A Context-Aware Model for Dynamic Adaptability of Software for Embedded Systems

by

Camille Lydia Leota Jaggernauth

M.Eng., Simon Fraser University, 2008

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Engineering Science
Faculty of Applied Science

© Camille Lydia Leota Jaggernauth 2016

SIMON FRASER UNIVERSITY

Summer 2016

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced, without authorization, under the conditions for Fair Dealing. Therefore, limited reproduction of this work for the purposes of private study, research, education, satire, parody, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

Approval

Name:	**Camille Lydia Leota Jaggernauth**
Degree:	**Doctor of Philosophy**
Title:	***A Context-Aware Model for Dynamic Adaptability of Software for Embedded Systems***
Examining Committee:	**Chair:** Andrew Rawicz Professor

Bozena Kaminska
Senior Supervisor
Professor

Fabio Campi
Supervisor
Lecturer

Craig Scratchley
Internal Examiner
Lecturer
School of Engineering Science/Faculty of Applied Sciences

Shahriar Mirabbasi
External Examiner
Professor
Electrical and Computer Engineering
University of British Columbia

Date Defended/Approved: July 14, 2016

Abstract

The scientific contributions of this thesis are three-fold. Firstly, a novel specialized embedded systems software architecture for context-awareness is presented. This architecture is developed for use on a resource constrained hardware platform (single-processor micro-controller with low processing power and limited memory space such as a legacy processor or a low power processor typically found in wireless sensor networks or energy-aware or cost-aware solutions) and is low latency. For firmware applications with many sources of context, a specialized architecture is important to achieve code readability, modularity, extensibility and maintainability. Context in embedded systems firmware development is defined as changeable and characterizing information such as sensor data. The embedded systems software architecture is a layered model with context and cognitive planes which focus on dynamic adaptability. The context plane features a micro-architecture, which includes context collectors, context controllers and a context task based coordinator. A second focus was on dynamic architecture adaptability in the form of a cognitive engine (cognitive plane) which processes real-time updates to its user-configurable module. Dynamic adaptability improves the application software's flexibility and responsiveness according to different user requirements or varying operational conditions. Thirdly, the concept of context-aware map logic (CAML) is introduced. Cognitive engine updates are performed using these logic maps which are derived from/inspired by fuzzy cognitive maps (FCM) and GPS (global positioning system) coverage maps. The logic maps feature phi, delta, timer, complement, latched and momentary operands. No previous work has been done on the use of fuzzy cognitive maps specifically with linguistic weights for enabling dynamic, resource constrained firmware adaptability. Fuzzy cognitive maps are at the intersection of fuzzy logic and neural networks. An industrial application, Novax's Accessible Pedestrian System (APS) and simulations using the Rapita suite of tools are presented for architecture's proof of concept and evaluation.

Keywords: embedded software architecture; adaptability; context-aware model; fuzzy cognitive maps; wireless sensor networks.

Dedication

Thanks be to God for all His mercies in my life (Jeremiah 33:3) and for Sarah and Jasmine.

Acknowledgements

Thanks be to God for all His mercies and blessings in my life.

Thanks to my family for their love and support. Until we meet again Grandma/Grandpa.

Thanks to my friends for all their encouragement.

Thanks to Dr. Kaminska for the opportunity and guidance during the doctoral studies.

Thanks to everyone who played a crucial role in helping me get to this point.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ADC	Analog to Digital Converter
APS	Accessible Pedestrian System
ATE	Automatic Test Equipment
CAML	Context Aware Map Logic
CiBER	Centre for Integrative Bio-Engineering Research
ECG	Electrocardiogram
FCM	Fuzzy Cognitive Map
FSM	Finite State Machine
GPS	Global Positioning System
IMS	Integrated Multi-Sensor
MVC	Model-View-Controller
OS	Operating System
QoS	Quality of Service
RF	Radio frequency
SFU	Simon Fraser University
UML	Unified Modelling Language

Glossary

Architecture	Overall structure, logical components, and the logical interrelationships of a system.
Context	Information used in the situational characterization of an entity, including explicitly provided user information.
Firmware	Embedded systems software.
Framework	Basic structure for a certain class of applications (focus on design reuse.
Infrastructure	Well-established, pervasive, reliable, and publicly accessible set of technologies that act as a foundation for other systems.
Library	Generalized set of related algorithms (focus on code reuse).
Middleware	Middleware enables applications by filling in the functionality gap between the application, OS, network and hardware layers.
nesC	Network embedded system C.
TinyOS	Open source operating system designed for low powered wireless devices.
Toolkit	Large number of reusable components for common functionality (builds on frameworks).
Service Infrastructure	Middleware technologies that can be accessed through a network.

Chapter 1.

Introduction

1.1. Context-Aware Model for Embedded Systems Software

The motivation for this research is to demonstrate a novel dynamically adaptable context-aware architecture to improve the application software's flexibility and responsiveness according to different user requirements or varying operational conditions which is specifically designed for low footprint (code-space), single processor, potentially energy-aware, solutions typically found in either wireless sensor networks, mature systems, cost-aware solutions or legacy implementations. This adaptability is similar to modifiable configuration parameters but extended to configuration logic and achieved by translating the adaptable code functionality into context-aware logic maps. Here the context-aware logic maps and their syntax is inspired by fuzzy cognitive maps (FCM). A specialized, context-aware, embedded systems software (henceforth referred to as firmware) architecture is important to achieve code modularity, maintainability and extensibility.

Figure 1.1 shows the basic components of a FCM - concepts (C) and weights (e) connected by causal links. The adaptability focuses on changing firmware operation by changing the strengths of weights or rewiring the operation of the logic map by adding or removing concepts. An example of use is a mobile phone application communicating with a battery operated wearable (e.g. wristwatch with physiological sensors (temperature, heart-rate, accelerometer) [3], [4]). Different patients would require different physiological parameters to be monitored at different times and hence a customized logic map would be downloaded to the device for each of these scenarios. If for example, the position (accelerometer) reading/concept is not required and hence not present in the logic map, this device can be powered off or put in lower power mode, realizing energy savings.

Additionally, the battery usage/energy required for a logic map update is low when compared to upgrading the entire firmware. This is because there are less instructions/bytes to be received and processed with the logic map as compared to the firmware executable (longer “on” period for transceiver). The context aware map logic is for use with customizable user logic and not core functionality. This level of flexibility allows for the application to be more responsive to end-user incremental changes. A comparison of the proposed solution versus the state of the art is presented in detail in Section 2.6.

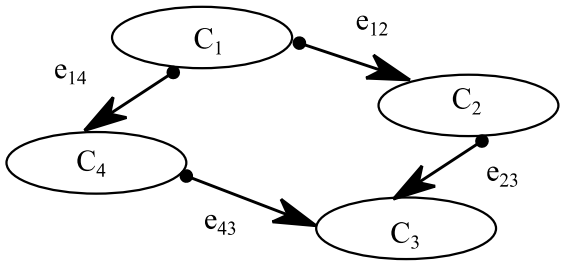


Figure 1.1. Generic FCM with concepts C and weights e.

1.2. Solution

This section will describe our micro-architecture components and data-flow. Figure 1.2 shows the new context layer which lies above the application layer (usually a classic MVC (Model View Controller) architecture standard to many legacy embedded applications).

Figure 1.3 shows our micro-architecture within the context-aware layer. The components of the micro-architecture are context collectors, context controllers, a context-coordinator engine and a cognitive engine.

Context Collector - The collectors can either interface directly with the sensor device drivers for reading the sensor data or through a hardware abstraction layer.

Context Controller - The context controller is responsible for integrating all the related contexts to produce a meaningful control sequence.

Context Coordinator - The context coordinator engine manages inter-context component and inter-layer messaging. The context coordinator is also responsible for scheduling of tasks.

Cognitive Engine - The cognitive engine enables dynamic adaptability.

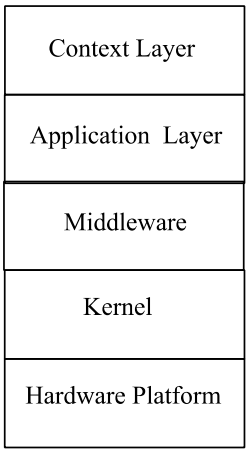


Figure 1.2. Multi-layer context-aware model

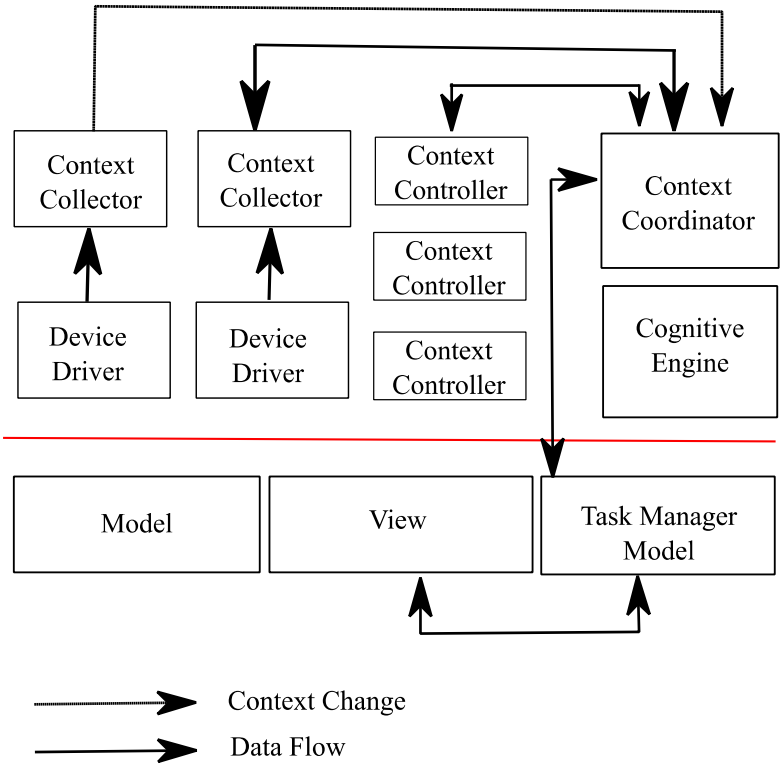


Figure 1.3. Proposed architecture

Data flow (Figure 1.4) occurs via events (e.g. context changes) and queries.

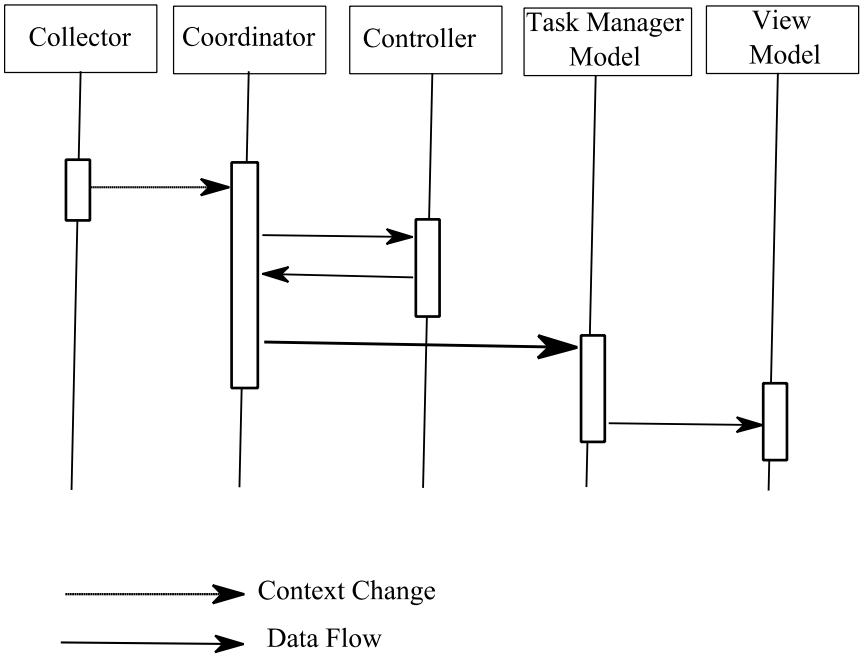


Figure 1.4. Proposed architecture data flow

1.3. Proof-of-Concept Demonstrations

The thesis research model is applied to 3 scenarios, dispatching, home health-care and accessible pedestrian system applications.

A dispatching application running on a mobile dispatch terminal (embedded system) found in cabs is used in accessing customers for taxi-car drivers. Taxi cabs can receive customers via street-hire, auction, reserve, closest cab and stand jobs. Taxis equipped with GPS are capable of determining the current zone of a driver. As the car moves from zone to zone it is eligible to bid on jobs from that zone, be it auction jobs (left over jobs), reserve jobs (future jobs), closest cab jobs (current jobs and also dependent on how long the driver has been idle) or stand jobs (dependent on proximity to the stand). Jobs are filtered based on driver (multi-lingual) and vehicle attributes (roof rack, lift equipped, minibus etc.), or driver state (active, suspended).

A home health care application is used to monitor the elderly. It may comprise a wearable, wirelessly enabled embedded system with sensors which are capable of measuring physiological data.

Novax produces accessible pedestrian systems (APS), Figure 1.5. A typical scenario is a pedestrian approaches an intersection and pushes the APS' button, the red led goes on and the button plays "wait to cross." The button receives the walk signal from the traffic controller and plays "walk sign is on." The button receives the flashing don't walk (or ped clear) signal from the traffic controller and plays an audible countdown (10, 9, 8 down to 1).

Figure 1.5. APS photo courtesy of Novax Industries Corp

1.4. Scientific Contribution

The scientific contribution of this thesis is to demonstrate a novel dynamically adaptable context-aware architecture to improve the application software's flexibility and responsiveness according to different user requirements or varying operational conditions which are specifically designed for low footprint (code-space), single processor, potentially energy-aware solutions, typically found in either wireless sensor networks, mature systems, cost-aware solutions or legacy implementations.

This adaptability is similar to modifiable configuration parameters but extended to configuration logic and achieved by translating the adaptable code functionality into

context-aware logic maps. Here the context-aware logic maps and their syntax is inspired by FCMs. The adaptability focuses on changing firmware operation by changing the strengths of, or rewiring the operation of the logic map. When a user e.g. mobile phone application communicating with a battery operated, wristwatch wearable equipped with physiological sensors, requires different functionality, a different logic map is downloaded.

The battery usage/energy required for a logic map update is low when compared to upgrading the entire firmware (because there are less instructions/bytes to be received and processed) especially for cases where the changes are not foundational functionality necessitating a firmware upgrade. And the coding of the logic into maps allows for the application to be responsive to end-user incremental changes.

The presented efforts in this thesis and other relevant work participated by the thesis author have been converted into the following publications:

[1] Camille Jaggernauth, Bozena Kaminska and Douglas Gubbe, "Context-Aware Model for Dynamic Adaptability of Software for Embedded Systems," International Journal of Computer, vol 19, no 1 (2015), pp 91-113.

[2] Camille Jaggernauth, "Modeling Returned Biomedical Devices in a Lean Manufacturing Environment," Modeling and Simulation of Complex Social Systems, Intelligent Systems Reference Library, Springer, 2014.

[3] Camille Jaggernauth, Yindar Chuo, Benny Hung, Philip Lin, Bozena Kaminska, “Test Firmware Architecture for a Flexible Wireless Physiological Multi-Sensor”, IEEE SMC 2011.

[4] Camille Jaggernauth, Yindar Chuo, Benny Hung, Philip Lin, Bozena Kaminska, “Optimized, Practical Firmware Design for a Novel Flexible Wireless Multi-Sensor Platform for Body Activity and Vitals Monitoring”, IEEE CES 2011.

[5] Y Chuo; M Marzencki; B Hung; C Jaggernauth; K Tavakolian; P Lin; B Kaminska, “Mechanically Flexible Wireless Multisensor Platform for Human Physical

Activity and Vitals Monitoring" IEEE Transactions on Biomedical Circuits and Systems (October 2010), 4 (5), pg. 281-294.

[6] Bozena Kaminska, Yindar Chuo, Marcin Marzencki, Benny Hung, Camille Jaggernauth, Kouhyar Tavakolian, Philip Lin, "Complete Platform for Remote Health Management,"IIIS 2009.

1.4.1. Research Intent

Potential applications of the thesis research include:

- Custom logic based on different customer requirements. The transit priority industry (APS test case (Figure 1.5)) is characterized by highly custom firmware solutions.

- Special one of occasions – different functionality depending on time of year, emergency scenarios.

- In field trouble-shooting – connection/correct hardware operation testing versus actual, business logic defined, device operation.

- Debug logic for special cases – log the values of certain collectors e.g. signal state, point of program execution when atypical conditions occur - brown-outs, voltage ripple, bad flash checksum.

- Learning by teaching the inputs (remote control application, Section 5.1.4).

In the state of the art comparison (Chapter 2) of the thesis research there are no easily compared numerical analysis mentioned. Dey [6] mentions 1) quality of service with respect to application specific desired coverage or resolution for location and 2) component real-estate and Rehman [65] considers scalability while Gamez [13] tracks latency with respect to self-adaptation for a non-resource constrained solution (desktop/cell-phone). For the thesis research model, three software metrics were chosen that are meaningful to a resource-constrained solution implementation to highlight the value of the work - code complexity, latency and code size.

A specialized architecture is important to achieve code readability, modularity, extensibility and maintainability. An architecture for a resource-constrained hardware

platform (single processor, lower clock speeds, low memory) which has high latency (too many abstraction layers) will affect the overall time taken to execute instructions. This in turn lowers the achievable sampling rate for querying hardware peripherals resulting in less precise device readings with the potential for missing exception events. An overly complex solution will also affect the overall hardware sampling rate if there is extensive catch-all functionality that needs to be traversed.

Additionally, overly complex solutions also affect code readability, extensibility and maintainability. Hence latency and code-complexity are key metrics for evaluating embedded software architecture. Code size is also an important consideration for processors with low integrated memory footprint. These metrics are described in greater detail in Section 5.2.

1.5. Organization of Thesis

The rest of the thesis will proceed as follows, Chapter 2 will present background foundational material on embedded systems, firmware, context-awareness and adaptability. Chapters 3 will discuss the proposed thesis model's context collectors' and context controllers' development including a section on a wireless, physiological, integrated multi-sensor's (IMS sensor) operational and ATE firmware. The concepts of context collectors and context controllers were realized while developing firmware for this sensor. Chapter 4 will introduce the concept of FCMs. Linguistic FCMs are the inspiration for the thesis' cognitive engine component. The cognitive engine is also detailed in chapter 4. Chapter 5 will focus on the application of the research architecture to three test cases 1) transport dispatching application, 2) home-based health monitoring system and 3) Novax's Accessible Pedestrian System (APS, Figure 1.5) as well as an evaluation of the new model. Chapter 5 will also include sub-sections on learning and power consumption and optimization. After Chapter 6, where the inclusion of a complex social system is considered, Conclusions and Future work are presented.

Chapter 2.

Background

This thesis research is highly specialized and background materials on the overlapping fields of the research - embedded systems, firmware architecture, context-awareness, adaptability and fuzzy cognitive maps are briefly introduced below. Selected material in this chapter has been excerpted from the author's research published in Jaggernauth et al. [1].

2.1. Embedded systems

Embedded systems are dedicated-purpose computing systems as opposed to multi-purpose computing systems (personal computers, workstations). Embedded systems are widely used in a variety of but not limited to consumer (toys, smart phones, tablets, music players, fitness/health-care solutions) (Figure 2.1), automotive (safety-critical systems) and industrial applications (plant control).

Figure 2.1. Toy laptop with embedded system

Technologies used in embedded systems include micro-processors, microcontrollers, digital signal processors, complex programmable logical devices, application-specific integrated circuits or field programmable gate arrays. Embedded systems interface with their environment using sensors and actuators.

2.1.1. Challenges in Embedded Software Design

Some important considerations in embedded software design include (a) the complexity of the hardware solution (overly complex hardware results in the system becoming too expensive while marginal hardware choices yield a system which is not powerful enough to meet the computing requirements), (b) minimizing power-consumption to either save on battery life in battery powered applications or reduce heat dissipation in non-battery powered applications. (Reducing power consumption can be achieved by slowing down the non-critical sections of the code. Additional solutions are presented in Chapter 8), (c) designing for upgradability in new product generations and (d) reliability which is key in safety critical applications and reduces expensive and impractical (e.g. the deployed system may not have a keyboard or monitor) in-field debugging.

2.1.2. Introduction to Middleware

Middleware enables applications (whether it be firmware, client/server or desktop) by filling in the functionality gap between the application, OS, network and hardware layers. These tasks typically include: complex high-level sensing tasks, communicating tasks to the network, data storage and reporting, abstractions mechanisms to ensure heterogeneity. From a programming approach, middleware can be typically classified as: virtual machines, databases, event based, application-driven, component-based, tuple spaces, tuple channels and mobile agents

The context-aware model presented in this thesis can be implemented as hybrid event-based, application driven middleware. (In our proof-of-concept examples, the thesis research model is the layer above the application level as seen in Figure 1.2) Event based middleware generates an event based notification to the interested

application on the basis of an event detection or pattern of event detection (compound events). Event based middleware is complex but a simplified publish-subscribe solution (as designed with nesC running on TinyOS) alleviates this disadvantage. The application driven approach allow developers to perform network fine-tuning based on application requirements thus improving parameters like QoS. The downside to this approach is crafting specialized and not general purpose middleware. Application driven middleware are described in greater detail in [5],[7],[8].

2.2. Firmware architecture

Firmware architecture provides a framework that supports firmware modularization [9]. The benefits for firmware modularization are well documented and include software reliability, faster development time and debugging, a means to achieve and manage complexity, improved testability and portability. Figure 2.2 shows the firmware architecture for the IMS (integrated multi-sensor) detailed in Chapter 3.

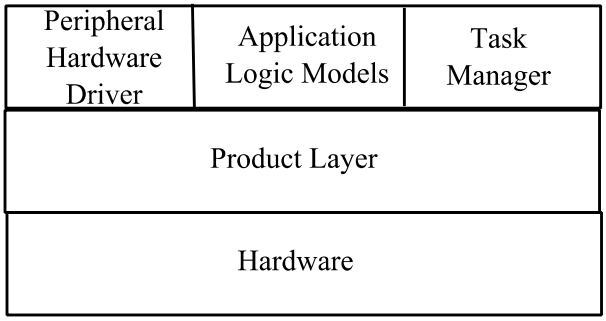


Figure 2.2. IMS Firmware Architecture

Firmware modularity allows for separation of concerns which is a key to context-aware systems [10]. There is a downside however in that firmware modularization increases overall code size due to encapsulation and layers of function calls. Excessive encapsulation could also result in increased code latency and problems with scalability. We discuss this latency for the thesis research model in Chapter 5.

2.3. Context-Awareness

For this research, context in embedded systems firmware development is defined as changeable and characterizing information such as sensor data (IR - infrared, GPS, accelerometer) or profile attributes (user, vehicle, device, etc.). This definition is adapted from Dey's [6] interpretation of context (information used in the situational characterization of an entity, including explicitly provided user information) as well as Mayrhofer's [6] definition of aspects of context - geographical, physical, organisational, social, emotional, user, task, action, technological, time attributes.

Pantsar-Syvaniemi et al. [12] outline the classifications for context modeling - key value models, markup scheme models, graphical models, object-oriented models, logic based models and ontology based models. The thesis research model is object-oriented.

In firmware development, middleware is an extensively researched area. Recent developments in the field of context-awareness for middleware includes: CASS (Context-Aware Substructure) a server based middle-ware used with mobile devices, C-CAST a context-awareness system using a consumer-provider broker model, MidSen a multi-application bridge for wireless sensor networks, WiSeKit enables dynamic behaviour in wireless sensor networks using adaptation and reconfiguration, COPAL defines a domain specific language (COPAL DSL) for context-provisioning plans and providing automatic code generation and a macro language (COPAL ML).

Additional context-aware modelling research includes the use of ContextUML (Unified Modelling Language) by Prezerakos, auto-generation of context-aware AmI models by Serral, and the work done by Segarra et al, using multilevel model based on observing, deciding, planning and executing [13]. Pantsar-Syvniemi in reference [12] introduces a micro-architecture that performs context-monitoring and context reasoning and context adaptation via a semantic database. Gamez in reference [13] describes a middleware solution that performs context acquisition, context storage, context analysis using predefined plans to achieve adaptability.

Dey presents a toolkit and framework (Figure 2.3) for the rapid prototyping of context-aware applications featuring (a) widget (separates concerns by hiding the

hardware functionality), (b) interpreter (raises the level of abstraction, manages the interaction and provides reusable building blocks), (c) discoverer (a registry of existing capabilities including which features are currently available for use), (d) service (executes actions) and (e) aggregator (collects related context information) components. The thesis research model features context collectors, context controllers, a context-co-ordinator and a cognitive engine [6].

