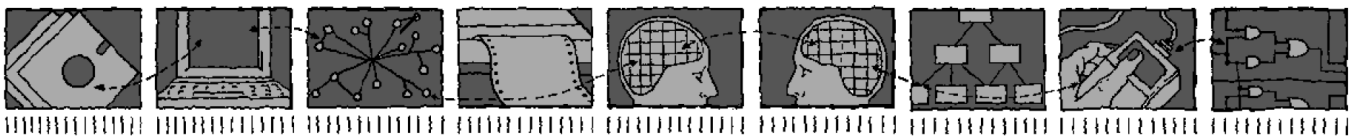
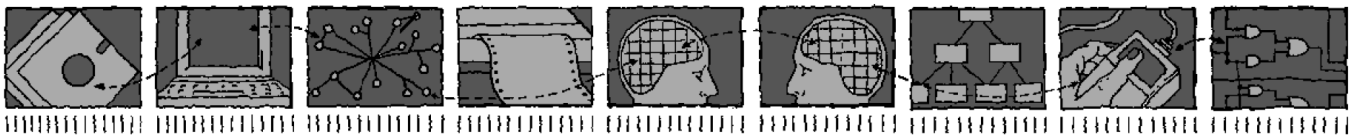


Department of Computing Science and Mathematics
University of Stirling



Handling Emergent Conflicts in Adaptable Rule-Based Sensor Networks

Jesse Michael Blum

Technical Report CSM-196

ISSN 1460-9673

October 2012

Department of Computing Science and Mathematics
University of Stirling

Handling Emergent Conflicts in Adaptable Rule-Based Sensor Networks

Jesse Michael Blum

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland

Telephone +44 1786 467 421, Facsimile +44 1786 464 551
Email jmb@cs.stir.ac.uk

Technical Report CSM-196

ISSN 1460-9673

October 2012

Abstract

This thesis presents a study into conflicts that emerge amongst sensor device rules when such devices are formed into networks. It describes conflicting patterns of communication and computation that can disturb the monitoring of subjects, and lower the quality of service. Such conflicts can negatively affect the lifetimes of the devices and cause incorrect information to be reported. A novel approach to detecting and resolving conflicts is presented.

The approach is considered within the context of home-based psychiatric Ambulatory Assessment (AA). Rules are considered that can be used to control the behaviours of devices in a sensor network for AA. The research provides examples of rule conflict that can be found for AA sensor networks.

Sensor networks and AA are active areas of research and many questions remain open regarding collaboration amongst collections of heterogeneous devices to collect data, process information in-network, and report personalised findings. This thesis presents an investigation into reliable rule-based service provisioning for a variety of stakeholders, including care providers, patients and technicians. It contributes a collection of rules for controlling AA sensor networks.

This research makes a number of contributions to the field of rule-based sensor networks, including areas of knowledge representation, heterogeneous device support, system personalisation, and in particular, system reliability. This thesis provides evidence to support the conclusion that conflicts can be detected and resolved in adaptable rule-based sensor networks.

Acknowledgements

Doing post-graduate research and writing a thesis can be a long and challenging process. It is with humble gratitude that I wish to acknowledge the help and support that I received along the way.

Firstly, it gives me great pleasure to thank Professor Evan Magill for his guidance and encouragement. Evan provided me with a good deal of freedom to pursue my interests, but was also always able to steer me in the right direction. We had a great deal of stimulating conversations that I will remember fondly with a chuckle. I would also like to thank Dr. Mario Kolberg for his advice and the discussions we had.

One of the best things about the University of Stirling is the friendliness of our institution. I have really appreciated the interactions I've had with our staff and students. When I first started mulling over undergraduate topics with Professor Ken Turner, for instance, little did I know that I would continue to work here for years to come. Yet, it was his encouragement that propelled me on to post-graduate work, and I really appreciate the feedback and guidance he has provided me over the years. Other staff, such as Dr Savi Maharaj, Dr Carron Shankland, and Kate Howie, have warmly invited me in to their offices in order to look at my work and provide me with extremely valuable insights. I learned a lot from formal research chats and seminars, of course, but it was the invigorating discussions I had with Dr David Cairns, Dr Marwan Fayed, Dr Andrea Bracciali, and Kevin Swingler that helped make my time at Stirling memorable. Not only were Graham Cochrane, Sam Nelson, Grace McArthur, Linda Bradley, and Frank Kelly always very helpful, but they were also good humoured and made being at Stirling an enjoyable experience.

Our wee gang of PGTipsters changed over the years but was always close. Dr Paul Godley helped show me the ropes for getting research done and lecturing masters students. I always knew that I could count on the support of Dr Larry Tan and Dr Gavin Campbell, and I look back on our games of Pool, Go, and Settlers fondly. Russell Hunter and I had great conversations about quantum computing, neuron spiking, and many other topics. Basketball with Soufiene Benkirane and tea with Rosalyn Porter were always fun, and I could always count on their support.

