involves the discoverer reading the IDs provided by the DS2401 chips on the modules through a separate 1-wire communication network. Both the power control and ID retrieval methods demand additional wiring and power control infrastructure, but remove timing challenges brought about by the slotted approaches.

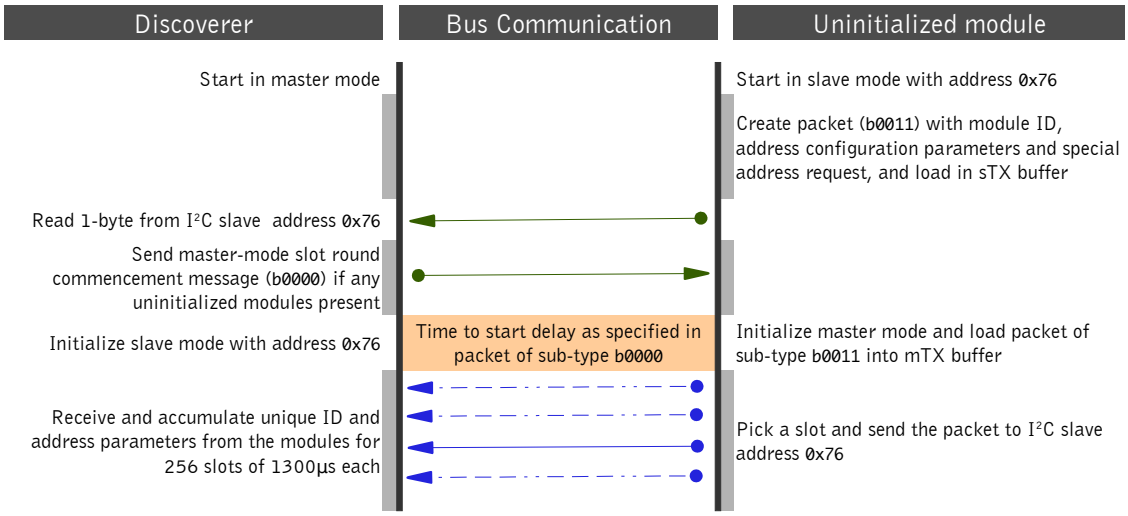


FIGURE 4.15: Slotted approach with participation in master mode. The illustration shows the process of module discovery using a slotted round process where the uninitialized modules participate in I^2C master mode. The green arrows show the direction of packet transfer as invoked by the discoverer in the master mode and the blue arrows show the direction of packet transfer as invoked by the uninitialized modules in the master mode. The broken arrows illustrate several such transfers that the discoverer receives from other uninitialized modules.

A. Slotted Approach with Participation in Master Mode

This slotted technique works in the absence of collision sensing but requires modules to switch between the master and slave modes during the process. Figure 4.15 illustrates the discovery process. As an initial step, the newly connected uninitialized modules (Algorithm 4) join the bus in I^2C slave mode with I^2C address $0x76$ (Figure 4.8) and load their slave transmit buffers (sTX) with a message of type $0x01$ (Figure 4.7). The sTX is the buffer from which data is placed on the I^2C bus when requested by a master. We will use the slave receive buffer (sRX) notation to denote the complementary buffer, into which data received from the master is placed. In the master mode, the buffers will be called mTX and mRX, respectively. As illustrated in Figure 4.16, the uninitialized module creates a packet with a maximum length of 12 bytes. At this stage, several modules may be connected to the bus with the same address, $0x76$, and an I^2C packet sent to this address will be received by all the modules simultaneously. The initialization packet created by a module has three fields in its data segment. The first field is 1-byte long and carries the

FIGURE 4.16: Packet of sub-type b0011. This packet is created by a module interested in getting initialized. The packet contains the 7 byte ID of the module, which is further broken into a 6 - byte ID segment and a 1-byte family type. The module can request for a preferred address to be allocated using the optional (marked green) "req. add" field. The parameter S denotes address type: 0 - if only static address is possible & 1 - if both static and dynamic address are possible; parameter R denotes address memory: 1 - if it can remember address after brownout reset & 0 - otherwise; parameter QQ denotes additional addresses being requested for other modules limited to a maximum of four. The length of the packet is dependent on the presence of the optional "req. add" field.

Algorithm 4: Uninitialized module participation algorithm for slotted discovery in I^2C master mode

Initialize *slave mode* with address $0x76$;
Read self-DSN;
$sTX \leftarrow$ Packet(type=$b0001$, sub-type=$b0011$); `// Load packet into slave transmit buffer`
while *uninitialized* **do** `// Until initialized with an address, stay active on the bus.`
    if *Packet (type=*$b0001$*, sub-type=*$b0000$*) is received* **then** `// Discoverer calling the start of the slots`
        Initialize *master mode* ;
        $mTX \leftarrow$ Packet(type=$b0001$, sub-type=$b0011$) ;
        Set *mySlot* = slot_selector(); `// Select a slot using the slot selection algorithm`
        Sleep until start of slots; `// Wait till start of the slot rounds`
        Sleep until *mySlot* $\times$ 1300 μs; `// Sleep till chosen slot`
        Transmit Packet(type=$b0001$, sub-type=$b0011$) to I^2C address $0x76$;
        Sleep until end of slots;
        Initialize *slave mode* with address $0x76$;
    `...// other conditions follow`

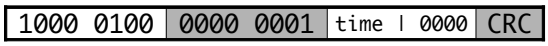


FIGURE 4.17: Packet of sub-type b0000. This packet is sent by the discoverer toward calling uninitialized modules for participation in slotted discovery in master mode. The packet contains a 4-bit time field that is used to specify an initial setup period following which the first slot starts. The setup period is provided to allow the modules to make the I^2C mode transitions and populate their buffers.

address configuration data. The lower nibble of this field (Figure 4.16), $b0011$, specifies the subtype of the current message type $0x01$. The upper nibble specifies three parameters; namely, address mode, address memory and address space request. The address space request parameter allows a module to request allocation of up to four additional addresses at the same protocol address level as itself. The four additional addresses have the same protocol address length and ancestors as the module. A module having obtained a collection of addresses may disconnect a part of the bus and perform initialization of other modules in isolation, which eventually become its siblings. Several simultaneous and parallel initializations speed up the aggregate formation process. Seven bytes of unique ID follow the address configuration data and are obtained from the DS2401 chip.

Algorithm 5: Discoverer algorithm for slotted rounds with uninitialized modules in I^2C master mode

```
1 Initialize Master mode;
2 Set uninitModPres = Success of 1-byte Read from address 0x76;
3 if uninitModPres is True then // Uninitialized modules available for discovery
4     mTX ← Packet(type=b0001, sub-type=b0000); // master mode slotted discovery
      call
      Transmit Packet(type=b0001, sub-type=b0000) to I2C address 0x76;
5     Initialize slave mode with address 0x76;
6     Sleep until start of slots; // Wait till start of the slot rounds
      for 256 slots of 1300 µs each do // Listen on I2C bus for transmissions from
      uninitialized modules in master mode
7         Accumulate packets with ID and address configuration information;
8     Initialize Master mode;
```

The 1-byte requested address following the module ID is optional and is incorporated for permitting modules to request a specific 7-bit bus-level I^2C address. We envision this 7-bit address request to be used for a couple of reasons. First, we have allocated eight addresses (Figure 4.8) for

the future, including for the purpose of accommodating modules with unmodifiable slave address. We call them static addresses. These modules can participate in the module initialization process and may be allowed to operate in the system in the absence of any address collisions. It is important to note that the scope of the address and address collision, if any, is limited to the particular level of hierarchy and specific I^2C bus. The modules with static addresses are required to have the ability of responding to I^2C global calls. Second, this provides a means for quick startup and initialization of a previously created and functional aggregate, by reducing the time for reallocation of new addresses, and updates to individual module addresses and local address tables.

Simultaneously, the discoverer begins the process with a scan for new modules as described in Algorithm 5. The module performs the scan in the master mode and it involves reading one byte of I^2C data from slave address $0x76$. The discoverer module, at this time, does not read a full packet of data, but only a single byte. The presence of uninitialized modules is signified to it by an I^2C bus level acknowledgement signal immediately following the transmission of the slave address of $0x76$ and a 1-byte data value other than $0xff$. If uninitialized modules are present, the discoverer calls for the slotted discovery step by sending the round call message specified in Figure 4.17. This message is four bytes long with a 1-byte data segment.

The initialization process is performed at an I^2C bus speed of 100kbps and the use of optional headers is disallowed. The 1-byte address configuration data specifies in the upper nibble the delay in the start of the time slots in milliseconds from the end of transmission of the current message, followed by the message sub type $b0000$ in the lower nibble as shown in Figure 4.17. The delay is restricted to a maximum of 16 ms. On receipt of the message, the uninitialized modules populate their mTX buffer with the ID information loaded earlier into their sTX buffer and switch from slave mode to master mode. In parallel, the discoverer switches from master mode to slave mode with the address $0x76$.

If collision sensing in multi-master mode is mandated, then a simple mechanism for module discovery can be employed. After the specified start delay, the algorithm can require all the uninitialized modules to begin the transfer of their ID packets. With the modules losing the ensuing bus arbitration required to attempt re-transfer, the approach discovers all the uninitialized modules in time linear in the number of modules. The determinism stems from the fact that one module wins each arbitrated transfer attempt. However, the I^2C specification does not strictly require master capable modules to be multi-master capable, so this approach is not always feasible. Instead, we employ a multiple time-slot approach in order to relax the requirement of collision sensing in multi-master I^2C bus transmission.

The slotted round incorporates 256 non-overlapping time-slots of 1300μs each, and at a 100kbps data rate, the slot duration provides a minimum guard band of 200μs between two successive slots. With their I^2C modes reversed, the uninitialized modules participate in the discovery round by transmitting the above ID packet (Figure 4.15) in the master mode in one of the 256 slots. To select the slot, the modules use a slot selection algorithm as described in Algorithm 6. With the W bit set to 0, the packet is addressed to slave address $0x76$ (which is the first byte of the I^2C frame) by the uninitialized modules and is received by the discoverer.

At the end of the 256 slots, the modules again reverse their I^2C modes and at this time, the discoverer has collected as many as 256 uninitialized module IDs and address requests. In the event of multiple modules transmitting in the same slot, the packet received by the discoverer is a binary-and superposition of the individual packets due to the electrical characteristics of the I^2C bus. The packet thus received is bound to have an incorrect CRC, and is consequently discarded by the discoverer.

Slot Selection

The slotted mode is not collision free. Two or more modules picking the same slot, in the absence of collision sensing, will lead to incorrect packets being transmitted to the discoverer. However, the approach mitigates this challenge by providing modules with one of several slots to choose from. In the event of a collision, the modules will remain non-initialized at the end of the current initialization process and continue to participate in a future processes until initialized. The slot-selection algorithm attempts to diversify the slot picked by a module so that it eventually finds a collision-free slot. A pseudorandom number generator (PRNG) would suffice for slot selection. Unfortunately, when implemented in aggregate class of modules, the PRNGs are neither economical nor sufficiently diversified in their outputs. The similarity of the random numbers, and consequently, the slot numbers is prevalent in modules powered on simultaneously, perhaps a side-effect of using an internal counter whose value is a function of time since booting. This leads to modules consistently choosing the same slot in successive initialization processes and colliding with one another.

Algorithm 6: Slot selection algorithm

```
// slotAttempt and silentSlots initially set to 0 at module boot up
1 if silentSlots is equal to 0 then
2     if slotAttempt < 0x06 then // participating discovery rounds
3         Set chooseSlot = slotAttempt byte of ID; // select the particular byte of
          the 7-byte ID as the slot number
          Set slotAttempt = slotAttempt + 1;
4     else
5         Set silentSlots = random number between 1 and 4;// choose a random number of
          rounds for not participating
          Set slotAttempt = 0;
6 else
      // nonparticipating discovery rounds
      Set silentSlots = silentSlots - 1;
```

In order to overcome the above issue, we have devised a slot selection algorithm (Algorithm 6) that utilizes the unique ID of a module, one byte at a time (steps 2 - 4), in the successive initialization

processes. We also designate a random number, between one and four, of initialization processes after every six unsuccessful attempts as non-participating processes (steps 5 - 9). This allows to break from lack of slot selection diversity between two modules. The modules may be picking the same slot in successive processes due to their IDs being self-similar or circularly-shifted.

B. Slotted Approach with Participation in Slave Mode

A direct extension of the SMBus-ARP with time division multiple access to the bus, the slotted process in slave mode can in fact suffice for discovering all modules. This approach has all steps similar to the algorithm described for the master-capable modules earlier except for modifications to the slot rounds. Unfortunately, this method is more demanding than the master mode slot rounds for a few reasons. To understand the challenges faced in the above approach, we outline the slave mode time-slot process below.

Algorithm 7: Discoverer algorithm for slotted rounds with uninitialized modules in I^2C slave mode

```
1 Initialize Master mode;
2 Set uninitModPres = Success of 1-byte Read from address 0x76;
3 if uninitModPres is True then // Uninitialized modules available for discovery
4     mTX ← Packet(type=b0001, sub-type=b1101); // slave mode slotted discovery
      call
      Transmit Packet(type=b0001, sub-type=b1101) to I2C address 0x76;
5     Sleep until start of slots; // Wait till start of the slot rounds
      foreach of the 256 slots of 1300 µs do // Read and accumulate packets from
      unassigned modules
6         Read a packet on I2C bus from slave address 0x76;
```

To begin, instead of the modules picking a slot and transmitting in the master mode, the modules respond on the I^2C bus on receiving a packet of sub-type $b1101$ (Figure 4.19), during one selected slot and remain disconnected from the bus otherwise as shown in Figure 4.18. The slave modules can disconnect from the bus either by going to a sleep state or by selectively disabling their I^2C functionality. In this process, the discoverer attempts to read one packet from the I^2C address $0x76$ for every time slot, completing 256 reads at the end of the slotted process. Collisions on the bus

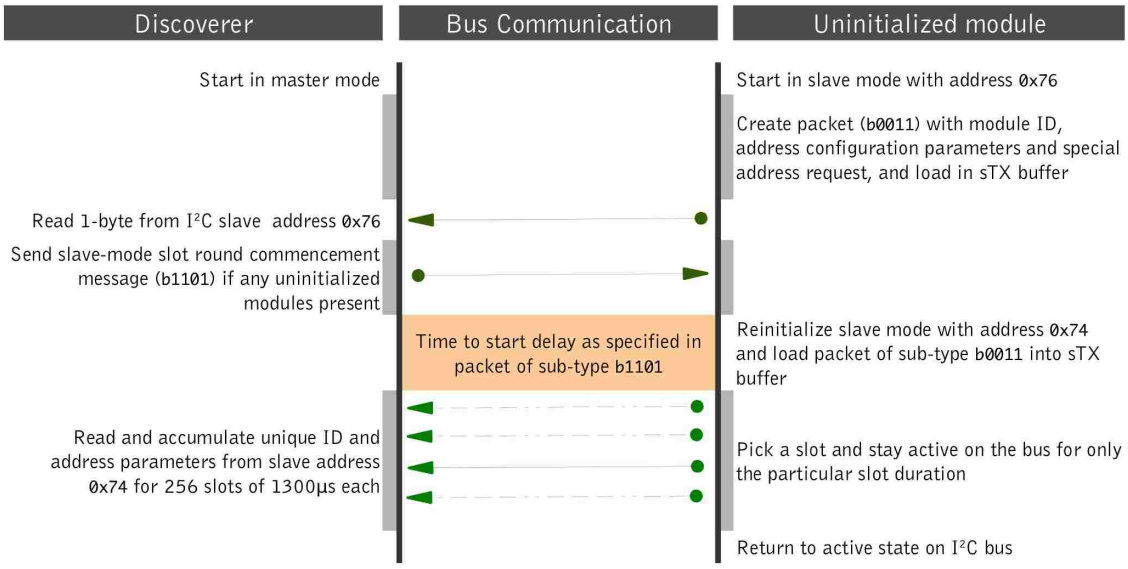


FIGURE 4.18: Slotted approach with participation in slave mode. The illustration shows the process of module discovery using a slotted round process where the uninitialized modules participate in I^{2}C slave mode.

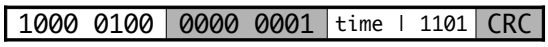


FIGURE 4.19: Packet of sub-type b1101. This packet is sent by the discoverer toward calling uninitialized modules for participation in slotted discovery in slave mode. The packet contains a 4-bit time field that is used to specify an initial setup period following which the first slot starts. The setup period is provided to allow the modules to populate their buffers, reinitialize slave mode with address 0x74 and initialize their bus agnostic mode corresponding to the chosen slot.

can occur when two modules select the same slot, and similar to the master mode case described earlier, the damaged packets received by the discoverer are discarded.

The success of the above approach hinges on modules remaining disconnected from the bus during all slots other than their chosen time slot. One of the issues arises from modules that power on during this process and connect to the bus in their slave mode with I^{2}C address 0x76. Consequently, all the modules transmit on the bus when the discoverer reads from address 0x76, leading to collisions during all subsequent slots. This issue may be overcome with a slight modification to the protocol that requires the modules to switch to a different predefined restricted slave address, 0x74, when they participate in the slot rounds.

Algorithm 8: Uninitialized module participation algorithm for slotted discovery in I^2C slave mode

```
1  Initialize slave mode with address 0x76;
2  Read self-DSN;
3  sTX ← Packet(type=b0001, sub-type=b0011); // Load packet into slave transmit
   buffer
   while uninitialized do // Until initialized with an address, stay active on the
   bus.
4      if Packet (type=b0001, sub-type=b1101) is received then // Discoverer calling the
       start of the slots
5          Initialize slave mode with address 0x74; // Reinitialize slave mode with
           alternate slave address
           Set mySlot = slot_selector(); // Select a slot using the slot selection
           algorithm
           Deactivate I2C communication;
6          Sleep until start of slots; // Wait till start of the slot rounds
           Sleep until mySlot × 1300 µs; // Sleep till chosen slot
           Activate I2C communication;
7          while Packet(type=b0001, sub-type=b0011) transfer not completed do // wait
           until one packet transfered to discoverer
               // nothing to do, discoverer is reading a packet
8          Initialize slave mode with address 0x76;
9      ...// other conditions follow
```

Our practical implementations of this approach have been challenging primarily due to the complexities of precise timing of the 256 reads by the discoverer. The difficulties arise due to challenges in allocating precise timer modules for this purpose. Simultaneously, several off-the-shelf processors that implement the basic I^2C slave mode often have one or two precision timer modules. Allocating one of these timers for the purpose of communication is tangential to the main goals and requirements of the resource constrained modules. We are currently investigating the feasibility of this approach by modifying the duration and number of the slots.

C. Bus Access Mediation through Power Control

The power control based approach is one of the simplest module discovery procedures. This technique begins with all the modules turned off by the discoverer as showing in Figure 4.20. The discoverer now turns one module on and, after a guard band of 1s, reads a packet of data from the slave address $0x76$. Simultaneously, the uninitialized module on powering up creates and loads its sTX buffer with the ID packet shown in Figure 4.16. Hence, when the discoverer attempts a

Algorithm 9: Discoverer algorithm in power control approach

```
1 Initialize Master mode;
2 Power on a uninitialized module;
3 Wait for 1 second for the module to stabilize; // do other jobs
  Read a packet on I2C bus from slave address 0x76;
```

Algorithm 10: Uninitialized module participation algorithm for discovery using selective power on approach

```
1 Initialize slave mode with address 0x76;
2 Read self-DSN;
3 sTX ← Packet(type=b0001, sub-type=b0011); // Load packet into slave transmit
  buffer
  while uninitialized do // Until initialized with an address, stay active on the
  bus.
  |  // nothing to do, discoverer will read a packet
4 |  ...// address assignment and verification conditions follow
```

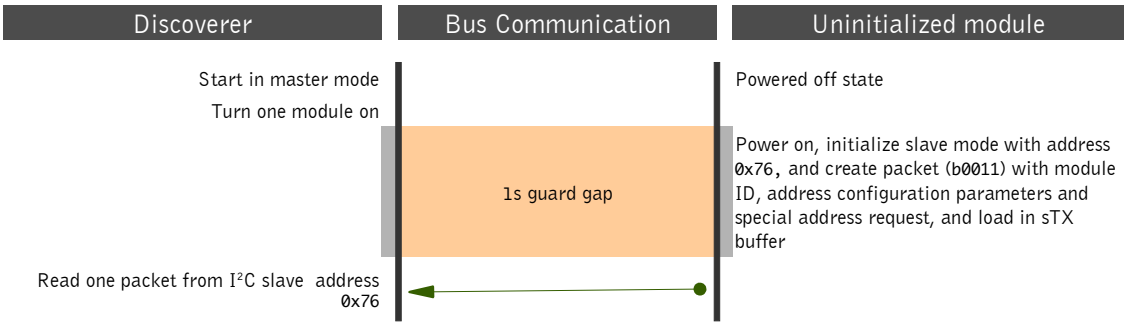


FIGURE 4.20: Uninitialized module discovery through selective power on. The illustration describes the process of gathering ID and address configuration information from an uninitialized modules by creating I^2C bus exclusivity by selective power on.