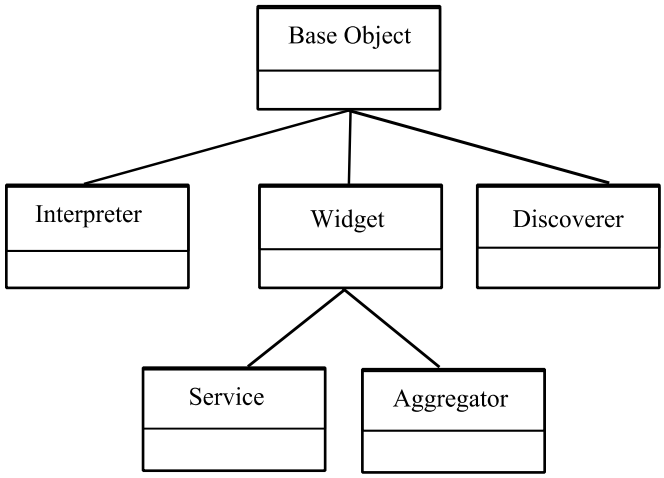


Figure 2.3. Dey's context-aware framework

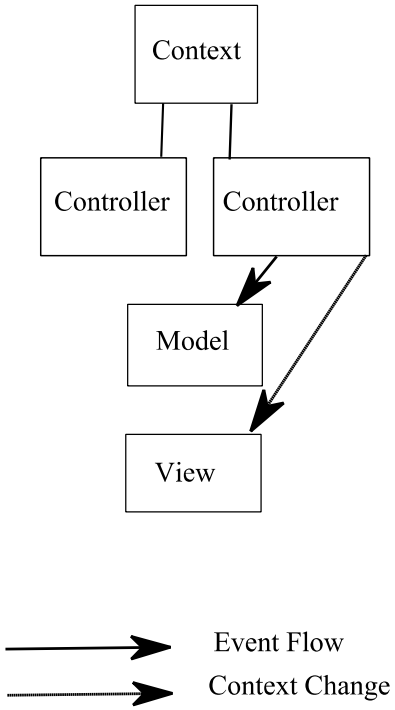


Figure 2.4. Rehman's context-aware architecture

A classic firmware architecture is the MVC of which there are many variants [63]. In the MVC architecture the model is responsible for data, logic and rules management, the view is for output representation and the controller converts accepted input for model/view processing. MVC has typically been used for user interface implementation.

Rehman's architecture for interactive context-aware applications (Figure 2.4) is based on this Model View Controller architecture where each context has two controllers (on/off context) and individual model and view components. When the defining event is presented to the appropriate context, the corresponding controller activates itself. The thesis research model assumes an underlying MVC application.

2.4. Adaptability

Inverardi et al. [10] propose a future for software in adaptability and dependability. They define adaptability as system changes according to changes in context and in terms of the four W's - why are there changes, what remains unchanged, when do the changes

occur and who manages these changes. Three examples are selected which demonstrate adaptability through topological changes and changes in interaction behaviour. Synthesis (automatically building reliable connectors), Graph Grammer (topological evolution) and ArchJava (topological and behaviour evolution) [14]. Gamez in reference [13] demonstrate adaptability in their work by showing when a context change is detected, a unique plan is selected and system reconfiguration occurs (e.g. sensor deactivation, reducing monitoring frequency or the installation of new services). In the thesis research model adaptability is achieved using both topological change (by adding new concepts to the FCM) and changes in interaction behavior (by modifying the values of the weights (FCM causal links)).

2.5. Fuzzy cognitive maps

Fuzzy Cognitive Maps (FCM), Figure 1.1, model variables or concepts that are interconnected by causal relationships and are figuratively represented by a signed directed graph with feedback. A causal relationship is defined where a relative change in one concept causes a relative change in a corresponding concept. FCMs have fuzzy logic and neural network components. The fuzzy logic element specifies degrees of causality and the neural networks component describes an artificial neuron (concept) processing where single or multiple inputs are transformed into a corresponding output value.

Fuzzy logic is a range of values between 0 and 1, unlike binary logic which can either be 0 or 1. Fuzzy logic is defined by a) set of membership functions for each input and output and b) rules which are applied to the membership function to yield a "crisp" output. Fuzzy logic is useful in capturing human knowledge/experience or modelling systems that are not easily mathematically expressed [2].

Fuzzy Cognitive Modelling is used for enabling dynamic adaptability because it allows for the consideration of a large number of complex inter-relationships in addition to being flexible and responsive to factor changes. FCMs are also an effective tool for modelling complex social systems (e.g. for decision making) as well as for applications in sciences and engineering. Fuzzy cognitive maps have been recently used in modelling lean manufacturing [2], collaborative planning, forecasting and replenishment approach

[15] affordance based planning [16] and medical decision making [17], [18], [19]. The FCM application to the thesis research is presented in greater detail in Chapter 5.

2.6. State-of-the-art comparison

In summary, the thesis research differs from the previously presented works because of the focus on an adaptability solution for memory-constrained, low power, single core embedded processors. The previous works focused on server, desktop computer platforms [10] or multimedia, multi-core embedded processors (e.g. mobile phones) [13]. Our model is specifically for embedded software (i.e. firmware running on electronic devices) whereas the previous works focused either on a semantic database, a multi-application bridge, domain specific language or UML language [10], [13], [14], [16]. Our cognitive engine is based on topological change using FCMs as opposed to topological change using petri-nets, graphs or architecture description language as presented in Inverardi [14].

Chapter 3.

Model development - Context collector and context controller components

3.1. Introduction

This chapter will describe our micro-architecture components and data-flow development specifically the context collectors and context controllers. This chapter will also describe operational and test firmware development for a wireless, physiological, multi-sensor. Selected material in this chapter has been excerpted from the author's research published in Jaggernauth et al. [1],[3],[4].

Figure 3.1 shows the new context layer which lies above the application layer (usually a classic MVC [63] architecture standard to many legacy embedded applications) technology in all applications).

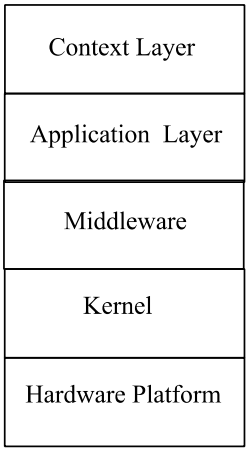


Figure 3.1. Multi-layer context-aware model

3.2. Materials and Methods.

3.2.1. Components

Figure 3.2 shows our micro-architecture within the context-aware layer. The components of the micro-architecture are context collectors, context controllers, a context-coordinator engine and a cognitive engine.

The criteria followed for determining the components in the micro-architecture as well as making the architecture adaptable is taken from Parnas research on rules for module decomposition [52] and software design principles for ease of expansion and contraction [53]. For our basic framework, Parnas stipulates that subsets need to be identified (in our system, contexts) and the idea of information hiding of “secrets” using modules interfaces and definitions (context collectors, context controllers). Parnas defines “secrets” as design decisions that are likely to change which are located in specialized components (the “on-the-fly” adaptability by expansion and contraction of the cognitive map in the cognitive engine). Parnas also identifies the “uses” structure (requires the presence of) which is reflected in the “requires” functionality of the cognitive map in our system (“requires” functionality “wires” the cognitive map concepts/contexts together to form the map).

Context Collector - The collectors can either interface directly with the sensor device drivers for reading the sensor data or through a hardware abstraction layer. The context collectors determine a change in context and alert the context coordinator (Figure 3.2 and Figure 3.3). Context collectors are also responsible for local storage. Storage could potentially reduce expensive energy transmission to improve battery life e.g. by limiting the need for real-time status updates to the server in low battery scenarios. The context collector is separate from the database structure (module) of the model where the general purpose variables are stored.

Context Controller - The context controller is responsible for integrating all the related contexts to produce a meaningful control sequence.

Context Coordinator - The context coordinator engine manages inter-context component and inter-layer messaging. The context coordinator is also responsible for scheduling of tasks.

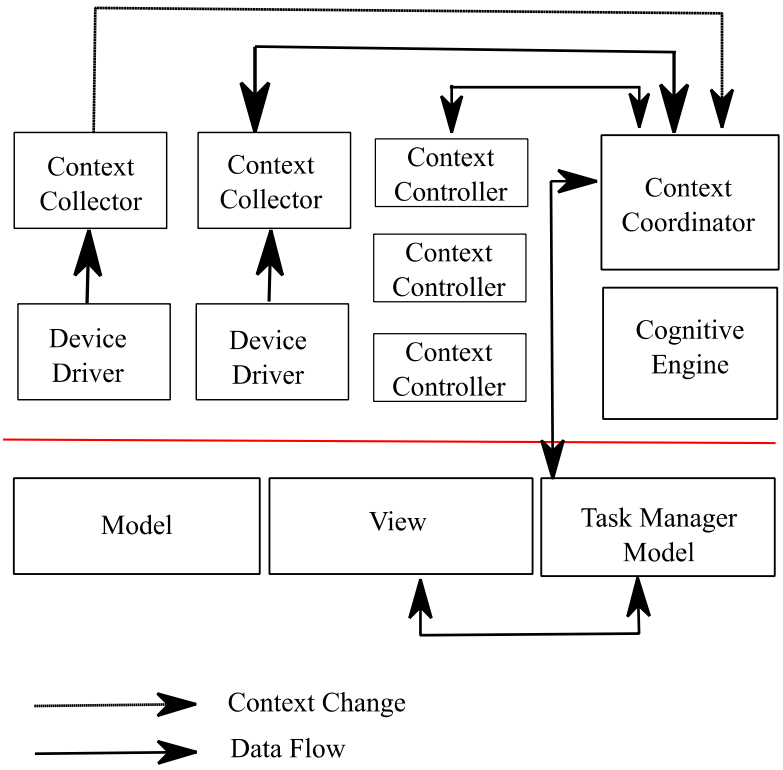


Figure 3.2. Proposed Architecture

MVC - The models in Figure 3.2 could be physical objects or abstract data structures that are used by the applications. The view model could be a line based display, a graphics display, no display but with output to a hyper-terminal application. The controllers could be responsible for interfacing with hardware peripherals e.g. printers.

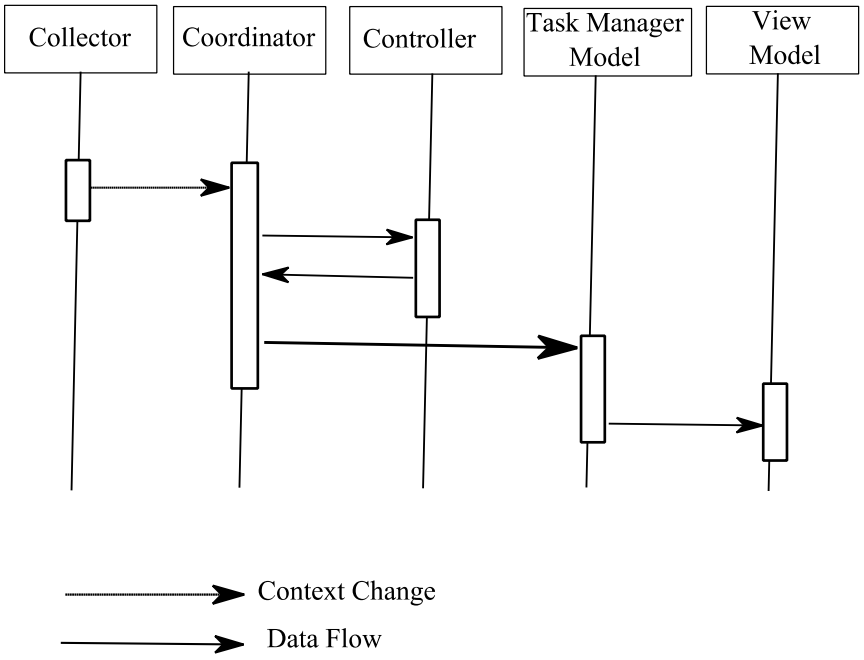


Figure 3.3. Data-flow sequence diagram for architecture

Cognitive Engine - The cognitive engine enables dynamic adaptability and is described in detail in a later section

3.2.2. Dataflow

Data flow (Figure 3.3) occurs via events (e.g. context changes) and queries. Based on sensor data received (e.g. via interrupts) a determination is made as to whether a context change has occurred. This context change is stored together with other useful environmental variables and the context coordinator engine is notified. At the context coordinator engine level, a task or control variable related to the context change is activated. Alternatively, an event is triggered when a timer expires in the context-engine and queries can be sent to context controllers or context collectors. There are two types of events - implicit and explicit. An implicit event occurs when a context-change occurs automatically e.g. a car driving to a new area. An explicit event occurs when a context-change occurs deliberately e.g. a new user logs on or vehicle attributes are changed or a shift sign-off occurs.

3.3. Firmware for IMS Sensor

This section details optimized and practical firmware development for a novel, wireless, multi-sensor, system for body activity and vitals monitoring, which improves user comfort and enables use on active mobile subjects with improved battery life. The sensor is one of the case studies in our context-aware research and this section outlines its proof-of-concept implementation.

The development of the device drivers for reading temperature, accelerometer and heart-rate is the groundwork that forms the basis of the context collectors in our thesis research model.

A firmware architecture is also introduced for the sensor application. Our context-aware layer would reside above the application layer. Selected material in this section has been excerpted from the author's research published in Jaggernauth et al. [4].

3.3.1. Introduction

Physiological information for vitals and activity monitoring is invaluable in a variety of personal and emergency healthcare applications. In response to today's health care devices' challenges of wired connections, device rigidity and behavioural modifications on physiological signals [20], a novel multi-sensor wireless node was developed at CiBER (Centre for Integrative Bio-Engineering Research) 55mm by 15mm, 3mm thick, to be attached to the patient's chest, for measuring body motion, activity-intensity, tilt, respiration, cardiac vibration, cardiac potential (ECG), heart-rate, body surface temperature, see Figure 3.4 [21]. CiBER research work includes the development of nano-optics technology, flexible electronics, wireless sensor platforms, optical devices and optical signal processing technology.

This section details practical and optimized biomedical firmware development for data acquisition and processing for the device which has commercially viable potential, (as opposed to a proof-of-concept implementation [22]). The firmware architecture design was driven by unique flexible nature of the device which determined the selected electronic components and layout. In a wireless sensor environment conserving power is

a crucial requirement [23], hardware and firmware optimizations employed, choice of hardware (low-power micro-controller with integrated (vs. peripheral) RF), hardware window comparator functionality (vs. software filtering), minimum number of executed firmware instructions, sleep modes, optimized, on-device, ECG source code, use of flattened ring buffers instead of a linked list for data storage, channel sampling using frequencies in fixed integer multiples etc. Additionally, correct scheduling of computationally intensive operations is implemented to prevent high priority tasks from getting delayed or starved (e.g. incremental heart rate computation). Events are used to maximize the overall efficiency of the code.

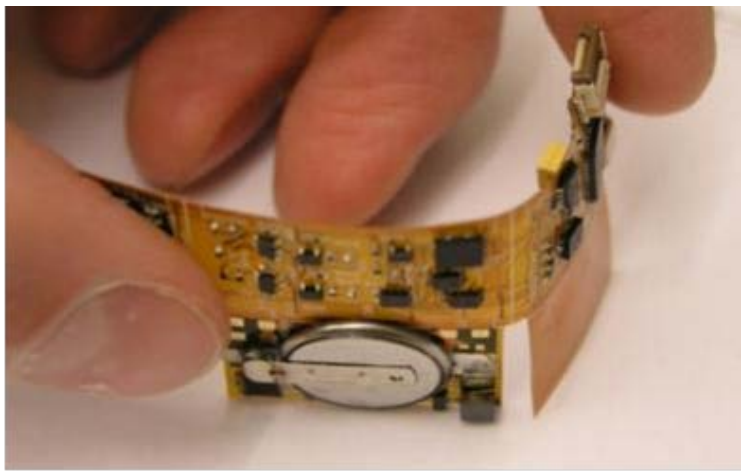


Figure 3.4. Unfolded assembled device

3.3.2. System Setup

The system is composed of three main elements – the wireless multi-sensor node, (Figure 3.4), for attachment to a patient in order to acquire vital data, a mobile phone as a data display and control unit, and radio interface bridge attached to the phone, (Bluetooth / Zigbee interface). The data gathered by the wearable node is transmitted (using Zigbee protocol) to the bridge, which then communicates with the phone using the Bluetooth standard. Control messages are generated by the phone and sent to the node through the bridge.

3.3.3. Firmware Architecture Design of the Multi-Sensor Node

The firmware of the system is designed to be implemented on the TI-CC2430 integrated circuit (IC) 2.4 GHz RF ZigBee micro-controller. The functionality of the multi-sensor node includes sampling, post processing and streaming multiple physiological signals to the ZBridge node. Figure 3.5 below demonstrates the block diagram for the architecture of the multi-sensor endpoint application. To briefly summarize – the sensor node application has three functional layers' categories - driver library (e.g. shared selectable signal conditioning and acquisition), management library (e.g. buffers, transmission modes, messages) and the event processing loop (e.g. scheduling). Sample firmware sources (for the switch and adc device drivers) are found in Appendix A and B.

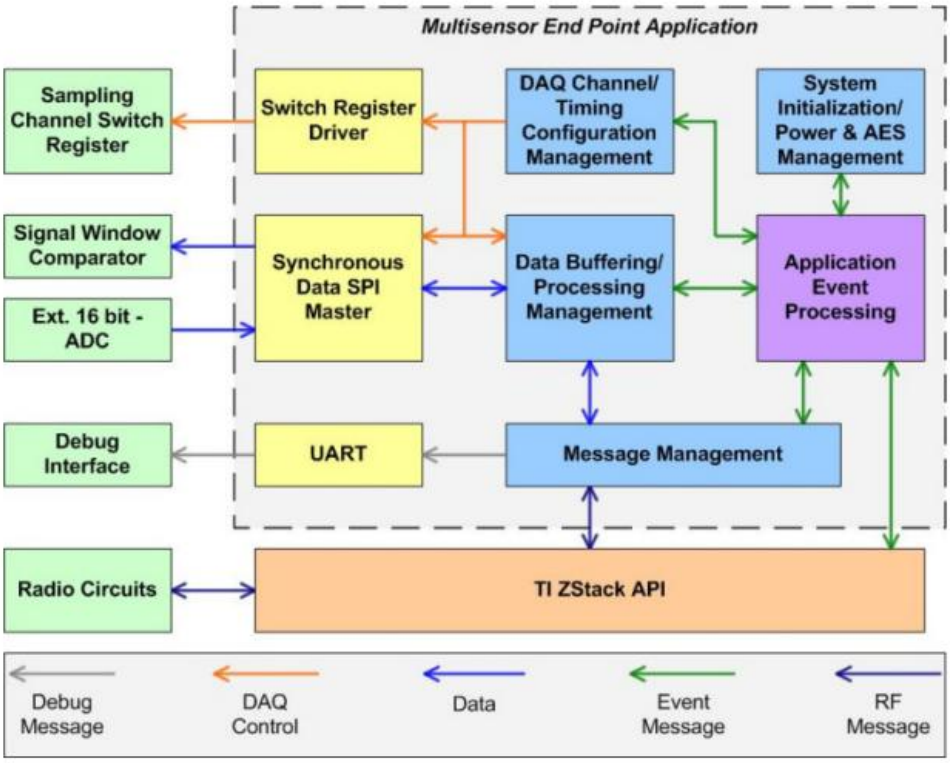


Figure 3.5. Multi-Sensor Node Endpoint Firmware Block Diagram

3.3.4. Signal Processing Algorithms

Efficient and compact signal processing algorithms are used to process ECG signals (QRS complex detection, band-pass filtering, differentiation and averaging) and detect heart rate within the node, prior to wireless data transmission, reducing energy-intensive energy transmissions [28].

3.3.5. Results and Discussion

Preliminary testing of the devices allowed for verification of the firmware driver for the individual data acquisition components including the ADC and the switching between devices. It was possible to obtain readings for the temperature, accelerometer and to capture the ECG signal and to verify heart rate processing code. A/D converted readings obtained from the firmware for temperature and positioning for the unfolded and upright sensor are posted in Table 3.1 below. The accelerometer sensitivity is 0.66 V/g.

Challenges were encountered because the flexible substrate was susceptible to breakage primarily where there was hand soldering work and during the testing phase, resulting in some of the circuit paths needing to be manually bypassed in order to get readings. This breakage made it impossible to verify the device on a patient and thus perform clinical tests.

Experimental tests were conducted to determine the optimal ZBridge power consumption and radio range ratio. It was determined that the transmission range decrease with decreasing transmission power, is much more pronounced than the decrease in power consumption. Based on these results, a much higher than required level is selected to allow for increased user comfort and transmission reliability.

Table 3.1. Sensor Readings

Position	Thermistor	X	Y	Z
Flat	40 mV	1.63 V	1.63 V	2.00 V
Lengthwise	40 mV	1.30 V	1.49 V	1.70 V
Widthwise	40 mV	1.67 V	1.86 V	1.83 V

We were able to develop firmware and test a practical novel multi-sensor on flexible substrate, which has potential to be a marketable product.

As will be discussed in later chapters the context collectors of the thesis model are supplied by the ECG (heart rate context collection), thermistor (temperature context collection), accelerometer (activity context collection) of the physiological sensors. Some of the optimization techniques presented in this section will also be applied to the cognitive engine development.

Future work includes using assembly language to code processor intensive functions for improved optimizations, further analysis with the window comparator aspect of the device, and to verify a fully operational device (in future device generations).

3.4. Automated Testing

3.4.1. Introduction

This section introduces a test firmware architecture for the previous' section physiological sensor. The test firmware is responsible for establishing the device's basic correct hardware operation. From the application to the thesis research standpoint, the concept of querying inputs is a precursor to reading context collectors and the actuation of outputs dove-tails with the idea of activating context collectors. This section also provides another firmware architecture example which is modular and extensible. The context-layer would reside above of this application layer. Selected material in this section has been excerpted from the author’s research published in Jaggernauth et al. [3].

This section introduces an ATE (automatic test equipment) component-styled, test firmware architecture for the device. It ensures reliability of the hardware, such that the

end application can be evaluated without the firmware developer needing to interface with the sensor at an electronic component and connection level [29]. Additionally, this section proposes a manufacturing test mode for in-field trouble-shooting at the end user site. The section will introduce the system setup, application firmware architecture of the multi-sensor node and Zigbee-Bluetooth bridge (ZBridge). It will include detailed hardware and firmware optimizations used (as opposed to summarized in [4]), followed by preliminary results which necessitated the development of the test firmware discussed in the subsequent sections. The IEEE has proposed the ISO/IEEE 11073 family of standards for commercial medical device communication compliance, however the Bluetooth-Zigbee intra-device communication presented in this section discusses a proprietary protocol for application specific operation primarily for research purposes.

At the hardware manufacturer, before encapsulation (packaging), the device is initially subjected to testing requirements including post-assembly verification of the hardware components, testing of modules, reliance on KGO (known-good-die) and hierarchical test flows. Specifically this involves testing for malfunctioned chips and interconnections and testing the subsystem modules sensors (performance characteristics vibration sensor tests, electrode conductance and conductance frequency), signal conditioning (filter and amplification characteristics), data acquisition and processing / radio (digitization, range and manufacturer specification), powering (device power consumption vs. radio range) as well as detailed cost analysis for repairing or discarding [8].

3.4.2. ATE Firmware Architecture

In response to difficulties encountered with breakage during post-encapsulation and transit to application development site as well as suggested future works [8], automated test equipment (ATE) firmware was developed to run diagnostic test routines using the sensor’s embedded micro-controller.

ATE firmware is critical because of limited real-estate available on flexible devices for populating test hardware for BIT (built-in-tests) and BOM (built-on-tests) due to the stringent power requirements that need to be met for low-power wireless sensor devices.

The multi-sensor has built in cables (micro-connectors patterned directly on the polymer substrate) to access test bus groups allowing access to the various device subsystems. The test points are described in Tables 3.2 and 3.3 below. Following verification, the micro-connectors can be detached using cutting tools.

Table 3.2. Sensor Test Points I

Sensors	Filters	Switch A
Accel x (O)	High Pass (I)	SW-A1 (O)
Accel-y (O)	High Pass (O)	SW-A2 (O)
Accel-z (O)	Low Pass (I)	SW-A3 (O)
Temp (O)	Low Pass (O)	SW-A4 (O)
Elec-1 (O)	Band Pass 1 (I)	SW-A5 (O)
Elec-2 (O)	Band Pass 1 (O)	SW-A6 (O)
Elec-R (O)	Band Pass 2 (I)	SW-A7 (O)
Elec-R (O)	Band Pass 2 (O)	SW-A8 (O)
	Comparator in (I)	
	Comparator out (O)	

Table 3.3. Sensor Test Points II

Switch B	MCU	Power
SW-B1 (O)	DATA (I)	BATT(+) (O)
SW-B2 (O)	CLOCK (I)	BATT(-) (O)
SW-B3 (O)	RST (I)	GND (O)
SW-B4 (O)	RXD (I)	VCC (O)
SW-B5 (O)	TXD (O)	BATT CHG (I)
SW-B6 (O)	Vcc (O)	
SW-B7 (O)	GND (O)	
SW-B8 (O)	SCLK (O)	
	MISO (I)	
	SS (O)	

Automated test firmware can be designed to stand-alone or work with a manufacturer supplied test jig with LabVIEW interface or manual probing (including proper ESD handling procedures, visual inspection) with serial terminal interface.

The ATE command subset is presented in Table 3.4 below.

UML

UML is an object oriented language for describing specifications. UML describes classes and objects elements and graphically represents association, aggregation, composition and generalization relationships. Three types of events specified in UML are signal, call and time-out events. UML also defines state-machine states and transition. Timing is expressed as sequence and collaboration diagrams [55].

Table 3.4. Command Result

Switch B	MCU
VER	Firmware Version
WRAX, WRBY	Turn on Switch A, B, X,Y 1 digit hexadecimal mask
RDADC	Read the ADC value
WRCLK	Output Clock Signal
WRSS	Output Chip Select
RADIO	Radio Detected Check

The actual test procedure followed was manual probing and signal measurement, using the oscilloscope and digital multi-meter which used a terminal program to issue ATE commands and capture data. The signal path was followed as per the multi-sensor schematic and each of the components was probed to determine compliance with operational requirements.

Verification of the accelerometer operation involved turning on the switch enables (accelerometer, filter) and comparing the measured ADC input and digitized SPI-read

output with the accelerometer output for all 3 axes. The same procedure was followed for ECG electrodes as well as for the thermistor.