Special mention, of course, needs to go to the clique: Andy "Dave" Abel, Jamie "The Cheerful Pirate" Furness, and Claire "Shadow" Maternaghan. We had great times together and I really appreciate that they listened to my issues and the valuable encouragement they offered. In particular, I am indebted to Andy, my colleague and office-mate, for the many, many hours he spent cheering me up, and for listening to me ceaselessly rant about the word processor that must not be named. He saw me through a number of troubles, from moving house one Hallowe'en night, to picking my mom up from the airport in the middle of a terrible blizzard.

I consider it an honor to have worked with my colleagues on the PAM and DAMES projects. I thank the Engineering and Physical Sciences Research Council (EPSRC) for generously funding my PhD work

and for the support that I have received.

Lastly, and most importantly, I owe my deepest gratitude to my family. My mother Sara Blum, my father Wayne Blum, and my grandmother Charlotte Sugarman were always there to listen to my concerns, encourage me to keep going, and even provide editorial feedback. My children, Ayden and Sophia, inspired and amazed me every day. Most of all, though, I want to thank my wife, Brit Wellens, for her faith in me, endless patience, and love.

Thank you all,

Jesse

Contents

List of Figures

Listings

Abbreviations and Acronyms

A-C	Antecedent-Consequent
AA	Ambulatory Assessment
AD	Activate Delayed
ADL	Activities of Daily Living
ADT	Automatic Data Transfer
ADTOU	Alternative Data Transfer On Unavailable
AI	Activate Immediate
AIert	Alert
API	Application Programming Interface
ASD	Activate Sensor Delayed
ASI	Activate Sensor Immediate
BCRU	Broadcast Control Rule Update
BDT	Buffered Data Transfer
BSN	Body Sensor Network
BVH	Brief Visit Home
CAES	Context Aware Experience Sampling
CDS	Context Detection Service
CFB	Call Forwarding when Busy
CFU	Call Forwarding Unconditional
CLIPPER	CoLlaborative Information Processing Protocol and Extended Runtime
CRUH	Control Rule Update Handling
CS-EMA	Context Sensitive Ecological Momentary Assessment
CSIP	Collaborative Signal and Information Processing
CSV	Comma Separated Value
CTS	Context Triggering System
DaM	Data Management
DCS	Data Collection System
DD	Deactivate Delayed
DeM	Device Management
DFU	Data Forward Unconditional
DI	Deactivate Immediate
DND	Do Not Disturb
DNMU	Do Not Monitor Unconditional

DNMX	Do Not Monitor X
DNNU	Do Not Notify Unconditional
DNNX	Do Not Notify X
DQC	Data Quality Control
DRF	Data Recording Frequency
DSTGP	Data Storage Through Group Processing
DSTP	Data Storage Through Processing
DSU	Data Storage Unconditional
DTOU	Data Transfer On Unavailable
E-C-A	Event-Condition-Action
EC	Event Calculus
ED	Environment Detection
EMA	Ecological Momentary Assessment
EPSRC	Engineering and Physical Sciences Research Council
ESM	Experience Sampling Method
ESP	Experience Sampling Program
EVDO	Evolution-Data Optimized
FI	Feature Interaction
GPRS	General Packet Radio Service
GPS	Global Positioning System
HCI	Host Controller Interface
HRF	Human Readable Format
ICUS	Incoming Control Update Screening
IDS	Inbound Data Screening
iESP	Intel Experience Sampling Program
L2CAP	Logical Link Control and Adaptation
LAN	Local Area Network
LD	List Devices
LI	Looping Interaction
m-health	Mobile Healthcare
MAI	Multiple Action Interaction
MTI	Missed Trigger Interaction
MU	Monitor Unconditional
MX	Monitor X

NU	Notify Unconditional
NX	Notify X
OCS	Originating Call Screening
ODS	Outbound Data Screening
OS	Operating System
OSGi	Open Services Gateway Initiative
Prompt	Prompt
PAM	Personalised Ambient Monitoring
PDA	Personal Digital Assistant
PIU	Personal Information Update
PM	Power Management
PMAT	Purdue Momentary Assessment Tool
PS	Poll Sensor
QMI	Query for Missing Information
RD	Register Device
RDS	Redirect Data Stream
RDSA	Report Device States All
RDSX	Report Device State
RDTOU	Retry Data Transfer On Unavailable
RFID	Radio-Frequency Identification
RL	Report Location
ROP	Rule-Oriented Programming
RSS	Report Subject State
SAI	Sequential Action Interaction
SDP	Service Discovery Protocol
SDS	State Detection Service
SI	Subject Interaction
SPF	Set Poll Frequency
STI	Shared Trigger Interaction
STS	State Triggering System
SyA	Symptom Analysis
TCRUX	Transmit Control Rule Update X
TS	Time Synchronisation
UML	Unified Modeling Language

VHP	Vertical Handover Performance
WHM	Wearable Health Monitor
WSN	Wireless Sensor Network
WSNs	Wireless Sensor Networks

Chapter 1

Introduction

The main technical challenge [of digital healthcare] remains
being able to monitor and analyse
activities of daily living to inform decisions
about changes in patterns of activity
to bring health benefits.
— The Royal Society [Society, 2006]