1000 1011	0000 0001	time | 1100	ID1	ID2	ID3	ID4	ID5	ID6	ID7	CRC

FIGURE 4.21: Packet of sub-type b1100. This packet is sent by the discoverer toward singling out a particular uninitialized module and requesting participation in a discovery process. The discoverer uses the ID of the module that it has acquired through other means toward this call. The 4-bit time parameter denotes the setup time guaranteed before the 2ms packet reading window.

read, the uninitialized module transmits the above packet without any collisions on the bus and is successfully received by the discoverer.

D. Module ID Retrieval through Additional 1-wire Infrastructure

In the 1-wire infrastructure based technique as shown in Figure 4.22, the discoverer uses an additional 1-wire infrastructure to read the 1-wire DS2401 DSNs. This technique begins with the discoverer collecting the IDs of all the uninitialized modules. The modules may be powered on or off at this time as the ID retrieval happens without any mediation by the on-board processor of the module [AS97], however, it is important to prevent both the module and the discoverer from reading the ID simultaneously. Hence, we stipulate a one second guard gap from the time of module power up when reading by the discoverer is prohibited. Following the guard gap, the discoverer can obtain the unique ID of the module, however, the additional configuration data like static address requirements, request for a particular address etc. of the module are unknown. Towards gathering these details, the discoverer now sends a selective-occlusion packet of sub-type $b1100$ as illustrated

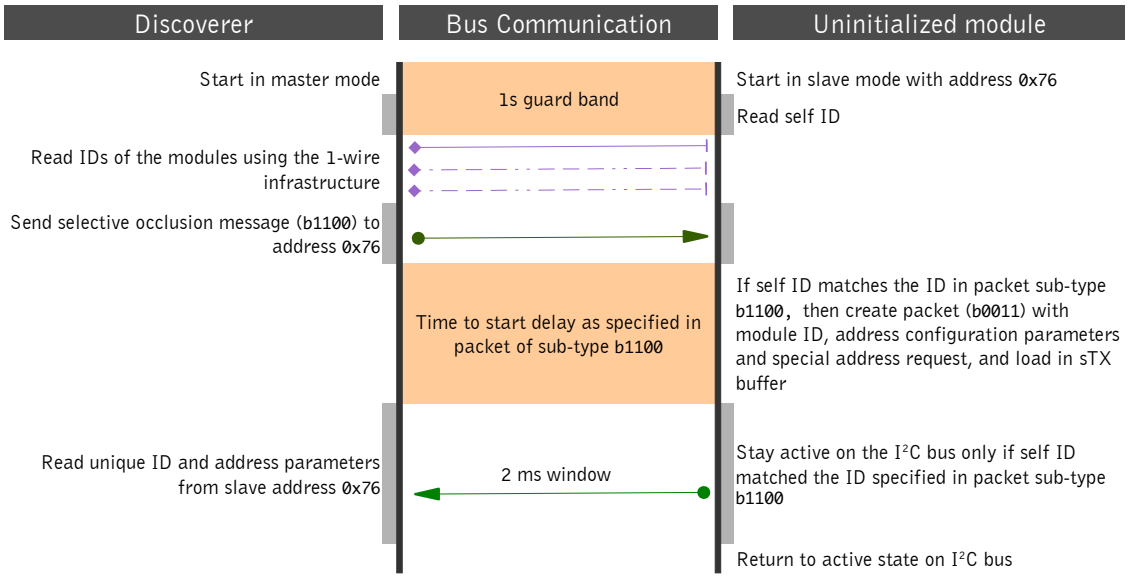


FIGURE 4.22: Uninitialized module discovery with additional 1-wire infrastructure. The illustration describes the process of gathering ID and address configuration information from an uninitialized module with support from the 1-wire infrastructure. The reading of the DSN on the 1-wire bus is shown distinctly using the violet arrow.

in Figure 4.21.

The selective-occlusion packet carries the unique ID of the module, in whose additional configuration information the discoverer is interested. Additionally, it carries a byte of configuration parameters. The lower nibble of this configuration byte carries the sub-type $b1100$, and the upper nibble, time in milliseconds. The time parameter specifies the guard gap between completion of the transmission and a read attempt by the discoverer from slave address $0x76$. This packet is received by all the uninitialized modules and interpreted differently based on ID match.

If the ID of the module matches the address in the packet, the module loads the ID and address request packet in its sTX buffer immediately. All other modules disconnect from the bus for a period of 2 ms at the end of the guard gap. This ensures that when the discoverer reads during the 2 ms window, only the module with the matching ID responds on the I^2C bus. It has to be noted that collisions, if any, are detected due to packet CRC error and the discovery may be attempted

Algorithm 11: Discoverer algorithm in the approach with support from 1-wire infrastructure

```
1 Initialize Master mode;
2 Read IDs of the modules using 1-wire infrastructure;
  foreach Uninitialized module ID do
3     mTX ← Packet(type=b0001, sub-type=b1100) with module ID;
      Transmit Packet(type=b0001, sub-type=b1100) to I2C address 0x76; // Notify
      modules of forthcoming read and targeted ID
      Wait for duration specified in the packet; // do other jobs while the
      uninitialized modules setup their buffers
      Read a packet on I2C bus from slave address 0x76;
```

Algorithm 12: Uninitialized module participation algorithm for discovery with support from additional 1-wire infrastructure

```
1 Initialize slave mode with address 0x76;
2 Read self-DSN;
3 sTX ← Packet(type=b0001, sub-type=b0011); // Load packet into slave transmit
  buffer
  while uninitialized do // Until initialized with an address, stay active on the
  bus.
4     if Packet (type=b0001, sub-type=b1100) is received then // Discoverer calling the
      start of the slots
5         if ID in Packet (type=b0001, sub-type=b1100) not equal to self-DSN then // Check
          for ID match
6             Deactivate I2C communication;
7             Sleep until start of window + 2ms; // Wait till completion of the window
              Activate I2C communication;
8     ...// other conditions follow
```

FIGURE 4.23: Packet of sub-type b0010. This packet is sent by the discoverer toward assigning protocol addresses to an uninitialized module. The parameter S denotes address type: 0 - if the static only constraint cannot be accepted & 1 - if static address is accepted; parameter T denotes address memory: 1 - if address can be remembered and the module can reuse the address on reset & 0 - otherwise; parameter QQ denotes number of additional addresses as requested by the module that are accepted and allocated. The addresses following the ID7 byte of the module are in the multi-byte (represented in blue) protocol address format. The address allocated for the module is the first address in this sequence of addresses. Five such addresses may be carried in a packet and three additional addresses are illustrated by the contracted boxes.

again. The guard gap in the process is anywhere between 1 to 16 ms. The length of the guard gap is to be set by the discoverer appropriately based on its CPU load and the speed of the uninitialized module.

At the end of module discovery step of the initialization process, the discoverer has collected the unique IDs of several uninitialized modules. We see a variety of uses for these IDs including, for example, authenticating module for use in an aggregate as a handle for the module meta-data, and perhaps, most importantly, as a handle for the look up table (LUT) containing module protocol addresses. The IDs are also used by other modules interested in seeking a particular module's protocol address. The discoverer now assigns the modules addresses based on availability from the pool of assignable addresses as follows.

4.8.2 New Module Address Assignment

The address assignment step of the module initialization process starts with the discoverer in the master mode and the uninitialized modules in the slave mode with the I^2C address $0x76$. The discoverer after allocating the addresses as requested by the module in the earlier step, assigns the address using the packet in Figure 4.23. This packet is sent to the address $0x76$ after a minimum guard band of 10ms following the 256 timed slots.

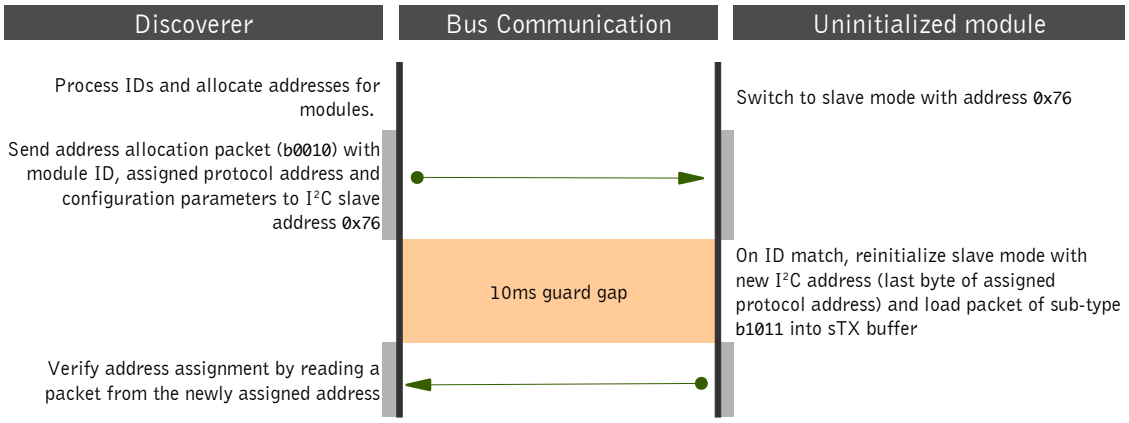


FIGURE 4.24: New module address-assignment. The address assignment process starts with the uninitialized module in slave mode and the discoverer in master mode. The process involves two communications on the I^2C bus, one to communicate the address information to the module and the second to verify the proper completion of the process.

As shown in the illustration, the 7-byte ID of the module is followed by a few addresses in the protocol address format. The first address in this sequence is the address of the module with the ID. The additional addresses that may be allocated based on the previous request follow this address. The number of additional addresses is less than or equal to the request made by the uninitialized module earlier.

The 1-byte address configuration field preceding the ID carries the packet sub-type $b0010$ and three parameters signifying the number of additional addresses being provided, acceptance of static address request and validity of the address into next power-up of the module. However, in the event that an address is unavailable, the module is sent the reserved address $0x76$ requesting participation in another round, or sent the address $0x00$ indicating the lack of any addresses and denying the module any address allocation in the future. In the case of modules requesting static addresses, the static address is not repeated in this message and the additional addresses follow the ID fields.

The address-assignment packet is received by all uninitialized modules and the modules set their slave addresses as specified in the packet if their ID matches the one in the packet. Following the

FIGURE 4.25: Packet of sub-type b1011. The packet created by a module on accepting the assigned slave address. The packet is similar to one in Figure 4.23, except for the sub-type field and the CRC.

address setting, the module places an address assignment confirmation message of sub-type $b1011$ as shown in Figure 4.25 in its sTX buffer. The assignment confirmation packet is the same as the received assignment packet, with the sub-type field and, consequently, packet CRC modified. In order to verify the correct assignment of the slave address, after a minimum guard gap of 10 ms, the discoverer reads a packet of message from the newly assigned protocol address. In the case of errors in assignment, the assigned address may be revoked by sending another address assignment message of sub-type $b0010$ with the protocol address $0x76$.

Algorithm 13: Discoverer address assignment step

```
for Every discovered unique ID do
    if Address allocation for ID successful then // Assigning address
        mTX ← Packet(type=b0001, sub-type=b0010) with allocated protocol addresses;
        Transmit Packet(type=b0001, sub-type=b0010) to I²C address 0x76;
        Sleep for 10ms;
        Read into a packet from the newly assigned I²C address VerifyPacket;
        if ID, Address configuration and Protocol Addresses of sent and verify packets are
        unequal then // Verify assignment process
            mTX ← Packet(type=b0001, sub-type=b0010) with address 0x76;
            Transmit Packet(type=b0001, sub-type=b0010) to newly assigned I²C address;
            // Revoking address assignment
    else if No address assignment possible then
        mTX ← Packet(type=b0001, sub-type=b0010) with address 0x00;
        Transmit Packet(type=b0001, sub-type=b0010) to I²C address 0x76; // Notify
        module of no possibility of present or future address assignment
    else
        // Do not sent module any message, module will participate in next
        discovery process
```

Following the successful assignment of an address, a module is considered to be initialized and

Algorithm 14: Uninitialized module address assignment step

```
1  while uninitialized do // Until initialized with an address, stay active on the
   bus.
2      ...// other conditions
3      if Packet (type=b0001, sub-type=b0010) is received then // Discoverer assigning
       address
4          if ID in Packet (type=b0001, sub-type=b0010) equal to self-DSN then // Check for
           ID match
5              Set selfAddress = LSB(First protocolAddress); // Assigned I2C address is
               the least significant byte of the first multi-byte protocol
               address in the packet
               if selfAddress is equal to 0x00 then
6                  Halt;// Call shutdown procedure as module cannot be
                   initialized
7              else
8                  Initialize slave mode with address selfAddress;
9                  sTX ← Packet(type=b0001, sub-type=b1011); // Load packet into
                   slave transmit buffer
                   while Packet(type=b0001, sub-type=b1011) transfer not completed do
                   // wait until one packet transfered to discoverer
                       // nothing to do, discoverer is reading a packet
```

is free to communicate with other modules. A module that requested multiple addresses earlier may, at this time, isolate a section of the bus and proceed with the discovery process if any of the addresses were allocated. Similarly, modules that form bridges between hierarchies proceed toward module initialization in the lower hierarchy. We have implemented two kinds of packets (not described here) in this regard, one toward probing and obtaining ID related information of a module from authenticating modules, and the second toward conveying the protocol address allocation to other modules.

This section concludes the descriptions and discussions on the important components of the aggregate framework that were designed and developed in this thesis work. All our earlier discussions leave unexamined the effects of the choices made in the aggregate framework on the derived systems. Systems built using the framework will inherit several of the aggregate characteristics and may have inherent features due to additional choices made by the designer. While it is beyond the scope of this research to develop an understanding of the descending systems, we develop equations in the following evaluation section that show the interplay between the various parameterized intrinsic properties of the aggregate's components.

4.9 Evaluation

In this section we present an analytical treatment of the embedded aggregates framework. The arbitrary message packet sizes and the lack of job scheduling specifications provide substantial freedom to aggregate implementers, fostering the growth of an open ended and flexible system. However, the numerous conceivable resultant architectures and implementations make the evaluation of the aggregate framework tedious.

In order to present to the reader an understanding of the dynamics of the aggregate's communication and functioning, in this assignment we undertake a study of the various factors

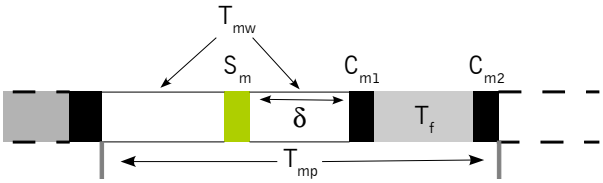


FIGURE 4.26: Function module single data send timeline. The data to be sent is loaded into the buffer in time slot S_m and retrieved by bus master during the polling slot for the module. The transmission time for the data is T_f long and during this period the module is unable to perform other operations. A function module is able to perform functions related to its core responsibilities during δ period.

that influence system performance and the interplay between them. We initially derive the ideal case communication costs for a generic aggregate function module and a generic communication module. Next, we use these costs toward understanding the packet transfer times for a couple of source-destination pairs. Finally, we derive a relationship that establishes the maximum number of modules that can be accommodated on a bus for a simple envisioned communication pattern.

4.9.1 Module Communication Cost

In the packet communication cost analyses below, we assume that every communication attempt is successful and do not account for retransmissions.

4.9.1.1 Generic Function Module

A function module distributes its time between performing tasks related to its core functionality and communication-specific tasks. Let us consider a simple situation where the module sends one message in a polling period as shown in Figure 4.26.

Let

$$S_m = \textit{time for loading data into buffer}$$

$$C_{m1} = \textit{time for context switching from core funtions to communications}$$

$$C_{m2} = \textit{time for context switching from communications to core funtions}$$

$$T_f = \textit{I}^2\textit{C frame transfer time}$$

$$T_{mw} = \textit{time available for core activity}$$

$$T_{mp} = \textit{time between two polling slots for the module}$$

$$T_{mc} = \textit{time spent on communication}$$

$$\delta = \textit{time from loading the data into buffer to frame transfer}$$

$$m = \textit{size of the useful data in the message}$$

$$l = \textit{length of destination address in protocol address format}$$

$$e = \textit{length of the extended header segment}$$

$$d = \textit{total length of the } I^2C \textit{ frame}$$

$$D = \textit{data rate of the bus in bits/sec}$$

Following the availability of the data intended for sending, the data is setup in the buffer in the interval shown in green. Due to the polling nature of data transfer to its master, a module can experience a delay between the time the data is ready in its buffer and the time it is retrieved by the master. This delay is denoted by δ and the module is able to perform its core functionality in this waiting period. The delay δ is bounded by T_{mp} and can be minimized by synchronizing the module's activities to its polling slot. The period T_{mp} can be unique to modules as they may require different durations depending on their core activity and use case. The packet is fetched by its parent in the T_f time period flanked by the context switching windows on either side. When pulls and pushes are gathered and collocated, the time period T_f has to account for data sent and received. Otherwise, multiple data exchange intervals flanked by context switching times will occur.

Here, the time periods S_m, C_{m1} and C_{m2} are dependent on the clock frequency of the module.

Let

$$F = clock\ frequency\ of\ the\ module,$$

then

$$C_{m1}\ \&\ C_{m2} \propto \frac{1}{F}$$
$$S_m \propto \frac{d}{F}\ .$$

In a given communication attempt, the I^2C frame transfer time is dependent on the size of the data frame and the speed of the bus. The size of a data packet depends on the size of the header, message size, destination address length, extended header segment length and trailer size. In an embedded aggregate packet, the total size of header and trailer is 3 bytes. Additionally, for each packet sent on the bus, the I^2C frame carries an additional 1-byte I^2C destination address. Therefore the length of the frame is given by

$$d = m + l + e + 3 + 1,$$

and time for frame transfer is

$$T_f = \frac{d \times 8}{D}$$
$$= \frac{(m + l + e + 4) \times 8}{D}\ .$$

We define the communication time as the sum of time spent on transferring the I^2C frame, context switching costs and data setup costs as follows.

$$T_{mc} = S_m + C_{m1} + T_f + C_{m2}$$
$$= S_m + C_{m1} + \frac{(m + l + e + 4) \times 8}{D} + C_{m2}$$

In this simplified example, only one packet of data is sent in this polling period. However, if several packets are exchanged during the polling instance, then additional time for exchange of

the subsequent frames and guard gaps are involved. In a function module, the module is free to perform activities related to its core objectives during times of no communication. Hence in a polling period, we have

$$T_{mw} = T_{mp} - T_{mc} = T_{mp} - \left(S_m + C_{m1} + \frac{(m+l+e+4)\times 8}{D} + C_{m2} \right). \tag{4.1}$$

Let us define r as the ratio of time spent on core activity to that of polling time period.

$$r = \frac{T_{mw}}{T_{mp}} = \frac{T_{mp} - \left(S_m + C_{m1} + \frac{(m+l+e+4)\times 8}{D} + C_{m2} \right)}{T_{mp}} \tag{4.2}$$

The above ratio gives the factor of time available in a polling time period for the module to pursue its core functionality.

4.9.1.2 Generic Communication Module

A communication module primarily mediates the communication between the function modules by pulling and pushing data. Additionally, the module performs other functions including discovering new modules and internal module housekeeping activities amongst others. The breakdown of the time spent on pushing a packet of data to a slave by a communication module is illustrated in Figure 4.27. The figure shows an ideal situation from a communication module's perspective. The slave module being sent data is ready for communication, and hence no additional time costs, due to context switching overheads in the slave module, are incurred in the transmission process. In the cases where such overheads exist, the gray packet transfer time undergoes elongation (see Figure 4.29) as the slave module utilizes flow-control mechanisms while context switching before the exchange of data. In this discussion, we assume a simple case where the slave module has switched its focus to communication at the time of being sent the data.

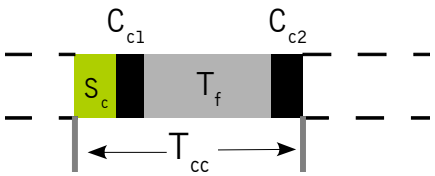


FIGURE 4.27: Communication module data pushing timeline. A push of data involves the time for buffer setup, data transfer on the bus and additional context switching times. A pull of data does not require the initial buffer setup time (marked green) in the master module.