Depending on the device, a different filter combination was enabled. Using the above procedure, it was possible to adequately test the device. Issues arose with repeated manual probing during connectivity tests which resulted in the shifting of hand-soldered traces as well as the clouding over of the device's protective conformal coating.

3.4.3. Manufacturing Test Mode

The new manufacturing test mode module introduces additional automated testing into the final application firmware to ensure that all components are functioning or to tag components that need to be repaired.

The manufacturing mode is also key to in-field troubleshooting. The manufacturing mode is triggered by sending <MANF> command over the device serial link and exited by sending <IMS>. A Test Menu (Figure 3.6) is output to the Tera Term display over the serial link. During testing an option was selected to run the corresponding self-test to ensure that the particular device component was still operational as well to provide actual read data. Figure 3.7 shows the results.

3.4.4. Results and Discussion

In tandem with the testing and verification of the hardware functionality of the first batch of devices, a second batch of devices was manufactured and delivered. The second batch of the devices yielded better and more consistent and complete results. Consequently, it was possible to conduct tests on patients and sufficient quality ECG signals were captured with clear indication of QRS waves, as well as sufficiently accurate and rapidly responsive heart rate calculation. Body activity signals (standing, sitting, lying left, lying supine, lying right and lying prone) were demonstrated along with synchronous ECG signals.

```
PLEASE SELECT COMPONENT FOR TESTING:

   1. RADIO

   2. UART

   3. BATTERY

   4. TEPERATURE

   5. ACCELEROMENTER

   6. ELECTRODES
```

Figure 3.6. Test Menu for Manufacturing

```
OPTION 1 SELECTED: RADIO OK

OPTION 2 SELECTED: TEST STRING PRINTED

OPTION 3 SELECTED: TEMPERATURE RANGE OK

OPTION 4 SELECTED: X,Y,Z, RANGE OK
```

Figure 3.7. Manufacturing Test Mode Results

The ATE firmware is an invaluable tool for enabling the use of fragile flexible devices. Using the ATE firmware it was possible to test the device's components and connections. The ATE firmware could also be used post assembly at the hardware manufacturer's site in addition to the current usage which is post encapsulation at the firmware development site. The manufacturing mode adds another layer of functionality

to the multi-sensor firmware, approaching a complete firmware solution for driving the novel device.

The use of the ATE command subset presented in this section parallels the context collectors command subset presented in later sections.

Future work includes the next phase of system testing as well as updating the test firmware to support the next generation of devices.

Chapter 4.

Model development – FCM and Cognitive engine component

4.1. Introduction

This section describes FCMs and cognitive engine component development. The key feature of the proposed thesis model is to enable firmware adaptability by introducing a cognitive engine component with reconfigurable map logic inspired by FCMs.

No previous work has been done on the use of FCMs with linguistic weights for enabling dynamic, low memory constrained, firmware adaptability. Selected material in this chapter has been excerpted from the author's research published in Jaggernauth et al. [1].

FCMs enable adaptable firmware logic by adjusting the strengths of links between concepts (e.g. the context collector variables change value) or updating the topology of the map by adding or removing concepts (remote update via a supervisor or system operator). Fuzzy cognitive maps (FCM), Figure 1.1, model variables or concepts that are interconnected by causal relationships. A causal relationship is defined where a relative change in one concept causes a relative change in a corresponding concept. There are numeric and linguistic FCMs. The numeric FCM can be characterized as either bivalent, trivalent or logistic signal depending on its transfer function. Finite State Machines are bivalent and trivalent Fuzzy Cognitive Maps.

The type of Fuzzy Cognitive Map used in this research is the Kardaras FCM [30]. Kardaras FCM was selected because it uses linguistic (as opposed to numeric) weights in decision analysis. Firmware sources for linguistic and numeric FCMs are found in Appendix C and Appendix D. For completeness, 3 case-studies on numeric FCMs are presented in section 5.4, Appendix G and Appendix H.

4.2. Materials and Methods

4.2.1. Kardaras FCM

In this section the Kardaras FCM functionality is described with respect to assignment of weights, causality and decision analysis.

Assignment of Weights - In the Kardaras Cognitive Map [30] model 4 weights are used

Undefined < Weak < Moderate < Strong

Causality - There are 4 causality relationships in Kardaras FCMS - affects, requires, multiples and stops. Only affects, requires (described below) and stops (implied) are relevant to the thesis research model. In Figure 4.1 Variable V_1 affects Variable V_2 and Variable V_2 requires Variable V_1. Generally, if X affects Y then an increase/decrease X results in an increase/decrease Y. If X requires Y then increase/decrease X does not result in an increase/decrease Y AND an increase/decrease Y requires an increase/decrease X.

Decision Analysis - In the Kardaras model, decision analysis (Equations 1-10) occurs by firstly identifying the causal path I, followed by determining the polarity of the path, the degree of belief and the most believed effect. The polarity of the path S is (+) if the number of negative polarity relationships is even or zero.

The degree of belief of the path Φ (phi) is determined by the minimum of the fuzzy linguistic weights along the path. The most believed effect Δ (delta) is the path which yields the maximum degree of belief.

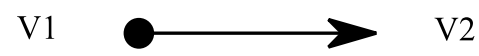


Figure 4.1. Affects and requires causality

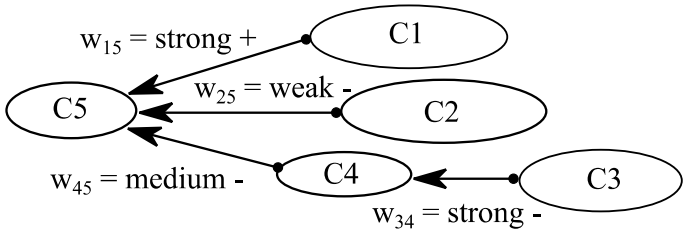


Figure 4.2. FCM with linguistic weights

Using the example presented in Figure 4.2 we can identify the different causal paths to get to C_5 - I_1, I_2, I_3. Equations 1-10 also show the results of evaluating polarity S, degree of belief Φ and most believed effect Δ on I_1, I_2, I_3.

$$I_1 = (C_1) \quad (1)$$

$$I_2 = (C_2) \quad (2)$$

$$I_3 = (C_4, C_3) \quad (3)$$

$$S_1 = \{\text{strong } +\} = + \quad (4)$$

$$S_2 = \{\text{weak } -\} = - \quad (5)$$

$$S_3 = \{\text{medium } -, \text{strong } -\} = + \quad (6)$$

$$\Phi_1 = \text{strong} \quad (7)$$

$$\Phi_2 = \text{weak} \quad (8)$$

$$\Phi_3 = \text{medium} \quad (9)$$

$$\Delta = I_1 \quad (10)$$

4.3. Results and Discussion

4.3.1. Application to Development of the Cognitive Engine

In our research we consider a binary FCM with the weights of 0 and 1 instead of weak, medium and strong. Φ reduces to the logical AND operation so min (A, B, C) translates to (A AND B AND C).

Likewise, Δ reduces to the logical OR operation so max (A, B, C) translates to (A OR B OR C).

For logic map evaluation purposes, Φ and Δ can be used to evaluate multiple variable if/else/else if statements.

Therefore:

if (A AND B) {do x}

else if (A AND B') {do y}

else if (A' AND B) {do z}

can be expressed as

max (min (A, B), min (A, B'), min(A'B))

or

$$\Delta (\Phi_1, \Phi_2, \Phi_3)$$

In this form, the basic evaluation building block of our logic map, which forms the basis of the cognitive engine, is realized.

4.4. Conclusion of the Work Presented in this Chapter

In this chapter we presented background material on FCMs: the different types of FCMs available and introduced the Kardaras FCM with its weights, causality and decision analysis feature set. We showed the application to the thesis research where we use FCM to enable dynamic adaptability of the cognitive engine by using the delta and phi functionality to code if and else statements.

Chapter 5.

Proof of Concept Implementation and Performance Profiling

5.1. Introduction

This chapter will preliminaril5y consider the application of the thesis architecture to three test cases 1) transport dispatching application, Figure 5.1 and Figure 5.2, 2) home-based health monitoring system, Figure 5.5 and Figure 5.6 and 3) Novax's Accessible Pedestrian System (APS), Figure 5.7. Sub-topics of learning and power consumption and optimization are briefly described. The cognitive engine operation is discussed in detail. Performance profiling using three software metrics - code complexity, latency and code size are outlined.

5.1.1. Transport Mobile Dispatching

A dispatching application running on a mobile dispatch terminal (embedded system) found in cabs is used for getting customers for taxi-car drivers. Taxi cabs can receive customers via street-hire, auction, reserve, closest cab and stand jobs. Taxis are equipped with GPS are capable of determining the zone that a driver is currently in. As the car moves from zone to zone it is eligible to bid on jobs from that zone, be it auction jobs (left over jobs), reserve jobs (future jobs) or closest cab jobs (current jobs and also dependent on how long the driver has been idle), stand jobs (dependent on proximity to the stand). Jobs are filtered based on driver (multi-lingual) and vehicle attributes (roof rack, lift equipped, minibus etc.), or driver state (active, suspended). The dispatching application architecture and sequence diagram are shown in Figure 5.1 and Figure 5.2. The hardware platform for this dispatching application is a low-cost terminal usually associated with a single-processor solution and monochromatic lcd display. Different customers will have different price-points. This is a solution for the low-end price-points e.g. customers in the developing world or for legacy applications.

Model and Controller - Model and Controller functionality could include interfacing with peripherals (camera, debit machine, printer, pager, or modem), system tools models (time, string formatting), and storage models (table data, application data, zone boundary data).

View - functionality would be window display (primary job display as well as secondary status and map display).

Context Collector - The sensor context is supplied by the GPS receiver. The virtual area context is the GPS geographically derived area. The application contexts (driver attributes, vehicle attributes, and status) provide software contexts from the server application. Examples of information collected for the status context include job id, taxi status, previous job number, GPS (x,y), stand tokens, number passengers, break count.

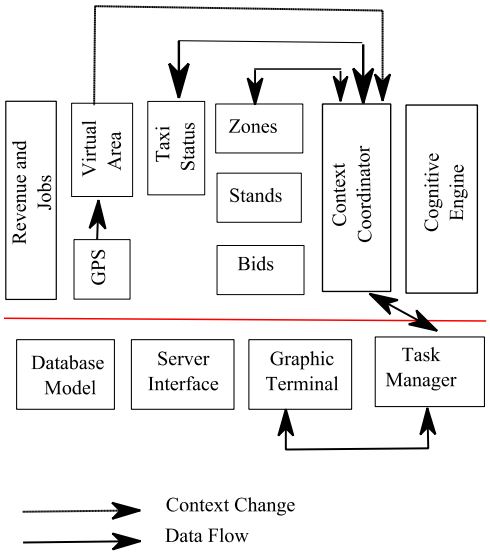


Figure 5.1. Dispatching application architecture

Context Controller - The context controller components for the dispatching application are - stands, areas, payment, trip, and time. Tasks performed by the area or stand context controllers are determining current area (shown in Figure 5.1), performing bookin operations (stands and zones), handling operational constraints (has the terminal been forced signed off, is there a bid in progress, is there a held auction job/ reserve job, emergency state, number of type of zonal jobs (e.g. soon to clear) allowed).

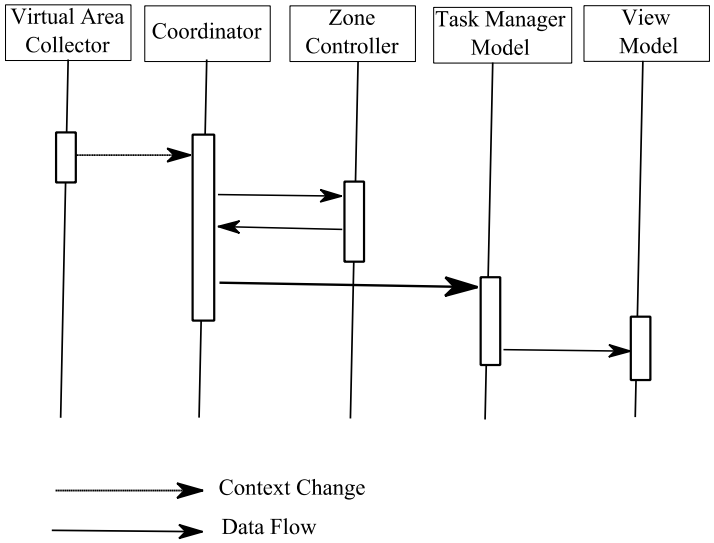


Figure 5.2. Dispatching application data-flow sequence diagram

Context Coordinator engine - Some of the decisions that the context engine perform are the determination of frequency of collection, constraints factors affecting collection e.g. - for status update related information, the effect of dormancy on status updates, co-ordination with the application layer as well as the contextual elements.

Dispatching Application and Latency

A simplified subset of the dispatching application outlined in the preceding section and Figures 5.1 and 5.2 was profiled. The details (procedure calls, messaging) are also described in Section 5.2.1. This subset was used to evaluate the latency introduced by the extra layers of the context collectors/context-coordinators/context controllers in the new architecture and compared against a standard MVC architecture implementation. The code was written in C and preliminarily verified on an x86 (DOS) platform. The source code is in Appendix H. The scenario profiled was the case where gps context data is used to evaluate the zone the vehicle is in currently. The sequence diagram for this test case in the new architecture is shown in Figure 5.4 and the sequence diagram for older MVC architecture is shown in Figure 5.2. The system model in the older architecture (Figure

5.4) is replaced by the context collector, context-coordinator and context controller models (Figure 5.2) in the new architecture.

There are several state-of the art tools available for latency model evaluation. The one chosen for this project was Rapita RVS (Rapid Verification Suite) [66]. Performance analysis of this thesis research model was confined to source code simulations as opposed to on-target verification. On target timing would be the same as the simulation because there was no operating system or multi-threaded application. Rapita's Rapitime works by instrumenting the source code during the preprocessor build stage in order to enable execution of performance analysis and generate a report on the subsequent trace data (Figure 5.3).

The types of performance analysis undertaken by Rapitime include:

- Worse Case Execution Time (W-ET)
- Maximum Execution Time (M-ET)
- High Water Mark Execution Time (H-ET)
- Execution Time Profiles

The Execution Time profiles are further divided into Self Execution Time (SelfET), Sub Routine Execution Time (SubET) and Overall Execution Time (OverET). Additional analysis is provided with respect to Test coverage, Loop bounds, Call Trees and Context Information.

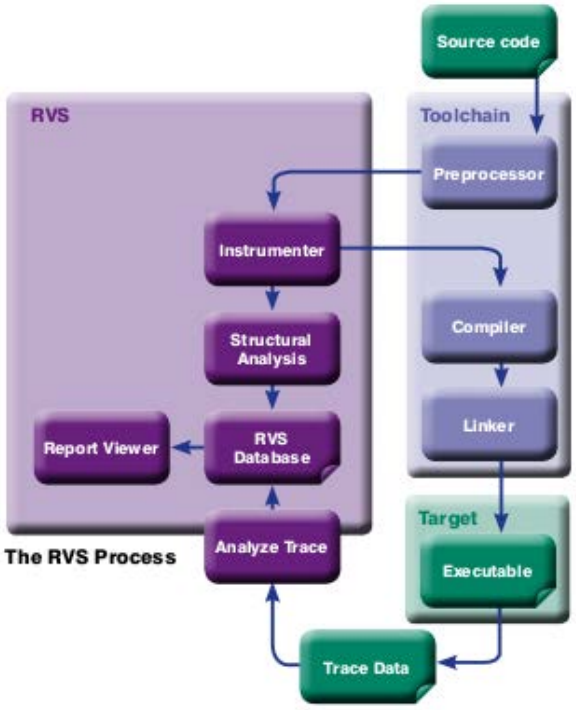


Figure 5.3. The RVS process, Rapita©

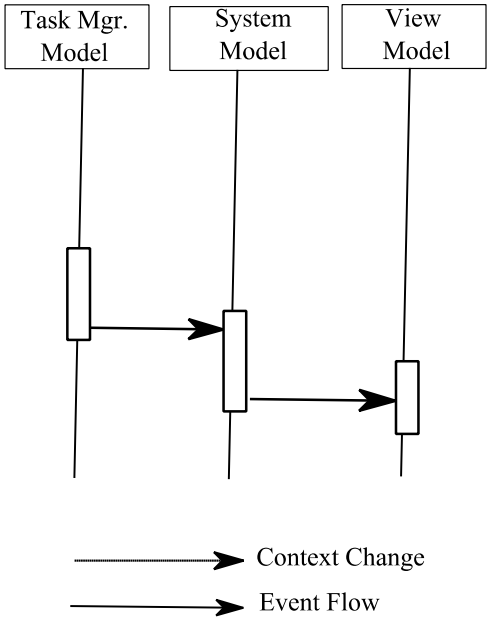


Figure 5.4. Standard MVC architecture

Using the Rapita tool, the code analysis results were simulated using an ARM processor engine. For Rapita the execution time is represented in terms of execution cycles.

5.1.2. Home Health Care

In this section a home health care application used to monitor the elderly is considered. There are many off-the-shelf products available for this purpose and this test-case may also apply to the IMS sensor described in Chapter 3.

The home health care solution may comprise a wearable, wirelessly enabled embedded system with sensors which are capable of measuring physiological data. The health-care application architecture and sequence diagram are shown in Figure 5.5 and Figure 5.6.

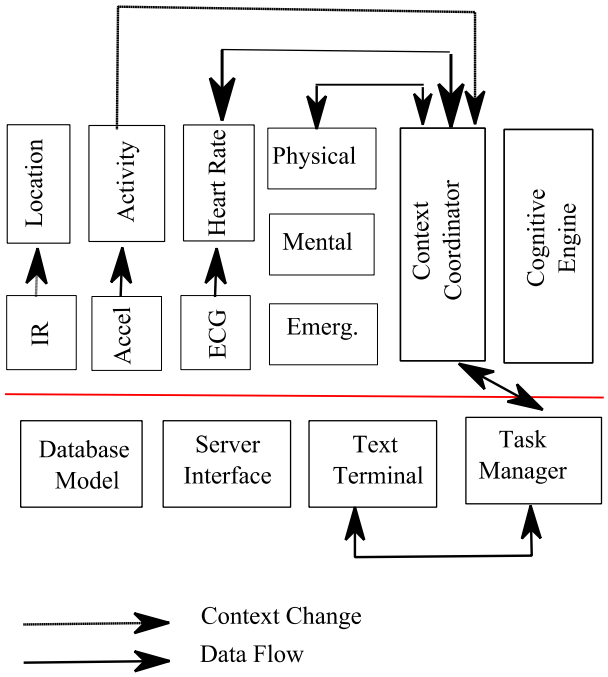


Figure 5.5. Health-care application architecture

Context Collector - The sensor contexts are supplied by the ECG (heart rate context collection), thermistor (temperature context collection), accelerometer (activity context collection), IR (position context collection).

Context Controller -The context controllers are responsible for e.g. generating alarms, logging events.

Cognitive Engine – Potential uses of the cognitive engine are described in Section 6.

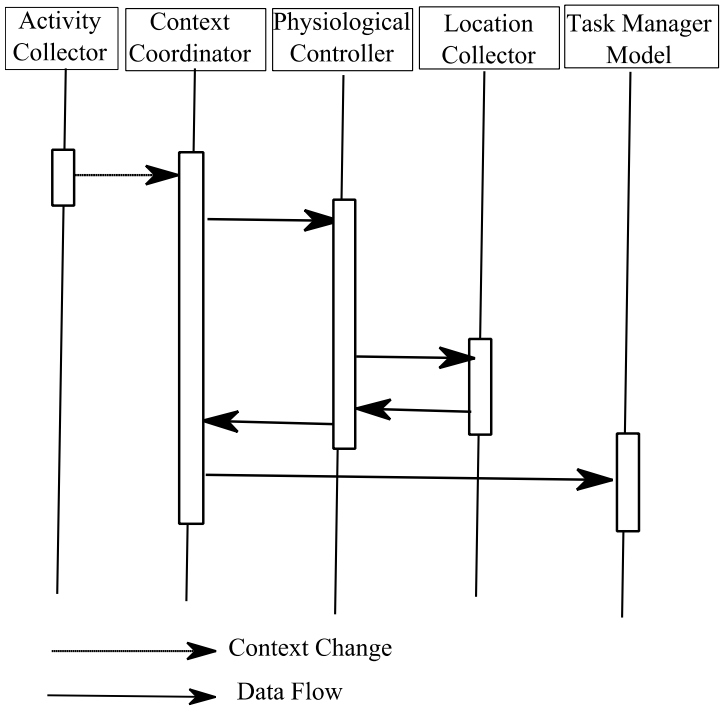


Figure 5.6. Health-care application data-flow sequence

5.1.3. Novax Accessible Pedestrian Systems

Novax produces accessible pedestrian systems (APS), Figure 5.7. A typical scenario is a pedestrian approaches an intersection and pushes the APS' button, the red led goes on and the button plays "wait to cross." The button receives the walk signal from the traffic controller and plays "walk sign is on." The button receives the flashing don't

walk (or ped clear) signal from the traffic controller and plays an audible countdown (10, 9, 8 down to 1). The APS hardware platform is a MCU 8-bit, 128 KB Flash embedded processor.

This aps firmware application has the following components:

Model - Defines the file manager, timers, configuration utility, logging, manufacturing modes.

Controller - Hardware abstraction layer for the button, uart, spi, led, flash, power line communications, vibe, audio, digital I/O.

Context collectors – Variables that track for button state, pedestrian state, audio conflict etc. Each variable is assigned a unique numeric identifier.

Context controllers - Command sequence for actuating the controllers directly at an application level. Some commands are listed in Table 5.1 below.

Figure 5.7. APS photo courtesy of Novax Industries Corp

Table 5.1. Command sequence for actuating controllers

Command	Response
N1 <n>	Insert n into log
N2	Display log in entirety
N3	Erase log
N4 <mask>	Set specified GPIO on
N5<mask>	Set specified GPIO off
N6 <n>	Turn off led n
N7 <n>	Turn on led
N8 <val>	Enable vibe to <val> level
N9 <val>	Disable vibe
N10 <n>	Play sound sequence n
N11	Stop sound sequence

FCM Applications to Research

In Section 4.3.1 we showed

max (min (A, B), min (A, B'), min(A'B))

or

Δ (ϕ1, ϕ2, ϕ3)

In this form, the basic evaluation building block of our logic map is realized. The format of our logic map is shown in Table 5.2 below. In the preliminary research implementation, the logic map is limited to 16 characters. We additionally introduce the concepts of:

Complement – A variable complement is achieved by using the negative value of the variable i.e. x' or (NOT x) is expressed numerically as –x.

Table 5.2. Logic Map Fields

Field Id	Data
1	Coverage map line number
2	Number of variable to be evaluated
3-4	Unique variable id
5	Timer
6	Momentary/Latch
7	Calculated phi for this path
8	Determined latch
9-16	Context controller command sequence

Latched – Once a path's ϕ has been evaluated to 1, 1 is stored in the Latch field of the FCM. Henceforth, this path is always evaluated as having a ϕ as 1 regardless of the state of the input variables. Regular operation is momentary operation where the path's ϕ reflects the real-time evaluation of the input variables.

Timers – If there are two aps buttons working in tandem (typical intersection configuration) it may be required to evaluate the respective aps cognitive engine's logic simultaneously or on odd rotation. This odd rotation would be achieved by setting the value of the timer field. E.g. if the timer field for APS y is set to 1, the cognitive engine logic would be evaluated on (clock_ticks mod 2) == 1, if the timer field for APS z is set to 0, the cognitive engine logic would be evaluated on (clock_ticks mod 2) == 0. A real-time example would be to have APS y play one sound and APS z play the subsequent sound e.g. APS y plays 1,3,5,7 of the audible count-down and APS z plays 2,4,6,8 of the audible countdown.

APS Cognitive Engine

The APS cognitive engine was written in C and preliminarily verified on a MCU 8-bit, 128 KB Flash embedded processor. The ϕ and Δ state diagrams are shown in Figures 5.8 and 5.9 respectively. The pseudo code for the cognitive engine is shown in Figure 5.10. The source code for the cognitive engine (only) is available upon request.

The test scenario involves defining different logic for different values of audio conflict and stuck button occurring together. Table 5.3 describes the functionality we want to implement, where depending on the value of the context collector variables either the led is turned off, or the vibe is turned off or an error condition is logged. An audio conflict is a sound playing in the wrong pedestrian state e.g. a walk sound playing during the don't walk pedestrian state.

We define the following context collectors - BAC is button audio conflict where 1 is audio conflict detected and 0 is no audio conflict detected. BS is stuck button where 1 is stuck button detected and 0 is no stuck button detected. The context collectors are program variables that have unique identifiers, BAC id is 1 and BS id is 2.