1.1 Introduction

This thesis presents a study of conflicts that can emerge amongst low-level sensor device rules when such devices are formed into networks to perform complex tasks such as Ambulatory Assessment (AA). The thesis has been written in consideration of a body of work in areas including Feature Interaction (FI) for call control, home automation, and AA. In particular, the underlying research was carried out in collaboration with the Personalised Ambient Monitoring (PAM) project. PAM was an Engineering and Physical Sciences Research Council (EPSRC) funded pilot study that involved researchers from the University of Stirling, the University of Southampton and the University of Nottingham. The project team investigated the feasibility of reducing the incidence of debilitating psychiatric episodes through personalised ambient monitoring of patients in their homes. The study involved an infrastructure composed of on-body and environmental sensors that performed longitudinal monitoring of subjects.

This research began as a study into how Wireless Sensor Network (WSN)s could be used in AA. Assessment personalisation was selected as a key concept, and the research concentrated on questions concerning rule-oriented programmability of system rules. The work further concentrated on detecting and resolving conflicts that could emerge between device rules. An approach based on the Event Calculus (EC) (described in chapter 2) was developed to improve device reliability. This thesis contributes this approach including device rules for tasking a WSN with monitoring and information processing duties. It also provides analytical tools for discovering potential inconsistencies in the usages of the rules.

The starting point for the approach presented in this thesis is work conducted as part of the PAM project to perform ambient monitoring for the mentally ill and in particular those with Bipolar Disorder. Rule-based sensor network infrastructure was developed to monitor with the explicit consent of the patients. Moreover it is expected to report its findings to the patient and so all the data facts and rules regarding the monitoring components are open to inspection by the patient and support staff. The infrastructure was developed to allow various devices for individuals and to be registered with the network and change over time as an individual's mental state changes. The approach presented in this thesis is an attempt to consider the nature of conflicts that can emerge in a rule-based sensor network and to describe how some of these can be minimised or resolved.

The approach presented in this thesis is part of the PAM project's feasibility study. It is not to be taken on the order of a full clinical trial, but rather on the examination of a small technical trial of equipment that the PAM team performed on themselves and a small-scale trial involving a single patient.

The approach is a justified response to the particular characteristics of the type of monitoring required by the users of the PAM project.

Performing reliable behaviour monitoring is a research challenge of our age, as the quotation at the beginning of this chapter suggests. The social sciences and medical professions have long used interviews and questionnaires as the gold-standard collection instruments for their studies, but as Stone *et al.* [2007] pointed out, a growing number of researchers have rejected these in favour of more reliable instruments. The problem with interviews and questionnaires is that they rely on human recall, which has been shown to be untrustworthy. People can intentionally lie and cheat. Usually, however, they make mistakes owing to biases in their memories, confabulations, and failures to follow procedures accurately. If scientific inquiry into how people think, feel, and behave was limited to using questionnaires, then what it means to be human might never be understood very well. Fortunately, questionnaires are not the only tools available to researchers anymore.

A newly emergent set of methods called AA analyses behaviours, physiological changes, experiences and environments of people as they go about their daily lives. Mobile and sensor technologies are being used independently to study the health and safety of people in their natural environments. The combination of mobile phone technology with home monitoring devices, may however, be able to provide us with greater understanding of subject mood and behaviour.

1.2 Conceptual Overview

This section provides an initial overview of topics that will be explored in more detail in the rest of the thesis.

Heterogeneous devices are capable to some degree of sensing, computing, storing data, and communicating with each other (and the outside world) as part of an *ad hoc* network[1]. Such devices may be mobile or stationary, and they may draw power from a variety of sources. When networked together each device can be thought of as a node in a network. Each device has an Operating System (OS) programmed in a low-level language to control hardware. Some of the nodes are capable of supporting rule engines to control their behaviours along with the behaviours of the less capable nodes.

Figure 1.1 shows an example network for monitoring subject behaviour and physical activity. The figure combines the hardware infrastructure elements (PAM-i) with software architecture (PAM-a) elements to depict four interlinked monitoring environments for on-body, environmental, mobile phone and personal computer monitoring. The figure shows that each of the four areas has a device capable of communicating with one or more of the other devices. The mobile phone is shown to support a rule engine and is programmed through its rules.

Declarative languages may be used to express device rule programs, which can be intuitive for domain expert use. For instance, a rule for monitoring a room might be expressed as "Stop monitoring if the subject is not present". Collections of such rules can be formed into programs to control the devices in a WSN but conflicts may emerge amongst the device rules. This thesis shows that five types of conflicts (Sequential Action Interaction (SAI), Looping Interaction (LI), Missed Trigger Interaction (MTI), Shared Trigger Interaction (STI) and Multiple Action Interaction (MAI)) of concern to the FI community can arise in such WSNs.

A novel approach to programming sensor networks is presented herein that uses rules written in a declarative language called the EC to load actions into devices. This approach requires access to all the device rules in the network in order to analyse them in such a way as to minimise the risk of various forms of conflicts when collectively capturing and processing data. This thesis considers the analysis approach using example rules for device control, contextual monitoring and notification.