Let

$$S_c = \textit{time for loading data into buffer}$$

$$C_{c1} = \textit{time for context switching from other activites to communications}$$

$$C_{c2} = \textit{time for context switching from communications to other activities}$$

$$T_f = I^2C \textit{ frame transfer time}$$

$$T_{cc} = \textit{time spent on communication}.$$

As in the previous case of function-specific modules, S_c, C_{c1} and C_{c2} of a communication module are dependent on the clock frequency of the module. We make a distinction in the case of these overheads (and use different notation) between the function and communication modules due to the different nature of the modules' core activities. With d as the size of the data frame, we have the following.

$$C_{c1} \ \& \ C_{c2} \propto \frac{1}{F}$$
$$S_c \propto \frac{d}{F}$$

Therefore, the time spent on pushing a frame of size d to a slave is

$$T_{cc} = S_c + C_{c1} + T_f + C_{c2} = S_c + C_{c1} + \frac{(m+l+e+4)\times 8}{D} + C_{c2}\ . \tag{4.3}$$

In the case where the data is pulled from the slave, the buffer setup cost is not incurred in the communication module and hence the time for communication is reduced by S_c.

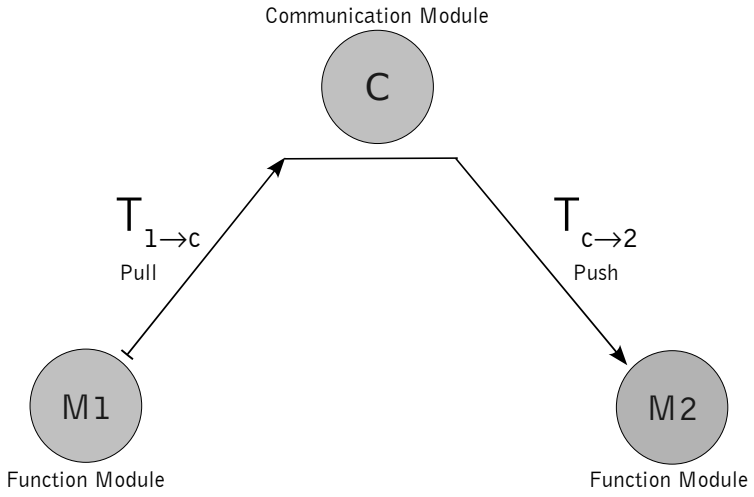


FIGURE 4.28: Packet transfer between two function modules on the same physical I^{2}C bus. The data flow for a packet transfer from module M1 to module M2 is illustrated by the black unidirectional arrows. The time taken for each leg of the path is also shown.

4.9.2 Data Transfer Time

Previously, we studied the communication time periods of the modules in isolation. Here, we estimate the time required for transferring a packet of data from one function module to another.

4.9.2.1 Communication Between Two Function Modules on the Same Physical Bus

Let us consider a packet transfer between two function modules on the same physical I^{2}C bus as shown in Figure 4.28. The function modules operate in I^{2}C slave mode on a bus and hence the communication between them is mediated by the communication module in the master mode. The time for packet transfer from one module to another is calculated from the time the packet is available in the buffer and ready for transmission at the sender to the time it is delivered to the recipient. In this example, the slave modules are not already ready for data exchange and hence a context switching penalty is incurred.

Figure 4.29 shows the causality of the events with respect to the data flow between the modules. In the figure, the times δ_1 and δ_2 denote the time gap between data availability in the buffer and the

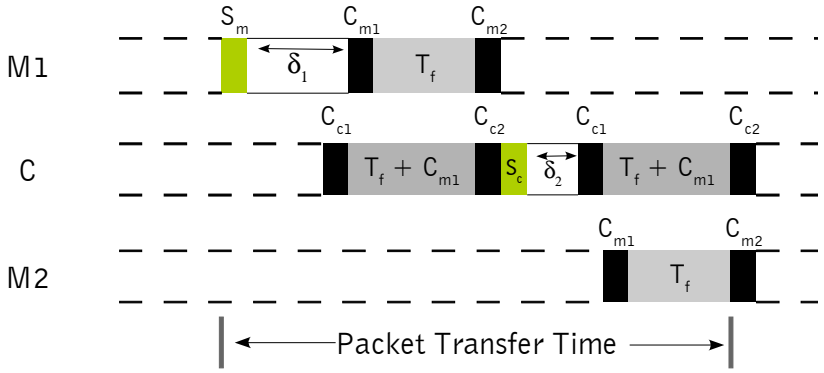


FIGURE 4.29: Timeline: Two function modules on the same physical I^{2}C bus. The figure shows a non-ideal situation where the modules are not synchronized and the function modules switch to communication mode on I^{2}C interrupt. In the time durations δ_1 and δ_2, the communication master may be communicating with other modules or performing different tasks. The modules are able to perform other tasks, including communications unrelated to the particular transfer, during the time interval denoted by the dotted lines.

occurrence of the exchange.

$$\delta_i = \textit{time from loading the data into buffer to frame transfer.}$$

For the sake of simplicity, let us assume that all modules have the same clock frequency F and the function modules have the same context switching penalties C_{m1} and C_{m2}. In the data transfer, we assume a case where the modules are performing their core tasks at the time of interruption by communication calls. The time for transfer of packet from M1 to C is then given by the following.

$$T_{1\rightarrow c} = \delta_1 + C_{m1} + T_f$$

Here, the data available for transfer at M1 has remained in the buffer for time δ_1 before the master C starts clocking the data out. The time cost C_{m1} is paid for context switching from regular core operations to communications due to single threaded nature of the module's CPU. For the second leg of transfer, based on the embedded aggregate protocol addressing and routing, the size of the packet is reduced by one byte due to modifications to the packet's address segment. However, for the sake of simplicity, we treat the packet sizes as constant, making the time estimates more

conservative. Now, for transfer from C to M2, we have

$$T_{c\to 2} = C_{c2} + S_C + \delta_2 + C_{c1} + C_{m1} + T_f \ .$$

In the second leg of the data transfer, we initially incur time penalties for context switching after the first leg, loading data into buffer and context switching for push at the communication module. Next, similar to the first leg, we also pay an additional penalty of C_{m1} at M2. Our notion of data transfer concludes with the data being received by the recipient in its buffer and hence do not consider C_{m2} at M2. Now, the total packet transfer time is give by the following.

$$T_{1\to 2} = T_{1c} + T_{c2} = S_c + C_{c1} + C_{c2} + 2 \times (C_{m1} + T_f) + \delta_1 + \delta_2 \tag{4.4}$$

In the above expression, the values for δ_1 and δ_2 are upper bounded by the polling time periods for modules M1 and M2.

4.9.2.2 Communication Between Two Function Modules with a Common Ancestor Two Levels of Hierarchy above Them

We now consider the case of communication between two modules on two different I^2C communication buses as show in Figure 4.30. With the assumptions and rationale in the previous single bus example, the times for packet transfer along the various legs are as follows.

$$T_{1\to c1} = C_{m1} + \delta_1 + T_f$$

$$T_{c1\to c} = S_c + \delta_{c1} + C_{c1} + T_f$$

$$T_{c\to c2} = C_{c2} + S_c + C_{c1} + \delta_{c2} + C_{c1} + T_f$$

$$T_{c2\to 2} = C_{c2} + S_c + C_{c1} + \delta_2 + C_{m2} + T_f$$

The total time for packet transfer is merely a summation of the above time periods as given below.

$$T_{1\to 2} = T_{1\to c1} + T_{c1\to c} + T_{c\to c2} + T_{c2\to 2}$$

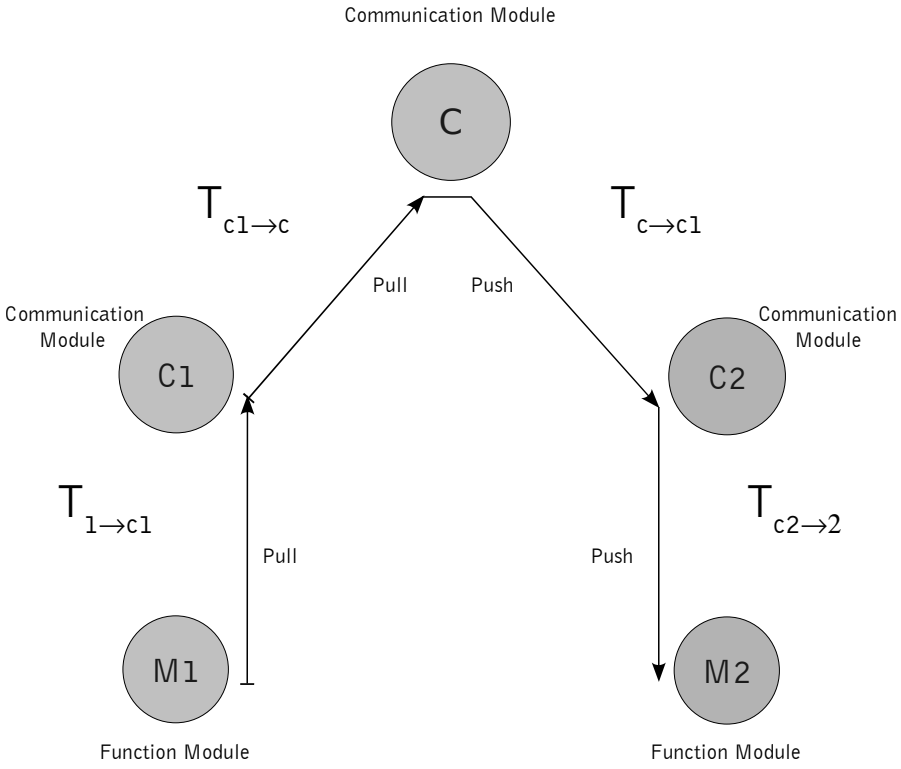


FIGURE 4.30: Common ancestor two levels of hierarchy above. The data flow for a packet transfer from module M1 to module M2 is illustrated by the black unidirectional arrows. The time taken for each leg of the path is also shown. The timeline for this case is an extension of the timeline illustrated in Figure 4.29. Due to the depth of the tree, the protocol address length of the modules M1 and M2 is 2 bytes or longer.

$$
\begin{aligned}
T_{1\to 2} = {} & C_{m1} + \delta_1 + T_f \\
& + S_c + \delta_{c1} + C_{c1} + T_f \\
& + C_{c2} + S_c + C_{c1} + \delta_{c2} + C_{c1} + T_f \\
& + C_{c2} + S_c + C_{c1} + \delta_2 + C_{m2} + T_f
\end{aligned}
\tag{4.5}
$$

4.9.3 Bus Module Capacity for a Simple Case

The equations derived earlier in this section do not provide any insight on the number of modules that can be connected to a bus while achieving the communication objectives. The restrictions on the number of modules, from a communications point of view, depend on the sizes of the packets, the frequency of communication, data rate of the bus and peak bus occupancy amongst others. As such, these parameters change based on the needs of the modules and are hard to predict. However, in order to gain some understanding of the system, let us consider a simple case where all the modules connect to the same physical I^2C bus. Let us also assume that the modules have the same processor speeds, and the function modules have one packet of size d for sending once every polling period T_{mp} to another module on the same hierarchy. Let us also assume that all the communications to a function module are not clustered, and a packet is delivered to the destination immediately (non-slotted delivery of messages) following the pull from the source. Let

$$\eta = \textit{number of modules on the bus.}$$

In this case, T_{mp} is the time available for a function module to perform its core activity and communicate with other modules. To satisfy this polling period of the function module, then, this is the time available for the communication module to poll all the other function modules and round robin to the module again. The time taken by the communication module to poll a function

module is given by Equation (4.3) as follows.

$$T_{cc} = S_c + C_{c1} + \frac{(m+l+e+4)\times 8}{D} + C_{c2}$$

Now, for every pull, the communication module has to push a data packet. Hence, in an ideal case where the modules' activities are synchronized and no context switching overhead is incurred, the maximum number of modules that can be accommodated on the bus is given by the following.

$$\eta = \left\lfloor \frac{T_{mp}}{2 \times T_{cc}} \right\rfloor = \left\lfloor \frac{T_{mp}}{2 \times \left(S_c + C_{c1} + \frac{(m+l+e+4)\times 8}{D} + C_{c2} \right)} \right\rfloor \tag{4.6}$$

The above equation provides an estimate on the maximum number of modules that can be accommodated on a bus with the modules communicating uniformly and consistently in each time period. This estimate is, however, conservative as generally every module may not have data to send or receive in all polling slots. Furthermore, aggregate systems may be composed of modules with heterogeneous polling/update frequency requirements, alleviating the load on the communication bus and tolerating increase in the number of modules.

Assuming that every function module receives one message from a sibling in the duration of a polling period, the ratio r for a function module equals the following.

$$r = \frac{T_{mw}}{T_{mp}} = \frac{T_{mp} - 2 \times \left(S_m + C_{m1} + \frac{(m+l+e+4)\times 8}{D} + C_{m2} \right)}{T_{mp}} \tag{4.7}$$

In this section, we articulated and explored the implications of the various embedded aggregate design decisions on its performance. We derived seven equations (3.1 to 3.7) that are representative of the temporal and communication aspects of the aggregates. More broadly viewed, these equations can be used by an aggregate implementer to make various module and system architecture level decisions toward realizing an embedded aggregate of chosen number of modules and admissible packet exchange times.

4.10 Chapter Summary

In this chapter we developed suitable physical and logical architectures for the embedded aggregates. In the architecture design process, a holistic approach was followed toward choosing the topologies and an existing underlying embedded communication protocol which can be extended to support the aggregate class of systems. The holistic approach involved a multi-pronged effort toward choosing and refining the choices on various fronts toward finally making a selection of protocols and topologies that are mutually complementary and compatible, providing several of the expected features of the communication infrastructure. We conclude that a logical star topology, using polling on each bus of a hierarchical I^2C network, is best suited in the embedded aggregates context, providing reasonable flexibility through limited compromise on communication independence.

Reconciling the requirements of the aggregates communication framework and the facilities provided by the I^2C bus, we extended and developed packet structures, types of packets, module address formats, packet routing, group send and receive (broadcast and leafcast) methods from communication primitives, flow control techniques and mechanisms to discover and initialize aggregate modules. We also presented a basic analysis of the embedded aggregates framework, deriving equations that model communications and functioning of the aggregates. Supported by this ground work, we discuss the design and development of blades and tiles, embedded aggregates for HCI researchers, in the following chapter.

Chapter 5

Embedded Aggregates for Physical User Interface Researchers

> Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry.
>
> Richard P. Feynman

The multidisciplinary nature of interaction device prototyping presents a challenging design and development environment. Towards making the process of UI design, development and prototyping more accessible and flexible from the different UI researcher's point of view, the designers of several UI prototyping toolkits have incorporated modular designs in realizing their hardware platforms. The abstraction and isolation of concerns provided by the modular toolkits have decoupled some of the otherwise strong linkages between the various design realms, leading to faster and easier realization of UI prototypes. Taking inspiration from these prototyping platforms, this researchwork aims to further increase the freedoms enjoyed by the UI researchers by exploring an alternative style and dimension of prototyping platform hardware modularity[1] (discussed in § 5.2 in page 135).

Modularity is naturally occurring[2] [WE03], and scholars argue that "modular" and "integral/consolidated" are end-points on a continuum [Mil05, HSW05]. In man-made systems, the level of modularity - a point on this continuum, is chosen by the designers based on the objectives to be fulfilled. Our specific objectives include scalability, reusability, reliability and flexibility, and we explain these objectives in the context of blades and tiles in § 5.3. We derive our modules

[1] Several analytical methods to evaluate the level of modularity of a system have been developed [NBR98, VG99, TW99, Mik06, HOW07], however researchers assert that there is no widely adopted measure or methodology to evaluate modularity [GPZ03, CC10]. Consequently, in this thesis we do not concern ourselves with the exact analytical measure of modularity of the various platforms.

[2] On a related front, talking about modularity and hierarchy in nature, Simon takes an interesting stand saying "I shall not try to settle which is chicken and which is egg: whether we are able to understand the world because it is hierarchic, or whether it appears hierarchic because those aspects of it which are not elude our understanding and observation." [Sim62]

informed by research in several fields including product design and medicine (human sensory system). Towards developing our envisioned modular platform for interaction device prototyping, we explored the broader field of embedded systems. Our studies indicated that modular systems of the particular scale and features that we envisioned were predominantly created as one-off systems. Convinced of the requirement for a unified architecture and framework toward realizing this class of systems, we explored, characterized and developed the concept of embedded aggregates. Using embedded aggregates as the foundation, we have created blades and tiles, a modular system for interaction device prototyping, pilot deployments and realizing evolving research interaction devices.

A blade is an embedded aggregate module which is designed in accordance to a set of specifications including physical size, interfacing connectors, power characteristics and communications. The implementation of a blade is guided by a variety of HCI motivations. As independent hardware components with individual processing capabilities, blades are intended to be created with self-sufficient specialized interaction-related functionalities and associated API calls. Tiles ground and reinforce the communication fabric and extend a platform to house blades. The creation of a tile is guided by both the electrical needs and HCI motivations. As user interfaces and interactive artifact-prototyping elements, tiles (Figure 5.8) also form the meeting ground between the electronics and tangible components underlying specific interaction modalities (e.g., switches, LEDs, rotary encoders) and the blades (functional components) supporting their underlying implementations.

We begin this chapter with a discussion on our rationale behind our hardware partitioning approach, following which we describe the implementation of blades and tiles electronics. Next, we describe the firmware stack for blades that implements both our embedded aggregates protocol and mechanisms and the blade-specific functionality. The software stack for interfacing bladed devices to a generic computer plays an important role in the decoupling process, and a discussion on the software stack which highlights the design decisions follows the firmware section. The

chapter finally ends with an evaluation of blades and tiles platform.

5.1 Disentangling the Strong Linkages

Interaction devices have evolved into complex electronic entities. Broadly speaking, many interaction devices are composed of a shell that provides the physical form and electronics housed inside the shell that extend the functionalities[3]. This section treats an interaction device at the level of an electronic hardware entity that is purposely constructed and installed for gathering input and disseminating output in a human-perceivable form.

Let us take a typical physical UI composed of several interaction modalities like switches, slides, rotating knob and displays. The realization of a modality in the device can be broadly broken down into two parts, the human engaging part and the underlying electronic implementation. For example, as shown in Figure 5.1, a modality that provides a rotating knob has the human contacting dial portion (a physical object) and an underlying electronic implementation that senses the position of the knob and converts it into digital information. The underlying electronic portion typically involves a embedded processor like a micro-controller and supporting transducer[4] related electronic components.

Traditionally, in the production version of an interaction device, the back-end electronics of the various modalities are collocated and the data processing requirements are met using a few micro-processors on board the device. The human-engaging parts of the interaction modalities may be exposed on the surface of the device (ex. dials, slides, switches) or may be contained within them (ex. gyroscopes and accelerometers). The positioning, arrangement and aesthetics of exposed

[3] It has to be noted that several HCI researchers working on Tangible UI, Ubiquitous UI and related areas pursue interfaces that are not discrete entities with turn-taking interaction sequences, but entities seamlessly woven into the continuum of physical space around the user with a coincidence of input/output space. Here, for the sake of simplicity, we consider a discrete device in order to understand the anatomy of a physical UI and identify the sources of entanglements in the prototyping process.

[4] A transducer is a device, usually electrical, electronic, electro-mechanical, electromagnetic, photonic, or photo-voltaic, that converts one type of energy or physical attribute to another for various purposes including measurement or information transfer (for example: pressure sensors) [from `http://en.wikipedia.org/wiki/Transducer`].

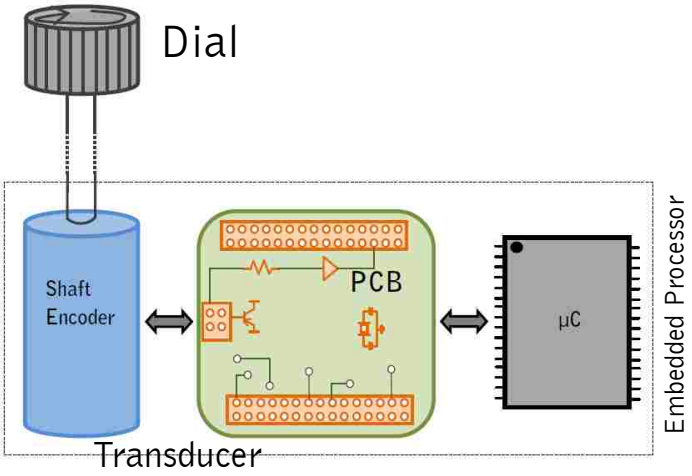


FIGURE 5.1: Example implementation of a rotor modality. The portion of the implementation that is not directly contacted by a user is shown enclosed in the dotted box.

human-engaging components plays an important role in the functioning and usability of the device, and falls in the design domain of the interaction and product designers.