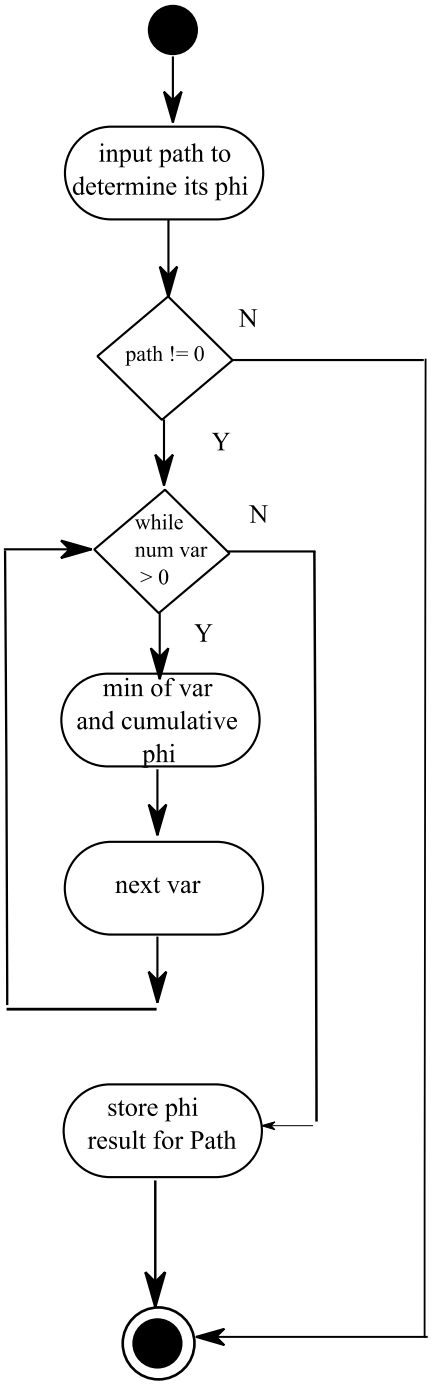


Figure 5.8. FCM ϕ state-diagram

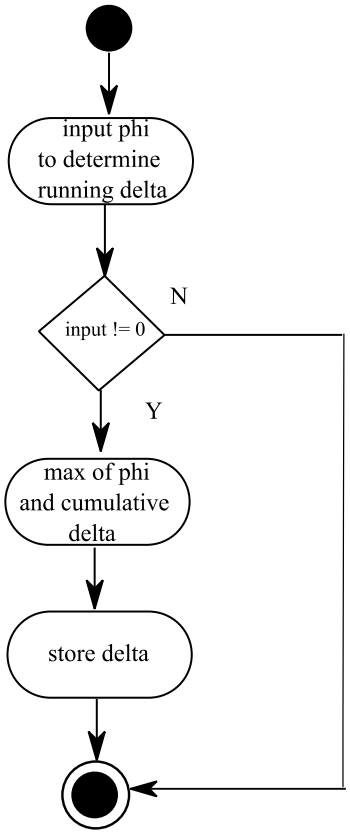


Figure 5.9. FCM Δ state-diagram

```
Loop as per timer defined in cognitive map

      For each line in the cognitive map

          Evaluate phi for each line

          Store phi in cognitive map

          Store latch if specified by coverage map



      For each line in the cognitive map

          Determine delta

          If momentary (as defined by coverage map)

                Execute first delta

                Break

          If latched (as defined by coverage map)

                Execute all latched values for valid delta
```

Figure 5.10. Pseudo code for overview cognitive engine

Table 5.3. Logic Map Setup

BAC	BS	Action	Logic Coverage Map Entry
1	1	Turn off led	1 2 1 2 0 0 0 0 'N' '6' '0' 0 0 0 0 0
0	1	Turn off vibe	3 2 -1 2 0 0 0 0 'N' '9' 0 0 0 0 0 0
1	0	Log error 9	2 2 1 -2 0 0 0 0 'N' '1' ' ' 'e' 'r' 'r' ' ' '9'

In order to rapidly prototype the proof of concept, simulated commands for updating the coverage map and manually overwriting the value of the context collectors via a Tera-term terminal utility were used. (This is instead of incorporating the coverage map logic download into the Novax Intellicross™ configuration utility which is generally to change user parameters e.g. sounds, volume levels). The context coordinator was configured to run the cognitive engine once instead of continuously on a periodic timer. The input command sequence is shown in Table 5.4 and the results are shown in Figure 5.17.

Table 5.4. Input command sequence for testing cognitive engine

Input Command Sequence	Expected Result
V0 000	Initialize all context collector variables to zero
V1	Load logic coverage map line 1
V2	Load logic coverage map line 2
V3	Load logic coverage map line 3
V0 111	Force values BAC=1, BS=1 to execute V1 context controller commands
V0 101, N3, N2	Force values BAC=0, BS=1 to execute V2 context controller commands, delete, display log
V0 110, N2	Force values BAC=1, BS=0 to execute V3 context controller commands, display log

5.1.4. Learning

This section discusses how learning by teaching the inputs can occur in the thesis model using an industrial remote control scenario. An industrial remote control is typically a transmitter/receiver pair where the toggling of switches/buttons/dials on the receiver results in the actuation of the corresponding output on the transmitter. The contexts here could be the switch status. The context collectors would acquire and store information on the switch status and the context controllers would be responsible for the actuation of the output. Adaptability could mean altering the switch pattern that is allowed to actuate a given output.

The hypothetical remote control example chosen is a land clearing machine (a “bush hog” used to cut trees and grass) [64].

The context controllers C1, C2 and C3 (Figure 5.11) represent Engine Start output, Horn output and Enable Forward Motion outputs respectively.

The context collector switches C4, C5, C6, C10 (Figure 5.11) are Engine Start switch, Horn switch, Enable Forward Motion switch and Break Release switch.

Figure 5.11 shows the current context-aware map logic for device operation.

- Toggling of the Engine Start (C4) switch turns on the Engine Start (C1) output.
- The horn (C2) output is turned on by toggling the horn (C5) switch.
- Toggling the Enable Forward Motion (C6) switch turns on the Enable Forward Motion (C3) output.

The training of the inputs would occur by the end user selecting which output is to be trained from a systems parameter menu similar to the one shown in Table 5.5. Once the output is selected, the system enters a training mode. In order to build the context-aware logic map, the user then toggles the combination of switches required to activate the selected output.

The context aware map logic structure allows for easily modifying the device (in-field) operation by remapping which switches activate which outputs. Figure 5.12 shows the modified functionality.

- In order to be on, the Engine Start (C1) output now requires toggling of the Engine Start (C4) switch in addition the Horn switch (C5).
- Also, turning on the horn (C2) output at any time by only toggling the horn (C5) switch, is now disabled.
- Turning on the Enable Forward (C3) output now requires toggling the Brake Release (C10) switch instead of the Enable Forward Motion (C6) switch.

Table 5.5. Training Outputs Menu

PLEASE SELECT OUTPUT TO BE TRAINED	
OUTPUT 1	[0,1]
OUTPUT 2	[0,1]
OUTPUT 3	[0,1]
OUTPUT 4	[0,1]

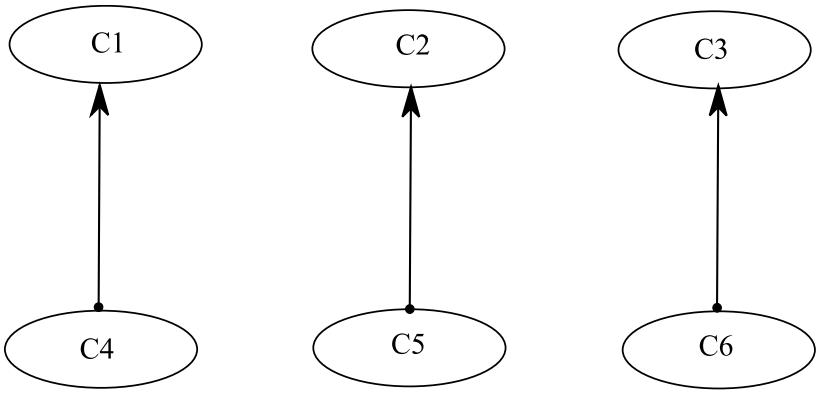


Figure 5.11 Current Controller Layout

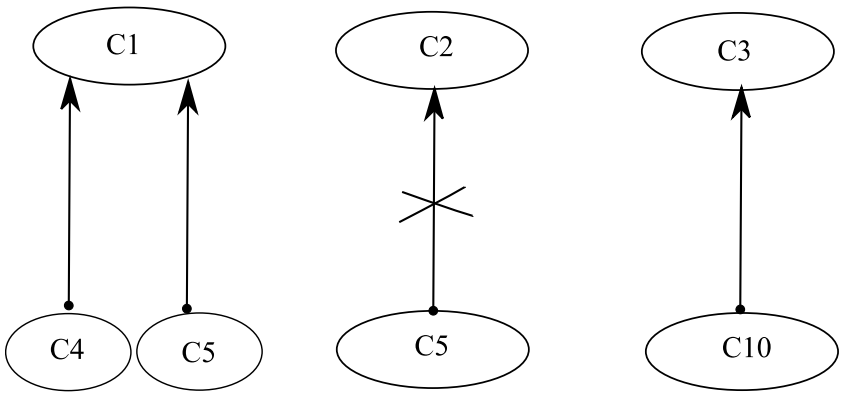


Figure 5.12 Controller Layout Update after Simple Learning

5.1.5. Power Consumption and Optimization Techniques

Power consumption is a very important topic in embedded systems - especially for battery operated devices. In our thesis model, the use of the context-aware logic map precludes catch-all functionality by specifying a select set of contexts to be queried. This translates to powering off or putting in low power mode hardware devices not specified in the context-aware logic map – a power-saving technique. This power-savings has not been numerically quantified but techniques for measuring the power consumption for future work include:

- Series resistor profiling in a dc circuit, where power in Watts = Volts x Amps. Two multimeters are used - one across the battery and one to measure current [55].
- Commercially available solutions like Lauterbach [56], ST Micro product STHORM, Wind River's Simics and Power Top (performance analysis tool) and IDEs (integrated development environment) that do power consumption profiles.

Measuring power and voltage

In order to measure power consumption of the code, the routine is called in a super loop in the program and average voltage and current in the circuit are measured to determine the power. Figures 5.14 and 5.15 show the source listing for measuring power consumption and Figure 5.13 shows the circuit setup [55].

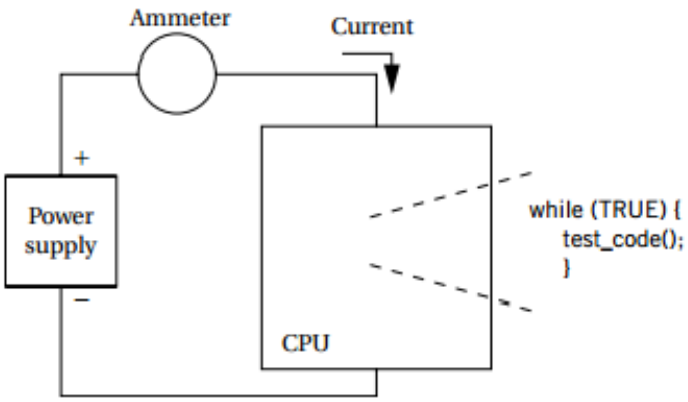


Figure 5.13. Circuit Setup for Measuring Power Consumption

```
while (1)
        {
        test_routine1_for_power_consumption ();
        }
```

Figure 5.14. Source Listing A

```
while (1)
        {
        test_routine2_for_power_consumption ();
        }
```

Figure 5.15. Source Listing B

Optimization Techniques

Additionally, as future work, some further optimization techniques to reduce power consumption can be considered. Some standard techniques include:

- Expression Simplification using laws of algebra to simply expressions.
- Procedure in lining reduces procedure linkage instructions when a cache is present.
- Loop Transformation unrolling loops, completely or partially, removes loop overhead. (Appendix E).
- Instruction Selection choosing instructions that have lower execution times.
- Understanding and Using Compiler Features using the -O
- Interpreters and JIT compilers - interpreters (e.g. Forth) translate a small piece of the program at a time and may result in overall smaller size (program plus interpreter) that the native machine code [55].

5.2. Results and Discussion

In this section the latency results are compared for our architecture versus the standard MVC architecture. Also the results of the APS cognitive engine operation are presented. The FCM and dispatching sources are presented in Appendices C, D and H.

5.2.1. Latency Results

Figure 5.16 shows the key routines associated with the new architecture coord_main call tree and the old architecture sys_main call tree. The call tree lists the number of functions defined, as well as the order in which they are called (which gives insight into the messaging sequence (shown in Figures 5.2 and 5.4.)) The new architecture has 9 functions and the old architecture has 5 functions. Figure 5.16 also shows a detailed listing of the latencies in execution cycles. In the new architecture the context coordinator (coord_main) queries the context collector (coll_context_change) for gps data (gps_get_serial_data) from the gps driver (gps_driver). The context-coordinator then passes this data (coord_handle_communication) to the zone context controller which processes it and then notifies (zone_handle_communication) the context coordinator of

the zone. The context coordinator the notifies (coord_handle_communication) the task manager of the zone information which passes the information (tsk_handle_communication) to the windows display controller (wnd_update). In the old architecture sys_main is responsible for querying the gps (gps_get_serial_data, gps_driver) and determining the zone, after which the task manager is notified ((tsk_handle_communication)) and forwards the information onto the windows display (wnd_update).

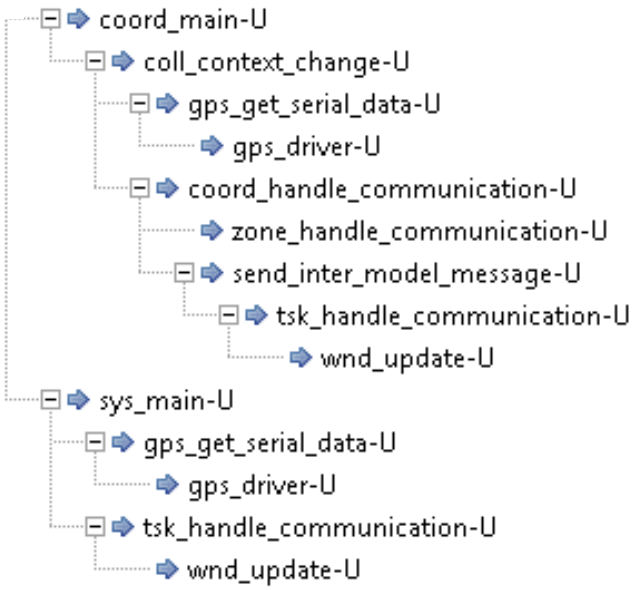
From the simulation it was shown that the benchmark statistics for Overall Execution Times are the same for the new and old architectures, i.e. Min OverET, Average OverET, High Water OverET, Max OverET, Worse Case OverET are identical. For latency analysis the Worse Case Execution Times are used.

Overall Latency - In this research overall latency is defined as the sum of functional latency, abstraction latency and messaging latency. The overall latency in the new architecture is larger. coord_main requires an overall execution time (OverET) of 8216 execution cycles and sys_main has a 4579 OverET. The increased latency is because of additional overhead in contextual processing.

Functional Latency - The core logic functionality latency is assumed to be the same for both the new and old architectures as the core logic is only organized differently.

Abstraction Latency - Overall the abstraction latency is larger in the new model. From Figure 5.16 the new model has the additional coll_context_change routine with 1757 SelfET, in addition to the routines shared with the old architecture gps_get_serial_data, gps_driver, and wnd_update.

Messaging Latency - The messaging latency is larger in the new model because there are more components and there is communication between the components and between the context layer and the application layer. From Figure 5.16 the new model has the additional coord_handle_communication 2436 SelfET, zone_handle_communication (not profiled) and send_inter_model_message 1299 SelfET. The routine shared with the old architecture is tsk_handle_communication 1155 SelfET.



Function Name	W-ET SelfET	W-ET SubfET	W-ET OverET
coll_context_change	1,757	5,889	7,646
coord_handle_communication	2,436	2,454	4,890
coord_main	570	7,646	8,216
gps_driver	0	0	0
gps_get_serial_data	999	0	999
send_inter_model_message	1,299	1,155	2,454
sys_main	2,425	2,154	4,579
tsk_handle_communication	1,155	0	1,155
tsk_main	1,105	12,795	13,900
wnd_update	0	0	0
zone_handle_communication	0	0	0

Figure 5.16. Rapita simulation results

5.2.2. Discussion

In this section the new model will be evaluated for code size (footprint), latency and code complexity as well as a discussion on the advantages and disadvantages of our model, future work and concluding remarks.

Code Size

The cognitive engine functionality is designed to be used with micro-processors with low memory requirements. Our Novax APS test-bed has 128KB flash memory. The size taken up by the cognitive engine is 138 bytes, ϕ function is 246 bytes and Δ function is 76 bytes. The code real-estate used to achieve dynamic adaptability is very small.

Code Complexity

The cognitive engine can be evaluated for code complexity using the McCabe Cyclomatic Complexity Metric. This metric measures complexity with respect to the number of linearly independent paths in the source listing. Dynamic allocation of the logic maps reduces the number of hard-coded paths. Only the logic that is needed is built into the map. There is no catch-all functionality which would significantly increase the number of linearly independent paths and consequently increase code complexity [54]. The use of FCM reduces the overall code complexity.

Advantages and Disadvantages

Based on the above findings, the advantages of our system include the light-weight FCM design characterized by low code space and reduced complexity. None of the previous works from Chapter 2 was developed to operate under such constrains. Our design can be scalable limited to only the memory space available on the processor and the number of context controllers and context collectors defined. The proof of concept

introduced limitations on logic map size only for rapid prototyping purposes. Our logic map design allows for more free-form changes with greater control over the device and it is not limited to e.g. specific predefined plans as used by Gamez [13].

The disadvantages of our research include limitations on the complexity of the logic map operator feature set e.g. it currently cannot support sophisticated machine learning algorithms or advanced mathematical operations. Another drawback is the end user needing application specific knowledge e.g. the context collector variables unique identifier numbers. This direct access to program variables may also pose security issues. The logic map design is based on the use of cyclic executive timing (or time slices) and not for use with a real-time operating system as there is no provision for mutexes to mitigate deadlocks or race-conditions.

The increased architecture latency is not significant enough to be prohibitive to the solution's adoption.

5.2.3. Cognitive Engine Result

The results in Figure 5.17 show the correct operation of cognitive engine for interpreting and executing the logic map. As per Table 5.4, rows 1-7 were entered sequentially at Terra-term and the results displayed. The code is written such that the command currently being executed is also displayed to the terminal e.g. Vzo is shown on the screen when the led is turned off, Vv appears when the vibe is turned off. Observing the APS hardware shows corresponding physical outputs also being actuated. When the contents of the BAC and BS variables are updated or the logic map is updated a confirmation of the contents is also printed to the screen. Before [V0 110] is evaluated the log is deleted and then displayed to verify 0 events logged. After [V0 110] is executed the log contents show error code 9 registered.

```
>V0 000
VAR = 0 0 0
> V1
1 = 1 2 1 2 0 0 0 0 86 122 48 0 0 0 0 0
> V2
2 = 2 2 1 -2 0 0 0 0 76 109 32 101 114 114 32 57
> V3
3 = 3 2 -1 2 0 0 0 0 86 118 0 0 0 0 0 0
> V0 111
VAR = 1 1 1
> N6
> V0 101
VAR = 1 0 1
> N9
Vibe off
> N3
> N2
0,*,01/01/01,00:02:31,,Log started,3
0 events
> V0 110
VAR = 1 1 0
> N1 err 9
> N2
0,*,01/01/01,00:02:31,,Log started,3
0,*,01/01/01,00:02:42,a,*** Marker ***,9
1 events
>
```

Figure 5.17 Results displayed on terminal program (Tera-term)

5.3. Conclusion of the Work Presented in this Chapter

The purpose of the research was to develop a context-aware embedded firmware model for dynamic adaptability. Simple well organized modularized components in a context-aware layer which resides above the application layer accomplished this. Dynamic adaptability was achieved using Fuzzy Cognitive Maps in order to provide user configurable logic for a flexible and enhanced application operation. The thesis research model was shown to be suitable for use with resource constrained embedded processors found in either wireless sensor networks or mature or legacy or cost-aware applications with single core processors.

Chapter 6.

Inclusion of A Complex Social System

The thesis model can be augmented to include a complex social system. This scenario would be for health care applications for physiological monitoring. Figure 6.1 shows the physiological biomedical device's Severity FCM and Figure 6.2 shows the complex social system to be integrated as part of Figure 6.1 FCM. The link between both models is the “biomedical device” concept.

From Figure 6.1, input information could have been acquired via implicit sensor monitoring or explicit voice command request/response processing. Potential contexts to be monitored include:

- Medication - Did you take your medication?
- Falls - You are lying flat. Did you fall?
- Mental Health - Are you feeling depressed this morning?
- Daily Care - Was your daily care performed?
- Temperature - What is your temperature? Is it within the normal range?
- Heart Rate - Was your heart racing or outside of the normal zone? This could be indicative of fearful situation (break-in) or a health-condition.
- Blood Pressure – What is your blood pressure? Is it within the normal range?

Depending on the medical condition the patient has only certain contexts needs to be monitored. If the patient is able-bodied but is a heart patient who forgets to take his or her medication and is prone to depression then only the medication, mental health and heart-rate context need to be monitored for comprising the severity fcm (Figure 6.3).

If the patient is elderly, prone to falls and suffers from high blood pressure then the falls, blood pressure, medication context needs to be monitored for severity (Figure 6.4).

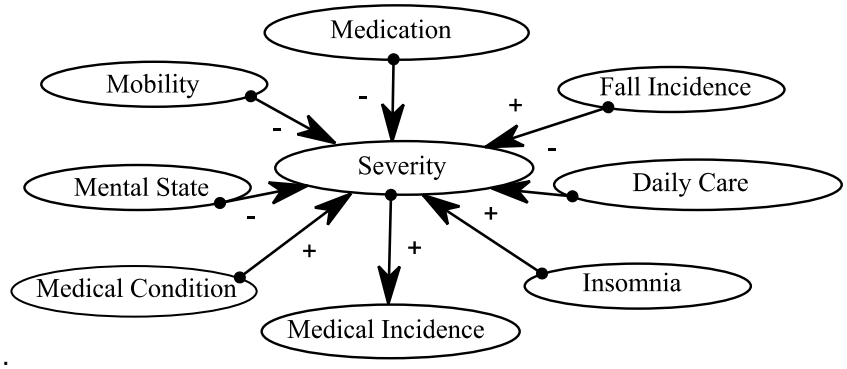


Figure 6.1 Severity FCM

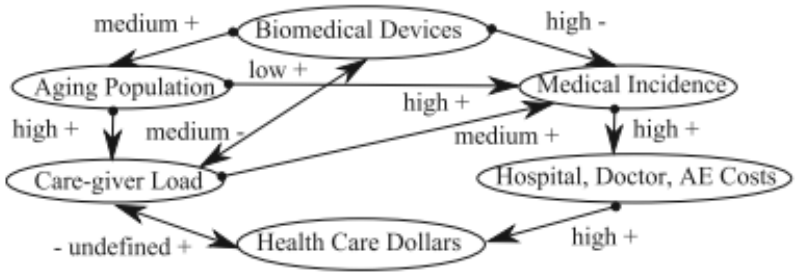


Figure 6.2 Social System FCM

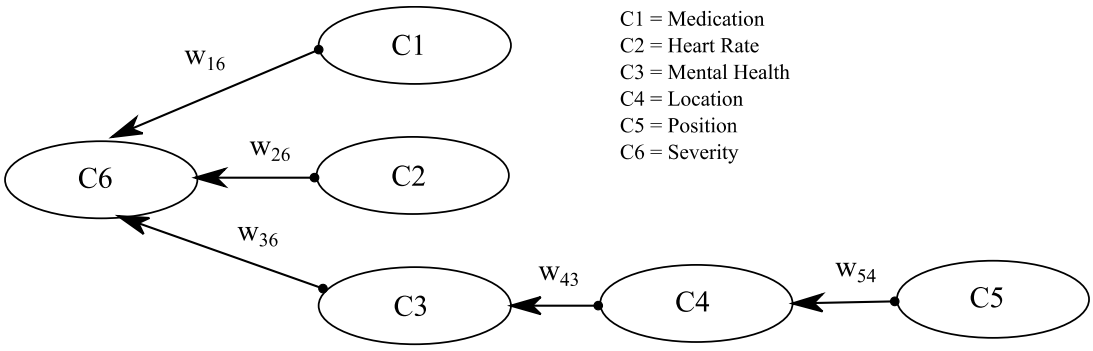


Figure 6.3 Expanded Severity FCM I

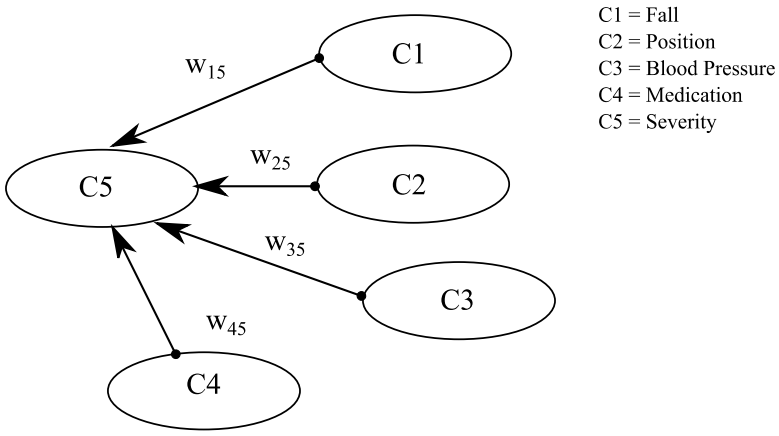


Figure 6.4 Expanded Severity FCM II

6.1. Model Verification

The second model in Figure 6.2 was already verified as part of a larger model [2]. The specific cases of first model in Figure 6.1 can be verified using simulation data as would be the case for any software model. The general functionality and software primitives were verified in the previous instalment. The below example (Figure 6.5, Figure 6.6) shows how the Severity FCM can be setup. Information collected by the severity FCM can be used to evaluate the effectiveness of in-home biomedical devices for physiological monitoring e.g. how has the devices usage reduced hospital admissions. The effectiveness of these devices in reducing hospital admissions is described in [2].

```
#define SEVERITY 1

#define MEDICATION 2

#define HEARTRATE 3

#define MENTALHEALTH 4

#define LOCATION 5

#define POSITION 6
```

```
void demo (void)
   {
   //these are the possible outcomes
   head = NULL;

   //insert the individual outcomes
   insert_outcome(SEVERITY, "E0");  //decision analysis outcome
   set_outcome_type (SEVERITY, decision_analysis_type);

   //insert decision analysis paths
   insert_path(MEDICATION, "A0");  //decision analysis element
   insert_path(HEARTRATE, "B0");  //decision analysis element
   insert_path(MENTALHEALTH, "C0");  //decision analysis element

   //insert decision analysis path elements
   insert_path_element(MEDICATION, "A0");  //decision analysis element
   insert_path_element(HEARTRATE, "B0");  //decision analysis element
```

```
insert_path_element(MENTALHEALTH, "C0");  //decision analysis element

insert_path_element(LOCATION, "C0");  //decision analysis element

insert_path_element(POSITION, "C0");  //decision analysis element


//add weight for decision analysis

add_weight(MEDICATION, get_weight(MEDICATION));

add_weight(HEARTRATE, get_weight(HEARTRATE);

add_weight(MENTALHEALTH, get_weight(MENTALHEALTH));

add_weight(LOCATION, get_weight(LOCATION));

add_weight(POSITION, get_weight(POSITION));


//decision analysis

fcm_polarity (search(SEVERITY));

fcm_degree_of_belief (search(SEVERITY));

fcm_most_believed_effect (search(SEVERITY));

}
```

Figure 6.5 Severity FCM Example 1 Source Listing

```
#define SEVERITY 1

#define MEDICATION 2

#define HEARTRATE 3

#define MENTALHEALTH 4

#define LOCATION 5

#define POSITION 6

#define FALL 7

#define BLOODPRESSURE 8



void demo (void)

   {

   //these are the possible outcomes

   head = NULL;


   //insert the individual outcomes

   insert_outcome(SEVERITY, "E0");  //decision analysis outcome

   set_outcome_type (SEVERITY, decision_analysis_type);
```

```
//insert decision analysis paths

insert_path(MEDICATION, "A0");  //decision analysis element

insert_path (POSITION, "B0");  //decision analysis element

insert_path(BLOODPRESSURE, "C0");  //decision analysis element

insert_path(FALL, "D0");  //decision analysis element


//insert decision analysis path elements

insert_path_element(MEDICATION, "A0");  //decision analysis element

insert_path_element(POSITION, "B0");  //decision analysis element

insert_path_element(BLOODPRESSURE, "C0");  //decision analysis element

insert_path_element(FALL, "D0");  //decision analysis element


//add weight for decision analysis

add_weight(MEDICATION, get_weight(MEDICATION));

add_weight(POSITION, get_weight(POSITION);

add_weight(BLOODPRESSURE, get_weight(BLOODPRESSURE));

add_weight(FALL, get_weight(FALL));


//decision analysis
```

```
fcm_polarity (search(SEVERITY));

fcm_degree_of_belief (search(SEVERITY));

fcm_most_believed_effect (search(SEVERITY));

}
```

Figure 6.6 Severity FCM Example 2 Source Listing

Conclusions

The purpose of the thesis research was to develop a context-aware embedded firmware model for dynamic adaptability. Simple well organized modularized components in a context-aware layer which resides above the application layer accomplished this. Dynamic adaptability was achieved using Fuzzy Cognitive Maps in order to provide user configurable logic for a flexible and enhanced application operation. The thesis research model was shown to be suitable for use with resource constrained embedded processors found in either wireless sensor networks for mature or legacy or cost-aware applications with single core processors.