To contextualise sensor networks for behaviour monitoring, consider a scenario involving researchers interested in studying the impact of Bipolar Disorder on subjects conducting their usual activities of daily living. Bipolar disorder is a severe psychiatric disorder characterised by patients being in patterned (possibly cyclic and/or recursive) affective states, including mania, hypomania, eurythmia, depression and mixed states. Not only may the quality of their lives may be reduced significantly by the disorder, but, according to Yatham *et al.* [2004] and Das Gupta & Guest [2002], there are also high costs for society to pay.

[1] In addition, nodes may also actuate changes in the environment (such as opening and shutting windows). Actuation, however, is outside the scope of this thesis and only information processing capabilities are considered here.

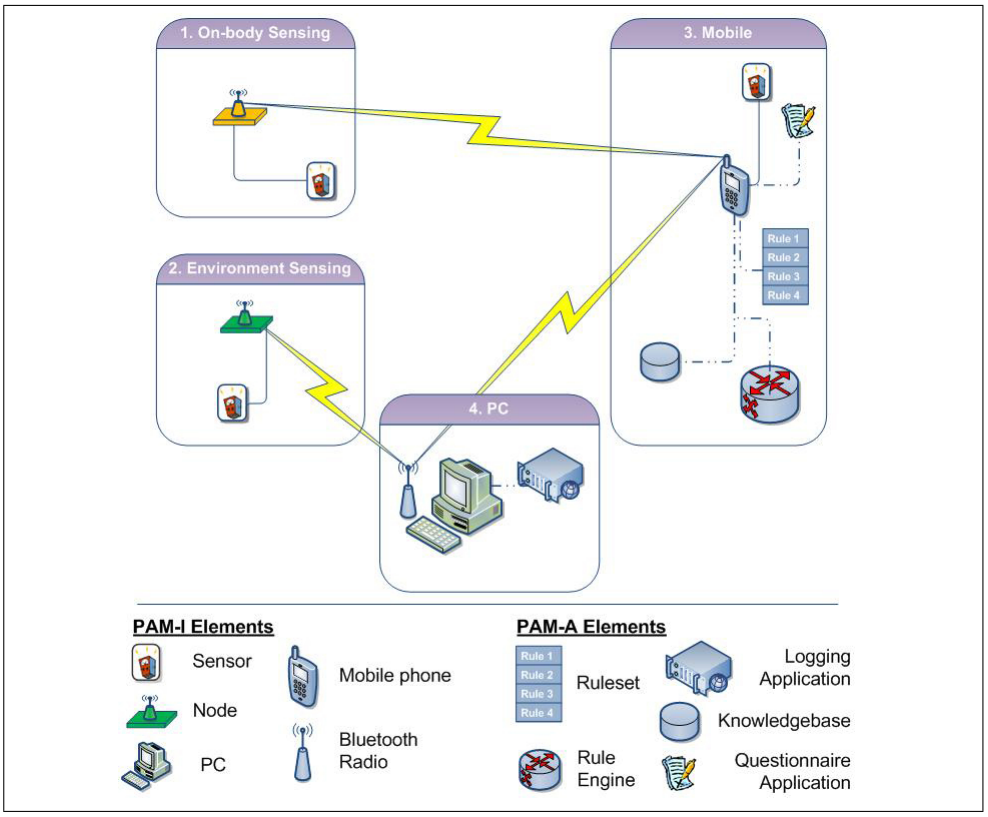


Figure 1.1: WSN for personalised monitoring.

Sensor networks can be used by researchers to better understand the patterns of a person's behaviour. Researchers, for instance, may be interested in observations relating to prodromes that lead to manic or depressive episodes, as well as the frequency and duration of episodes. They can place small inter-communicating sensors in the homes of subjects and provide them with wearable sensors and mobile phones. These devices may be independently programmed by various team members using different rules to control data sampling frequencies, communication, analysis and storage. This thesis discusses the importance of personalisable device configurations in AA because subject reactivity to devices causes skewed or inaccurate data. Reactivity occurs when subjects vary their behaviours owing to reactions to the monitoring methods according to Fahrenberg [2006]. Reactivity problems can be reduced through the use of device configurations personalised to the subjects. For instance, subject schedules could be taken into account when prompting for information. Whilst earlier approaches to AA provided mechanisms for setting certain configuration options, the approach presented in this thesis goes further by suggesting a rule-orientated device programming method that can be used to personalise a greater range of system aspects.

1.3 Research Questions

Given the views presented above this work examines the following questions.

1. What rules can be used to control the behaviours of devices in a sensor network for AA?
2. Can examples of rule conflict be found for AA sensor networks?
3. How can AA sensor network rule conflicts be detected and resolved?

1.4 Thesis Statement and Research Objectives

This thesis proposes an approach to detect and resolve conflicts in rule-based sensor networks where device control rules, knowledge management rules, conflict detection rules and device-level priority rules are loaded into an analytical framework. The framework evaluates the control and knowledge rules against the conflict rules to determine which of the evaluated rules conflict with each other. To resolve the conflicts, device-level priority statements are loaded into the analytical framework and evaluated to determine which of the conflicting rules should be disabled in a given situation.