The success of an user interface depends on the coming together of the form and functionality of the device, and hence requires several rounds of refinement during the prototyping process leading up to a pilot implementation. During the prototyping phase, the product and interaction designers iterate on the spatial placement, construction, physical attributes like shape, size, texture and feel of the exposed components of the modalities and the interaction sequence. The hardware developers iteratively enhance the functionality of the hardware (through component and firmware revisions) and attributes like efficiency, size and transducer design amongst others. Concurrently, the software developers build and extend the system software toward supporting a larger set of external systems and applications. For a successful product to be designed and realized, innovations and progress in all the above directions are vital and a prototyping process that allows an unencumbered development is best suited. In an ideal setting, each of the involved practitioners should be able to modify and refine their implementation independently and progressively toward creating a successful device.

During the development of an interaction device, the various design domains impact the decisions and implementations in the other domains. For example, the modifications to the interaction device's electronics may be brought about by the hardware developers enhancing their design or as a consequence of iterations by interaction and product design experts. More specifically, the changes in the positioning and number of interaction modalities requires extensive modifications to the hardware in cases where the built-in processing power, number of available electronic-component interfacing circuits and rigidity of the physical PCB pose restrictions. The modifications in the hardware domain in terms of processors used and power requirements require modifications in the firmware. Also, depending on the changes in the electrical, logical and protocol specifications of the interface between the interaction device and an external system, the external system software requires modifications[5]. The level of independence in the design process hence plays an important role in the smoothness and speed of the prototyping phase.

In a typical UI design, considerable immunity in the software domain from hardware modifications, and vice-versa, are extended by the use of time-tested design techniques including APIs and abstractions in the software stack and in the firmware at the interface between the hardware and software. Consequently, the strongest ties are encountered between the form and functionality (electronic hardware implementation). In such a situation, the consolidation of electronic components and processing duties leads to additional technical challenges and time penalties as each iteration of the form and aesthetics may require iterations of hardware and firmware. Toolkit and prototyping platform developers have consequently adopted a modular design of their platforms.

The modular design breaks the hardware both physically into smaller components and extends a level of abstraction due to the logical segmentation and isolation of functions. This resultant

[5] An interaction experience, in addition to engagement with the physical UI, also involves the software-based graphical UI on the external system, if any. Hence, changes in the interaction experience and sequence also summon appropriate GUI software development and modifications. In this thesis, however, we do not concern with the coupling between software and graphical UI interface design requirements on the external system front.

freedom relaxes the strong linkages between the design realms and increases the isolation and prototyping freedom of the various practitioners. We believe that the current level of disengagement can be further enhanced without loss of prototype performance by carefully architecting the modules with a different level of modularity. In the following section, we qualitatively place the various toolkits (commercially available and currently being developed) on the modularity spectrum and develop module boundaries of our blades.

5.2 Modular Blades

Modularization may be approached from diverse directions, however a strong causality ties together the aim of modularization, definition of modules, method of modularization and the measure of modularity [CC10]. Blades are modules in our system. How does one break a UI into bladed modules? What goes into modular blades and what, if any, remains integral? Researchers have extensively studied and written about this modular-integral dichotomy. Relevant to these questions and speaking about products, in his highly cited paper titled "The Role of Product Architecture in the Manufacturing Firm," Ulrich defines a product architecture as "the scheme by which the function of a product is allocated to its physical components" [Ulr95]. Following this definition, Ulrich suggests a one-to-one mapping from function elements in the functional structure to the physical components of the product. So how does one map the functionalities of a UI to the physical blades? Towards answering these questions we consider a variety of related research in modular product design and we will break an interface into blades in this section.

Several analytical methods have been proposed toward breaking a electronic product into modules [BC00]. In the case of electromechanical[6] systems like physical user interfaces, a "functional structure" that is a set of sub-functions interconnected by flows is widely used [PB96].

[6]Some researchers argue that complete representations of mechanical and electromechanical systems are hard, if not impossible to achieve [FD89, HK98].

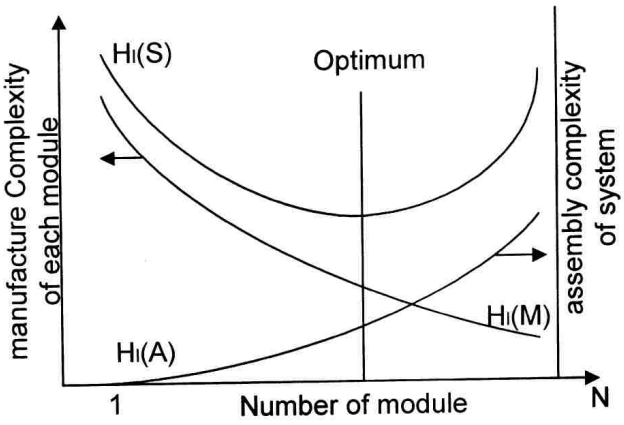


FIGURE 5.2: Number of modules and complexity. Tsai and Wang's complexity variety in different module types in a system. [TW99]

In this approach, the functional dependence of the various components is drawn and the system is broken where there is a smaller set of interconnecting flows between adjacent and connected components. This approach proceeds to reduce the coupling between components and better separation of concerns.

VanHilst and Notkin suggest breaking a system into modules where each module encodes only one design choice [VN96]. Based on Garlan *et al.'s* stand [GAO95], they argue that larger modules are hard to reuse in part because of the number of decisions they encode. On the other hand, in an extreme case, this scheme of partitioning can lead to increased complexity. Tsai and Wang present the effects of number of modules on the complexity of each module and overall assembly cost of a system through a compelling graphic as reproduced in Figure 5.2 [TW99]. They argue that for a given system, while the number of modules is low (segmentation into modules), the manufacturing complexity of each module ($H_1(M)$) is high, but the complexity of assembling the system ($H_1(A)$) with modules is low. At the other end of the spectrum, the complexity of manufacturing each module decreases as the number of modules is increased, however the assembly complexity of the system spikes. As a compromise, they suggest the existence of an optimal level of modularity that minimizes the sum of the two complexity costs ($H_1(S)$).

Shilling shares a similar perspective and presents insightful remarks stating that as the number of modules increases, it becomes more difficult for a user to choose appropriate components or assemble those components into the required configuration [Sch00]. Shelling also argues that for a system with several modules to be successful, the customer must have a great understanding of how the components work both individually and together, and be willing to distinguish and understand the differences in their quality, performance and functionality. Hence, as the number of available modules increases, a greater emphasis is placed on the user, and often a balance between integration that provides additional functionality in the form of preassembly and modularization may have to be achieved by platform developers.

Another criteria to be considered is the answer to the question "Modularity for what?". From a life-cycle point of view, Gershenson *et al.* explore modularity toward reducing product retirement costs, increasing product maintainability and increasing recycling output [GPA99]. They write: "Modularity allows for the reduction of service costs by grouping components so those less reliable components are easily accessed. In addition, grouping components into modules by how they are recycled can greatly reduce product retirement costs". Dahmus *et al.* take a similar stand and further suggest that systems may be divided into modules such that some modules may be made in multiple levels to offer wide variety [JDO01]. From a similar perspective, Gupta and Veerakamolmal suggest partitioning such that the optimum cost-benefit ratio for end-of-life retrieval is achieved [VG99]. These positions and considerations are relevant to us as we are interested in developing user interfaces and interactive artifacts in an evolutionary fashion, gradually upgrading and replacing parts of the interface while reducing replacement costs and electronic wastage.

In his seminal paper, Parnas motivates the notion of a module as a "responsibility" [Par72]. Huang and Kusiak suggest modularization such that concurrent engineering of various components may be carried out and the development time and cost may be reduced [HK98]. Furthermore, they also

suggest the feasibility of collaborative design through the internet by using modules created by others. Both Parnas' and Huang and Kusiak's approaches suggest breaking a system into modules such that each one of them solidifies one direction of expertise. Another important consideration is the threshold of interactivity. While breaking a UI into modules, it is important to consider a fragmentation where a modality or closely related modalities are not separated into modules such that the UI user perceives a lack of fluidity in the interaction ([SJ08], pp. 274-275).

The various suggestions toward defining module boundaries discussed here present a multi-dimensional situation of conflicting compromises. A common philosophy amongst all the pursuits is the segregation into functional blocks with maximum self-sufficiency (minimum dependence on other modules), and well defined interfaces and boundaries. As an interaction device engages with our sensing and actuation abilities, we decided to look for ideas to locate these well defined boundaries by studying human sensory and motor skills. It has been argued by neurologists [Res94, Car98, EK01, ZCE08, MLF$^+$09] that the different parts of the central nervous system perform the actions of sensing, processing and responding to stimuli in a modular, parallel, independent, interactive and interdependent fashion. In her discussions on the human brain, Rita Carter notes in the Mapping the Mind [Car98],

> "The human brain is made of many parts. Each has a specific function: to turn sounds into speech; to process colour; to register fear; to recognize a face or distinguish a fish from a fruit. But this is no static collection of components: each brain is unique, ever-changing and exquisitely sensitive to its environment. Its modules are interdependent and interactive and their functions are not rigidly fixed: sometimes one bit will take over the job of another... The whole is bound together in a dynamic system of systems that does millions of different things in parallel."

We suggest that anatomically an interaction device is composed of several interaction modalities

and related components. These modalities functionally engage with humans by sensing their actions and generating stimuli perceivable by them through electro-* transducers[7]. Looking at the one-to-one mapping between a bodily organ and the part of the UI that engages with it, it is compelling to consider breaking a UI into self-sufficient modality-specific modules, where the different modules engage the sensory systems one-on-one and simultaneously, as a possible solution.

The above solution is attractive for several reasons. First, it nicely yields itself to expert development of individual modules. Researchers, who are experts in a particular kind of interaction modality, may develop modules extending the modality functionality. These modules can be utilized by HCI practitioners who may be geographically separated. Multiple modules providing the same function, with some intended differences (cost, size etc.) may be created and developed into a library of modules (similar to [JDO01]). With an orientation toward developing blades that isolate and extend individual interaction modalities and simultaneously keeping an eye for the threshold of interactivity, we have developed several blades as discussed later in this chapter. In the next section, we describe some of our design goals for blades and tiles.

5.3 Design Goals

Several architectural and functional characteristics are required of the underlying hardware-core toward achieving the decoupling goal. These characteristics are inherited by the blades and tiles by virtue of the embedded aggregate design. In this section, we describe these goals in the blades and tiles context. Our goals are scalability, flexibility, re-usability and reliability of bladed hardware.

- **Scalability:** In blades, by scalability, we refer to the capability of adding bladed interaction

[7]Electro-* transducers are devices that convert from/to electrical signals.

modalities with minimal change in architecture and implementation of the rest of the blades and tiles system.

- **Reusability:** Our interpretation of re-usability has two implications.
  - **Hardware Re-Usability:** An interaction modality built into a bladed interaction device is not specific to the device and is readily re-used in other bladed devices. In other words, a hardware component or hardware design can be re-used to realize the interaction modality in different devices with no modifications.
  - **Firmware Re-Usability:** Two interaction devices implementing a modality use the same firmware for realizing the modality. Put differently, the firmware for implementing a modality is modality-specific and independent of the overall interaction device.
- **Reliability:** The reliability goal addresses challenges both in the context of a distributed system and also certain issues specific to blades and tiles. Blades use a pair of interconnects to link to tiles. The type and placement of these interconnects are the same across all blades. Also, tiles are built with generic connectors to accept any blade. In practice, each of the four function-blade-slots on a tile can be configured to accommodate modality-specific I/O circuits (e.g., RFID coils, joy stick, LEDs, switches). Hence certain constraints on blade placements are introduced by the customizations of a tile. These constraints, together with our expectations that in-field systems will develop failures in blades due to use and end-users bear the responsibility for replacing faulty blades with newer ones, raise reliability concerns.

  By reliability, we hence refer to the system's ability to continue working with faulty blades, possibly with graceful degradation, and also protect against hardware failures due to insertion of a blade in an incompatible slot.

- **Flexibility:** By flexibility, we refer to architectural features that facilitate hot-swapping, interchangeable use of blades from a library with negligible software modifications, and the arrangement of blades and tiles in numerous patterns. Bladed systems can function as stand-alone, independent devices, as network-dependent devices, or as clients to host PCs and gateway devices. For instance, connectivity to a PC or external system can be provided by a gateway communications blade. Our gateway-comm library provides blades with Bluetooth-Serial, RS-232, and USB-Serial communication capabilities. Devices can use any of these implementations with negligible implications on the rest of the system.

The above additional characteristics and functionalities are contributed by the blades and tiles hardware and the system-software architecture. We now proceed to discuss the realization of these goals through the blades and tiles hardware, firmware and software design.

5.4 Hardware

The hardware implementation of blades and tiles is motivated and guided by several factors and considerations. These include aspects of blade size, choice of electronic components and fault-tolerant design amongst others. In the following pages, we discuss these factors and introduce the physical and electrical aspects of the blades and tiles hardware.

5.4.1 Blades

A blade is an embedded aggregate module that is designed in accordance to a set of specifications including physical size, interfacing connectors, power characteristics and communications. As independent hardware components with individual processing capabilities, blades are intended to be created with self-sufficient specialized interaction-related functionalities and associated API calls. Aside from power, communications, and external interactors, for which they depend on the

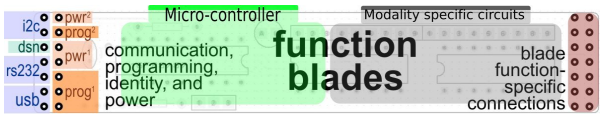


FIGURE 5.3: Concept sketch of a blade. The figure shows the logical parts, and pin-outs of a function blade. Blades contain two sets of interconnects, one generic across all blades with power, programming, and communication pins, and the other blade specific. The circuits on a blade comprise an intelligent processor and interaction specific circuits.

tile, they are oblivious to the electronic realization of other modalities and other blades. Complex interaction devices are envisioned to be built using networks of blades.

Blades are composed of two types of components, as shown in Figure 5.3: a micro-controller for local processing, and electronic components and circuits required to realize the interaction modality. Blades currently use Microchip's PIC series of micro-controllers, but are fundamentally envisioned to be processor independent. Following the embedded aggregate design specifications, each blade is uniquely identified (electronically) by a six byte ID hard-coded in a DSN chip (Dallas DS2401).

Blades fall into three main classes as show in Figure 5.4: core, function, and resource blades. Core blades are instances of embedded aggregate modules that primarily focus on communication in the blades and tiles system. These modules are capable of I^2C master mode and function as system gateways and segregate the system into tile-level hierarchies. Additionally, at the tile level, core blades ensure safe operation of the system by interrogating blade IDs before soft power-up, detecting faulty blades and supporting active power management.

Function/interaction blades implement specific interaction modalities. They are broadly classified as sense, actuate, and display blades. Most blades fall in this category. Resource blades implement supporting resources to augment function and core blades. Examples include battery power, additional memory, supporting computational capabilities and data encryption.

Blade Type	Core		Function							Resource	
Blade	Gate-Way	Intra -Comm	Stepper	Servo	Haptic	RFID	Switch & Led	LCD	Relay & Power	Proto-typing	Battery
Family	Communication & Control		Actuation			sensing	Display		Power control	Multi-purpose	Energy
Implemented and Used	Yes, 1-Blade wide		Yes, 2-Blades wide	Yes, 1-Blade wide							No, Proposed
Purpose and abilities	Control & System-		1 Stepper Motor	4 Servo Motors	1 Haptic Rotor	R\W 4 RFID tags using TDMA	10 Switches & Leds	1 LCD Display	4 Solid State Relays	General Proto-typing	Power for Standalone operation
	Root communica-to and gateway port	Hierarchical Communication & Control									

FIGURE 5.4: Blades classification. Blades are classified as core, function and resource blades. The table shows the various sub-classifications and provides brief details on the various blades.

5.4.1.1 Physical Aspects - Size and Shape

The physical size and shape of blades play an important role in their utility to the HCI community. As the blades form the underlying core electronics for interaction devices, their physical characteristics dictate the conceivable forms of end UIs. In a discussion on affordances of objects [Gib86], Gibson articulates the effects of the size of an object on its graspability, writing,

> " The affordances of what we loosely call objects are extremely various… Some are graspable and other[s] not. To be graspable, an object must have opposite surfaces separated by a distance less than the span of the hand. A five-inch cube can be grasped, but a ten-inch cube cannot. "

Also, in their study on forms of UIs, Ullmer *et al.* survey and discuss the evolution and convergence of the sizes of several tangible interfaces [UIJ05] and write,

> " For example, a number of tangible interfaces have converged on "modes" of cubical or rectangular objects of 10cm or 5cm per side. For instance, systems by Frazer [FFF80], Anagnostou et al. [ADP89], Suzuki and Kato [SK95], and Shieβl [Sch02] all independently converged upon cubes of roughly 10cm/side not far from the "five-inch cube" referred to by Gibson [Gib86]. Similarly, a number

> of token+constraint systems [UIG98] (e.g., mediaBlocks) have converged on tokens of roughly 5cm/side. These sizes seem to reflect the anatomy of the human hand. In classifications of hand postures by Cutkosky and Howe [CH90], the 10cm cube corresponds to a power grasp, while the 5cm sizes corresponds to a precision grasp. ”

We share opinions with Gibson and Ullmer *et al.*, and have hence developed blades of length 10cm, width in the multiples of 2cm and 1cm depth. We do not, however, restrict the length and width of blades to 10 x 2cm, and expect them to be realized in sizes that are factors of the two measurements.

For implementing interaction devices of our choice, we have found the blade width a promising trade off of physical real estate with ability to accommodate standard components. As a specific example underlying blade width, 2cm width will exactly accommodate a 28-pin standard-width through-hole integrated circuit. This is the maximum width common among through-hole components, and the size of (e.g.) the popular IB Technology family of RFID reader hybrid modules. In cases where 2cm width is insufficient, we suggest integer multiples of width (4cm, 6cm, etc.). E.g., our stepper motor blade is 4cm wide, on account of wider driver and power components. Overall, the standardization of the blade sizes, through the physical modularity thus extended, knits closely with the goal of quick integration into systems and replacement by other modules.

5.4.1.2 Electrical Aspects

Blades are an embodiment of the embedded aggregates concept. In the development of the embedded aggregates' design criteria, the electrical aspects of the system were purposefully left unspecified. The embedded aggregates framework was developed to be utilized toward building systems with constraints and features as deemed appropriate by the implementer. In the blades

instance of the aggregates, we have implemented the electrical aspects of the systems as detailed below.

Power

Our current implementation of blades operate with a 5v DC power supply and interface with the communication bus at 5v TTL logic. Two channels of power are fed to the blades through the tiles, and the channels are electrically isolated when required. The first channel (low-current) is used as the source for the on-board processor and the second (high-current) for powering the modality related circuits (Figure 5.3). This separation of the power sources is done toward achieving two end goals: fine-grained power-control and prevention of loss of power to modules' CPUs due to faulty modality payloads in any of the blades. The second goal has special resonance with efforts in building fault-tolerance in blades and tiles. The fine grained power control is facilitated by the additional power control and ID circuitry housed in the tiles. Together, the power and ID circuitry allow us to implement PC ACPI [HP04] like power states in blades and improve fault tolerance by identification and isolation of faulty blades. Additionally, some blades themselves are capable of performing local power management by actively monitoring and moderating power distributed (from the high-power source) to the additional on-board circuitry.

Components

We currently build all blades and tiles with through-hole electronic components. Through-hole components [Wik09] can be easily attached to circuit boards (PCB) by developers with minimal soldering experience. While through-hole components have limitations in terms of size and density, we have found that they provide considerable flexibility for prototyping and small fabrication runs. Simultaneously, we envision smaller blades to be built with surface-mount components, with the added advantage that they are better suited for machine (mass) assembly.

Interconnect

Blades connect to tiles through two 14 pin through-hole interconnects. The layout of the interconnects and the number and order of the pins has undergone several revisions. In our current design, we have one 14-pin interconnect common across all blades (Figure 5.3); the other is blade-specific. The common interconnect provides linkage to power, in-tile + in-circuit blade programming, blade ID interrogation, and blade communication. The configuration of this interconnect has been refined over many iterations to support in-tile-in-circuit programming of different micro-controllers with off-the-shelf programmers. Drawing inspiration from the increasing number of communication protocols being implemented in embedded processors, our communication-related pins are purposefully laid out to facilitate incremental improvements and interchange between complementary communication protocols when situations demand.