While our research was developed for and is recommended for use in resource constrained embedded systems, because of its light-weight design it can also be used with more complex multi-core, multimedia, embedded processors or desktop or sever solutions. Our design, where the context-aware functionality is layered above a standard MVC architecture, easily extends existing solutions built on a standard MVC architecture. The FCM cognitive engine can be used to change user functionality depending on the customer needs, time of day, or special occasion (e.g. in the APS case, heavier pedestrian traffic requiring one-of, infrequent operational change).

Future Work

Our recommendations for further research includes the development of a formal SDK (system development kit) for the context-aware adaptable architecture similar to Dey's tool-kit [10], expanding the cognitive logic map to be more that sixteen characters per line by using variable length fields, exploring multi-line context controllers, and investigating compression techniques for context controller command sequences, incorporating a checksum (for wireless download), providing an expanded instruction set to augment our phi, delta, latch, momentary complement and timer functionality (potentially based on research into the current range of FCM operators) and expanding from a binary to ternary or n-ary FCM. In our example the maps were stored in RAM variable space but they can also be stored in external, peripheral. Further research can also be conducted on the inclusion of a complex social system finite state machine learning, evaluation of optimizations on the basis of power consumption and optimization on the cognitive engine

References

[1] Camille Jaggernauth, Bozena Kaminska and Douglas Gubbe, “Context-Aware Model for Dynamic Adaptability of Software for Embedded Systems.” International Journal of Computer, vol 19, no 1 (2015), pp 91-113.

[2] Camille Jaggernauth, “Modeling Returned Biomedical Devices in a Lean Manufacturing Environment.” Modeling and Simulation of Complex Social Systems, Intelligent Systems Reference Library, Springer, 2014.

[3] Camille Jaggernauth, Yindar Chuo, Benny Hung, Philip Lin, Bozena Kaminska, “Test Firmware Architecture for a Flexible Wireless Physiological Multi-Sensor.” IEEE SMC 2011.

[4] Camille Jaggernauth, Yindar Chuo, Benny Hung, Philip Lin, Bozena Kaminska, “Optimized, Practical Firmware Design for a Novel Flexible Wireless Multi-Sensor Platform for Body Activity and Vitals Monitoring.” ICCE 2011.

[5] Y Chuo, M Marzencki, B Hung, C Jaggernauth, K Tavakolian, P Lin, B Kaminska, “Mechanically Flexible Wireless Multisensor Platform for Human Physical Activity and Vitals Monitoring.” IEEE Transactions on Biomedical Circuits and Systems October 2010, 4 (5), pp. 281-294.

[6] Bozena Kaminska, Yindar Chuo, Marcin Marzencki, Benny Hung, Camille Jaggernauth, Kouhyar Tavakolian, Philip Lin, “Complete Platform for Remote Health Management.” IIIS 2009.

[7] Salem Hadim, M. Franklin, Nader Mohamed, and D. Culler, “Middleware: Middleware challenges and approaches for wireless sensor networks.” IEEE Distributed Systems, 2006, 7(3).

[8] Salem Hadim, M. Franklin, Nader Mohamed, and D. Culler, “Middleware for wireless sensor networks: A survey.” COMSWARE, 2006.

[9] N. Medvidovic, “Software architectures and embedded systems: A match made in heaven.” IEEE Software, 2005.

[10] Abowd G. D. Dey, A.K. and D. Salber, “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.” Human-Computer Interaction, 2001, 16(2), pp 97-166.

[11] Y. Chuo et al., “Mechanically flexible wireless multisensor platform for human physical activity and vitals monitoring.” IEEE Transactions on Biomedical Circuits and Systems – BIODEVICES, 2010, 4(5), pp 281-294.

[12] S. Pantsar-Syvaniemi, “Embedded systems - theory and design methodology.” In Computational Models of Complex Systems. InTech, 2012.

[13] N. et al Gamez. “Context-awareness in wireless sensor networks: A middleware solution.” Sensors, 2012, 12(7), pp 8544-8570.

[14] P. Inverardi and M. Tivoli, “The future of software: Adaptation and dependability.” LNCS 5413, Springer-Verlag Berlin, Heidelberg, 2009. 5413, pp 1-31.

[15] Z. Buyukozkan, G. Vardaloglu, “Analyzing of collaborative planning, forecasting and replenishment approach using fuzzy cognitive map.” International Conference on Computers and Industrial Engineering, 2009.

[16] A. Scalmato, A. Sgorbissa and F. Mastrogiovanni, “Affordance-based planning for assisting humans in daily activities.” Sixth International Conference on Intelligent Environments (IE), 2010.

[17] D.K. Iakovidis and E. Papageorgiou, “Intuitionistic fuzzy cognitive maps for medical de-cision making.” IEEE Transactions on Information Technology in Biomedicine, 2011.

[18] E. et al. Papageorgiou. “A fuzzy cognitive map based tool for prediction of infectious diseases.” FUZZ-IEEE, 2009.

[19] N.I. Papandrianos, D. Apostolopoulos, P. Vassilakos, E. Papageorgiou, “Complementary use of fuzzy decision trees and augmented fuzzy cognitive maps for decision making in medical informatics.” International Conference on Biomedical Engineering and Informatics, BMEI, 2008.

[20] P. W. Franks, U. Ekelund, S. Brage, N. Brage and N. J. Wareham, “Reliability and validity of the combined heart rate and movement sensor Actiheart.” European Journal of Clinical Nutrition, 2005, 561-570.

[21] Y. Chuo and B. Kaminska, “Multiparameter single locus integrated multilayer polymer micro-sensor system.” Proc. International Conference on Biomedical Electronics and Devices, BIODEVICES(1), 2008, 36-43.

[22] F. Touati N. Hamza and L. Khriji, “An optimized embedded architecture for multi-purpose wireless biomedical system using ZigBee technology.” International Conference on Signals, Circuits and Systems, 2008.

[23] B. Rubio, “Programming approaches and challenges for wireless sensor networks.” IC-SNC, 2007.

[28] Y. Chuo, A. Vaseghi, K. Tavakolian, F.M. Zadeh and B. Kaminska. “Development of a novel contactless mechanocardiograph device.” International Journal of Telemedicine and Applications, 2008.

[29] B. Kaminska and Y. Chuo, “Testing multilayer flexible wireless multisensor platforms.” Journal of Electronic Testing: Theory and Applications, 2009, pp 1-6.

[30] D. Kardaras and B. Karakostas, “The use of fuzzy cognitive maps to simulate the information systems strategic planning process.” Information and Software Technology, 1999, 41(4), 197-210.

[31] W. Stach and L. Kurgan, “Modeling Software Development Projects Using Fuzzy Cognitive Maps.” Proceedings of the 4th ASERC Workshop on Quantitative and Soft Software Engineering. 55–60 (2004).

[32] E. Papageorgiou, et al., “A Fuzzy Cognitive Map based tool for prediction of infectious diseases.” FUZZ-IEEE, (2009).

[33] A. Pantelopoulos, “Prognosis – A Wearable Heath-Monitoring System for People at Risk: Methodology and Modeling.” IEEE Transactions on Information Technology in Biomedicine, May 2010.

[34] M. Nii, K. Nakai et al., “Extraction of Human Behavior from Human Activity Monitoring Data using MEMS Sensors.” 2011 IEEE Workshop on Robotic Intelligence In Informationally Structured Space (RiiSS).

[35] Chee Seng Chan, Honghai Liu, D. Brown and N. Kubota, “A fuzzy qualitative approach to human motion recognition.” Fuzzy Systems, 2008. FUZZ-IEEE 2008.

[36] T. Shany and S. Redmond, “Sensors-Based Wearable Systems for Monitoring of Human Movement and Falls.” IEEE Sensors Journal, March 2012.

[37] H. Ghasemzadeh and R. Jafari, “Physical Movement Monitoring Using Body Sensor Networks: A Phonological Approach to Construct Spatial Decision Trees.” 2011 IEEE Transactions on Industrial Informatics.

[38] J Pärkkä, L. Cluitmans and M. Ermes, “Personalization Algorithm for Real-Time Activity Recognition Using PDA, Wireless Motion Bands, and Binary Decision Tree.” IEEE Transactions on Information Technology in Biomedicine, 2 Sep 2010.

[39] M. Ermes, J. Parkka, J. Mantyjarvi and I. Korhonen, “Detection of Daily Activities and Sports With Wearable Sensors in Controlled and Uncontrolled Conditions.” IEEE Transactions on Information Technology in Biomedicine, 7 Jan 2008.

[40] A.M Khan, Young-Koo Lee Lee and Kim Tae-Seong, “A Triaxial Accelerometer-Based Physical-Activity Recognition via Augmented-Signal Features and a Hierarchical Recognizer.” IEEE Transactions on Information Technology in Biomedicine, 2 Sep 2010.

[41] L. Atallah, B. Lo, R. Ali, R. King and Yang Guang-Zhong, “Real-Time Activity Classification Using Ambient and Wearable Sensors.” IEEE Transactions on Information Technology in Biomedicine, 3 Nov 2009.

[42] P. van de Ven et al., “Integration of a suite of sensors in a wireless health sensor platform.” IEEE Sensors 2009 Conference.

[43] M. Tentori and J. Favela “Activity-Aware Computing for Healthcare.” IEEE Pervasive Computing 11 Apr 2008.

[44] http://spectrum.ieee.org/biomedical/devices/wireless-health-care/0. Cited 15 Oct 2011.

[45] W. Hare. et al., “A Deterministic Model of Home and Community Care Client Counts in British Columbia. Health Care Management Science.”, 2009, 12(1), 80-99.

[47] D. Schrier, “Consequences of an Aging Population, Can Existing Levels of Social Services be Sustained,” Government of British Colombia publication. http://www.bcstats.gov.bc.ca/data/pop/pop/agingpop.pdf. Cited 24 Oct 2011.

[48] K. Stajduhar et al., “Interviewing family caregivers: Implications of the caregiving context for the research interview.” Qualitative Health Research, 2012.

[49] K. Stajduhar et al., “Planning for end-of-life care: Findings from the Canadian study of health and aging.” Canadian Journal on Aging. 27, 2008, 11–21.

[50] K. Stajduhar et al., “Situated/Being situated: Client and co-worker roles of family caregivers in hospice palliative care.” Social Science and Medicine, 2012.

[51] E. O’Connor, “Hardest job on the planet,” The Province, 21 October 2011.

[52] D.L. Parnas, “On the criteria to be used in decomposing systems into modules,” Communications of the ACM, Vol. 15, No. 12, December 1972, pp. 1053 - 1058

[53] D.L. Parnas, “Designing software for ease of extension and contraction,” IEEE Transactions on Software Engineering, 1979, SE-5(2), 128 – 138.

[54] T. McCabe, “A complexity measure.” IEEE Transactions on Software Engineering, 1976, 2(4), pp 308–320.

[55] Wayne Wolf, “Computers as Components Principles of Embedded Computing Design.” Morgan Kaufman Publishers, Chicago, London, 2008.

[56] Lauterbach, www.lauterbach.com.

[57] Jihong Pang, “Intelligent Modeling and Decision Making for Product Quality of Manufacturing System Based on Fuzzy Cognitive Map.” IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 1, No 2, January 2013, http://ijcsi.org/papers/IJCSI-101-2-501-506.pdf

[58] W. Stach and L., Kurgan, “Modeling Software Development Projects Using Fuzzy Cognitive Maps.” Proceedings of the 4th ASERC Workshop on Quantitative and Soft Software Engineering, 2004, pp 55–60.

[59] In Keun Lee, Hwa Sun Kim and Hune Cho, “Design of Activation Functions for Inference of Fuzzy Cognitive Maps: Application to Clinical Decision Making in Diagnosis of Pulmonary Infection,” Healthc Inform Res. 2012 Jun; 18(2): 105–114.

[60] S. Bueno et al, “Benchmarking main activation functions in fuzzy cognitive maps.” Elsevier, Expert Systems with Applications 36 (2009) 5221–5229.

[61] J. Smith, “The Doctor will see you ALWAYS.” IEEE Spectrum, October 2011, http://lifesciences.ieee.org/articles/55-the-doctor-will-see-you-always

[62] Bart Kosko, “Fuzzy Cognitive Maps,” International Journal of Man-Machine Studies, 24(1986) 65-75

[63] E. Glenn Krasner and Stephen T. Pope "A cookbook for using the model–view controller user interface paradigm in Smalltalk-80". The Journal of Object Technology (SIGS Publications), Aug–Sep 1988.

[64] Traxx Bush Hog, https://www.youtube.com/watch?v=K93fExVD4c0. Accessed 08 Aug 2016.

[65] K. Rehman, F. Stajano and G. Coulouris, “An architecture for interactive context-aware applications,” IEEE Pervasive Computing, 2007.

[66] Rapita Veification Suite. https://www.rapitasystems.com/. Accessed 03 August, 2016.

Appendix A

Switch Code

```
/*********************************************************************
* INCLUDES
*/
#include "switch_control.h"
#include "hal_sleep.h"

/*********************************************************************
* MACROS
*/
#define SC_BIT_GET(p,m) ((p) & m))
#define SC_BIT(x) (0x01 << (x))

/*********************************************************************
* CONSTANTS
*/

/*********************************************************************
* TYPEDEFS
*/

/*********************************************************************
* GLOBAL VARIABLES
*/

/*********************************************************************
```

```
* EXTERNAL VARIABLES
*/

/*****************************************************************
* EXTERNAL FUNCTIONS
*/

/*****************************************************************
* LOCAL VARIABLES
*/
int8 index;
int8 tempx[8];
/*****************************************************************
* LOCAL FUNCTIONS
*/



/*****************************************************************
* PUBLIC FUNCTIONS
*/

/*****************************************************************
* @fn      void SwitchControl_Init(void)
* @brief       Initialization of the switch control hardware abstraction
* @param       void
* @return      void
*/

void SwitchControl_Init(void){
```

```
                //1 Configurate pins to output by set bit
                //2 Select corresponding pins to GPIO by clear bit
                SR_1_PORT_DIR |= (0x02 | 0x10 |0x20);                //Set bit-1, bit-4, bit-5
                SR_1_PORT_SEL &=~(0x02 | 0x10 |0x20);    //Clear bit-1, bit-4, bit-5
                SR_2_PORT_DIR |= (0x01 0x02);//Set bit-0, bit-1
                SR_2_PORT_SEL &=~(0x01 |0x02);//Clear bit-0, bit-1
}

/*********************************************************************
* @fn      SwitchControl_Write()
* @brief       Write data to slelect switch

                8-bits data should shift out from MSP to LSP, left shift operaiton

                The following maps to the individual D-Latch in SCH

                ShiftData_0; QA
                ShiftData_1; QB
                ShiftData_2; QC
                ShiftData_3; QD
                ShiftData_4; QE
                ShiftData_5; QF
                ShiftData_6; QG
                ShiftData_7; QH
* @param      uint8 data
* @param      uint8 select
* @param      uint16 CPU block waiting, clock in ms base.
* @return   void
*/
void SwitchControl_Write(uint8 data, uint8 select, uint16 clock)
```

```
{

    //Every Write Sequence to the shift register should have the following

        //1. Pull done, Paralle Clock and Serial Clock before writing to the serial
        data

        //2. Output Serial Data

        //2.1. Shift Data bit to output pin

        //2.2. Delay Half of clock cycle

        //2.3. Pull up, Serial Clock,  ( Data should write input to the shift register)

        //2.4  Delay Half of clock cycle

        //2.5  Pull down Serial clock pin

        //3. Delay Half of clock cycle

        //4. Pull Up, Paralle Clock (New data should be available Shift Register
        Outputs)

        //5. Wait for Clock/2, and pull down the parelle clock again


        if(select == SR_1_SELECTED){

        SR_1_SRCK = LOW;

        SR_1_RCK  = LOW;

        data = 1;

        for(index = 7; index >=0; index--){

         //SR_DATA is set to "High" if data bit at index is 1

        //SR_DATA is set to "Low" if data bit at index is 0

        //Most Significant bit will shift out first.

        SR_DATA = SC_BIT_GET(data, SC_BIT(index)) ? HIGH : LOW ;

         //tempx[index] = SC_BIT_GET(data, SC_BIT(index)) ? HIGH : LOW ;

         halSleepWait(2);

         SR_1_SRCK = HIGH;

        halSleepWait(2);
```

```
SR_1_SRCK = LOW;
}
SR_1_RCK  = HIGH;
halSleepWait(2);
SR_1_RCK  = LOW;
}
else if(select == SR_2_SELECTED){
SR_2_SRCK = LOW;
SR_2_RCK  = LOW;    //?for testing purposes only
data = 1;
for(index = 7; index >= 0; index--){

SR_DATA = SC_BIT_GET(data, SC_BIT(index)) ? HIGH : LOW ;
//tempx[index] = SC_BIT_GET(data, SC_BIT(index)) ? HIGH : LOW ;
halSleepWait(2);
SR_2_SRCK = HIGH;
halSleepWait(2);

SR_2_SRCK = LOW; //? for testing purposes only
}
SR_2_RCK  =                                                    HIGH;
halSleepWait(2);
SR_2_RCK  =                          LOW;  //?for testing purposes only
}
}
```

Appendix B.