It is claimed that a conflict detection and resolution approach to the configuration of sensor networks, based on device rule evaluation, provides network design, implementation and usage benefits and is a suitable approach to modelling rule conflicts and resolutions.

To answer the previously stated research questions, the objectives of this research are:

- To determine and describe rules necessary and sufficient for AA sensor networks
- To examine AA sensor network examples for rule conflict
- To demonstrate and evaluate an approach to detecting and resolving rule conflicts within AA sensor networks

1.5 Contributions

This thesis contributes a study into detecting and resolving five types of conflicts (SAI, LI, MTI, STI and MAI) that can occur in WSNs. The thesis also contributes rule conflict investigation tools developed from the EC, a logic language that can be used to reason about temporal activities. This thesis presents a study into how these tools can be used to detect and resolve conflicts for digital healthcare and other complex monitoring tasks. The conflict investigation tools were developed to test the hypotheses described in the previous section. These were used to support positive conclusions about the hypotheses. Evidence is provided that shows that independently programmed device rules may lead to conflicts, but that it is possible to detect and correct them.

This work is the first to study rule conflicts in AA WSN systems. AA concerns provide novel and interesting challenges for WSN conflict detection. AA has gained ground as a method used in the behavioural sciences, however researchers such as Collins & Muraven [2007] have described technological barriers that have limited previous AA research. Device interaction is a threat that has heretofore been ignored by such researchers because of their focus on the use of individual devices. Given the increasing availability of sensors, it is reasonable to foresee that researchers will want to interconnect a variety of such devices in order to gain greater insights into their subjects. This, in fact, is the vision proposed by Intille [2007] of Context Sensitive Ecological Momentary Assessment (CS-EMA) which is discussed further in chapter 3. Device conflicts will increasingly become problematic in AA as researchers rely on a greater number of devices for their studies. Previous FI solutions, however, have not been used with these systems, and new approaches to establishing their reliability are required.

1.6 Thesis Organisation

The complexities of AA WSN systems are introduced in the first part of this document. Concepts in the areas of rule programming, EC, conflict analysis, WSN, and AA are reviewed in chapter 2. These concepts lay the foundation for the following chapters. Chapter 3 identifies AA rules for latter analysis. Chapter 4 provides details of the rules, describes their organisation into device control and knowledge exchange groups, and provides examples of conflicts that can emerge from their usage.

The second part examines a solution to conflict problems in WSN systems. Chapter 5 describes the theory behind conflict detection and resolution for WSN systems. Chapter 6 provides evaluation details of the conflict detection and resolution. Chapter 7 summarises the thesis and describes future work.

1.7 Summary

Performing reliable behaviour monitoring of subjects as they go about the activities of their daily lives is an important goal for society. Devices may be combined into WSN solutions to perform AA. In doing so, we must ensure that they are personalisable and work together in a manner that minimises conflicts between them.

This thesis presents an approach to detecting and resolving conflicts that can arise in such systems. It shows how rule sets can fulfill the goals of AA, but that these can conflict unless detected and resolved.

Chapter 2

Background

2.1 Introduction

This chapter describes background information concerning the concepts used in the rest of the thesis. In particular it describes rule-based programming, conflict detection and resolution, the Event Calculus (EC), Wireless Sensor Network (WSN) programming and Ambulatory Assessment (AA).

2.2 Rule-Based Programming

Rules can be used to control sensor networks and can simplify the reasoning processes about the network and the subjects of inquiry. The following is an examination of the meaning of rules and the benefits they bring to sensor network programming.

The term "rules" is short for the term "production rules". These were originally used to define formal grammars. In addition, rules have long been used to program intelligent systems as reported by Negnevitsky [2002], to program expert systems such as those discussed by Gonzalez & Dankel [1993] and Jackson [1999], along with, more recently, to program sensor networks, such as was reported by Terfloth *et al.* [2006]. Rules for all of these purposes are axioms of first-order predicate calculus used in connection with other axioms. They contain two parts: an antecedent and a consequent. These parts are connected within an inferential *if-then* framework such that *if an antecedent is true then the consequent should also be inferred as true.*

A declarative approach to programming systems can be used to chain inferences together to come to conclusions given starting facts about the world of discourse. Facts and rules are generally stored in accessible knowledge bases. In addition to reasoning, declarative approaches may descend from the realm of pure logic to actuate changes in the world, such as through specialised *if-then* frameworks like the one described by Fei & Magill [2008]. These type of rules are often viewed as event-orientated and, if so, are known as Event-Condition-Action (E-C-A) rules. Although syntax for various systems may differ, they can usually be read as: *for a given event, if a condition (or situation) is true then a particular action should be performed.* The perspective of what a rule is (E-C-A or Antecedent-Consequent (A-C)) may seem trivial, but it impacts on the flexibility of the rule system. The perspective also influences whether or not a system requires a hierarchy of rule types (as was used in the goals and policies work reported by Turner & Campbell [2009]), or whether rule composition suffices to describe a complete system. The work in this thesis mainly concentrates on the latter, more traditional approach.