5.4.1.3 Blade Evolution

As research prototypes for realizing research and pilot interaction devices, blades have undergone several revisions, both on hardware and firmware fronts. In retrospect, the revisions were considered to support changes in the pin-outs and adding power control features which were progressively developed. The size and shape of the blades have remained consistent over the iterations, however, the backward compatibility has been lost over the revisions. While initially all the blades were developed equally, with several major revisions to ensure compatibility with the blade ecology, later following the proof of concept phase, efforts were predominantly invested in exploring the underlying software and firmware architectures and developing the aggregate concept and framework. Figure 5.5 provides a brief overview on various blades and design iterations.

Type	Specific subtype	Blade	Blades produced	Systems built using the blade	Hardware revisions	Firmware revisions
Core Communication	External and internal communication	USB Gateway-comm	10	many	4	4
		Bluetooth Gateway-comm	4	few		
		RS-232 Gateway-comm	2	couple		
	Tile level hierarchical communication	Intra-comm	20	many	4	3
Function	Display	LCD	12	many	3	4
	Sensing	Switch-Led	25	many	4	3
		RFID	25	many	4	4
		Joystick	3	couple	1	1
		Slider	1	none	1	1
		Rotary encoder	2	none	1	1
	Actuation	Haptic Rotor	15	many	5	3
		Stepper driver	8	many	3	2
		Servo	4	few	1	1
		Power switch	3	few	1	1
	Prototyping	Breadboard blade	8	several	5	N/A
		A/D Blade	2	couple	1	1

FIGURE 5.5: Blades implemented. The table provides details on the various blades implemented and used.

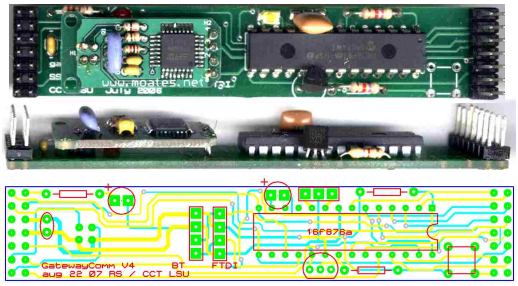


FIGURE 5.6: Gateway communication blade - USB-RS232 protocol. Communications with external systems are mediated by gateway blades. This figure shows a blade that extends the ability to connect to an external system using USB-RS232 protocol. The blade implements USB slave profile using a FTDI chip.

5.4.1.4 Example Blades

Gateway Communication Blade

Gatewaycomm blades act as the protocol translators to enable blades and tiles systems to communicate with other hardware systems. In certain incarnations the gatewaycomm blade doubles its role also as a blade performing message routing internally in blades and tiles system. From a responsibility point of view, the gatewaycomm blade then assumes the role of a conductor and an orchestrator.

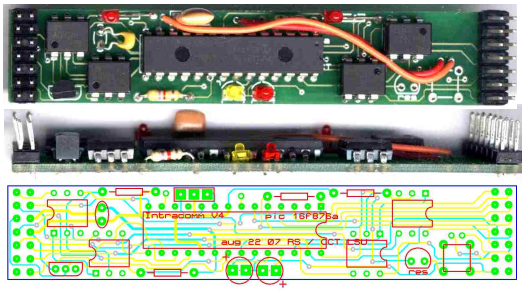


FIGURE 5.7: Intra-Communication blade. Shown here is an intra-communication blade implemented using a PIC 16F876A processor. The through-hole constraint led to the use of a microcontroller with a single I^2C port. Hence in order to serve on two I^2C buses, this design of an intra-communication blade utilizes two pairs of switches to multiplex the buses with the port. The module time-multiplexes between the buses and can also bridge the two buses while being transparent to the communication.

Intra Communication Blade

Intracomm blades, also called the dual-comm blades, perform communication routing functions in a blade network. Attached to a tile, they are responsible for tile level control, communication, power optimizations and fault tolerance. Intracomms, along with gatewaycomms, form the active components of the communication and control network that enables the other blades to coordinate their actions and cooperate in achieving concerted system level functionalities.

As discussed earlier, several other blades have been realized, tested and used in our UI research prototypes. Toward encouraging and fostering development of blades by a larger community of developers, basic blade PCB schematics incorporating a variety of processors is expected to be realized in the future. These schematics will have the connections between the embedded processor and the communication + programming interconnect routed. Blade developers can begin their development from this seed PCB, implementing circuitry specific to the envisioned modalities on them.

5.4.2 Tiles

In our envisioned use cases of blades, we often found them being pursued as clusters of blades rather than individual modules. In a discussion on the abacus in his doctoral thesis [Ull02], Ullmer observes,

> " . . . the pragmatics of mobility and managing numerous physical elements eventually pushed the abacus to a system of captive beads, abacus tokens remained discrete and spatially reconfigurable for much of the device's history. "

This evolution of the abacus has interesting implications for the blades and is partly inspirational to the tile based approach of grounding and packing blades together. One might recall that in our discussion on the embedded aggregates, the details on the physical implementation of the communication fabric was left unspecified. We can implement this fabric and connect the modules using a set of cables, however, in the blades version of the embedded aggregates, we realize the communication fabric using physical PCBs called Tiles. A tile is a piece of electronic hardware to which blades are affixed. Tiles typically contain no processing components and act as substrates providing power, communication and control circuits for the operation of the blades. As UI prototyping elements, tiles (Figure 5.8) also form the meeting ground between the electronics and tangible components underlying specific interaction modalities (e.g., switches, LEDs, rotary encoders) and the blades (functional components) supporting their underlying implementations.

Tiles incorporate three kinds of electronic elements: circuits for blade management and control (e.g., blade connection headers, power control and distribution), circuits that implement tile specific electronics (e.g., tile ID, tile interconnects, power converter, debug circuits) and underlying blade supported modality specific components (typically interactors like RFID coils and LEDs). Table 5.9 provides further details on various tiles and design iterations. Drawing inspiration

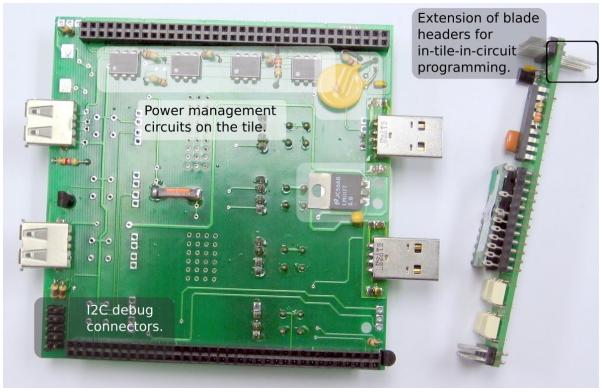


FIGURE 5.8: Bottom of a Tile. The underside of a tile houses the tile specific circuits consisting of blade management, communication and power circuits. This tile has been designed with some perforated PCB space for rapid prototyping.

Tile	Comments	Major Hardware Revision	Major Feature Changes
Access tile type 1 – PCB designed	This is the primary tile design. Other tiles are created by modifying the modality specific circuits and components.	1	Basic tile with four function blades.
		2	Power control version 1.
		3	Intra-comm blade added, communication hierarchy introduced.
		4	Power control version 2. Tile interconnects, tile status LEDs, blade ID and control circuits revised.
Access tile type 2 – PCB designed	Implemented by reproducing the LED and switch specific circuits of the switch-LED blade in a different blade slot from type 1 access tile. This change required limited firmware changes.	1	None.
Rotor tile – ad hoc realization	Realized by drilling a hole in the center of access tile 1 and attaching a haptic rotor. The haptic rotor blade was quickly introduced into one of the empty blade slots.	1	None.

FIGURE 5.9: Tiles implemented. The table presents details on the 3 types of tiles we have realized. A brief detail on the iterative feature changes is also provided.

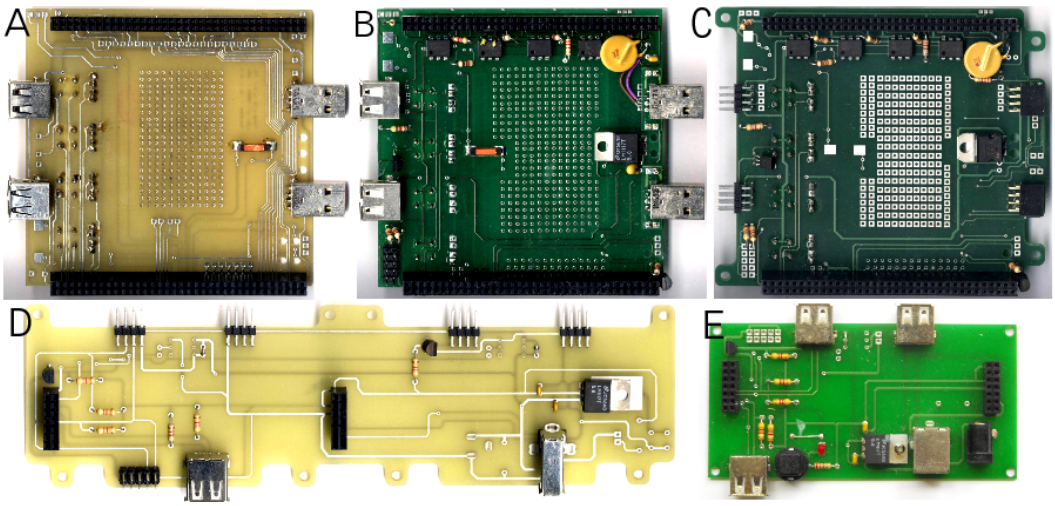


FIGURE 5.10: Tiles and Spines. Three generations of access tiles and two generations of spines are shown here. The exposed side of the boards where the blades are attached are shown here.**A.** The first version of the access tile lacked power control circuits. **B.** The third version of the access tile incorporates individual blade slot power switch. The switches are controlled by the intracomm on the tile and together with DSN interrogation circuits, limited hardware level fault tolerance is realized. **C.** The changes in the fourth version of the access tile includes physical and electrical redesign to accommodate mounting screws and right-angled connection headers for tile-tile interconnects. **D.** A full-spine to which two chains of hardware rev. 4 tiles can be connected. **E.** A half-spine that accommodates one chain of hardware rev. 3 tiles.

from DataTiles [RUO01], tiles are prototypically 10x10 cm, accommodating blades that are prototypically 10 cm long; 2 cm wide; and 1 cm deep. This size affords composition of multiple tiles in sizes that are ubiquitous. For example, the foot print of a 2x2 tile arrangement with a full-spine (Figure 5.10-D) holding them together, fits a letter size paper. These size relationships open up several avenues in the HCI context as discussed by us in [UDS$^+$10].

The blades and tiles hardware is designed to facilitate addition and removal of interaction components, both at the blade & tile levels of granularity. We employ a pair of 70 pin female headers to both electrically connect and mechanically attach blades to tiles. Five blades connect to these connectors with their two sets of 2x7 male headers. Tiles are designed to be chained together, a feature that may be used for creating interaction devices. Each tile connects to one tile on its left and one tile on its right through two sets of interconnects for each tile-to-tile linkage. Our initial tile design employed USB connectors (Figure 5.10-[A-B]) for interlinking them through which

USB, I^2C, 1-wire bus and DC power were run from one tile to another. In the newest revision of the tiles (Figure 5.10-C), however, we use right-angled 4 x 2 male-female breakaway headers. Our earlier choice for the USB connectors was motivated by their ubiquitous nature and our expectation of plugging tiles individually to PCs. However, since then, our usage contexts have converged toward matrices of tiles, and we developed spines that allow us to arrange one (Figure 5.10-E) or two (Figure 5.10-D) rows of tile chains. Additionally, the USB connectors exhibited data and power glitches induced by mechanical stress when the tiles were held by physical constraints in an interaction devices. The above convergence toward a tiles+spine based UI development led to the displacement of the USB connectors by the 4x2 headers, a choice that has also doubled the number of inter-tile conductors.

5.5 Firmware

In the blades and tiles setup, all processing power is located on the blades, with the tiles only containing some supporting active and passive electronic components. The firmware of a blade can be divided into two parts. First, there is the blade-specific portion implementing the blade's functionality and API. The remaining portion, called the base firmware, assembles communication capabilities like I^2C, RS-232, 1-wire etc., implements the embedded aggregates protocol and mechanisms, provides structure to the firmware and dictates the flow of control in the firmware. Over the years, we expect several base firmwares to be created for different processors and architectures in various programming languages.

Blades are envisioned to be developed in large numbers, individually by hardware and sensor/actuator experts. At the hardware level, in addition to creating the blade, the experts will be required to create the blade-specific portion of the firmware and utilize the base firmware of the specific micro-processor platform toward assembling the machine code for the blade.

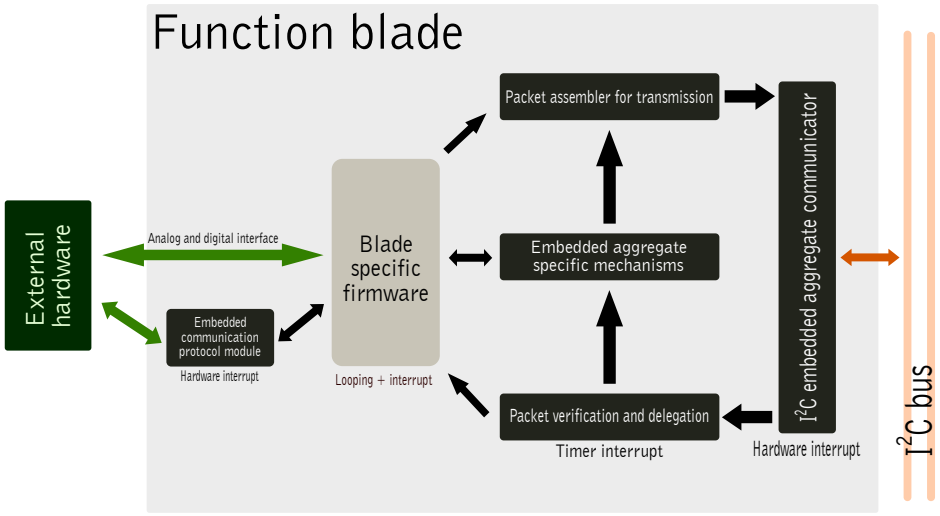


FIGURE 5.11: Function Blade firmware architecture. The firmware of the blade utilizes hardware and timer interrupts to time-slice and switch between tasks. We have written firmware communication modules in C that extend I^2C , RS-232 and 1-wire capabilities, protocol modules that implement embedded aggregate protocol stack, modules that implement communication buffers, and PIC micro-controller specific headers with pin-out customization for blades. A developer of a new blade can utilize these modules, and compile the firmware for a blade after implementing the blade specific function calls.

5.5.1 Architecture and Implementation

The organization and flow of control in the firmware allows a function blade to spend the majority of its time performing its core functionality. The communications over the I^2C network and the separation of communication packets received into blade-specific and aggregate-protocol related categories are performed using hardware and timer interrupts. Figure 5.11 shows the important portions of the firmware stack. A function blade, on power-up undergoes a blade recognition and initialization phase, at the end of which it is provided a protocol address. During the initialization phase, blades generally perform no core functionalities and the messages sent and received on the I^2C bus are embedded aggregate protocol related packets. Post-initialization, blades are able to perform their core functions, intermittently interrupted by communication or timer interrupts toward receiving data packets and processing them. We utilize linear or circular arrays toward implementing the internal send and receive communication buffers.

Blade firmware has been written in C language using CCS's proprietary compiler [CCS09] for PIC Micro-Controllers. With modifications (primarily I^2C and RS-232 communication calls), the code could be made compatible with other C compilers, both for PICs and other architectures. We have divided the firmware into modules that implement the various communication protocols, embedded aggregate mechanisms and manage micro-controller specific initializations and setups. The firmware architecture for the communication blades (gateway and intra-comm) differ from that of function blades in a limited fashion and implement additional control related mechanisms.

5.5.2 Embedded Aggregate Communication API Calls

The communication API is defined in the core communication library. This library provides both basic usb-serial, bluetooth-serial, I^2C and RS-232 primitives and API calls to support the embedded aggregates protocols and mechanisms. In the internal distributed communication context, primitives like send, receive, broadcast, and fetch are provided. To illustrate, a message is sent to a function blade by a core-communication blade by using the sendToBlade API call as follows.

```
...
sendToBlade(whichblade,message)
...
```

The receiving function blade is interrupted by the delivery of the message and processes the data through its local message dependent API calls.

5.5.3 Blade Function API

In addition to the aggregates communication API, blades firmware is written to support function API. The function API is composed of standard calls and calls specific to a blade (not instance of a blade). The standard calls include blade hardware and firmware version, blade hardware id, and

blade power control amongst several others. The blade specific calls expose the modality specific calls supported by the blade. For example, these include calls for various LED patterns supported by the Switch-LED blade, a call for reading a particular RFID or to a call to generate a haptic feedback pattern by the haptic rotor blade. To illustrate with an example, the following are two of the API calls specific to the switch-led blade.

```
...
setLedOff(whichled)
pwmLed(whichled)
...
```

To summarize, over 26 different standard API calls have been implemented, which provide the mechanisms to realize the distributed design paradigm and provide the skeleton to support many blade specific API calls. The API calls have been revised twice since the earliest calls were designed in 2006. The standard and common API calls are implemented in over 1000 lines of C code in firmware, and about a similar LOC of python in software. Including the blade specific libraries, over 4000 lines of C code has been revised over three revisions of firmware development.

5.6 Software

Software is an important part of every UI, and in the case of physical UIs, system software plays an important role in integrating the physical UI device to programs, applications and data on the PC to which it is attached. The architecture of the system software plays a very important role in the blade+tile based interaction devices toward realizing the decoupling goals and has undergone several iterations. These have supported interaction device research; four semesters of class use; and applications to several end-user contexts, including co-located and distributed collaborative visualization and microscopy imaging. For portability, system software has been written in Python, Cython, Tcl, and Java.

We have observed our software converging toward two distinct modes of usage. The first implementation mode couples bladed hardware as mixed hardware/software library elements of medium- to high-end computational systems. The second combines blades as library elements for small computational systems – e.g., as function-specific co-processors for Arduino, Rabbit core modules, or other micro-controller-based systems. Thus far, our efforts have concentrated on the first mode of use. This typically takes the form of integration with a supporting PC or gateway computer.

The software stack for blades has several functionalities and responsibilities, some of which concern themselves with communicating with the bladed system, and others that allow integration and connection of bladed interaction devices with other applications in the system. This thesis work limits itself to the portions of software to the extent of being able to support communication with interaction devices made of several blades. In this section, we will describe the current architecture of the software stack, whose specific design is vital to the decoupling vision, with particular emphasis on the mechanisms closer to the hardware device.

Broadly speaking, we wish to support a blades and tiles ecology where hardware developers can create new blades that are rapidly integrated into other bladed systems. Particularly, we envision a situation where blades are developed by hardware experts, supported appropriately by software architects and quickly integrated into an interaction device by the product and interaction designers. Towards accomplishing this vision, a software stack for blades has to extend the flexibility to recognize the blades, initialize the modalities as part of an interaction device and communicate with the blade toward accomplishing modality related tasks. Currently, we are progressively converging toward a layered software architecture with blade specific software modules.