ADC Code

```
#include "ioCC2430.h"

#include "hal_types.h"

#include "hal_defs.h"

#include "spi_driver.h"

//#include "FlashUtils.h"

#include "hal_sleep.h"


#define STATIC


/***************UART and SPI copnfiguration macros from ./external/.../hal.h*****************/


/********************************** Configure UARTs ************************************/

#define IO_PER_LOC_UART1_AT_PORT0_PIN2345() do { PERCFG = (PERCFG&~0x02)|0x00; } while (0)

#define IO_PER_LOC_UART1_AT_PORT1_PIN4567() do { PERCFG = (PERCFG&~0x02)|0x02; } while (0)


#define IO_PER_LOC_UART0_AT_PORT0_PIN2345() do { PERCFG = (PERCFG&~0x01)|0x00; } while (0)

#define IO_PER_LOC_UART0_AT_PORT1_PIN2345() do { PERCFG = (PERCFG&~0x01)|0x01; } while (0)


// Macro for getting the clock division factor

#define CLKSPD  (CLKCON & 0x07)


// baud rate macro definitions

#define BR_2400         1
```

```
#define BR_4800         2
#define BR_9600         3
#define BR_14400        4
#define BR_19200        5
#define BR_28800        6
#define BR_38400        7
#define BR_57600        8
#define BR_76800        9
#define BR_115200       10
#define BR_153600       11
#define BR_230400       12
#define BR_307200       13

//*****************************************************************************
// Macro for setting up an SPI connection. The macro configures the appropriate
// pins for peripheral operation, sets the baudrate if the chip is configured
// to be SPI master, and sets the desired clock polarity and phase. Whether to
// transfer MSB or LSB first is also determined. _spi_ indicates whether
// to use spi 0 or 1. _baudRate_ must be one of 2400, 4800, 9600, 14400, 19200,
// 28800, 38400, 57600, 76800, 115200, 153600, 230400 or 307200.
// Possible options are defined below.

// The macros in this section simplify UART operation.
#define BAUD_E(baud, clkDivPow) (       \
    (baud==BR_2400)   ?  6  +clkDivPow : \
    (baud==BR_4800)   ?  7  +clkDivPow : \
    (baud==BR_9600)   ?  8  +clkDivPow : \
    (baud==BR_14400)  ?  8  +clkDivPow : \
    (baud==BR_19200)  ?  9  +clkDivPow : \
```

```
    (baud==BR_28800)  ?  9  +clkDivPow : \
    (baud==BR_38400)  ?  10 +clkDivPow : \
    (baud==BR_57600)  ?  10 +clkDivPow : \
    (baud==BR_76800)  ?  11 +clkDivPow : \
    (baud==BR_115200) ?  11 +clkDivPow : \
    (baud==BR_153600) ?  12 +clkDivPow : \
    (baud==BR_230400) ?  12 +clkDivPow : \
    (baud==BR_307200) ?  13 +clkDivPow : \
    0  )


#define BAUD_M(baud) (         \
    (baud==BR_2400)   ?  59  : \
    (baud==BR_4800)   ?  59  : \
    (baud==BR_9600)   ?  59  : \
    (baud==BR_14400)  ?  216 : \
    (baud==BR_19200)  ?  59  : \
    (baud==BR_28800)  ?  216 : \
    (baud==BR_38400)  ?  59  : \
    (baud==BR_57600)  ?  216 : \
    (baud==BR_76800)  ?  59  : \
    (baud==BR_115200) ?  216 : \
    (baud==BR_153600) ?  59  : \
    (baud==BR_230400) ?  216 : \
    (baud==BR_307200) ?  59  : \
    0)

#define SPI_SETUP(spi, baudRate, options)           \
   do {                                     \
```

```
  U##spi##UCR = 0x80;                       \
  U##spi##CSR = 0x00;                       \
                                            \
  if(spi == 0){                             \
    if(PERCFG & 0x01){                      \
      P1SEL |= 0x3C;                        \
    } else {                                \
      P0SEL |= 0x3C;                        \
    }                                       \
  }                                         \
  else {                                    \
    if(PERCFG & 0x02){                      \
      P1SEL |= 0xF0;                        \
    } else {                                \
      P0SEL |= 0x3C;                        \
    }                                       \
  }                                         \
                                            \
  if(options & SPI_SLAVE){                  \
    U##spi##CSR = 0x20;                     \
  }                                         \
  else {                                    \
    U##spi##GCR = BAUD_E(baudRate, CLKSPD);    \
    U##spi##BAUD = BAUD_M(baudRate);           \
  }                                         \
  U##spi##GCR |= (options & 0xE0);             \
} while(0)
```

```
// Options for the SPI_SETUP macro.
#define SPI_SLAVE             0x01
#define SPI_MASTER            0x00
#define SPI_CLOCK_POL_LO      0x00
#define SPI_CLOCK_POL_HI      0x80
#define SPI_CLOCK_PHA_0       0x00
#define SPI_CLOCK_PHA_1       0x40
#define SPI_TRANSFER_MSB_FIRST 0x20
#define SPI_TRANSFER_MSB_LAST  0x00

/*****************************************************************************/
#ifdef PROTOTYPE_DATAFLASH_SUPPORT
  #define SLAVE_CHIP_SELECT()    (P2 &= 0xFE)
  #define SLAVE_CHIP_DESELECT()  (P2 |= 1)
  #define IOCHARBUF              U0DBUF
  #define UARTCSR                U0CSR
#else
  #define SLAVE_CHIP_SELECT()    (P1 &= 0xEF)
  #define SLAVE_CHIP_DESELECT()  (P1 |= 0x10)
  #define IOCHARBUF              U1DBUF
  #define UARTCSR                U1CSR

#endif

#ifdef CC2430_BOOT_CODE
STATIC __near_func void ConfigureSpiMaster(SpiMasterConfigOptions opt);
#else
STATIC void ConfigureSpiMaster(SpiMasterConfigOptions opt);
#endif
```

```
ST_uint8  cRX=0;
ST_uint8  cTX=0;
ST_uint8  temp=0;
ST_uint8  cInput[3]={0,0,0};

/*******************************************************************************
Function:     _spiInit(void)
Arguments:
Return Values:There is no return value for this function.
Description:  This function is a one-time configuration for the CPU to set some
              ports to work in SPI mode (when they have multiple functions. For
              example, in some CPUs, the ports can be GPIO pins or SPI pins if
              properly configured).
              please refer to the specific CPU datasheet for proper
              configurations.
*******************************************************************************/
#ifdef CC2430_BOOT_CODE
__near_func
#endif
void _spiInit( void )
{
   // The TI reference design uses UART1 Alt. 2 in SPI mode
   IO_PER_LOC_UART1_AT_PORT1_PIN4567();
   SPI_SETUP(1, BR_115200, SPI_MASTER       | \
                     SPI_CLOCK_POL_LO | \
                     SPI_CLOCK_PHA_0  | \
                     SPI_TRANSFER_MSB_FIRST);
```

```
    // There is some awkwardness here. The SPI_SETUP macro above set P1.4 as a
    // peripheral pin in support of SPI. But the reference design uses P1.4 as
    // the /CS for the external dataflash part. We need to reconfigure the pin
    // as a GPIO pin, set the direction register to output, and set it high to
    // deselect the part.
    P1SEL &= ~BV(4);    // set P1.4 as GPIO pin
    P1DIR |=  BV(4);  // set P1.4 as output
    P1    |=  BV(4);  // set output high
}

/******************************************************************************
Function:     ConfigureSpiMaster(SpiMasterConfigOptions opt)
Arguments:    opt configuration options, all acceptable values are enumerated in
              SpiMasterConfigOptions, which is a typedefed enum.
Return Values:There is no return value for this function.
Description:  This function can be used to properly configure the SPI master
              before and after the transfer/receive operation
Pseudo Code:
   Step 1  : perform or skip select/deselect slave
   Step 2  : perform or skip enable/disable transfer
   Step 3  : perform or skip enable/disable receive
******************************************************************************/
STATIC
#ifdef CC2430_BOOT_CODE
__near_func
#endif
void ConfigureSpiMaster(SpiMasterConfigOptions opt)
{
/*
```

```
    if(enumNull == opt)
      return;
*/
    if(opt & (SpiMasterConfigOptions)MaskBit_SelectSlave_Relevant)
    {
      (opt & (SpiMasterConfigOptions)MaskBit_SlaveSelect) ? SLAVE_CHIP_SELECT() :
SLAVE_CHIP_DESELECT();
    }
/**
    if(opt & MaskBit_Trans_Relevant)(opt & MaskBit_Trans) ?
EnableTrans():DisableTrans();
    if(opt & MaskBit_Recv_Relevant) (opt & MaskBit_Recv)  ? EnableRcv():DisableRcv();
**/
}

/*****************************************************************************
Function:     Serialize(const CharStream* char_stream_send,
              CharStream* char_stream_recv,
              SpiMasterConfigOptions optBefore,
              SpiMasterConfigOptions optAfter
              )
Arguments:    char_stream_send, the char stream to be sent from the SPI master to
              the Flash memory, usually contains instruction, address, and data to be
              programmed.
              char_stream_recv, the char stream to be received from the Flash memory
              to the SPI master, usually contains data to be read from the memory.
              optBefore, configurations of the SPI master before any transfer/receive
              optAfter, configurations of the SPI after any transfer/receive
Return Values:TRUE
Description:  This function can be used to encapsulate a complete transfer/receive
```

```
            operation
Pseudo Code:
   Step 1  : perform pre-transfer configuration
   Step 2  : perform transfer/ receive
    Step 2-1: transfer ...
        (a typical process, it may vary with the specific CPU)
        Step 2-1-1:  check until the SPI master is available
        Step 2-1-2:  send the byte stream cycle after cycle. it usually involves:
                a) checking until the transfer-data-register is ready
                b) filling the register with a new byte
    Step 2-2: receive ...
        (a typical process, it may vary with the specific CPU)
        Step 2-2-1:  Execute ONE pre-read cycle to clear the receive-data-register.
        Step 2-2-2:  receive the byte stream cycle after cycle. it usually involves:
                a) triggering a dummy cycle
                b) checking until the transfer-data-register is ready(full)
                c) reading the transfer-data-register
   Step 3  : perform post-transfer configuration
*****************************************************************************/
#ifdef CC2430_BOOT_CODE
__near_func
#endif
Bool Serialize(const CharStream* char_stream_send,
               CharStream* char_stream_recv,
               SpiMasterConfigOptions optBefore,
               SpiMasterConfigOptions optAfter
               )
{
  uint16 inLen, outLen;
```

```
uint8 *pIn, *pOut;

if (char_stream_send)  {
  pOut   = char_stream_send->pChar;
  outLen = char_stream_send->length;
}
else  {
  outLen = 0;
}

if (char_stream_recv)  {
  pIn   = char_stream_recv->pChar;
  inLen = char_stream_recv->length;
}
else  {
  inLen = 0;
}

ConfigureSpiMaster(optBefore);

//while (outLen--)  {
//  while (UARTCSR & 1)  ;
//  IOCHARBUF = *pOut++;
//}

// wait for last Tx to finish.
while (UARTCSR & 1) ;

inLen = 3;
```

```
  P1 &= ~BV(4);    // set P1.4 as GPIO pin

  while (inLen--)  {
    // dummy write
    IOCHARBUF = 0; //camille - this could be failing the loopback
    while (UARTCSR & 1) {};
    cInput[inLen] = IOCHARBUF;
  }

  P1 |=  BV(4);

  ConfigureSpiMaster(optAfter);

    return TRUE;
}

/*****************************************************************************
Function:     _spiWrite
Arguments:
Return Value:
Description:
Pseudo Code:
*****************************************************************************/
void _spiWrite ( void )
{
    CharStream char_stream_send;
    ST_uint8  cWREN = 0x55;

    // Step 1: Initialize the data (i.e. instruction) packet to be sent serially
```

```
    char_stream_send.length = 1;
    char_stream_send.pChar  = &cWREN;

    // Step 2: Send the packet serially
    Serialize(&char_stream_send,
            ptrNull,
            enumEnableTransOnly_SelectSlave,
            enumDisableTransOnly_DeSelectSlave
            );
}


/*****************************************************************************
Function:     _spiRead
Arguments:
Return Value:
Description:
Pseudo Code:
*****************************************************************************/
void _spiRead ( void )
{
    CharStream char_stream_recv;
    ST_uint8  cRX;

    // Step 1: Initialize the data (i.e. instruction) packet to be sent serially
    char_stream_recv.length = 1;
    char_stream_recv.pChar  = &cRX;

    // Step 2: Send the packet serially
```

```
    Serialize(ptrNull,
            &char_stream_recv,
            enumEnableRecvOnly_SelectSlave,
            enumDisableRecvOnly_DeSelectSlave
            );
}


/*****************************************************************************
Function: _spiReadWrite
Arguments:
Return Value:
Description:
Pseudo Code:
*****************************************************************************/
void _spiReadWrite ( void )
{
    CharStream char_stream_send;
    CharStream char_stream_recv;
    //cTX = 0x33;
    //cRX = 0;

    // Step 1: Initialize the data (i.e. instruction) packet to be sent serially
    char_stream_send.length  = 1;
    char_stream_send.pChar   = &cTX;

    char_stream_recv.length  = 1;
    char_stream_recv.pChar   = &cRX;
```

```
    // Step 2: Send the packet serially
    Serialize(&char_stream_send,
            &char_stream_recv,
            enumEnableTansRecv_SelectSlave,
            enumDisableTansRecv_DeSelectSlave
            );
}
```

Appendix C.

FCM Code I

```
//The polarity should be stored in all the elements
//All path elements store the property of the path
//In case a path element is deleted
void fcm_polarity (node *to_be_search_outcome)
   {
   node * current_path;
   node * current_path_element;
   int polarity=1;

   //go through each path in the outcome and
   current_path = to_be_search_outcome->next_path;  //new path
   while (current_path != NULL)
      {
      //determine the polarity of each path
      polarity *= current_path->polarity;
      //printf("%s:%d\n", current_path->name, polarity);
      current_path->s = 0;

      current_path_element = current_path->next;
      while (current_path_element != NULL)
         {
         polarity *= current_path_element->polarity;
         //printf("%s:%d\n", current_path->name, polarity);
         current_path_element->s = 0;
         current_path_element = current_path_element->next;
```

```
        }//end while individual path element

      //then store the value of the polarity in the head element
      //additions/deletions to the list will result in auto-recalculation
      current_path->s = polarity;

      //advance to the next path
      current_path = current_path->next_path;
      polarity = 1;
      }
   };

//Degree of belief is the mininum value of the path
//All path elements store the property of the path
//In case a path element is deleted
void fcm_degree_of_belief (node *to_be_search_outcome)
   {
   //go through each path in the outcome
   //and determine the degree of belief
   //then store the value in the
   //head path element
   node * current_path;
   node * current_path_element;
   int phi=strong;

   //go through each path in the outcome and
   current_path = to_be_search_outcome->next_path;  //new path
   while (current_path != NULL)
      {
```

```
        //determine the polarity of each path
        phi = min(current_path->weight, phi);
        current_path->phi = undefined;

        current_path_element = current_path->next;
        while (current_path_element != NULL)
           {
           phi = min(current_path_element->weight, phi);
           current_path_element->phi = undefined;
           current_path_element = current_path_element->next;
           }//end while individual path element

        //then store the value of the polarity in the head element
        //additions/deletions to the list will result in auto-recalculation
        current_path->phi = phi;

        //advance to the next path
        current_path = current_path->next_path;
        phi=strong;
        }
    };

//Most Believed Effect is path with maxiumum belief
//This is the characteristic of the outcome
//The name of the path should be stored
void fcm_most_believed_effect (node *to_be_search_outcome)
    {
    //go through each path head and determine the maximum value.
    node * current_path;
```

```
node * current_path_element;
int delta=weak, prev_delta=weak;

to_be_search_outcome->delta = undefined;
prev_delta = delta = undefined;
to_be_search_outcome->name_delta[0] = 0;

//go through each path in the outcome and
//outcome is not a path
current_path = to_be_search_outcome->next_path;
while (current_path != NULL)
   {
   //determine the polarity of each path
   delta = max(current_path->phi, delta);

   //if delta has changed
   //store the name of the new path
               if (prev_delta != delta)
      {
      strcpy(to_be_search_outcome->name_delta, current_path->name);
      to_be_search_outcome->delta = delta;
      }

   current_path->delta = 0;
   prev_delta = delta;

   //advance to the next path
   current_path = current_path->next_path;
   delta = prev_delta = weak;
```

```
        }
    };

//The sole purpose of this routine is to test extern
void fcm_demo (void)
{
    print_node(head);
};



int f_x (int x)
{
    //bivalent function
    return ((x > 0) ? 1 : 0);
}

void state_machine_engine (void)
{
    //assume already have prepopulated list

    int num_concept = 0;
    int index_x, index_y;
    int cognitive_map[10][10]; //states and the link strengths
    char node_name[30];
    int x[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //state in calculation progress
    bool equilib = false;
    int num_state_to_equilib = 0;

    //get the cognitive map
```

```
/*
inputfile.open("input.txt", ios_base::in);
outputfile.open("output.txt", ios_base::out);

//cout << "Please enter the number of concepts: ";
inputfile >> num_concept;
//cout << endl << "Setting up the cognitive map" << endl;
for (index_x=0; index_x < num_concept; index_x++)
    {
    inputfile >> node_name[index_x];
    for (index_y=0; index_y < num_concept; index_y++)
        {
        //cout << index_x << " " << index_y << ":";
        inputfile >> cognitive_map[index_x][index_y];
        //cout << endl;
        }
    }


outputfile << ",";
for (index_x=0; index_x < num_concept; index_x++)
    {
    outputfile << node_name[index_x] << ",";
    }

outputfile << endl;
outputfile << "0,";

for (index_x=0; index_x < num_concept; index_x++)
```

```
    {
    outputfile << cognitive_map[index_x][index_x] << ",";
    }

  outputfile << endl;
  */

  while (!equilib)
    {
    num_state_to_equilib++;
    //calculate the first iteration, print out first iteration

    //outputfile << num_state_to_equilib << ',';

    for (index_x=0; index_x < num_concept; index_x++)
      {
      for (index_y=0; index_y < num_concept; index_y++)
        {
          if(index_y != index_x)
            {
              x[index_x] += cognitive_map[index_y][index_y] *
cognitive_map[index_x][index_y];
            }
        }
        //update the x state
        x[index_x] = f_x(x[index_x]);
      }
    //outputfile << endl;
```

```
        //update the state values
        equilib = true;
        for (index_x=0; index_x < num_concept; index_x++)
           {
           if (cognitive_map[index_x][index_x] != x[index_x])
              {
              cognitive_map[index_x][index_x] = x[index_x];
              equilib = false;  //if at least one state has changed then equilibrium was not achieved
              }
            x[index_x] = 0;
           }
        }
}

void state_machine_evaluate_outputs (node *head_state_machine){};
void state_machine_build (node *head_state_machine){};
void state_machine_evaluate_links (node *head_state_machine){};


void state_machine_and (node *to_be_search_outcome)
   {
   //go through each path in the outcome
   //and determine the degree of belief
   //then store the value in the
   //head path element
   node * current_path;
   node * current_path_element;
   bool and_state=true;
```

```
//go through each path in the outcome and
current_path = to_be_search_outcome->next_path;  //new path
while (current_path != NULL)
   {
   //determine the polarity of each path
   and_state = min(current_path->state, and_state);
   current_path->path_state = undefined;

   current_path_element = current_path->next;
   while (current_path_element != NULL)
      {
      and_state = min(current_path_element->state, and_state);
      current_path_element->path_state = undefined;
      current_path_element = current_path_element->next;
      }//end while individual path element

   //then store the value of the polarity in the head element
   //additions/deletions to the list will result in auto-recalculation
   current_path->path_state = and_state;
   //printf("path debug %s %d\n", current_path->name, current_path->path_state);

   //advance to the next path
   current_path = current_path->next_path;
   and_state=true;
   }
};
```

```
void state_machine_or (node *to_be_search_outcome)
  {
  //go through each path head and determine the maximum value.
   node * current_path;
   node * current_path_element;
   bool or_state=false, prev_or_state=false;

   to_be_search_outcome->outcome_state = undefined;
   prev_or_state = or_state = undefined;



   //go through each path in the outcome and
   //outcome is not a path
   current_path = to_be_search_outcome->next_path;
   while (current_path != NULL)
      {
      //determine the polarity of each path
      or_state = max(current_path->path_state, or_state);

      //if delta has changed
      //store the name of the new path
                  if (prev_or_state != or_state)
         {
         to_be_search_outcome->outcome_state = or_state;
         }

      current_path->outcome_state = 0;
      prev_or_state = or_state;
```

```
    //advance to the next path
    current_path = current_path->next_path;
    or_state = prev_or_state = false;
    }
};
```

Appendix D.

FCM Code II

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//node.h

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

typedef enum {

    false = -1,

    true = 1,

} bool;



enum linguistic_weight {

   undefined = -1,

   weak = 1,

   medium = 2,

   strong = 3,

};



enum fcm_type {
```

```
   process_control_type = 1,
   decision_analysis_type = 2,
   state_machine_type = 3,
};


enum parameter_type {
   cc = 1,
   dbs = 2,
   user = 3,
   cfg = 4,
};




struct node {
    int data;
    int weight;
    bool state;
    bool path_state;
    bool outcome_state;
```

```
    int polarity;
    int type;
    int phi;
    int delta;
    int s;
    char name[10];
    char name_delta[10];
    struct node * next;
    struct node * next_path;
    struct node * next_outcome;
    int parameter_id;    //unique identifier
    int parameter_type;  //cc, dbs, user, cfg
    int sequence;
};

typedef struct node node;

node * getnode(void);
void insert_beg(int x);
```

```
void insert_end(int x);

void insert_mid(int n_no,int x);

int delete_node(int kv);

node * search(int kv);

void reverse(void);

void sort(void);

void empty(void);

void display(void);

int count(void);



void insert_path (int x, char *name);

void insert_path_element (int x, char *name);

node * search_path (char *name);

void init_path (void);


void insert_outcome (int x, char *name);

//void insert_outcome_path(int x, char *name_outcome, char *name_path);
```

```
//void insert_outcome_path_element(int x, char *name_outcome, char *name_path);

node * search_outcome (char *name);


//Split up branches into parallels

void print_path (void);

void print_node (node *);

void print_decision_analysis (node *);

void add_weight(int, int);


void set_outcome_type (int, int);

void update_state(int, bool);


void node_test (void);
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//node.c

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
#include "link.h"
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>

node *head,*save,*current;

node node_heap[20];

void update_state (int kv, bool state)
   {
   node *addstatepls = search(kv);
   if (addstatepls != NULL)
      {
      addstatepls->state = state;
      //printf("\n%s - %d %d", addweightpls->name, addweightpls->weight, addweightpls-
>polarity );
      }
   }

void add_weight (int kv, int weight)
   {
   node *addweightpls = search(kv);
   if (addweightpls != NULL)
      {
      addweightpls->weight = abs(weight);
      if (weight >= 0)
         {
```

```
            addweightpls->polarity = 1;
            }
        else
            {
            addweightpls->polarity = -1;
            }
        //printf("\n%s - %d %d", addweightpls->name, addweightpls->weight, addweightpls->polarity );
        }
    }

void print_node (node *printpls)
    {
    if (printpls == NULL)
        printf("\nNOT FOUND");
    else
        printf("\n data %d\n name %s", printpls->data, printpls->name);
    }

void print_decision_analysis (node *printpls)
    {
    if (printpls == NULL)
        {
        printf("\nNOT FOUND");
        }
    else
        {
        printf("\n data %d", printpls->data);
        printf("\n name %s", printpls->name);
```

```
        //characteristics of the outcome
        printf("\n delta name %s", printpls->name_delta);
        printf("\n delta %d", printpls->delta);

        printpls = printpls->next_path;
        //characteristics of the paths
        while (printpls != NULL)
            {
            printf("\n");
            printf("\n     name %s", printpls->name);
            printf("\n     s %d", printpls->s);
            printf("\n     phi %d", printpls->phi);
            printpls = printpls->next_path;
            }

        printf("\n\n");
        }
    }

void print_state_machine_analysis (node *printpls)
    {
    if (printpls == NULL)
        {
        printf("\nNOT FOUND");
        }
    else
        {
        printf("\n name %s", printpls->name);
```

```
        printf("\n outcome state %d", printpls->outcome_state);
        printf("\n");
        }
    }

void print_path (void)
    {
    node * current_outcome;
    node * current_path_link;

    //clrscr();
    current_outcome = head;
    if (current_outcome == NULL)
        printf("\aThe List Is Empty!\n");
    while (current_outcome != NULL)   //new outcome
        {
        printf("\n%d - %s",current_outcome->data, current_outcome->name);

        current = current_outcome->next_path;  //new path
        while (current != NULL)
            {
            printf("\n    %d - %s",current->data, current->name);
            //print out the individual head elements

            current_path_link = current;
            while (current_path_link != NULL)
                {
                printf("\n        %d - %s",current_path_link->data, current_path_link->name);
                current_path_link = current_path_link->next;
```

```
            }

          current = current->next_path;
          }

        current_outcome = current_outcome->next_outcome;
        }

    //getch();
    //getch();
    }

void node_heap_init (void)
    {
    int i;

    for (i=0; i<20; i++)
       {
       node_heap[i].sequence = -1;
       }
    }

node * getnode_node_heap (void)
    {
    int i;
    node * temp;

    temp = NULL;
```

```
    for (i=0; i<20; i++)
       {
       if (node_heap[i].sequence == -1) //element is free
          {
          temp = &node_heap[i];
          }
       }
    }


node * getnode()
    {
    node * temp;
    temp = (node *) malloc(sizeof(node));

    if (temp == NULL)
       {
       printf("\nMemory allocation Failure!\n");
       exit(1);
       }
    else
       return(temp);

    return(NULL);
    }


void node_test (void)
    {
```

```
    node *temp1;
    node *temp2;

    temp1 = getnode_node_heap();
    temp2 = getnode();
    }


void insert_beg(int x)
    {
    node *temp;
    temp = getnode();
    temp->data = x;
    temp->next = NULL;
    if (head == NULL)
        head = temp;
    else
        {
        temp->next = head;
        head = temp;
        }
    }

void init_path (void)
    {
    node *temp;
    head = getnode();
    strcpy(head->name, "HEAD");
    temp->data = 0;
```

```
    temp->next = NULL;
    temp->next_path = NULL;
    }


void insert_path_element (int x, char *name)
    {
    //firstly determine the head of the particular path
    node *temp1 = search_path(name); //head
    node *temp2 = getnode(); //new node
    node *temp3 = NULL; //save scratch

    strcpy(temp2->name, name);
    temp2->data = x;
    temp2->next = NULL;
    temp2->next_path = NULL;
    temp2->next_outcome = NULL;

    //head first, insert new @ end of list
    while (temp1 != NULL)
       {
       temp3 = temp1;
       temp1 = temp1->next;
       }
    temp3->next = temp2;
    }


void insert_path (int x, char *name)
```

```
{
/*all paths have their own heads all heads are connected together inserting a path or
//initiating a path means connecting that paths head element to all head elements
paths are inserted at the beginning*/

node *temp, *temp1;

temp1 = NULL;
temp = getnode();
current = search_outcome (name);

strcpy(temp->name, name);
temp->data = x;
temp->next = NULL;
temp->next_path = NULL;
temp->next_outcome = NULL;

if (current == NULL)
   {
   //current = temp;
   //no corresponding outcome found
   //at this stage always assume
   //there is an outcome
   //printf("could not find name\n");
   return;
   }
else
   {
   //insert at the end of the list
```

```
        //current is the head of the list
        while (current != NULL)
           {
           temp1 = current;
           current = current->next_path;
           }
        temp1->next_path = temp;
        }
     }

//exact match on name
node * search_path (char *name)
    {
    current = search_outcome (name);

    if (head == NULL)
       {
       }
    else
       {
        while (current != NULL)
           {
           if (strcmp(current->name, name) == 0)
              {
              break;
              }
           else
              {
              current = current->next_path;
```

```
            }
        }

    if (strcmp(current->name, name) == 0)
        {
        return(current);
        }
    }
  return (NULL);
  }

void set_outcome_type (int x, int type )
  {

  }

//exact match on outcome only
void insert_outcome (int x, char *name)
  {
  /*create a new outcome at the beginning*/

  node *temp;
  temp = getnode();

  strcpy(temp->name, name);
  temp->data = x;
  temp->next = NULL;
  temp->next_path = NULL;
  temp->next_outcome = NULL;
```

```
    if (head == NULL)
        {
        head = temp;
        }
    else
        {
        temp->next_outcome = head;
        head = temp;
        //printf("%s%s\n", head->name, (head->next_outcome)->name);
        }
    }

//void insert_outcome_path_element(int x, int y, char *name)
//{
/*Insert a particular path element for a particular outcome*/
//}

//void insert_outcome_path (int x, int y, char *name)
//{
/*Insert a particular path for a particular outcome*/
//}

node * search_outcome (char *name)
    {
    /*search for a particular outcome */
    current = head;

    /*name 2nd char is the outcome id*/
```

```
    name++;

    if (head == NULL)
       {
       }
    else
       {
        while (current != NULL)
          {
          if (strstr(current->name, name) != NULL)
             {
             break;
             }
          else
             {
             current = current->next_outcome;
             }
          }

       if (strstr(current->name, name) != NULL)
          {
          return(current);
          }
       }
    return (NULL);
    }


void insert_end(int x)
```

```
    {
    node *temp;
    temp = getnode();
    temp->data = x;
    temp->next = NULL;
    if (head == NULL)
        head = temp;
    else
        {
        current = head;
        while (current != NULL)
            {
            save = current;
            current = current->next;
            }
        save->next = temp;
        }
    }



void reverse()
    {
    node *temp;
    current = save = head;
    if (head == NULL)
        {
        printf("\a\nThe Linked List is Empty!\n");
        getch();
```

```
    }
  else
    {
    save = NULL;
    while (current != NULL)
      {
      temp = save;
      save = current;
      current = current->next;
      save->next = temp;
      }
    head = save;
    printf("\nThe Linked List Is Reversed!\n");
    getch();
    }
  }


void sort()
  {
  int temp;
  current = save = head;
  if (head==NULL)
    {
    printf("\aLinked List Is Empty!\n");
    getch();
    }
  else
    {
```