There are alternative forms for expressing rules. For instance, the purely logical form is:

The consequents c_1 or $c_2$or ... or c_m are to be inferred if the antecedents a_1 and a_2 and ... and a_m are true.

This can also be expressed in clausal form as:

c_1 *or* c_2 *or ... or* $c_m \longleftarrow a_1$ *and* a_2 *and ... and* a_m.

The Horn clause subset, as described by Kowalski [1974], limits the number of consequents of a rule to, at most, one. This has a number of benefits regarding the simplification of the logic such as reducing the state space. In clausal form this looks like:

$c_1 \longleftarrow a_1$ *and* a_2 *and ... and* a_m.

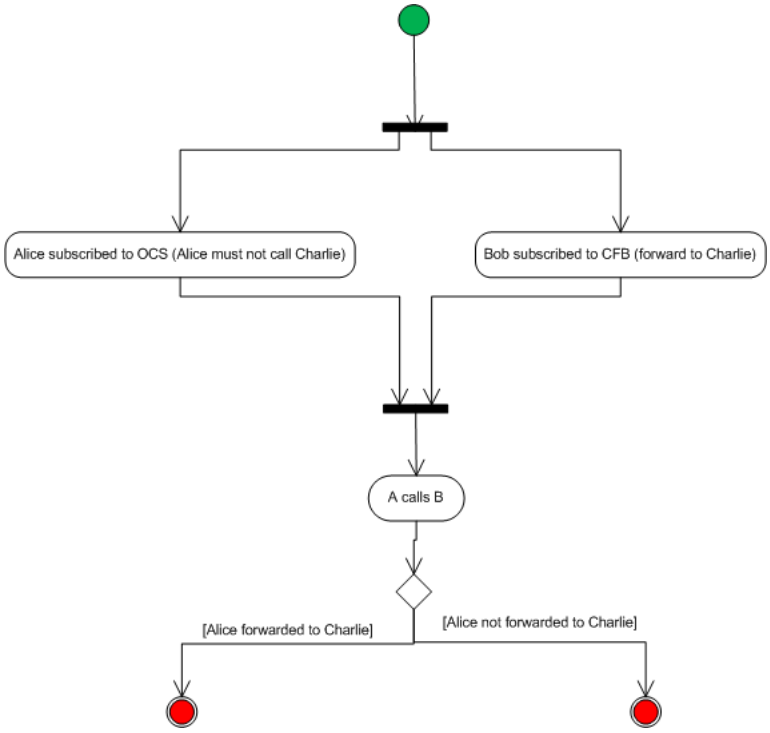


Figure 2.1: The Interaction of OCS with CFB.

The Prolog programming language is used throughout this document. The authors of Prolog, Colmerauer & Roussel [1996], reported that their initial goal was to develop a tool that could use first-order predicate logic to analyse the syntax and semantics of the French language. What they developed became a system that is able to both represent knowledge (facts and rules) and interpret execution of rules given facts. Prolog uses certain conventions which will be adhered to in this document, including that known atoms begin with lower case letters as distinct from variables (logical placeholders) that begin with an upper case letter or an underscore. In addition, Horn clause subset rules are used in Prolog, and these are represented by replacing $\longleftarrow$ with ":-" as in:

c_1 :- a_1 *and* a_2 *and* ... *and* a_m.

2.3 Conflict Detection

Rules controlling multiple interacting devices can conflict with each other, that is to cause each other to behave in manners that violate their requirements, specifications, or the assumptions of their users. Such conflicts are not software bugs in the traditional sense, since the rules are operating correctly within the context of the individual devices. Detecting and resolving these conflicts within sensor networks is the main concern of this thesis, and has been informed by a form of system feature conflict of concern to researchers since the 1980s, called Feature Interaction (FI). FI researchers are concerned with problems within systems that are exposed when services and features are composed. FI was originally discussed by Bowen *et al.* [1989] with regard to telecommunication systems. Subsequently, FI has been extended into other domains such as the examination of Internet applications by Crespo *et al.* [2007] and home automation by Wilson *et al.* [2007] and Nakamura *et al.* [2009].

A feature is defined by Calder *et al.* [2003a] as "... a component of additional functionality – additional to the core body of software." The core body of the software can be viewed as the services offered by the software; each service offering groups of features. Marples [2000] defined FI as: "The change in operation of any Feature which can be attributed in part or in whole to the presence of any other Feature within the operational environment." Marples goes on to point out that this definition ignores the FI quality[1].

[1] The quality of FI is the degree to which the FI brings advantages or disadvantages to the system.