Computer software for blades is split into 4 layers as shown in Figure 5.12. At the lower levels consisting of Layers 1 and 2, we have implemented a device driver and blade interpreter tasked

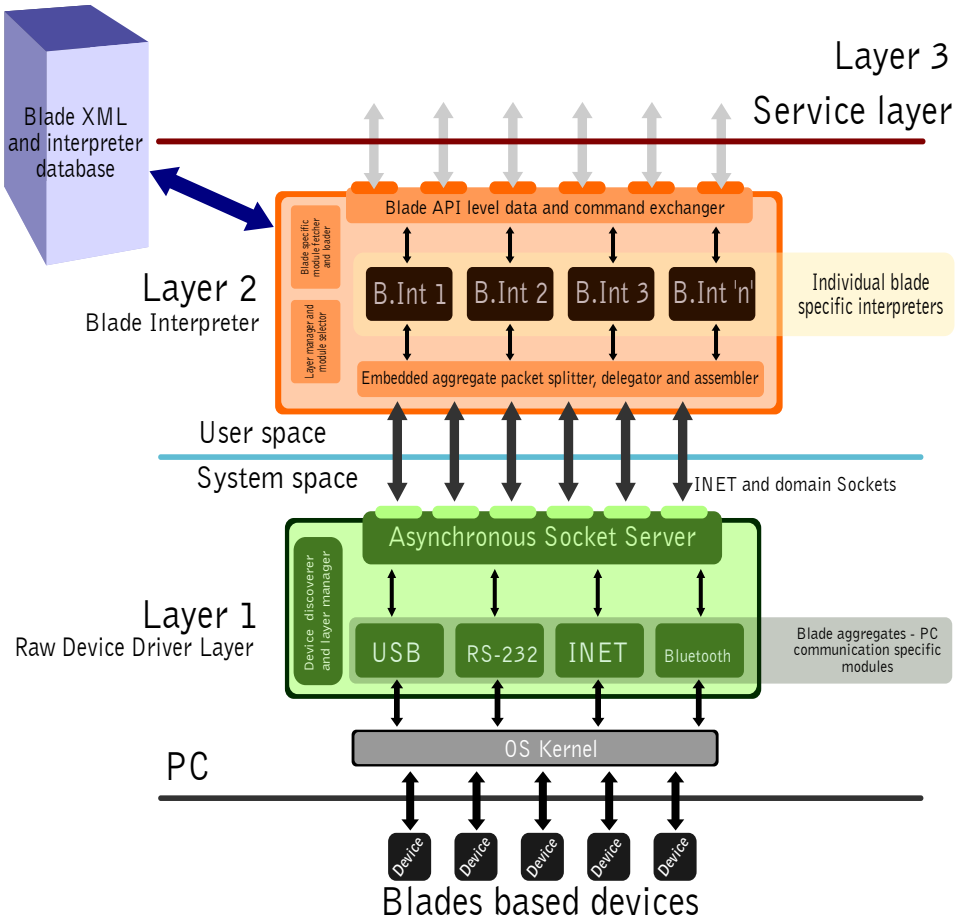


FIGURE 5.12: Computer software stack for interfacing blades and tiles. The software is developed in four layers. As part of this thesis work, layers 1 and 2 were designed and are being developed.

with exchanging raw data with a blades+tiles interaction device. In the next level at Layer 3, the service layer involves a tangibles interaction server which collects and redistributes events to the appropriate target applications. We are also implementing additional services including session handling, and management + coordination models for interaction devices and the target applications to which they are bound at the application layer at the top of the stack at Layer 4. The software contributions as part of this thesis are in Layers 1 and 2.

5.6.1 Layer 1: Raw Device Driver Layer

Layer 1 is closest to the hardware and implements low level communications for interacting with bladed systems. In a bladed system, following the embedded aggregates design, the communications with a PC or external device is mediated by a gateway-communication blade. These blades implement a growing number of communication protocols including USB, RS-232, Bluetooth and 802.11 family of protocols. Layer 1 establishes communications with bladed systems through the appropriate protocol and provides access to them using unified system level INET and Unix domain sockets.

The means for connecting to blade gateways using the various protocols are implemented in protocol specific software modules that are dynamically linked (.so unix files). The modular approach toward implementing protocol support allows hardware developers to rapidly develop new communication blades and promote them in software by implementing an appropriate communication software module. For example, we implemented USB-Serial and Bluetooth-Serial software communication modules which are used in the layer for accessing data from the appropriate blade gateways. The modules use low level system kernel interrupts and messages (inotify [Lov05, Dow], knotify etc.) to detect newly connected devices and establish connections with gateway communication blades.

The various software communication modules exchange data with the gateway blades by

packaging the blade message packets appropriately as dictated by the underlying communication protocol. The message packets follow the embedded aggregate message packet format. On acquiring the data from the hardware device, the modules present them as packets to an asynchronous socket module that implements INET and domain sockets. The socket module advertises new devices and presents a socket through which Layer 2 can connect to the device. Similarly, messages received by the socket module are passed to the communication modules and finally to the device. The software is split at this level as we envision Layer 1 to be run as a system level process that exposes aggregate based hardware in a unified fashion to other processes in the system.

5.6.2 Layer 2: Blade Interpreter

The design of the Layer 1 of the software stack supports the disentangled development of gateway blades. Placed between the interaction server at Layer 3 and the device driver at Layer 1, Layer 2 performs the task of converting message packets into interaction events and vice versa. The Layer 2 design plays a significant role in achieving considerable hardware developer flexibility and its design is closely related to the blades and tiles hardware architecture. Following the modularity principles, Layer 2 implements blade specific dynamically linked modules toward communicating with individual blades. Each blade specific module implements the blade specific API and interprets the data from the blade and packages data in blade specific packets.

In short, Layer 2 consists of a protocol interpreter that parses the embedded aggregates data packets sent by the device. The packets arriving from the device are broken down into individual blade specific sub-packets. The individual sub-packets carry the address of the blade software module in the embedded aggregate protocol address format. The address of the blade software module is equivalent to the protocol address of the blade prefixed with the reserved external gateway port address. This extension of the aggregate protocol addressing into the software realm abstracts

communications with both hardware blades and software modules. The blades are hence oblivious to the physical or virtual nature of external entities with which they communicate. The protocol interpreter on decomposing the data packet, passes the appropriate sub-packets to individual blade specific modules determined by the address. The interpreter additionally also delegates the function of fetching and loading blade specific modules when presented with the information of a new blade following the discovery process. The module associated with a blade and other blade details are specified in an XML file identified by the unique ID of the blade.

API

The software API for Blades consists of a set of both standard calls common across all bladed systems, and calls specific to the function blades that make the particular bladed instance. The standard calls include system discovery, initialization, blade authorizations, communication bandwidth setting, system reset, and data send and receive among others. These calls are implemented in Layer 2 of the software stack. On the hardware side, these calls are implemented as part of the API of the gateway-comm and additionally in any other communication oriented blade.

The blade specific software calls closely follow the appropriate blade functional API and provide a means for triggering blade functionality and digesting the data produced by a blade. These data/commands/requests may include replies to commands previously sent, status messages, input events sensed by the blades and commands to output using the blades. To illustrate how blade specific software API is generated from the firmware API, let us take the example of a Switch-LED blade. In our current implementation, a Switch-Led blade supports the following unique C firmware function:

```
...
setLedOn(whichled)
...
```

This function in firmware sets the appropriate LED on. To trigger this function in a Switch-LED blade, we send a packet of message with the identifier for the particular LED. It has to be noted that this message does not exactly contain setLedOn(whichled) in ACII in the data segment, but a simplified binary equivalent comprising of 2 bytes. The first byte represents the operation 'set led on' and the second byte represents the appropriate LED.

On receiving the message with the 2 bytes of data, the functions implemented in the switch-LED blade's base-firmware pass the data to the portions of the firmware that interpret and perform the blade/modality specific functions. The blade specific function handler has to trigger the appropriate LED on receiving this message. It is expected that the blade hardware and firmware developers provide details for the above blade specific call, like the parameters, order of parameters and binary data format of the particular function. The software developers subsequently implement software calls based on those details.

Given the one-to-one communication between the software and a blade, the blade address and the firmware API call are packaged together into a software API call in our implementation. For example the setLedOn call is translated to the following software API call:

```
...
setLedOn(whichSwitchLEDBlade, whichled)
...
```

Software and application developers who interface bladed interaction devices to applications on a computer may use this call to switch on the appropriate LED connected to a particular blade. This function is exposed by the blade-specific module in Layer 2. On calling this function, the blade specific module generates the above 2 bytes of data that will trigger the action in the appropriate blade and packages it in a complete embedded aggregates message packet. This packet is passed town to the Layer 1 of the software stack using the following call.

```
...
sendToBlades(message packet of setLedOn(whichBlade, whichled))
...
```

On receiving the data pushed down by the above call through the connections between the layers 1 and 2, Layer 1 either sends the packet to the bladed device. The packet may be sent as such or by combining several packets together and creating a compound packet. The packet is sent by the Layer 1 of the software stack to the gateway of the bladed system through the appropriate communication means (USB, Bluetooth etc.) with the help of the associated protocol module.

On receiving the data packet, the gateway blade sends the packet (after extraction and repackaging in the case of a compound packet) on the I^2C bus with the appropriate Switch-LED blade marked as the destination. The command below accomplishes the transfer by calling appropriate lower level I^2C commands.

```
...
sendToBlade(message packet of setLedOn(whichBlade, whichled))
...
```

On receiving the setLedOn command packet, the blade powers the appropriate LED ON, completing the chain of transmissions. During the process of transmissions like the above, the data packets may pass through several levels of bladed hierarchies and may require the invocation of additional functionalities of the intermediary bladed modules. The software stack is currently being extended to implement several calls toward supporting communication across multiple versions of underlying protocol and bandwidth heterogeneities.

5.7 Summary

User interaction with computing systems is today largely carried out through virtual graphical interaction venues, physical interaction modalities or a combination of them. The emergence of electronic hardware prototyping tools and user interface toolkits has reduced the prototyping and design times of electronic user interfaces and interaction artifacts. They have done so while leveraging on pre-built components, instead of building the underlying electronics from the basic circuitry. Over the years through the advancements in these toolkits, interface hardware development has been made much more accessible to non-electrical engineers such that they can build electronic hardware without knowing the very microscopic details of the system. Following the path of these developmental aids that provide methodologies for easy and more intuitive hardware development, we have designed and developed blades and tiles.

This Chapter introduced and presented the rationale behind the blades and tiles approach. The design of blades and tiles has involved careful considerations of several physical, electrical and hardware realization aspects. As a system, this toolset comprises of electronic implementations of various blades and tiles, firmware executed on blades, system software for interfacing bladed devices to computers and the embedded aggregates protocols and mechanisms.

We believe that the concept of a blade and a tile, firmware with user-sandbox, the concept of blade level APIs and a modular software-stack are among the core conceptual contributions in this Chapter. Consequently, this Chapter described in detail these core ideas and principles through examples and illustrations, while restricting discussions on the implementation aspects of these concepts. Furthermore, the blades and tiles system is an ongoing work and certain parts of the software stack and firmware are at varying levels of completion. While all implementations to support this thesis and its claims have been performed, several of these implementation pieces are at varying levels of cohesion with other parts. Consequently, the completed software stack and

firmware are not claimed as primary contributions of this dissertation work. It is expected that the ongoing work on blades and tiles will soon lead to a finished tool-set for use by the HCI – and hopefully, broader communities.

Chapter 6

Conclusion

> This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.
>
> Sir Winston Leonard Spencer-Churchill

With computers and computationally mediated systems becoming ubiquitous and indispensable in modern society, electronic user interfaces and interaction artifacts have come to increasingly occupy the physical space around us. These interfaces may be generic or specialized for a particular use-case, or designed especially for use in specific environments. Independent of the reason for their deployment and use, they all undergo similar steps of design and development. This process typically demands several iterations of the UI's form and functionality toward maturing the design.

The realization of a successful electronic user interface or interactive artifact needs the intervention and synergistic participation of practitioners of a diverse set of sciences and engineering. Currently in the academic research community, physical user interfaces and interactive artifacts are predominantly designed and prototyped either as one-off instances from the ground up, or using functionally rich hardware toolkits and prototyping systems. Like several other fields of research pursued by academicians, this prototyping aspect of interface realization has continued to foster research and advancement on account of both domain specific breakthroughs and due to innovations in the related areas of science and engineering. Motivated by challenges in the realization of specialized user interfaces incorporating several hundreds of interaction elements and in the pursuit of further enhancing the freedoms of various researchers in the prototyping process, this thesis work has made advancements and claims contributions as discussed in the following section. These contributions are spread across the specializations of HCI, computer networks and embedded systems.

6.1 Research Contributions

The contributions of this dissertation research are as follows.

- We identify the need for and motivate the use of an alternative modular hardware approach towards realizing highly specialized user interfaces and interaction artifacts.
- We introduce the concept of embedded aggregates, and we identify and characterize this class of systems.
- We present a holistic design of the embedded aggregate framework and develop a set of communication protocols, specifications and mechanisms that provide the foundations for realizing the aggregate class of systems.
- We develop an instance of the embedded aggregates called blades and tiles for use in the design and development of user interfaces and interaction artifacts, with aspirational generalization toward wider domains of application.

In short, this research has the following components.

- Physical, logical and electrical descriptions for developing a modular electronic hardware toolset and a library of such modules aimed toward UI design and prototyping.
- Communication protocols for realizing such a toolset with heterogeneous modules. This involves the design of the communication protocol suite with a holistic view including optimizations across the physical layer, MAC, and a compound datalink-transport layer.
- Algorithms and mechanisms toward realizing such embedded systems as open-ended designs with the core principles of scalability, flexibility and hardware-firmware reusability.

- A system software architecture for tethering electronic systems built using the multi-module framework to personal computers.

6.2 Future Work

This research work led to the design of a new I^2C based communication protocol suite for use in the embedded aggregates class of systems and the development of the blades and tiles modular hardware platform. In the process of design and development of these entities, we have learned several lessons, gained insights and have identified some future work directions. While some of these directions are more concrete and can be accomplished in the near term, a couple of them look far beyond into the future. In the following paragraphs we describe some of these future directions.

To begin with, on the blades and tiles hardware front, a few future design and development avenues look promising. Our current blade implementations primarily use Microchip's PIC16F87 and PIC16F876A. We have focused our hardware development efforts predominantly toward refining the design of the total system and implementing needed functional submodules on demand. Having produced a scalable and stable design, we have begun developing a new set of blades including a battery blade, Gumstix blade, slider blade, analog/digital blade among others.

Additionally, building on observations of other toolkits, we are considering blades based on AVR micro-controllers, ARM processors, and embedded Linux systems such as the Gumstix family of modules. Bladed systems in the future may include heterogeneous modules in terms of processing power, RAM, ROM and OS. These efforts will also require the development of firmware libraries and software libraries implementing the embedded aggregate protocols and mechanisms for the various processor architectures.

Another hardware level future work can be in the direction of increasing the scaling in blades. In

the current form of the blades and tiles hardware, we have successfully tested networks of ∼40 blades and a dozen tiles. For broader scaling, we believe the use of multiple power sources and I^2C bus isolation+extension using repeaters could provide possible paths forward. The pursuit of these two directions can lead to larger bladed networks that incorporate a wider variety of embedded processors.

The embedded aggregates framework is built upon the I^2C communication protocol. We chose I^2C on account of it being widely implemented in a multitude of microcontrollers. However, the I^2C protocol implemented in several microcontrollers does not support both the 1 Mbit/s Fast mode plus and 3.4 Mbit/s High Speed mode. Furthermore, in our experience with the microcontrollers, we encountered a lack of completeness in the implementation of the I^2C specification. Consequently, we designed the protocol stack to overcome some of the issues relating to the varying levels of protocol implementation and accommodate a wider variety of processors. However, this capability comes at the cost of performance. During the course of this thesis research, in parallel, protocols like SMBus [For03] and PMBus [Sys07] have been developed by others based on the I^2C specification. These new protocols demand stricter hardware implementations and introduce functionalities that are very useful in the aggregate context. As these protocols find their way into many new chips, perhaps a transition from I^2C can be made in the future with the added advantage of improved performance and more functional support at the hardware level.

Necessity is a powerful driver of innovation. In this thesis work, we have demonstrated the advantages and effort required toward developing the embedded aggregate class of systems. Given the applicability of the aggregate class of systems, it may be an useful exercise to design and develop an entirely new physical layer protocol in place of I^2C and its variants.

Another direction worth exploring is the use of RF based wireless communication between aggregate modules. In the chapter on the design of embedded aggregates, we argued for the

use of wired communication protocols in order to reduce the cost, power supply requirements and complexity of the individual modules. Over the last five years, during the course of this research, the cost of implementing wireless communication has steadily decreased. With the advent of software radios and ZigBee communication modules, the prospect of integrating wireless communication capabilities to aggregate modules is becoming more viable and inexpensive.

In the future, embedded aggregates could be an amalgamation of processors of heterogeneous capabilities and communication capabilities. In the current form, several of the packet structure and routing decisions in the embedded aggregate protocols have been made based on nature of the underlying I^2C communication infrastructure. These design decisions and mechanisms may have to be modified when used with wireless communication protocols as the protocols may intrinsically implement several of these features.

How might future blades look and function? How many blades would a futuristic blade cluster contain? Blades could take a variety of forms, including transitioning into the chip die or assuming nano-scale sizes or becoming an order of magnitude larger. In their current form, blades and tiles design has exploited the advantages of modular hardware design only outside a chip (using aggregates of chips). The emergence of IP cores and associated interconnection standards like Wishbone [H^+02], Avalon [Alt11], AMBA [ARM99] and IBM CoreConnect [IBM06] show promise of extending the blades and tiles designs to inside a chip.

With the development of more efficient cross-compilers, place-and-route algorithms and cheaper and easily programmable FPGAs and Hybrid-FPGAs, one may be able to switch between bladed modules with independent processors to bladed modules within a chip. A tool-chain may be able to synthesize our embedded aggregates firmware into a IP cores or byte-code or a combination of the two, and usable within diverse DSPs, heterogeneous multi-cored processors, FPGAs, Hybrid-FPGAs, stacked die ensembles or other emerging architectures expressing diverse computational

functionalities. This direction can open many avenues of possibilities and make the blades applicable in a wider variety of contexts.

This research exercise on communication protocols aimed to develop a lightweight protocol and communication infrastructure that was tractable by the wide range of processors envisioned for use in embedded aggregates. Moving into the realm of nano-scale blades may perhaps require much more modest and efficient protocols and ways to accomplish addressing and communication amongst amongst millions, billions, or perhaps far larger number of nano-blades. This seems a challenging and interesting direction, and research is already underway in the pursuit of such networks [Aky08].

Beyond the above described directions, universal communication [Jub08b, Jub08a] and open-source hardware are two interesting future pursuits. The openness of the embedded aggregate communication infrastructure and the notion that each module acts independently and in a peer fashion in the system, provides a good starting point for both these directions. In terms of universal communication, we envision a scenario wherein blades with minimum knowledge about one another and armed with only primitive communication mechanisms, form a network and develop mutual communication amongst each other, to accomplish larger tasks and form systems synergistically. In this regard, pursuing the routes similar to the Open Mind Common Sense project [Sin10] too can be interesting.

On the open-source hardware front, we envision the creation and sustenance of an ecosystem of hardware modules that can be connected together to create open systems. This begs for further extension of the aggregate protocols and mechanisms, and also extension of module implementation standards. To understand this scenario, let us compare it with open source Linux distributions (ex. Gentoo, Ubuntu, Slackware). On can find the following parallels. First, the hardware modules are akin to software and applications that take advantage of the OS and provide

useful functionality. Second, the hardware framework is similar to the OS itself and extends the basic infrastructure for communication between modules, and supplies power and other vital resources. Finally, the collection of modules together assume the notion of a Linux distribution. Can we foster a Linux like open source platform for hardware? A platform that is open to new modules developed by developers all around the world, a platform that comprises of open firmware, software, hardware and interconnect (protocol and physical interconnect).

6.3 Closing Remarks

Blades and tiles have invoked and reduced to practice the conceptual and theoretical work on embedded aggregates and embedded communication protocols in this dissertation. In response to a National Science Foundation (NSF) Major Research Instrumentation (MRI) proposal in 2005 aimed at creating a new architecture for building interaction devices and scientific instruments [Ull05], research in blades and tiles was undertaken during the early part of the graduate research leading up to this thesis. Our pursuits toward creating an electronic base capable of supporting the creation, and progressive refinement and retirement of generic and application specific user interfaces for computationally mediated scientific applications necessitated the use of a modular approach that differed from the concurrent approaches in user interface design and prototyping electronics.

Using a hardware architecture that embraces bus modularity has seemed to best support our interaction devices and interfacing artifacts. Achieving the bus modularity with electronic modules specifically designed to provide a certain interaction-related functionality has been a formidable challenge undertaken and accomplished in this dissertation research. The creation of blades and tiles with specific properties has required us to design and implement embedded communication protocols to a far greater extent than initially envisioned at the time of the NSF MRI research

proposal [1]. In the middle stages of this thesis, there was motivation toward open source release of bladed electronics, source code for communication libraries and system software in the interest of creating a community of developers. However, due to time and technical constraints such a release and efforts on organizing an initial developer group have strategically been deferred thus far, with recent steps taken to probe new activity in this direction.

Across the duration of these dissertation/thesis efforts, primarily due to the efforts of the dissertation author and with ample support from several graduate and undergraduate researchers, several blades and bladed interaction devices have been created and deployed. Other than use in user interface and tangible interaction research, blades and tiles have been employed in four installments of an undergraduate level HCI and embedded systems course focused on developing novel interaction devices. In their various hardware, protocol, firmware and software revisions, blades and tiles have been used by about roughly hundred students to create and demonstrate user interfaces towards various end applications. Students have worked at various levels of the blades and tiles system including developing new hardware blades, blade-specific firmware and software stack for interfacing bladed devices with other applications. The blades and tiles system is currently under active development in the areas of hardware, system software and protocol implementation in firmware for Microchip PIC processors.