```
    for (current=head;(current != NULL);current=current->next)
       {
       for (save=current->next;(save != NULL);save=save->next)
          {
          if ( current->data < save->data )
             {
                      temp = save->data;
                      save->data = current->data;
                      current->data = temp;
             }
          }
                }
    printf("\nThe Linked List Is Sorted Now!\n");
    getch();
    }
  }


//traverse the map and return the element based on key value
node * search(int kv)
  {
  node * current_outcome;
  node * current_path;
  node * current_path_element;
  current_outcome = head;
  if (head == NULL)
    {
    //printf("\a\nLIST IS EMPTY");
    //getch();
```

```
    }
else
    {
    while (current_outcome != NULL)
        {
        //search the path of each outcome for the link
        if (current_outcome->data == kv)
            {
            return(current_outcome);
            }
        //else traverse path
        current_path = current_outcome->next_path;  //new path
        while (current_path != NULL)
            {
            if (current_path->data == kv)
                {
                return(current_path);
                }
            current_path_element = current_path->next;
            while (current_path_element != NULL)
                {
                if (current_path_element->data == kv)
                    {
                    return(current_path_element);
                    }
                current_path_element = current_path_element->next;
                }//end while individual path element
            current_path = current_path->next_path;
            }//end while current path
```

```
        current_outcome = current_outcome->next_outcome;
        }//end while current outcome
      }//end else
   return (NULL);
   }//end proceedure


int delete_node(int kv)
   {
   int undel;
   current = head;
   if (head == NULL)
      {
      printf("\a\nLIST IS EMPTY");
      getch();
      }
   else
     {
     while ( (current != NULL) && (current->data != kv) )
        {
        save = current;
        current = current->next;
        }
     if (current->data == kv)
        {
        if (current == head)
           {
           undel = current->data;
           current = current->next;
```

```
            free(head);
            head = current;
            }
         else
            {
            undel = current->data;
            save->next = current->next;
            free(current);
            }
         printf("%d Value is Deleted!",kv);
         getch();
         return(undel);
         }
      else
         printf("%d value does not exists, cannot delete a non existing node",kv);
      getch();
      }
   return (NULL);
   }


void insert_mid(int n_no,int x)
    {
    int ct=0;
    node *temp;
    temp = getnode();
    temp->data = x;
    temp->next = NULL;
    current = head;
```

```
    while (ct < n_no )
       {
       save = current;
       current = current->next;
       ct++;
       }

    if (head == NULL) head = temp;
    else if (current == head)
       {
       temp->next = current;
       head = temp;
       }
    else
       {
       temp->next = save->next;
       save->next = temp;
       }
    }



int count()
    {
    int ct=0;
    current = head;
    while (current != NULL)
       {
```

```
        ct++;
        current = current->next;
        }
    return(ct);
    }


void display()
    {
    clrscr();
    current = head;
    if (current == NULL)
        printf("\aThe List Is Empty!\n");
    while (current != NULL)
        {
        printf("\n%d - %s",current->data, current->name);
        current = current->next;
        }
    getch();
    }


void empty()
    {
    current = head;
    while (current != NULL)
        {
        current = current->next;
        free(head);
```

```
        head = current;
        }
    printf("\nLinked List Is Empty Now!\n");
    getch();
    }



/*
main()
{
int ch,n,n_no;
head = NULL;
do
{
clrscr();
printf("\n0.  EXIT");
printf("\n1.  Insert the value at begining");
printf("\n2.  Display the Linked List");
printf("\n3.  Insert the value at End");
printf("\n4.  Empty the Linked List");
printf("\n5.  Count the Nodes of Linked List");
printf("\n6.  Insert the value after specified node");
printf("\n7.  Delete the Node of Linked List");
printf("\n8.  Search the Node of Linked List");
printf("\n9.  Sort the Linked List");
printf("\n10. Reverse the Linked List");
printf("\nEnter Your Choice\n");
scanf("%d",&ch);
```

```
switch (ch)
{
                case 1:

                printf("\nEnter The Value To Be Inserted At The Begining\n");
                scanf("%d",&n);
                insert_beg(n);
                break;

                case 2:

                display();
                break;

                case 3:

                printf("\nEnter The Value To Be Inserted At The End\n");
                scanf("%d",&n);
                insert_end(n);
                break;

                case 4:

                empty();
                break;

                case 5:

                printf("\nThere are %d Nodes in the Linked List",count());
```

```
getch();
break;

case 6:

printf("\nEnter The no. of node after which you"
      " want to Insert the value\n");
scanf("%d",&n_no);
if ( n_no > count() )
{
printf(" %d node(s) does not exists. There are only"
      " %d node(s) in the list",n_no,count());
getch();
}
else
{
printf("\nEnter The Value To Be Inserted after %d node(s)\n",n_no);
scanf("%d",&n);
insert_mid(n_no,n);
}
break;

case 7:

printf("\nEnter The Key Value To Be Deleted\n");
scanf("%d",&n);
delete_node(n);
break;
```

```
                case 8:

                printf("\nEnter The Key Value To Be Searched\n");
                scanf("%d",&n);
                search(n);
                break;

                case 9:

                sort();
                break;

                case 10:

                reverse();
                break;

}
}
while (ch!=0);
}
*/
```

Appendix E

Loop Transformation

A typical array element will look like:

```
//cognitive element struct
//sequence flag
//sequence = -1 if empty
struct node {
int data;
int weight;
bool state;
bool path_state;
bool outcome_state;
int polarity;
int type;
int phi;
int delta;
int s;
char name[10];
char name_delta[10];
struct node * next;
struct node * next_path;
struct node * next_outcome;
int parameter_id; //unique identifier
int parameter_type; //cc, dbs, user, cfg
int sequence;
};
```

Pseudo code for a simple sorting technique will look like:

```
//search array sequentially
//until an empty cognitive element
//is found


node * get_element (void)
{
int i;
node * temp;
temp = NULL;
for (i=0; i<num_elements; i++)
{
if (cog_array[i].sequence == -1) //element is free
                    {
                    cog_array[i].sequence = 1;
                    temp = &cog_array[i];
                    }
}
}
```

Know sorting routines which are an improvement on a simple sort are qsort and bubble sort.

Pseudo code for hybrid sorting routine would include:

```
//if the list has not been filled once yet
//(e.g. first few times the application
//is used, keep track of the next available space
//next available space is the element
//after the last filled element
next_element = cog_array_filled_once();
//when the list has been filled once,
//search between elements, using optimized
```

```
//sort for optimized results or a simple
//search
if (next_element == -1)
                    new_cog_element = get_element();
else
                    new_cog_element = cog_array[next_element];
```

Appendix F.

Intelligent Modeling and Decision Making for Product Quality of Manufacturing System Based on Fuzzy Cognitive Map

Purpose

The purpose of the project is to recreate the results of Pang's work [57] in modeling product quality of manufacturing systems. The modeling tool used is fuzzy cognitive maps (FCM). Pang creates a product quality model using the following 8 concepts – awareness, accuracy of the equipment, stability of the equipment, material strength, material hardness, system measurement, environment characteristics, feasibility of manufacturing methods. The model is used to make decisions based on the different states of the model's concepts and the effect of the concepts' contributions.

According to Pang the state and weight definition is given:

- QC1 quality awareness
- QC2 accuracy of the equipment
- QC3 stability of the equipment
- QC4 material strength
- QC5 material hardness
- QC6 system measurement
- QC7 environment characteristics
- QC8 feasibility of manufacturing methods.

We follow Pang's fuzzification of weights as presented in his paper. Generally, to obtain numeric FCM weights, 3 experts are usually consulted and their answers are combined as per a membership shown in the weight fuzzification.

Methodology

The project will proceed by concept data analysis, implementing the FCM, running the simulations, and comparing the simulations against Pang's research.

Currently there has been extensive research in FSM e.g. learning and probabilistic reasoning However, in this project research paper the basic implementation is illustrated.

Implementation

The FCM source code was adapted from the source code shown in Appendix C and Figure F.9. The source code was written in C++ and tested in a DOS environment. The input file (input.txt, Figure F.11) applied to the weight/adjacency matrix (Figure F.10) gives the output file (output.txt, Figure F.12) with the state listing per concept. These results are then compared to Pang's results in [57]).

Results and Discussion

As represented diagrammatically in Section 5.4, Equation 11, the next state of current concept is the summation of the effects of the contributing concepts (determined by multiplying their current states with their weight contribution). The value must be between 0 and 1 so an activation or "squishing" function or transfer function is applied. Upon closer examination, the transfer function was not supplied by the author. However, since the inputs can be calculated (using pervious states and the weight matrix) and the outputs are known it is possible to graph f(n) vs. n (f(n) = "squished" state, n = raw state) to determine the transfer function either by inspection (Figures F.2, F.3, F.4 show standard activation functions) or by curve fitting in Excel or Matlab.

Because the transfer function needed to be determined, the FSM code was verified using another example with all known parameters i.e. "Modeling Software Development Projects Using Fuzzy Cognitive Maps" [58] see Appendix D. The output states (output.txt, Appendix F) matched the published results in Appendix D. The proposed strategy for the determining the unknown transfer function was also verified by plotting f(n) vs. n for this test case. Figure F.5 shows the resulting plot resembling the expected sigmoid transfer function with c=5.

Figure F.1 shows the plot for attempt to determine Pang's transfer function. While the final value is constrained between 0 and 1 the figure does not match any of the standard activation functions

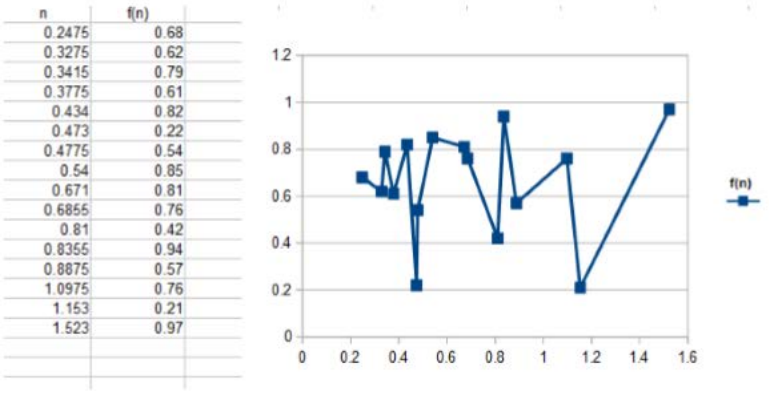
(Figures F.2 - F.4). Lee et al. researches the design of user-defined transfer functions in [59, 60] including sinusoids, so the conclusion was finally drawn that this was the case of a user-defined transfer function based on the expertise of the unique characteristics of the system being modeling. In the FCM source code, the transfer function was implemented as a multi-level step function (Figure F.4 is a 2 level step function) in the form of a lookup table (Appendix A - float f_x_lookup_user (float input)). There was no linear extrapolation between points because the available points were sufficiently distributed. The "connect-the-points" in Figure F.1 is for visualization purposes only.

Using the user-defined step function, it was possible to recreate Pang's results. Appendix C rows 0,1,2 match Iteration (1), (2), (3) in Figure F.1.

The system oscillates (this is also the case with some transfer functions e.g. sinusoids in [59, 60] with $\beta > 1$) and does not reach an equilibrium position. The individual concepts were plotted against state-number shown in Figure F.6, F.7 and F.8.

No meaningful analysis of the system is presented in Pang's original work except for generalized comments that the system accurately reflects the system being modelled.

By doing simple forward simulations (changing the value of the contributing components and observing the end value) on the system, based on inspection of Figure F.6, F.7 and F.8 the following can be observed: 1. QC8 (feasibility of manufacturing methods) settles to an equilibrium state and does not oscillate as would be expected. 2. QC2 (accuracy of equipment) oscillates with QC1 (quality awareness) and QC5 (material hardness) to reflect their positive causal contribution to QC2. 3. QC6 (system measurement) decreases with respect to Q2 (accuracy of equipment) and Q5 (material hardness), as would be expected.



n	f(n)
0.2475	0.68
0.3275	0.62
0.3415	0.79
0.3775	0.61
0.434	0.82
0.473	0.22
0.4775	0.54
0.54	0.85
0.671	0.81
0.6855	0.76
0.81	0.42
0.8355	0.94
0.8875	0.57
1.0975	0.76
1.153	0.21
1.523	0.97

Figure F.1 f(n) vs. n Product Quality of Manufacturing Systems

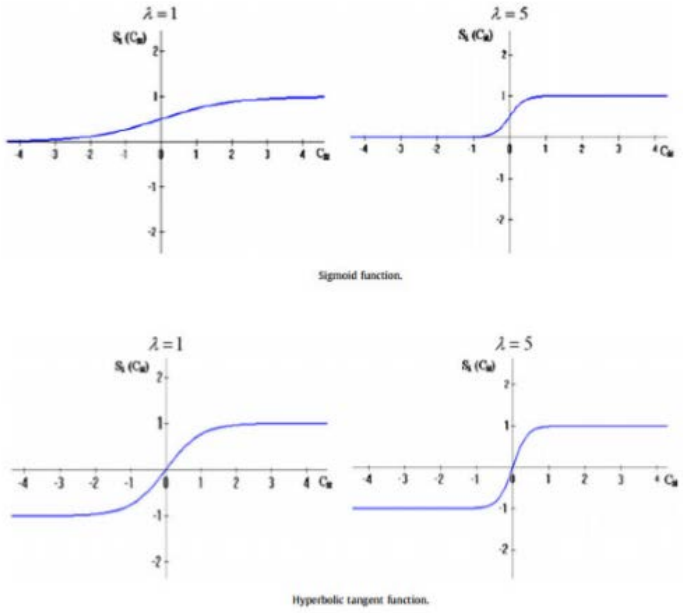


Figure F.2 Standard Activation Functions [59, 60]

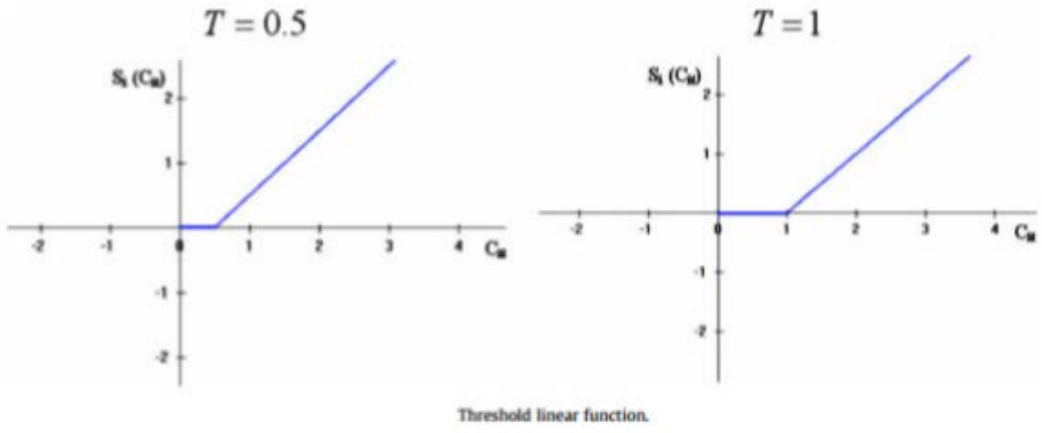


Figure F.3 Standard Activation Functions [59, 60]

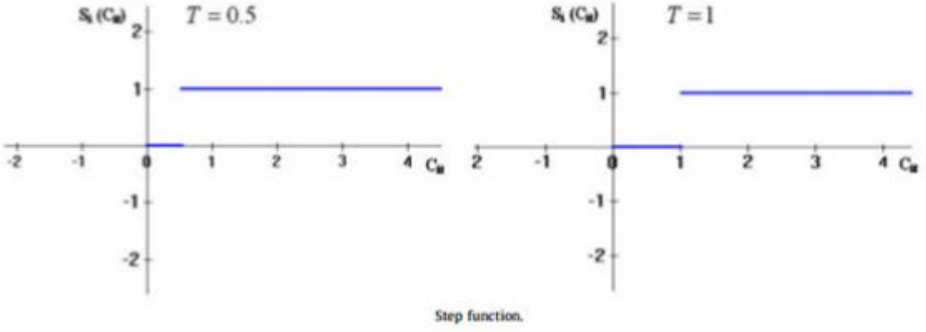


Figure F.4 Standard Activation Functions [59, 60]

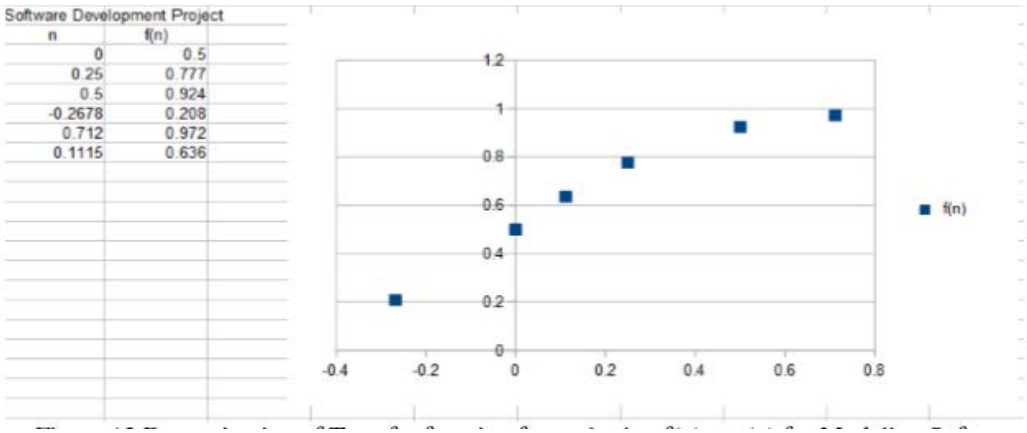


Figure F.5 Determination of Transfer function from plotting f(n) vs. (n) for Modeling Software Development (Appendix G)

state	C1	C2	C3
0	0.25	0.5	0.75
1	0.42	0.61	0.57
2	0.76	0.82	0.21
3	0.61	0.85	0.97
4	0.42	0.82	0.76
5	0.76	0.22	0.97
6	0.76	0.54	0.94
7	0.57	0.81	0.21
8	0.42	0.81	0.97
9	0.76	0.85	0.76
10	0.76	0.85	0.97
11	0.81	0.22	0.97
12	0.76	0.81	0.81
13	0.76	0.81	0.21
14	0.42	0.81	0.76
15	0.76	0.85	0.76
16	0.76	0.85	0.97
17	0.81	0.22	0.97
18	0.76	0.81	0.81
19	0.76	0.81	0.21
20	0.42	0.81	0.76

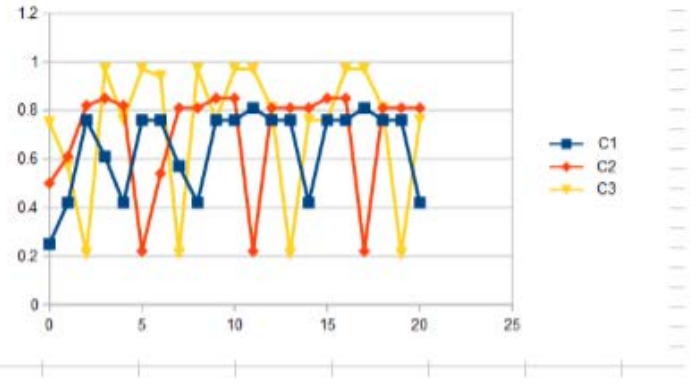


Figure F.6 C(n) vs. state(n) I

state	C4	C5	C6
0	0.5	0.35	0.8
1	0.68	0.54	0.62
2	0.79	0.81	0.22
3	0.54	0.57	0.42
4	0.82	0.42	0.85
5	0.61	0.42	0.82
6	0.68	0.76	0.62
7	0.82	0.76	0.81
8	0.22	0.57	0.81
9	0.82	0.42	0.85
10	0.61	0.76	0.22
11	0.54	0.57	0.81
12	0.62	0.76	0.82
13	0.22	0.76	0.81
14	0.22	0.57	0.81
15	0.82	0.42	0.85
16	0.61	0.76	0.22
17	0.54	0.57	0.81
18	0.62	0.76	0.82
19	0.22	0.76	0.81
20	0.22	0.57	0.81

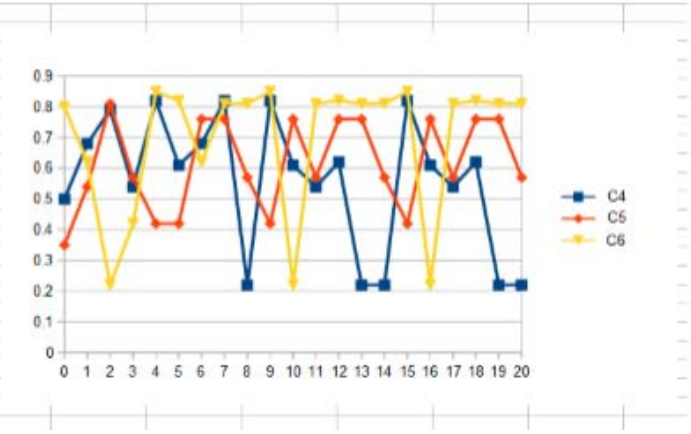


Figure F.7 C(n) vs. state(n) II

state	C7	C8
0	0.65	0.45
1	0.85	0.76
2	0.94	0.97
3	0.21	0.97
4	0.76	0.97
5	0.76	0.97
6	0.94	0.97
7	0.76	0.97
8	0.76	0.97
9	0.76	0.97
10	0.76	0.97
11	0.21	0.97
12	0.57	0.97
13	0.76	0.97
14	0.76	0.97
15	0.76	0.97
16	0.76	0.97
17	0.21	0.97
18	0.57	0.97
19	0.76	0.97
20	0.76	0.97

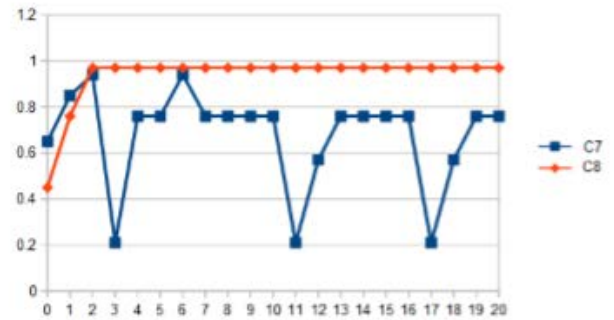


Figure F.8 C(n) vs. state(n) III

Conclusion

It was possible to recreate Pang's results by assuming a user defined activation function. It was observed that the system was not stable but oscillated i.e. the product quality of manufacturing systems varies. However, the concepts behavior matched what was expected as defined by the causal relationships in the weight matrix and the initial system state. There is also no meaningful analysis in Pang's original work.

The product quality is determined by the interrelationships of the concept variables. By doing forward simulations on Figures F.6, F.7 and F.8 it is possible to see how a change in one causal concept variable affects another corresponding resultant variable and that these interactions contribute to the overall “big picture” view of product quality of manufacturing systems.

```
float f_x_lookup_user (float input)
{
if (input <= 0.25)  return 0.68;
else if (input <= 0.33) return 0.62;
else if (input <= 0.35) return 0.79;
else if (input <= 0.38) return 0.61;
else if (input <= 0.44)  return 0.82;
else if (input <= 0.476)  return 0.22;
else if (input <= 0.49)  return 0.54;
else if (input <= 0.55)   return 0.85;
else if (input <= 0.68)  return 0.81;
else if (input <= 0.69) return 0.76;
else if (input <= 0.82)   return 0.42;
else if (input <= 0.84) return 0.94;
else if (input <= 0.89) return 0.57;
else if (input <= 1.1) return 0.76;
else if (input <= 1.2)  return 0.21;
else if (input <= 1.6)  return 0.97;
else return 0.97;
}


float f_x_lookup_exponential (float input)
{
if (input <= 0)     return 0.5;
else if (input <= 0.25) return 0.777;
else if (input <= 0.5) return 0.924;
else if (input <= -0.2678) return 0.208;
else if (input <= 0.712) return 0.972;
else if (input <= 0.1115) return 0.636;
}
```

Figure F.9 FCM source listing

8
0.25
0
0.35
0
0
0.60
0
0.15
0.15
0.50
0
0
0.40
0.25
0
0
0
0.65
0.75
0.8
0
0
0.25
0
0
0.25
0
0.5
0.35
0
0
0
0.50
0
0
0.15
0.35
0.15
0

Figure F.10 Manufacturing system quality fcm weight matrix

0.35
0
0.20
0
0
0.65
0.80
0
0
0
0.25
0
0
0.35
0
0.65
0.65
0.75
0
0
0.8
1.0
0.2
0
0.45

Figure F.11 Manufacturing system quality fcm input.txt

0	0.25	0.5	0.75	0.5	0.35	0.8	0.65	0.45
1	0.42	0.61	0.57	0.68	0.54	0.62	0.85	0.76
2	0.76	0.82	0.21	0.79	0.81	0.22	0.94	0.97
3	0.61	0.85	0.97	0.54	0.57	0.42	0.21	0.97
4	0.42	0.82	0.76	0.82	0.42	0.85	0.76	0.97
5	0.76	0.22	0.97	0.61	0.42	0.82	0.76	0.97
6	0.76	0.54	0.94	0.68	0.76	0.62	0.94	0.97
7	0.57	0.81	0.21	0.82	0.76	0.81	0.76	0.97
8	0.42	0.81	0.97	0.22	0.57	0.81	0.76	0.97
9	0.76	0.85	0.76	0.82	0.42	0.85	0.76	0.97
10	0.76	0.85	0.97	0.61	0.76	0.22	0.76	0.97
11	0.81	0.22	0.97	0.54	0.57	0.81	0.21	0.97
12	0.76	0.81	0.81	0.62	0.76	0.82	0.57	0.97
13	0.76	0.81	0.21	0.22	0.76	0.81	0.76	0.97
14	0.42	0.81	0.76	0.22	0.57	0.81	0.76	0.97
15	0.76	0.85	0.76	0.82	0.42	0.85	0.76	0.97
16	0.76	0.85	0.97	0.61	0.76	0.22	0.76	0.97
17	0.81	0.22	0.97	0.54	0.57	0.81	0.21	0.97
18	0.76	0.81	0.81	0.62	0.76	0.82	0.57	0.97
19	0.76	0.81	0.21	0.22	0.76	0.81	0.76	0.97
20	0.42	0.81	0.76	0.22	0.57	0.81	0.76	0.97

Figure F.12 Manufacturing system quality fcm state-listing, output.txt

Appendix G.

Modeling Software Development Projects Using Fuzzy Cognitive Maps

This appendix recreates the results in Stach's et al. work on modelling software development projects presented in [58]. The input (Figure G.1) and corresponding output (Figure G.2) is shown below.