Diagrams based on UML 2 activity diagrams are used throughout this document to depict examples of FI and rule conflict. These diagrams may contain the following elements:

- A single circle marks the initial state
- Black bars mark the splits or joins of concurrent activities
- Rounded-edged ovals represent activities
- Diamonds indicate optional paths
- Arrows represent the ordering of the activities
- Labelled invariant conditions within square brackets above arrows
- Dashed arrows indicate missing action caused by a conflict or interaction
- Double-edged circles mark the final states

A classic example of FI in telecommunications is depicted in figure 2.1. The example involves the user Alice subscribed to the feature Originating Call Screening (OCS), screening out calls to the user Charlie. The user Bob is subscribed to the feature Call Forwarding when Busy (CFB), forwarding calls to Charlie when busy. As is clear from the figure, an interaction occurs when Alice calls Bob when Bob is busy, because either the call from Alice is forwarded to Charlie, thereby invalidating OCS, or else the call is blocked, thereby invalidating CFB. In either case, the operation of one of the two features is invalidated by the presence of the other. This example demonstrates that multiple agents may be involved in FI. However this is not a requirement, and other examples are discussed that involve single devices or single users.

FI causation is reported in Cameron *et al.* [1994]. The main categories of causes are: violation of assumptions, network support limitations, and problems native to all distributed systems. Although that paper concentrates on call control networks, many of the lessons can be adapted for the types of sensor networks investigated in this work. For instance Cameron *et al.* pointed out that there may be timing problems (such as race conditions), and resource access problems. Certainly, modern sensor network designers need to be aware of these types of problems.

Features are often added by different developers at various stages of the software life cycle, which is problematic because features tend to be tested in isolation rather than in full integration with active environments. FI, therefore, requires more considered solutions than manual inspection of programming instructions or unit testing features. Three groups of FI solutions are described by Calder *et al.* [2003a]: focused software or service engineering approaches, offline formal methods for requirements or specification analysis, and online detection and resolution techniques. In addition a fourth group of solutions is offered that uses a hybridisation of offline and online techniques, which was further investigated by Calder *et al.* [2003b].

Online approaches are necessary in order to support extensible systems that are future proofed. They are, however, usually tightly connected to a given network, have trouble dealing with distributed logic, and they lack *a priori* knowledge. Such difficulties can lead to high resource usage when resolving a large number of feature conflicts. Online approaches, such as that of Kolberg [2004] have often had service privacy as a key goal and have attempted to avoid the direct exchange of sensitive business information. In contrast, the approach in this thesis requires direct access to the rules, however this is justified given that the envisioned device network pertains to AA, and therefore, the various device rules should be open for inspection by the subject and care providers.

Marples [2000] defined two categories of online techniques that can be used in FI analysis: feature managers and negotiation strategies. Feature managers are message analysis agents that review all messages of a network and determine actions to be taken (such as message passing or destruction). Feature managers tend to be centralised, although Calder *et al.* [2003a] hypothesised that distributed architectures for feature managers were also possible. Negotiation strategies allow features to intercommunicate in order to resolve interactions that may emerge through their usage. Wilson [2005] also suggested that indirect and arbitrated negotiations were possible such that a third party could be part of the negotiations.

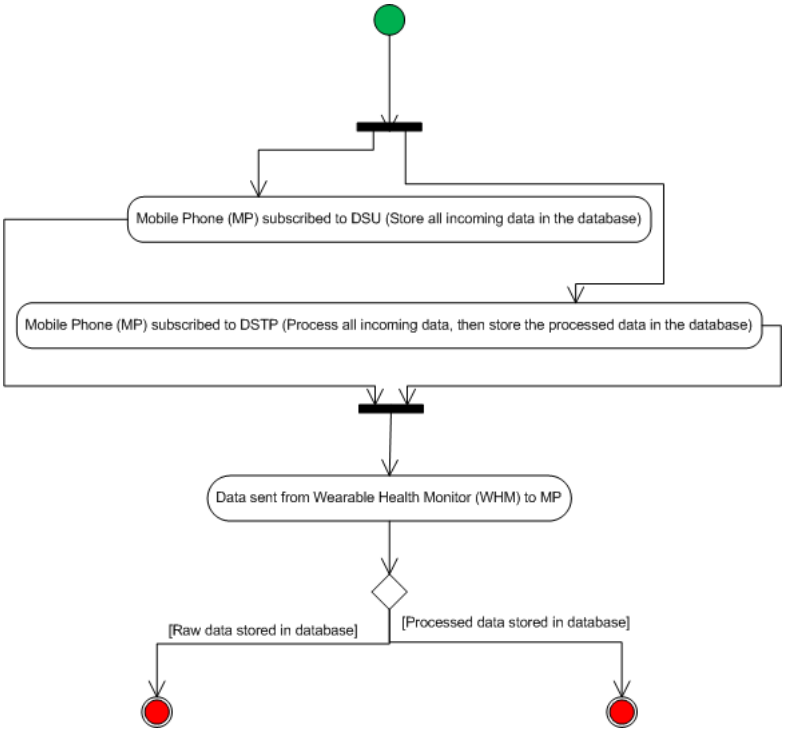


Figure 2.2: AA STI example. A mobile phone is subscribed to the competing features DSU and DSTP leading to STI.