Rapid technology change and low initial cost coupled with product life-cycles as low as 18 months and planned obsolescence have resulted in a fast-growing surplus of electronic waste around the globe [MPS03, KS05, WOKSK$^+$05]. Economies of mass production have dictated producing PCBs with minimum physical components, often involving chips that package almost all functionality of the device inside them. This high density circuitry, often with components, easy to produce, but hard and toxic to recover and recycle, is leading many countries to pass e-scrap laws. Furthermore, the changing global economic scene, environmental awareness and concerns

[1] In the initial proposal submitted to NSF in 2005, the word firmware appeared two times and the word protocol was never mentioned.

may soon lead to the creation of mandates that dictate the production, lifetimes and the right to own electronics.

Our modular design allows replacement and augmentations in parts, bringing with it a greener solution to building emerging interfaces. The amortized cost over the life cycle of the product, reduced waste footprint and its environmental impact can substantially outweigh the relatively high initial production costs. Recycling and reuse are very important for our own survival. Given enough time, Earth finds a way to chew and digest everything ever produced, however, the millions of years taken for their assimilation may prove fatal to us.

Bibliography

[Abr] N. Abramson. The Aloha System: Another Alternative for Computer Communications. In *Proceedings of the, Fall Joint Computer Conference, November 17-19, 1970.*, pages 281–285. ACM.

[ADP89] G. Anagnostou, D. Dewey, and A.T. Patera. Geometry-defining Processors for Engineering Design and Analysis. *The Visual Computer*, 5:304–315, 1989.

[AFM+00] D. Anderson, J.L. Frankel, J. Marks, A. Agarwala, P. Beardsley, J. Hodgins, D. Leigh, K. Ryall, E. Sullivan, and J.S. Yedidia. Tangible Interaction+ Graphical Interpretation: A New Approach to 3D Modeling. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 393–402, 2000.

[Ais79] R. Aish. 3D Input for CAAD Systems. *Computer-Aided Design*, 11:66–70, 1979.

[Aky08] Akyildiz, I.F. and Brunetti, F. and Blázquez, C. Nanonetworks: A new communication paradigm. *Computer Networks*, 52(12):2260–2279, 2008.

[ALM05] T. Arampatzis, J. Lygeros, and S. Manesis. A Survey of Applications of Wireless Sensors and Wireless Sensor Networks. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 719–724. IEEE, 2005.

[Alt11] Altera Corp. Avalon Interface Specifications. 2011.

[AN84] R. Aish and P. Noakes. Architecture Without Numbers - CAAD Based on a 3D Modeling System. In *Computer Aided Design 16*, volume 6, pages 321–328, 1984.

[Ark01] O. Arkin. ICMP Usage in Scanning–The Complete Know How. *The Sys-Security Group, June*, 2001.

[ARM99] ARM Limited. AMBA Specification (Rev 2.0). 1999.

[AS97] D. Awtrey and D. Semiconductor. Transmitting Data and Power Over a One-wire Bus. *Sensors-The Journal of Applied Sensing Technology*, 14:48–51, 1997.

[AT] B. Davies E. Aboba and D. Thaler. Multiple Encapsulation Methods Considered Harmful. Technical report, RFC 4840, April 2007.

[Bai93] G.S. Bailey. Iterative Methodology and Designer Training in Human-computer Interface Design. In *CHI'93: Proceedings of the Interact'93 and CHI'93 Conference on Human Factors in Computing Systems*, pages 198–205. ACM, 1993.

[BBLZ96] D. Baumer, W. Bischofberger, H. Lichter, and H. Zullighoven. User Interface Prototyping-concepts, Tools, and Experience. In *ICSE*, page 532. Published by the IEEE Computer Society, 1996.

[BC00] C.Y. Baldwin and K.B. Clark. *Design Rules: The Power of Modularity*. The MIT Press, Cambridge, MA, 2000.

[BC01] M.S. Blumenthal and D.D. Clark. Rethinking the Design of the Internet: The End-to-End Arguments Vs. The Brave New World. *ACM Transactions on Internet Technology (TOIT)*, 1:70–109, 2001.

[BEA+09] A. Bdeir, L. Eldin, A. Aesquite, Y. Chung, S. Russel, J. Banks, R. Valcich, J. Hoefs, and J. Sturges. Littlebits: Preassembled Circuitboards, Made Easy by Tiny Magnets. Technical report, LittleBits, October 2009.

[BEC09] O.C. Brown, P. Eremenko, and P.D. Collopy. *Value-centric Design Methodologies for Fractionated Spacecraft: Progress Summary From Phase 1 of the DARPA System F6 Program*. 2009.

[BEH02] O. Brown, P. Eremenko, and B.A. Hamilton. The Value Proposition for Fractionated Space Architectures. *Sciences*, 99:2538–2545, 2002.

[Beu00] J. Beutel. *Handbook of Medical Imaging: Physics and Psychophysics*. Spie Press, 2000.

[BG03] M. Beigl and H. Gellersen. Smart-its: An Embedded Platform for Smart Objects. In *In Proc. Smart Objects Conference SOC*, pages 15–17, 2003.

[BKM04] H. Baldus, K. Klabunde, and G. Müsch. Reliable Set-up of Medical Body-sensor Networks. *Wireless Sensor Networks*, pages 353–363, 2004.

[Blo04] C. Block. Conversational User Interfaces. *IT–Information Technology*, 46, 2004.

[BLY05] R. King B. Lo, S. Thiemjarus and G.Z. Yang. Body Sensor Network–A Wireless Sensor Platform for Pervasive Healthcare Monitoring. In *The 3rd International Conference on Pervasive Computing*, volume 13, 2005.

[BM02] R. Bush and D. Meyer. RFC3439: Some Internet Architectural Guidelines and Philosophy. *RFC Editor United States*, 2002.

[Bor08] J.O. Borchers. A Pattern Approach to Interaction Design. *Cognition, Communication and Interaction*, pages 114–131, 2008.

[Bos91] R. Bosch. *CAN Specification Version 2.0*. Bosch, 1991.

[Bot00] H. Boterenbrood. CANOpen, High-level Protocol for CAN-bus. volume 18, 2000.

[BP95] L.S. Brakmo and L.L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on, Selected Areas in Communications*, 13:1465–1480, 1995.

[Bro06a] O. Brown. Fractionated Space Architectures: A Vision for Responsive Space. Technical report, DARPA, 2006.

[Bro06b] O. Brown. The Value Proposition for Fractionated Space Architectures. Technical report, DARPA, 2006.

[BRSB03] R. Ballagas, M. Ringel, M. Stone, and J. Borchers. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments. In *CHI'03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 537–544. ACM, 2003.

[Car98] R. Carter. *Mapping the Mind*. University of California Press, 1998.

[CC10] D. Campagnolo and A. Camuffo. The Concept of Modularity in Management Studies: A Literature Review. *International Journal of Management Reviews*, 12:259–283, 2010.

[CCS09] CCS Inc. CCS C Compiler for Pic Microcontrollers. Web URL `http://wwwccsinfocom/contentphp?page=compilers`, September 2009.

[CDC98] J.H. Saltzer C. David, D.P. Reed and D.D. Clark. Active Networking and End-to-End Arguments. 1998.

[CH90] M.R. Cutkosky and R.D. Howe. Human Grasp Choice and Robotic Grasp Analysis. In *Dextrous robot hands*, pages 5–31. Springer-Verlag New York, Inc., 1990.

[CHP03] J. Collins, L.J Hansen, and J. Pollack. Self-configuring Network for Launch Vehicle and Satellite Avionics. In *28th Annual Gomactech Conference*, 2003.

[CK74] V. Cerf and R. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on, Communications*, 22:637–648, 1974.

[Col94] Collins, Dave. *Designing Object-Oriented User Interfaces*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[col09] *Aerospace System Value Models: A Survey and Observations*, 2009.

[Com08] D. Comer. *Computer Networks and Internets*. Pearson Prentice Hall, 2008.

[D+94] A. Dolenc et al. *An Overview of Rapid Prototyping Technologies in Manufacturing*. 1994.

[DBV+03] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). Technical report, RFC 3315, July, 2003.

[Dha03] A.P. Dhawan. *Medical Image Analysis*, volume 11. Wiley-Interscience, 2003.

[DM98] L. Dagum and R. Menon. OpenMP: An Industry Standard API for Shared-memory Programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.

[DMT00] R. Davis, N. Merriam, and N. Tracey. How Embedded Applications Using an RTOS Can Stay Within On-chip Memory Limits. In *12th EuroMicro Conference on Real-Time Systems*, pages 71–77, 2000.

[Dow] E.M. Dow. Monitor Linux File System Events With Inotify. Technical report.

[DP88] D. Dewey and A.T. Patera. *Geometry-defining Processors for Partial Differential Equations*, pages 67–96. Academic Press Ltd., 1988.

[DS68] R. Descartes and F.E. Sutcliffe. *Discourse on Method and the Meditations*. Penguin Classics, 1968.

[DS87] W.J. Dally and C.L. Seitz. Deadlock-free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on, Computers*, 100:547–553, 1987.

[DS00] D. Dietrich and T. Sauter. Evolution Potentials for Fieldbus Systems. In *IEEE International Workshop on Factory Communication Systems WFCS*, 2000.

[DT01] W.J. Dally and B. Towles. Route Packets, Not Wires: On-chip Interconnection Networks. In *Proceedings Design Automation Conference*, pages 684–689. IEEE, 2001.

[DT04] W.J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[DYN03] J. Duato, S. Yalamanchili, and L.M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2003.

[EBE04] M. Eisenberg, L. Buechley, and N. Elumeze. Computation and Construction Kits: Toward the Next Generation of Tangible Building Media for Children. *Proceedings of Cognition and Exploratory Learning in the Digital Age (CELDA), Lisbon, Portugal*, 2004.

[ECDG06] M. Eng, K. Camarata, E.Y.L. Do, and M.D. Gross. FlexM: Designing a Physical Construction Kit for 3D Modeling. *International Journal of Architectural Computing*, 4:27–47, 2006.

[EEG+02] M. Eisenberg, A. Eisenberg, M. Gross, K. Kaowthumrong, N. Lee, and W. Lovett. Computationally-enhanced Construction Kits for Children: Prototype and Principles. In *Proceedings of the Fifth International Conference of the Learning Sciences*, pages 23–26, 2002.

[EK01] P. Erdi and T. Kiss. The Complexity of the Brain: Structural, Functional and Dynamic Modules. *Emergent Neural Computational Architectures Based on Neuroscience: Towards Neuroscience Inspired Computing*, pages 203–211, 2001.

[Elm97] J.L. Elman. *Rethinking Innateness: A Connectionist Perspective on Development*. The MIT Press, Cambridge, MA, 1997.

[FD89] S. Finger and J.R. Dixon. A Review of Research in Mechanical Engineering Design. Part II: Representations, Analysis, and Design for the Life Cycle. *Research in Engineering Design*, pages 121–137, 1989.

[Fen81] T. Feng. A Survey of Interconnection Networks. *Computer*, 14:12–27, 1981.

[FFF80] J.H. Frazer, J.M. Frazer, and P.A. Frazer. Intelligent Physical Three-dimensional Modelling System. *Proceedings of Computer Graphics*, 80:359–370, 1980.

[FFF81] J. Frazer, J. Frazer, and P. Frazer. New Developments in Intelligent Modelling. *Computer Graphics*, 81:139–154, 1981.

[FFF82] J.H. Frazer, J.M. Frazer, and P.A. Frazer. Three-dimensional Data Input Devices. In *Proc. of Conference on Computers/graphics in the Building Process*, pages 409–416, 1982.

[Fod83] J.A. Fodor. *The Modularity of Mind*, volume 341. The MIT Press, Cambridge, MA, 1983.

[For03] SBS Implementers Forum. *System Management Bus (SMB) Specification*. SBS-IF, 2003.

[Fra95] J.H. Frazer. An Evolutionary Architecture. *Architectural Association*, 1995.

[Fre74] D. Freides. Human Information Processing and Sensory Modality: Cross-modal Functions, Information Complexity, Memory, and Deficit. *Psychological Bulletin*, 81(5):284, 1974.

[GAO95] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch: Why Reuse is So Hard. *Software, IEEE*, 12:17–26, 1995.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, MA, 1994.

[GF01] S. Greenberg and C. Fitchett. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. In *UIST'01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 209–218. ACM, 2001.

[GHSR97] P. Gu, M. Hashemian, S. Sosale, and E. Rivin. An Integrated Modular Design Methodology for Life-Cycle Engineering. *CIRP Annals-Manufacturing Technology*, 46(1):71–74, 1997.

[Gib86] J.J. Gibson. The Ecological Approach to Visual Perception. 1986.

[GO97] M.G. Gorbet and M. Orth. Triangles: Design of a Physical/digital Construction Kit. In *Proceedings of the 2nd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, pages 125–128. ACM, 1997.

[GOI98] M.G Gorbet, M. Orth, and H. Ishii. Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography. In *CHI'98: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1998.

[GPA99] J.K. Gershenson, G.J. Prasad, and S. Allamneni. Modular Product Design: A Life-cycle View. *Journal of Integrated Design & Process Science*, 3:13–26, 1999.

[GPZ03] J.K. Gershenson, G.J. Prasad, and Y. Zhang. Product Modularity: Definitions and Benefits. *Journal of Engineering Design*, 14:295–313, 2003.

[GSRRU07] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya. Self-Healing Systems–Survey and Synthesis. *Decision Support Systems*, 42(4):2164–2185, 2007.

[H+02] R. Herveille et al. WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores. *OpenCores Organization*, 2002.

[HBC+05] T. Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong, and W. Verplank. ACM SIGCHI Curricula for Human-computer Interaction. August 2005.

[Hea00] K.B. Heaton. Physical Pixels. Master's thesis, Massachusetts Institute of Technology, 2000.

[Hea03] S. Heath. *Embedded Systems Design*. Newnes, 2003.

[HK98] C.C. Huang and A. Kusiak. Modularity in Design of Products and Systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 28:66–77, 1998.

[HKB+06] B. Hartmann, S.R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee. reflective Physical Prototyping Through Integrated Design, Test, and Analysis. In *UIST'06: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, pages 299–308. ACM, 2006.

[HM06a] S.E. Hudson and J. Mankoff. Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape. In *UIST'06: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, pages 289–298. ACM, 2006.

[HM06b] S.E. Hudson and J. Mankoff. Rapid Construction of Functioning Physical Interfaces From Cardboard, Thumbtacks, Tin Foil and Masking Tape. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, pages 289–298. ACM, 2006.

[HN] J. Horst and J. Noble. Distributed and Centralized Task Allocation: When and Where to Use Them. In *Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*, pages 1–8. IEEE.

[HOW07] K. Hölttä-Otto and O. De Weck. Degree of Modularity in Engineering Systems and Products With Technical and Business Constraints. *Concurrent Engineering*, 15:113, 2007.

[HP04] I. Hewlett-Packard. Microsoft, Phoenix, and Toshiba. Advanced Configuration and Power Interface Specification. *ACPI Specification Document, Revision*, 3, 2004.

[HSW05] K. Hölttä, E.S. Suh, and O. De Weck. Trade-off Between Modularity and Performance for Engineered Systems and Products. In *ICED 2005: The 15th International Conference on Engineering Design*, pages 15–18, 2005.

[Hui98] C. Huitema. *IPv6: The New Internet Protocol*. Prentice Hall, 1998.

[HZ83] E. Horowitz and A. Zorat. Divide-and-Conquer for Parallel Processing. *IEEE transactions on computers*, pages 582–585, 1983.

[IBM06] IBM Corp. IBM CoreConnect, 2006.

[ID+04] T. Igoe, O. Dan, et al. Physical Computing: Sensing and Controlling the Physical World With Computers. 2004.

[Ish09] H. Ishii. Tangible User Interfaces. *Human-Computer Interaction: Design Issues, Solutions, and Applications*, 2009.

[IU97] H. Ishii and B. Ullmer. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 234–241. ACM, 1997.

[JD02] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation With TCP Throughput. *ACM SIGCOMM Computer Communication Review*, 32:295–308, 2002.

[JDO01] Gonzalez-Zugasti J.B. Dahmus, J.P and K.N. Otto. Modular Product Architecture. *Design Studies*, 22:409–424, 2001.

[Jer77] A.J. Jerri. The Shannon Sampling Theorem - Its Various Extensions and Applications: A Tutorial Review. *Proceedings of the IEEE*, 65(11):1565–1596, 1977.

[Jol06] D. Joliffe. Arduino Fever. *MAKE V7*, pages 52–53, 2006.

[Jub08a] Juba, B. and Sudan, M. Universal semantic communication I. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 123–132. ACM, 2008.

[Jub08b] Juba, B. and Sudan, M. Universal semantic communication II: A theory of goal-oriented communication. Technical report, Technical Report TR08-095, ECCC, 2008.

[K+02] F. Kordon et al. An Introduction to Rapid System Prototyping. *IEEE Transactions on Software Engineering*, pages 817–821, 2002.

[KA05] S. Kobayashi and M. Akamatsu. spinner: A Simple Approach to Reconfigurable User Interfaces. In *NIME'05: Proceedings of the 2005 Conference on New Interfaces for Musical Expression*, pages 208–211. National University of Singapore, 2005.

[KCJ02a] M.D. Gross K. Camarata, E.Y.L. Do and B.R. Johnson. Navigational Blocks: Navigating Information Space With Tangible Media. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, pages 31–38. ACM, 2002.

[KCJ02b] M.D. Gross K. Camarata, E.Y.L. Do and B.R. Johnson. Navigational Blocks: Tangible Navigation of Digital Information. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems*, pages 752–753. ACM, 2002.

[KEHO06] S. Kobayashi, T. Endo, K. Harada, and S. Oishi. Gainer: A Reconfigurable I/O Module and Software Libraries for Education. In *NIME'06: Proceedings of the 2006 Conference on New Interfaces for Musical Expression*, pages 346–351, 2006.

[KIMK00] Y. Kitamura, Y. Itoh, T. Masaki, and F. Kishino. Activecube: A Bi-directional User Interface Using Cubes. In *Proceedings. Fourth International Conference On, Knowledge-based Intelligent Engineering Systems and Allied Technologies*, volume 1, 2000.

[KK96] R. Kjeldsen and J. Kender. Toward the Use of Gesture in Traditional User Interfaces. 1996.

[KLLL04] S.R. Klemmer, J. Li, J. Lin, and J.A. Landay. papier-mache: Toolkit Support for Tangible Input. In *CHI'04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 399–406. ACM, 2004.

[KM97] K. Kramer and N. Minar. Stackables: Manipulable Distributed Displays. Master's thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 1997.

[KM07] J. Kramer and J. Magee. Self-Managed Systems: An Architectural Challenge. 2007.

[KPTT99] C. Kitts, F. Pranajaya, J. Townsend, and R. Twiggs. Emerald: An Experimental Mission in Robust Distributed Space Systems. In *The 13th Annual AIAA/USU Conference on Small Satellites*, 1999.

[Kra98] K.H. Kramer. *Moveable Objects, Mobile Code*. PhD thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 1998.

[KS05] H.Y. Kang and J.M. Schoenung. Electronic Waste Recycling: A Review of us infrastructure and technology options. *Resources, Conservation & Recycling*, 45(4):368–400, 2005.

[KSBY02] V. Kapilal, AG Sparks, JM Buffington, and Q. Yan. Spacecraft Formation Flying: Dynamics and Control. In *Proceedings of American Control Conference*, volume 6, pages 4137–4141. IEEE, 2002.

[KTON06] H. Kimura, E. Tokunaga, Y. Okuda, and T. Nakajima. Cookieflavors: Easy Building Blocks for Wireless Tangible Input. In *CHI'06: CHI'06 Extended Abstracts on Human Factors in Computing Systems*, pages 965–970. ACM, 2006.

[Lab09] BUG Labs. Bugbase. Technical report, BUG LABS, October 2009.

[LAH+04] J.C. Lee, D. Avrahami, S.E. Hudson, J. Forlizzi, P.H. Dietz, and D. Leigh. The Calder Toolkit: Wired and Wireless Components for Rapidly Prototyping Interactive Devices. In *DIS'04: Proceedings of the 5th Conference on Designing Interactive Systems*, pages 167–175. ACM, 2004.

[Lai01] L. Laixing. Formation Flying of Small Satellite and Its Orbital Configuration [j]. *Chinese Space Science and Technology*, 1, 2001.

[Lam80] S.S. Lam. A Carrier Sense Multiple Access Protocol for Local Networks. *Computer Networks (1976)*, 4(1):21–32, 1980.

[Lan92] R.N. Langlois. External Economies and Economic Progress: the Case of the Microcomputer Industry. *The Business History Review*, 66:1–50, 1992.

[LCC+09] K. Lorincz, B. Chen, G.W. Challen, A.R. Chowdhury, S. Patel, P. Bonato, M. Welsh, et al. Mercury: A Wearable Sensor Network Platform for High-fidelity Motion Analysis. In *Proc. 7th ACM Conference on Embedded Networked Sensor Systems (sensys' 09)*, 2009.