```
3
0.5
0.25
-0.5
0.5
0
0.5
1.0
-0.5
0
```

Figure G.1 Modeling software development project input.txt

0	0.5	0	0
1	0.5	0.77729	0.92414
2	0.20771	0.97235	0.6357
3	0.40762	0.89173	0.19904
4	0.64954	0.82004	0.45234
5	0.47357	0.94017	0.7681
6	0.3219	0.95706	0.50435
7	0.48386	0.88751	0.31364

Figure G.2 Modeling software development project output.txt

Appendix H.

Dispatching Source Code

zones.c

```
/* Dispatching program */

#include "zones.h"
#include "message.h"

void zone_init (void) {};
void zone_inside_context (void) {};
void zone_control_sequence (void) {};
void zone_event_alert_app (int type_of_alert) {};
void zone_handle_communication (char src, char dest, char type, char *package)
{
   SendInterModelMessage(ID_ZONE, ID_COORD, MESSAGE_EVENT, package);
};
```

---

wnd.c

```
/* Dispatching program */

#include <stdio.h>
#include <stdlib.h>
#include "wnd.h"
#include "message.h"
```

```
void wnd_init (void){};
void wnd_main (void){};
void wnd_handle_event_view (void){};
void wnd_handle_communication (char src, char dest, char type, char *package)
{
    printf("e\n");
    printf("%d\n", *package);
};
```

---

vca.c

```
#include <time.h>
#include "vca.h"
#include "gps.h"
#include "message.h"

clock_t start;
clock_t end;
char data;

void vca_init (void)
{
    start = clock();

};
void vca_build_map (void) {};
```

```
void vca_context_change (void)
{
    end = clock() - start;

    if (end/1000 == 1)
        {
        //random code
        gps_get_serial_data(&data);
        //if (data != NULL)
        //    {
            //send new data
            SendInterModelMessage(ID_VCA, ID_COORD, MESSAGE_EVENT, &data);
       //     }
        start = clock();
        }

};
void vca_log (int type_of_log) {};
void vca_handle_communication (char src, char dest, char type, char *package) {};
```

---

tsk.c

```
/* Dispatching program */


#include "tsk.h"
#include "message.h"
#include "wnd.h"
```

```
#include "link.h"

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

#include "gps.h"

void sys_init (void)
{

}

void sys_main (void)
{
char src = 1;
char dest = 2;
char type = 1;
char *package;

package[0] = 0;

//get gps data
gps_get_serial_data(&package);

//do zone analysis

//forward the info to the task for windowing display
tsk_handle_communication (src, dest, type, package);
```

```
}


void tsk_init (void)
{
//    coord_init();
    sys_init();
};

void tsk_main (void)
{
    //coord_main();

    sys_main();
}; void tsk_mgr_handle_event_view (void) {};

void tsk_handle_communication (char src, char dest, char type, char *package)
{
if (src == ID_COORD)
    {
    printf("d\n");
    wnd_handle_communication(ID_TSK, ID_WND, type, package);
    printf("---------------\n");
    }

};
```

message.c

```
/* Dispatching program */
/* Responsible as the mail center - routing */

#include "message.h"
#include "coord.h"

void SendInterModelMessage(char src, char dest, char type, char *package)
{
   if (dest == ID_COORD)
      {
      //incoming
      coord_handle_communication(src, dest, type, package);
      }
};
```

---

main.c

```
/* Dispatching program */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include "tsk.h"
```

```
#include "cog.h"

void main(int argc, char* argv[])
{

demo();

/*
    char c = getch();

    tsk_init();

    if (c == '1')
        {
        demo();
        //fcm_demo();
        }
    else
       {
       while (1)
           {
           tsk_main();
                       }
        }
*/
}
```

---

gps.c

```
/* Dispatching program */
#include <conio.h>
#include <time.h>
#include "gps.h"


int random_integer;



void gps_get_serial_data(char *str)
{
//check data ready flag
//if data is ready return gps string
//else return str = NULL
gps_driver();
*str = (char) random_integer;
};


char * gps_convert_gps_data (char *str) {return NULL;};
void gps_add_correction_info (char *str){};
void gps_init (void)
{
   srand(time(NULL));

};
```

```
void gps_driver (void)
{
//this is responsible for reading the gps data
//over the serial link
    random_integer = (rand() % 3) + 1;
//    printf("%d\n", random_integer);

};
```

---

coord.c

```
/* Dispatching program */

#include "coord.h"
#include "zones.h"
#include "message.h"
#include "tsk.h"


void coord_init (void)
{
    vca_init();
    gps_init();
};
void coord_main (void)
{
```

```
    vca_context_change();
};
void coord_to_controller_context_event_change (void) {};
void coord_to_app_context_event_change (void) {};
void coord_handle_communication (char src, char dest, char type, char *package)
{
if (src == ID_VCA)
    {
    zone_handle_communication(src, dest, type, package);
    printf("b\n");
    }
if (src == ID_ZONE)
    {
    printf("c\n");
    tsk_handle_communication(ID_COORD, ID_TSK, type, package);
    }
};
```

Appendix I.

Case Study: Human Activity Extraction and Fuzzy Cognitive Maps – A Systems Perspective

Numeric Fuzzy cognitive maps comprise concepts and weights (Figure 1.1). The value of the concepts is determined by the equation below (1) [31, 32]. This section introduces a numeric fcm case-study based on a systems perspective of human activity extraction. Additional numeric fcm case studies are Pang's et al. work on decision making for product quality in manufacturing systems [57] described in Appendix F and Stach's et al. work on modelling software development projects [58] presented in Appendix G.

$$\quad (1)$$

The FCM design is also characterized by the choice of transfer function (2, 3, 4) [38,40]. In [4], c controls how fast a concept saturates.

Bivalent $\quad (2)$

Trivalent $\quad (3)$

Logistic Signal $\quad (4)$

Severity Fuzzy Cognitive Map

The severity FCM is a predictor of medical incidence. The more severe the case the more likely there is to be a medical incident occurring. The severity FCM (Figure I.1)

could potentially be incorporated as a firmware module (inputs from devices and user) as part of a cognitive application layer as a means of adding cognition to the firmware. The FCM was developed in consultation with experts.

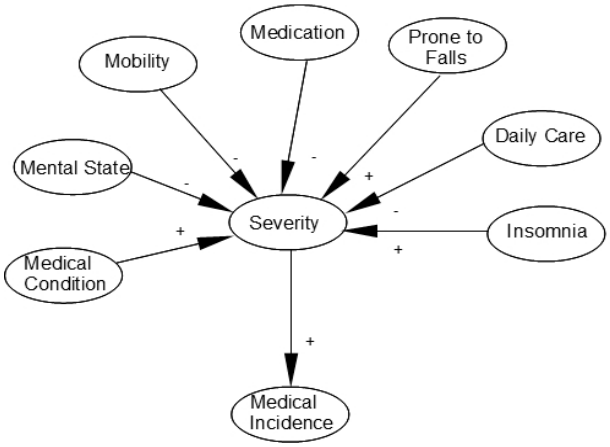


Figure I.1 Severity FCM

Mobility as wells as detection of falls can be determined using biomedical devices. The below section highlights research in human activity extraction techniques.

Human Activity Extraction

Recent advances in human activity extraction include the use of fuzzy qualitative and quantitative approach [33, 34, 35], decision tree classifiers [36, 37, 38, 39], artificial neural networks [36, 39, 40], statistical analysis [40, 41] and state-machines [42, 33] and the concept of context-aware computing [2], activity-aware computing [43] and affordance based resource allocation [9].

The below sections spotlights some of the most recent research. In [42] van de Ven et al. introduces the use of a state machine to summarize relative time spent sitting, lying, standing and walking but with no detailed implementation description.

In [34] Nii et al. human activity extraction research using fuzzy rules and in [33] Pantelopoulos et al. researches fuzzy finite state machines as part of the Prognosis, a fuzzy regular formal language.

This section introduces human activity state-machine extraction with novel detection of abnormal conditions. State machines have low computation costs and battery consumption for low power wireless solutions e.g. Zigbee. Abnormal conditions can be detected using e.g. an integrated multi-sensor which can detect heart rate and temperature. Abnormal conditions that can be detected using these physiological parameters are illness (fever), dead (no heart rate), pain (increased heart rate, reduced mobility). An increase in sedentary time (lying, sitting) as opposed to mobile time (standing, walking, running) is an indication of reduced mobility, corrected for cases of illness or pain. The human activity is characterized from observing the x, y, z coordinates of the accelerometer from 5 seconds of activity and based on the work done by Nii et al in [34].

Based on the graphical traces Table I.1 was extracted. The state machine is shown in Figure I.2.

Table I.1 Position vs. accelerometer reading

g	x	y	z
lying	-1	-0.33	0.25
sitting	-0.2	-1	0.17
standing	0.25	-1	0
running	$2 < x < 3$	$0.75 < y < 2.33$	2.33
walking	$0.83 < x < 2$	-0.17	$0.83 < z < 2$

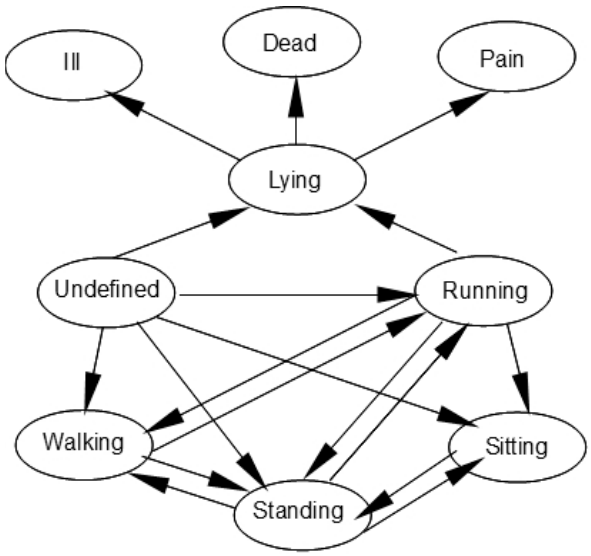


Figure I.2 Activity State Machine

Role of Biomedical Devices in Health-Care

In the IEEE Spectrum article “The Doctor will see you ALWAYS” by J. Smith [61], the usage of biomedical devices is explored for ambient monitoring and specifically its role in chronic disease management (which eats up 75% health care spending or US $1.9 trillion annually). The article highlights specific existing and potential future device usage scenarios. Highlighted in the article are a personal sleep coach (for monitoring sleep cycles, online if required) insulin pumps with glucose monitoring, wireless enabled scale, smart phone applications for chronic disease management and tele-health solutions (physiological, mental and medication monitoring). A 2008 trail run of a tele-health solution with veterans who experience multiple chronic conditions has found that device usage catches complications early on and has reduced hospital admissions by 19% and days spent in health care facilities by 25% [44].

Most recent population projections suggest that the senior population of British Columbia (B.C.), Canada, will double in the next 20 years. This will lead to in-creased usage of health care facilities and home and community care services (HCC- Home Care

Community Services and Accommodation Environments) as well as social services [45, 46].

Elderly people are not only afflicted by physical problems associated with aging but also mental illness (suicide, dementia, distress, stress) and substance abuse problems which require the use of health care systems for proper care [47].

Existing commercial solutions help family member monitor aging relatives as well as empowering professional caregivers by using sensors to track daily activity levels as well as a falls. Daily as well as routine changes and detection of asymptomatic behavior could indicate medication complications, congestive heart failure worsening or depression [44].

Friends and family provide $2.7 billion in free care and are the primary health-care system in Canada. [48, 49, 50]. The Statistics Canada 2008 report Eldercare estimates that 1 in 5 Canadians over 45 were caring for a senior in 2007 where 25% of senior caregivers are seniors themselves. A 1999 US study has found elderly caregivers of spouses have a 63% higher mortality and if a caregiver gets sick, more people entering the facility. Studies show stress of care-giving can be debilitating, harmful physical, mental, emotional consequences. In Canada, there are lack of measures available to ease the financial, emotional and psycho-social repercussions on caregivers [51].

Based on the literature review done above and the information provided by experts the below preliminary cognitive map with polarities was developed (Figure I.3).

Because actual numeric information was available from the data sources linguistic fuzzy cognitive weights were tentatively added to allow for further preliminary simulation analysis. As mentioned in the previous section there is currently no governmental policies to ease the financial, emotional and psycho-social effect on caregivers so this fuzzy linguistic weight is considered undefined.

The following nodes are used:

- C_0, Aging Population – the number of seniors in the population.

- C_1, Medical Incidence – senior related medical incidence, falls, depression, adverse medication effects e.g., forgetting to take medications, heart-attacks, strokes, addictions etc.

- C_2, Caregiver Load – number of man-hours involving in the care of the aged including primary care personnel, family members, temporary relief as well as the nature of the care-giving work and its debilitating effects.

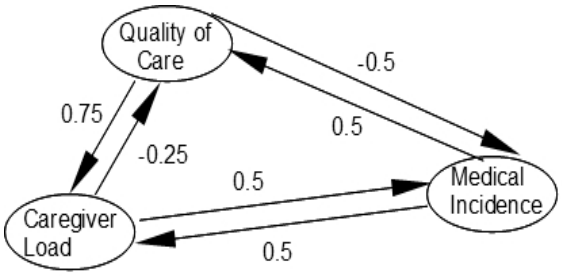


Figure I.3 Health-care FCM

The causal relationships can be interpreted as follows. An increase in quality of care results in a decrease of preventable medical incidence via greater supervision for example falls, forgetting to take medication, dementia patients wandering off e_{01}. An increase in medical incidence leads family member to look to improve the quality of care whether it be through additional help, government initiatives or new technologies e_{10}. The increase in medical incidence due to deteriorating health increases the caregiver load by increasing the tasks to perform as well as the number of hours needed to supervise the elderly e_{12}. caregivers of the elderly who are themselves elderly have higher medical incidence rates because of the debilitating effect of care-giving e_{21}. As the load on the caregiver increases the quality of care is compromised e_{20} and an increase in quality of care, increases the caregiver load e_{02}. The weights were established based on interviews with experts and literature reviews.

The developed model was simulated and the results of the dynamic simulation are shown in the figure below. A logistic signal was used with c = 5. The starting vector V_0 = (C_0, C_1, C_2) = (0.5, 0.375, 0.25).

The simulation is presented in below states and Figure I.4.

V_0 = (0.5, 0.375, 0.25)

V_1 = (0.651, 0.348, 0.943)

V_2 = (0.423, 0.674, 0.964)

V_3 = (0.617, 0.794, 0.963)

V_4 = (0.617, 0.794, 0.963)

V_5 = (0.628, 0.679, 0.987)

V_6 = (0.614, 0.71, 0.982)

V_7 = (0.633, 0.715, 0.983)

The model requires 13 states to achieve equilibrium V_{13} = (0.631, 0.707, 0.984).

The starting vector represents the state of the model at the beginning of care and the anticipated medical incidence as well as burgeoning caregiver load.

Figure I.4 Simulations on health-care FCM

The model shows that the caregiver load node achieves the highest value in the first state C_1. This reflects the situation where care-giving starts with potentially an informal caregiver (family member) and there is a drop in preventable medical incidence and the load of the caregiver rises significantly due to the learning curve and determining the expectations and extent of the work required. At maximum care-giver load, the quality of care drops triggering a rise in medical incidence. This could be seen as trying to adapt to the tasks which may be varying, the caregiver being an elderly spouse or elderly child who themselves is overwhelmed with stress, and the determination that there needs to another tier of help.

As the states progresses, under maximum care-giver load the quality of care is seen to oscillate due to the effect of medical incidence and at one-point drop below an acceptable level of care before settling to the equilibrium value at average quality of care, slightly higher than the initial state. The medical incidence peaks after the effect of drop in quality of care is fully realized and eventually settles to a value that is much higher than the original state. This could be interpreted to mean that as time progresses with maximum care giver load constraining the quality of care the medical incidence will still be high be it due to deteriorating condition of the elder or the elder caregiver as they struggle to maintain a consistent minimum acceptable level of care. This significant increase in medical incidence in light of quality of care would suggest that a other factors need to to

be present to achieve a higher quality of care and to drive down medical incidence and caregiver load.

The fuzzy cognitive map was augmented to include additional variables to investigate further improving the quality of care. The new FCM is shown below. New concepts introduced include:

- C_3, Biomedical Devices – devices catered toward the elderly market, this concept represents device adoption and usage. Include are automatic medication dispensers, wireless activity tracker, sleep tracker, wireless infra-red alarm, bed pad alarms, emergency medical alert dialer with wireless help panic transmitter, cameras for tele-health etc.

- C_4, Health-Care funding – government funds allocated to seniors related care, tax-credits for biomedical device purchases, tax-credit senior-related home renovations, government subsidized community and home-care services, caregiver tax credit, family caregiver tax credit.

The meaning and relationships between the previous nodes remain the same. The new relationships are described as follows. The adoption and usage of medical devices result in better diagnosis and disease management as well as staving off complications resulting better quality of care e_{30} and a reduction in preventable medical incidence (fall detection and precipitating of immediate assistance) e_{31}.

Likewise, with an increase in falls, the adoption of a biomedical device is more likely e_{13}. Additionally, as the quality of care increases biomedical devices could be employed to supplement monitoring and provide relief for caregivers e_{03}. The use of biomedical devices reduces the caregiver load by providing more efficient care and in some cases reducing or eliminating (tele-care) the number of hours needed to supervise the elderly e_{32}.

The well documented need of overwhelmed caregivers also provides a market driving the development of biomedical devices e_{23}. With access to more health care governmental resources and programs the quality of care increases e_{40}, tax-credits results

into savings for the elderly person or family caregiver which can then be translated into reducing the care-giver load e_{42}, or the financial means to invest in means into products, equipment, home renovations for reducing medical incidence e_{41}.

Governmental initiatives, grants, foster research, development and innovation in biomedical devices e_{43}. The increased elder-related medical incidence results in the government investing into more scientific research e_{14} as well as the need to alleviate the burden of a growing taxpayer base – the caregiver whose roles provides considerable relief on government agencies and programs e_{24}.

The weights were established based on interviews with experts and literature reviews (Figure I.5).

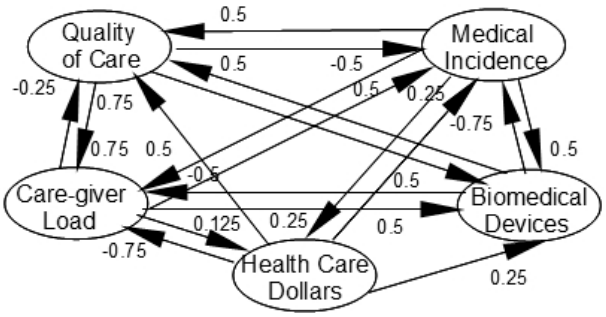


Figure I.5 Expanded FCM

The results of the dynamic simulation are shown in the figure below.

V_0 = (0.5, 0.375, 0.25, 0, 0.25)

V_1 = (0.826, 0.422, 0.867, 0.957, 0.651)

V_2 = (0.991, 0.064, 0.336, 0.997, 0.744)

V_3 = (0.993, 0.011, 0.196, 0.988, 0.572)

V_4 = (0.987, 0.006, 0.297, 0.976, 0.534)

V_5 = (0.983, 0.008, 0.326, 0.98, 0.548)

V_6 = (0.984, 0.009, 0.31, 0.981, 0.553)

V_7 = (0.984, 0.009, 0.306, 0.981, 0.551)

The model requires 11 states to achieve equilibrium V_{11} = (0.984, 0.009, 0.309, 0.981, 0.551). Only the first 7 states are depicted graphically (Figure I.6).

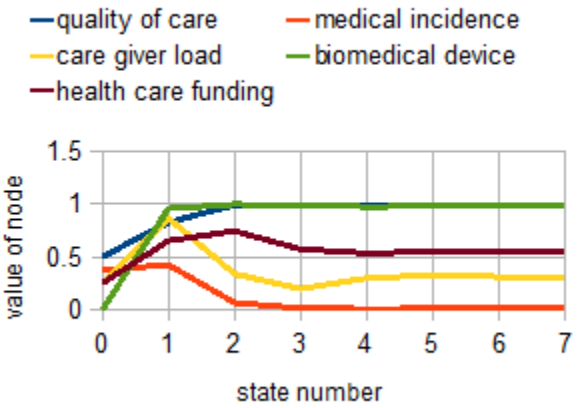


Figure I.6 Dynamic Simulation I

Trends observed with the inclusion of new concepts include a slight increase and then sharp drop in the pattern of medical incidence as well as the consistent increase in the quality of care. The peak in the caregiver load in the first state could be due to the time to research and receive available governmental funding, or source and setup the biomedical devices and the learning curve associated with device usage. In the subsequent states the increase in health care funding and use of biomedical devices drives down the caregiver load.

The healthcare funding curves shows the initial heavy subscription to available resources and then the settling out as these resources become maxed out (the one-time purchase of biomedical devices, completion of home renovations). This simulation represents where the care-giver is knowledgeable of existing governmental programs and technological solutions. Eventually the caregiver load significantly dips and then rises slightly reflecting the situation where the usefulness of the biomedical devices is fully realized in easing caregiver load. At this point biomedical devices need to keep pace with the market and address the current needs.

The second simulation uses the end states of the first state-machine, to see the effect of the additional concepts after the equilibrium state of the first fcm is achieved. V_0 = (0.631, 0.707, 0.984, 0, 0.25) and the model requires 11 states to achieve equilibrium V_{11} = (0.984, 0.009, 0.309, 0.981, 0.551). The graph shows the reduction in caregiver load initially. This simulation would represent the case when only after being overwhelmed the caregiver begins to consider other solutions (Figure I.7).

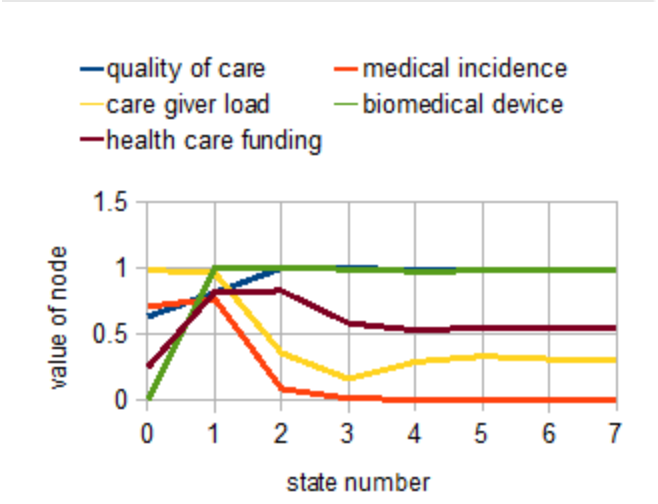


Figure I.7 Dynamic Simulation II

Previous knowledge of governmental programs allows for a significant decrease on the care-giver load and medical incidence as the care-giver taps into what is available.

Adjusting the starting vector V_0 = (0.631, 0.707, 0.984, 0.25, 0.25) to allow for the knowledge of biomedical solutions as well and the model requires 11 states to achieve equilibrium V_{11} = (0.984, 0.009, 0.309, 0.981, 0.551). The graph below shows immediate decrease in medical incidence (Figure I.8).

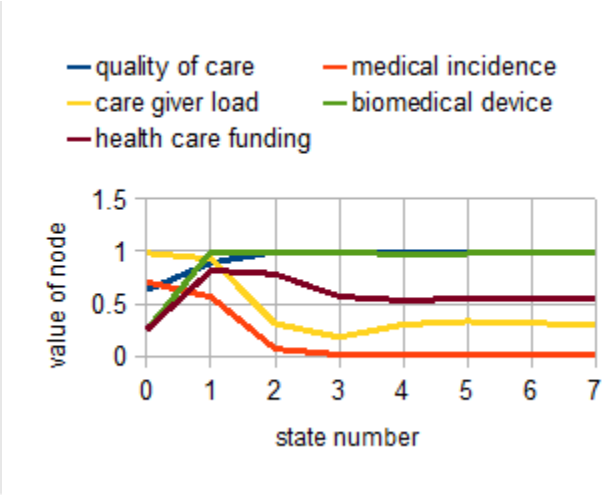


Figure I.8 Dynamic Simulation III

The simulation results of the two fuzzy cognitive maps show the temporal nature of the analysis used for an effective visual representation for all the stake-holders. Biomedical companies can look at this information and see the need to develop products that are as user-friendly as possible and with due consideration to human factors. Government agencies can look at this information to determine adequate promotion of available resources as important to the effectiveness of the programs intentions.