[LM95] J.A. Landay and B.A. Myers. Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 43–50. ACM Press/Addison-Wesley Publishing Co., 1995.

[Lov05] R. Love. Kernel Korner: Intro to Inotify. *Linux Journal*, 2005:8, 2005.

[Mak09] MakingThings. Teleo: Hardware Tools to Create Unique Interactive Devices. October 2009.

[Mar07] M. Marti. *Complexity Management: Optimizing Product Architecture of Industrial Products*. 2007.

[MB76] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, 19(7):395–404, 1976.

[McN99] T.S. McNerney. *Tangible Programming Bricks: An Approach to Making Programming accessible to Everyone*. PhD thesis, Massachusetts Institute of Technology, 1999.

[McT02] M.F. McTear. Spoken Dialogue Technology: Enabling the Conversational User Interface. *ACM Computing Surveys (CSUR)*, 34:90–169, 2002.

[McT05] M.F. McTear. Spoken Dialogue Technology: Toward the Conversational User Interface. *Computational Linguistics*, 31:403–405, 2005.

[MGGR02] J. Miller, J. Guerrero, D. Goldstein, and T. Robinson. Spaceframe-Modular Spacecraft Building Blocks for Plug and Play Spacecraft. *Breakthrough Technologies-The Foundation of the Future*, 2002.

[Mik06] J.H. Mikkola. Capturing the Degree of Embedded Modularity in Product Architectures*. *Journal of Product Innovation Management*, 23:128–146, 2006.

[Mil05] J. Miltenburg. *Manufacturing Strategy: How to Formulate and Implement a Winning Plan*. 2005.

[MLF+09] D. Meunier, R. Lambiotte, A. Fornito, K. Ersche, and E.T. Bullmore. Hierarchical modularity in human brain functional networks. *Frontiers in Neuroinformatics*, 3, 2009.

[MMC+00] P. Maglio, T. Matlock, C. Campbell, S. Zhai, and B. Smith. Gaze and Speech in Attentive User Interfaces. *Advances in Multimodal Interfaces ICMI 2000*, pages 1–7, 2000.

[Moo02] T. Moors. A Critical Review of End-to-End Arguments in System Design. volume 2, pages 1214–1219, 2002.

[MPS03] M. Macauley, K. Palmer, and J.S. Shih. Dealing With Electronic Waste: Modeling the Costs and Environmental Benefits of Computer Monitor Disposal. *Journal of Environmental Management*, 68(1):13–22, 2003.

[MW05] C. Mathieu and A.L. Weigel. Assessing the Flexibility Provided by Fractionated Spacecraft. *Space*, 2005.

[MXD+07] Y. Mei, C. Xian, S. Das, Y.C. Hu, and Y.H. Lu. Sensor Replacement Using Mobile Robots. *Computer Communications*, 30(13):2615–2626, 2007.

[NB93] J. Nielsen and M. Bellcore. Iterative User-interface Design. *Computer*, 26(11):32–41, 1993.

[NB96] B. Nichols and D. Buttlar. *Pthreads Programming*. O'Reilly Media, Inc, 1996.

[NBR98] P.J. Newcomb, B. Bras, and D.W. Rosen. Implications of Modularity on Product Design for the Life Cycle. *Transaction of the ASME Journal of Mechanical Design*, 120:483–490, 1998.

[NC93] Laurence Nigay and Joëlle Coutaz. A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion. In *Proceedings of the Interact'93 and CHI'93 Conference on Human Factors in Computing Systems*, CHI'93, pages 172–178. ACM, 1993.

[NF00] S. Nolfi and D. Floreano. Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. 2000.

[NL03] J. Nielsen and H. H. Lund. Spiking Neural Building Block Robot With Hebbian Learning. In *proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 2, pages 1363–1369, 2003.

[OC00] S. Oviatt and P. Cohen. Perceptual User Interfaces: Multimodal Interfaces That Process What Comes Naturally. *Communications of the ACM*, 43:45–53, 2000.

[OCW+00] S. Oviatt, P. Cohen, L. Wu, J. Vergo, L. Duncan, B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, et al. Designing the User Interface for Multimodal Speech and Pen-based Gesture Applications: State-of-the-Art Systems and Future Research Directions. *Human-Computer Interaction*, 15:263–322, 2000.

[OI04] D. O'Sullivan and T. Igoe. *Physical Computing: Sensing and Controlling the Physical World With Computers*. Course Technology, 2004.

[OMSJ06] C. Otto, A. Milenkovic, C. Sanders, and E. Jovanov. System Architecture of a Wireless Body Area Sensor Network for Ubiquitous Health Monitoring. *Journal of Mobile Multimedia*, 1:307–326, 2006.

[O'N09] M.G. O'Neill. *Assessing the Impacts of Fractionation on Pointing-Intensive Spacecraft*. PhD thesis, Massachusetts Institute of Technology, 2009.

[P+81] J. Postel et al. Internet Protocol. Technical report, 1981.

[Par72] D.L. Parnas. On the Criteria to Be Used in Decomposing Systems Into Modules. *Communications of the ACM*, 15:1053–1058, 1972.

[Pas02] A. Pasetti. *Software Frameworks and Embedded Control Systems*. Springer-Verlag Berlin, Heidelberg, 2002.

[Pax06] V. Paxson. End-to-End Routing Behavior in the Internet. *ACM SIGCOMM Computer Communication Review*, 36:41–56, 2006.

[PB96] G. Pahl and W. Beitz. *Engineering Design: A Systematic Approach*. Springer, 1996.

[PD93] B.J. Pine and S. Davis. *Mass Customization*. Harvard Business School, 1993.

[PD03] L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2003.

[PD07] L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2007.

[Pei97] C.S. Peirce. The Logic of Relatives. *The Monist*, 7:85–95, 1897.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. *ACM SIGCOMM Computer Communication Review*, 28:303–314, 1998.

[PG05] C. Pinart and G.J. Giralt. On Managing Optical Services in Future Control-plane-enabled IP/WDM Networks. *Lightwave Technology*, 23:2868–2876, 2005.

[PKSS02] B. Palmintier, C. Kitts, P. Stang, and M. Swartwout. A Distributed Computing Architecture for Small Satellite and Multi-spacecraft Missions. In *16th Annual AIAA/USE Conference on Small Satellites*, 2002.

[Plu82] D. Plummer. An Ethernet Address Resolution Protocol. Technical report, STD 37, RFC 826, MIT, 1982.

[Pot00] G.J. Pottie. Wireless Sensor Networks. In *Information Theory Workshop, 1998*, pages 139–140. IEEE, 2000.

[Rad97] J. Radatz. The IEEE Standard Dictionary of Electrical and Electronics Terms. IEEE, 1997.

[Rag06] C.S. Raghavendra. *Wireless Sensor Networks*. Springer Verlag, 2006.

[Raw08] A.O. Rawashdeh. Towards Decentralized Management of Graceful Degradation in Distributed Embedded Systems. In *IEEE Dependable Systems and Networks Conference*, 2008.

[RBKV02] D. Rus, Z. Butler, K. Kotay, and M. Vona. Self-Reconfiguring Robots. *Communications of the ACM*, 45(3):39–45, 2002.

[Res94] R.M. Restak. *The Modular Brain*. Scribner's, 1994.

[RPI04] H.S. Raffle, A.J. Parkes, and H. Ishii. topobo: A Constructive Assembly System With Kinetic Memory. In *CHI'04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 647–654. ACM, 2004.

[RUO01] J. Rekimoto, B. Ullmer, and H. Oba. Datatiles: A Modular Platform for Mixed Physical and Graphical Interactions. In *CHI'01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 269–276. ACM, 2001.

[Rus01] R.A. Russell. Tracking Chemical Plumes in Constrained Environments. *Robotica*, 19(04):451–458, 2001.

[San95] R. Sanchez. Strategic Flexibility in Product Competition. *Strategic Management Journal*, 16:135–159, 1995.

[SB08] N. Shah and O. Brown. Fractionated Satellites: Changing the Future of Risk and Opportunity for Space Systems. *High Frontier*, 2008.

[SBL+07] R. Shepherd, S. Beirne, K.T. Lau, B. Corcoran, and D. Diamond. Monitoring Chemical Plumes in an Environmental Sensing Chamber with a Wireless Chemical Sensor Network. *Sensors and Actuators B: chemical*, 121(1):142–149, 2007.

[SBM01] C. Sabol, R. Burns, and C.A. McLaughlin. Satellite Formation Flying Design and Evolution. *Journal of Spacecraft and Rockets*, 38:270–278, 2001.

[SBS+95] T. Sterling, D.J. Becker, D. Savarese, J.E. Dorband, U.A. Ranawake, and C.V. Packer. Beowulf: A Parallel Workstation for Scientific Computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14. CRC Press, 1995.

[Sch00] M.A. Schilling. Toward a General Modular Systems Theory and Its Application to Interfirm Product Modularity. *The Academy of Management Review*, 25:312–334, 2000.

[Sch02] S. Schießl. Digital Cubes, 2002.

[SCH05] F. Vahid S. Cotterell, R. Mannion and H. Hsieh. Eblocks: An Enabling Technology for Basic Sensor Based Systems. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pages 58–es. IEEE Press, 2005.

[SCV04] K. Downey S. Cotterell and F. Vahid. Applications and Experiments With Eblocks-electronic Blocks for Basic Sensor-based Systems. In *Sensor and Ad Hoc Communications and Networks*, pages 7–15. IEEE, 2004.

[SD+98] R. Hinden S. Deering et al. Internet Protocol, Version 6 (IPv6) Specification. Technical report, 1998.

[Sem] Dallas Semiconductor. DS2401 Silicon Serial Number. Technical report.

[SG06] E. Schweikardt and M.D. Gross. Roblocks: A Robotic Construction Kit for Mathematics and Science Education. In *Proceedings of the 8th International Conference on Multimodal Interfaces*, pages 72–75. ACM, 2006.

[SG08] E. Schweikardt and M.D. Gross. The Robot is the Program: Interacting With Roblocks. In *TEI'08: Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, pages 167–168. ACM, 2008.

[Sha10] Shaer, Orit and Hornecker, Eva. *Tangible User Interfaces*. Now Publishers Inc., Hanover, MA, USA, 2010.

[Sim62] H.A. Simon. The Architecture of Complexity. *Proceedings of the American Philosophical Society*, 106:467–482, 1962.

[Sin10] Singh, P. and Lin, T. and Mueller, E. and Lim, G. and Perkins, T. and Li Zhu, W. Open Mind Common Sense: Knowledge acquisition from the general public. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 1223–1237, 2010.

[SIW+02] E. Sharlin, Y. Itoh, B. Watson, Y. Kitamura, S. Sutphen, and L. Liu. Cognitive Cubes: A Tangible User Interface for Cognitive Assessment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing Our World, Changing Ourselves*, pages 347–354. ACM, 2002.

[SJ08] A. Sears and J.A. Jacko. *The Human-computer Interaction Handbook*. 2008.

[SK95] H. Suzuki and H. Kato. Interaction-level Support for Collaborative Learning: Algoblock—An Open Programming Language. In *CSCL'95: the First International Conference on Computer Support for Collaborative Learning*, pages 349–355, 1995.

[SKSL94] M. Swartwout, C. Kitts, P. Stang, and E.G. Lightsey. A Standardized, Distributed Computing Architecture: Results From Three Universities. In *19th Annual AIAA/USU Conference on Small Satellites*, pages 8–11, 1994.

[SMS06] B. Salemi, M. Moll, and W.M. Shen. Superbot: A Deployable, Multi-functional, and Modular Self-reconfigurable Robotic System. In *Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2006.

[Sni98] M. Snir. *MPI–The Complete Reference: the MPI Core*. The MIT Press, Cambridge, MA, 1998.

[SÖV09] D. Sandra, J.O. Östman, and J. Verschueren. *Cognition and Pragmatics*. 2009.

[SOW+95] M. Snir, S.W. Otto, D.W. Walker, J. Dongarra, and S. Huss-Lederman. *MPI: The Complete Reference*. The MIT Press, Cambridge, MA, 1995.

[SRC84] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-End Arguments in System Design. *ACM Trans. Comput. Syst.*, 2:277–288, 1984.

[SRP07] H. Sharp, Y. Rogers, and J. Preece. *Interaction Design: Beyond Human-computer Interaction*. Wiley Hoboken, 2007.

[Sta05] Stankovic, J.A. and Lee, I. and Mok, A. and Rajkumar, R. Opportunities and obligations for Physical Computing Systems. *Computer*, 38(11): 23 – 31, 2005.

[SW95] W.R. Stevens and G.R. Wright. *TCP/IP Illustrated (vol. 2): The Implementation*. Addison-Wesley, 1995.

[Sys07] System Management Interface Forum. *Power System Management Protocol Specification Rev 1.1*. SM-IF, 2007.

[TB80] F.A. Tobagi and V. Bruce Hunt. Performance Analysis of Carrier Sense Multiple Access With Collision Detection. *Computer Networks (1976)*, 4:245–259, 1980.

[TFCG06] A. Tatsch, N. Fitz-Coy, and S. Gladun. On-Orbit Servicing: A Brief Survey. In *Performance Metrics for Intelligent Systems Conference*, 2006.

[Tho05] J.P. Thomesse. Fieldbus technology in industrial automation. *Proceedings of the IEEE*, 93(6):1073–1101, 2005.

[TKYM04] H. Tanaka, Y. Kawahara, T. Yairi, and K. Machida. Research on Reconfigurable Space System Using Orbital Servicing Robots and Cellular Satellites. In *twenty-fourth International Symposium on Space Technology and Science*, pages 674–679, 2004.

[TP06] S.E. Thompson and S. Parthasarathy. Moore's Law: The Future of Si Microelectronics. *Materials Today*, 9(6):20–25, 2006.

[TT90] H. Thimbleby and W. Thimbleby. User Interface Design. *ACM Press Frontier Series*, 1990.

[TW99] Y.T. Tsai and K.S. Wang. The Development of Modular-based Design in Considering Technology Complexity. *European Journal of Operational Research*, 119:692–703, 1999.

[UDS+10] B. Ullmer, Z. Dever, R. Sankaran, C. Toole Jr, C. Freeman, B. Cassady, C. Wiley, M. Diabi, A. Wallace Jr, M. DeLatin, et al. Cartouche: Conventions for Tangibles Bridging Diverse Interactive Systems. In *TEI 2010: Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*, pages 93–100. ACM, 2010.

[UI97] B. Ullmer and H. Ishii. The Metadesk: Models and Prototypes for Tangible User Interfaces. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 223–232. ACM, 1997.

[UIG98] B. Ullmer, H. Ishii, and D. Glas. mediaBlocks: Physical Containers, Transports, and Controls for Online Media. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 379–386. ACM, 1998.

[UIJ05] B. Ullmer, H. Ishii, and R.J.K. Jacob. Token+Constraint Systems for Tangible Interaction with Digital Information. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(1):81–118, 2005.

[UKK+01] B. Ullmer, E. Kim, A. Kilian, S. Gray, and H. Ishii. Strata/ICC: Physical Models as Computational Interfaces. In *CHI'01 extended abstracts on Human factors in computing systems*, pages 373–374. ACM, 2001.

[Ull02] B.A. Ullmer. *Tangible Interfaces for Manipulating Aggregates of Digital Information*. PhD thesis, Massachusetts Institute of Technology, 2002.

[Ull05] Ullmer, Brygg. MRI: Development of Viz Tangibles and VizNet: Instrumentation for Interactive Visualization, Simulation, and Collaboration. URL `http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0521559`, September 2005.

[Ulr] Ulrich, K.T. and Tung, K. *Fundamentals of Product Modularity*.

[Ulr95] K. Ulrich. The Role of Product Architecture in the Manufacturing Firm. *Research Policy*, 24:419–440, 1995.

[USA09] USAF. Space Modular Systems. Technical report, FAS, March 2009.

[Ver02] R. Vertegaal. Designing Attentive Interfaces. In *Proceedings of the 2002 Symposium on Eye Tracking Research & Applications*. ACM, 2002.

[Ver03] R. Vertegaal. Attentive User Interfaces. *Communications of the ACM*, 46:31–33, 2003.

[VG99] P. Veerakamolmal and S.M. Gupta. A Combinatorial Cost-benefit Analysis Methodology for Designing Modular Electronic Products for the Environment. In *International Symposium on Electronics and the Environment, ISEE-1999*, pages 268–273. IEEE, 1999.

[VGRG07] N. Villar, K. Gilleade, D. Ramdunyellis, and H. Gellersen. The VoodooIO Gaming Kit: A Real-time Adaptable Gaming Controller. *Computers in Entertainment (CIE)*, 5:7, 2007.

[VM99] G. Von Dassow and E. Munro. Modularity in Animal Development and Evolution: Elements of a Conceptual Framework for Evodevo. *J. Exp. Zool.(Mol. Dev. Evol.)*, 285:307–325, 1999.

[VN96] M. VanHilst and D. Notkin. Decoupling Change From Design. *ACM SIGSOFT Software Engineering Notes*, 21:58–69, 1996.

[VSG02] Kristof V.L., Albrecht S., and Hans-Werner G. Pin & Play: Networking Objects Through Pins. In *UBICOMP'02*, pages 219–228, 2002.

[VSH11] N. Villar, J. Scott, and S. Hodges. Prototyping With Microsoft .Net Gadgeteer. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 377–380. ACM, 2011.

[VVS+03] K. Van Laerhoven, N. Villar, A. Schmidt, H.W. Gellersen, M. Hakansson, and L.E. Holmquist. Pin&play: the Surface As Network Medium. *Communications Magazine, IEEE*, 41:90–95, 2003.

[Wan06] B. Wang. *Information-Theoretic Methods for Modularity in Engineering Design*. PhD thesis, California Institute of Technology, 2006.

[WE03] M.J. West-Eberhard. *Developmental Plasticity and Evolution*. Oxford University Press, USA, 2003.

[WH02] H. Balakrishnan W.R. Heinzelman, A. Chandrakasan. Energy-efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10–pp. IEEE, 2002.

[Wik09] Wikipedia. Through-hole Technology. `http://enwikipediaorg/wiki/Through-hole_technology`, September 2009.

[WOKSK+05] R. Widmer, H. Oswald-Krapf, D. Sinha-Khetriwal, M. Schnellmann, and H. Bóni. Global Perspectives on E-waste. *Environmental Impact Assessment Review*, 25(5):436–458, 2005.

[WUF92] M.B. Wall, K.T. Ulrich, and W.C. Flowers. Evaluating Prototyping Technologies for Product Design. *Research in Engineering Design*, 3:163–177, 1992.

[YDR00] M. Yim, D.G. Duff, and K.D. Roufas. Polybot: A Modular Reconfigurable Robot. In *IEEE International Conference on Robotics and Automation ICRA'00*, volume 1, pages 514–520, 2000.

[YHE02] W. Ye, J. Heidemann, and D. Estrin. An Energy-efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings, Infocom. Twenty-first Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1567–1576. IEEE, 2002.

[YWMS+07] M. Yim, S Wei-Min, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular Self-reconfigurable Robot Systems [grand Challenges of Robotics]. volume 14, pages 43–52, 2007.

[ZCE08] P. Rosa-Neto J. Germann Z.J. Chen, Y. He and A.C. Evans. Revealing Modular Architecture of Human Brain Structural Networks by Using Cortical Thickness From MRI. *Cerebral Cortex*, 2008.

[ZCL07] V. Zykov, A. Chan, and H. Lipson. Molecubes: An Open-source Modular Robotics Kit. In *IROS*, 2007.

[Zig] Zigbee Alliance. Zigbee Specification.

[Zim80] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *Communications, IEEE Transactions on*, 28:425–432, 1980.

[ZSS05] D. Zarzhitsky, D.F. Spears, and W.M. Spears. Distributed Robotics Approach to Chemical Plume Tracing. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 4034–4039. IEEE, 2005.

Vita

Rajesh Sankaran was born in Chennai, India, on 8 January 1983 to Mrs. Visalakshi Sankaran and Mr. Sankaran Nagarajan Pattam. After graduating from Space Central School with AISSCE in 2000, he studied electronics and communication engineering at V.M.K.V Engineering College, Periyar University, India, from 2000 through 2004 toward obtaining his Bachelor of Engineering degree. He is currently a doctoral candidate in the Department of Electrical & Computer Engineering at Louisiana State University, Baton Rouge, where he has been a graduate student since Spring 2005. During his time at LSU, he has been a member of the Tangible Visualization group at the Center for Computation & Technology. In July 2011, he will be joining the Mathematics and Computer Science Division at the Argonne National Laboratory, Chicago, as a postdoctoral fellow.