Various types of FI have been reported. This thesis considers an extended form of Marples' FI taxonomy as described by Wilson *et al.* [2007]. It consists of five types of interactions: Shared Trigger Interaction (STI), Sequential Action Interaction (SAI), Looping Interaction (LI), Multiple Action Interaction (MAI) and Missed Trigger Interaction (MTI). The following describes them in more detail.

2.3.1 Shared Trigger Interactions

STI can be defined as the antecedents of multiple features being satisfied such that they each perform actions in response to the the same triggering event, and the operation of one or more of the features is different from how it would have reacted had it been the sole responder. For example, STI will occur in the AA example depicted in figure 2.2. If Alice's mobile phone is subscribed to Data Storage Unconditional (DSU) and Data Storage Through Processing (DSTP) then, when data arrives from Alice's wearable health monitor, the operation of one of the features will not be executed since either the raw data or the processed data would be stored in the database. The type of data (raw or processed) to be stored is unclear, and the determination will result from a race condition involving the speed that the two features receiving their data and performing their operations.

Multiple features, however, do not always interfere when responding to the same triggering event. In the above example, for instance, if the features write to separate databases, no operational problems will arise from them both reacting to the same triggering event.

2.3.2 Sequential Action Interactions

The definition of SAI is the operation of a feature triggered in response to the actions of another feature. Such chaining of features may indeed be qualitatively desirable, however they are still a form of FI. An example of SAI is depicted in figure 2.3. In this example, the mobile phone belonging to Alice (the user) is subscribed to the feature DSU. Alice wears a device called a Wearable Health Monitor (WHM) which is subscribed to the feature Automatic Data Transfer (ADT) configured to send data to Alice's mobile phone. When a signal is detected by the WHM, the device's ADT feature sends the data to the mobile phone, triggering its DSU feature. Whilst a desirable example, it is nevertheless an interaction between

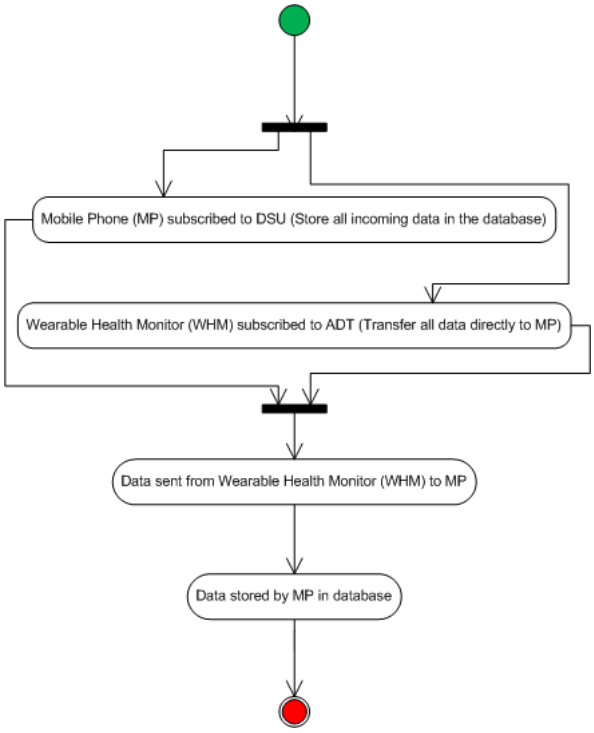


Figure 2.3: AA SAI example. A mobile phone is subscribed to the feature DSU and a wearable health monitor is subscribed to feature ADT. This leads to SAI.

the features. This potential for desirability, along with the interaction being spread across multiple devices, makes SAI difficult to resolve.

2.3.3 Looping Interactions

LI can be defined as a special case of SAI whereby the operation of the chained features leads to a redundant cycle. That LI is a special case of SAI is in agreement with Wilson [2005], who explained that SAI can potentially be beneficial, but LI will never be.

An example of LI in AA is given in figure 2.4. Here, the user Alice's mobile phone is subscribed to the feature Data Forward Unconditional (DFU) such that when Alice is at home the data stored on the mobile phone is forwarded to a data sink. The data sink is subscribed to the feature Data Transfer On Unavailable (DTOU) which is configured so that data sent to an unavailable sink will be redirected to Alice's mobile phone for temporary storage. In this case, an infinite loop occurs whereby DFU causes the phone to send data to the sink, which is unavailable and thus DTOU returns the data to the phone. This results in an unnecessary flooding of the network and the storage capabilities of the phone.

2.3.4 Multiple Action Interactions

If multiple features attempt to provide instructions for the same device, then the features are said to interact by way of MAI. In some cases the interaction may be benign, such as when both services send the same instruction to the device. Alternatively the interaction may be intolerable, such as when the services send conflicting instructions. An example of the latter is displayed in figure 2.5. In this example, a mobile phone is registered with services for not disturbing the user and for time synchronisation. If Do Not Notify Unconditional (DNNU) is activated then the Time Synchronisation (TS) message cannot be activated. Alternatively, if activation occurs, then the DNNU is not being respected. These services interact because they are both trying to control the same device at the same time.

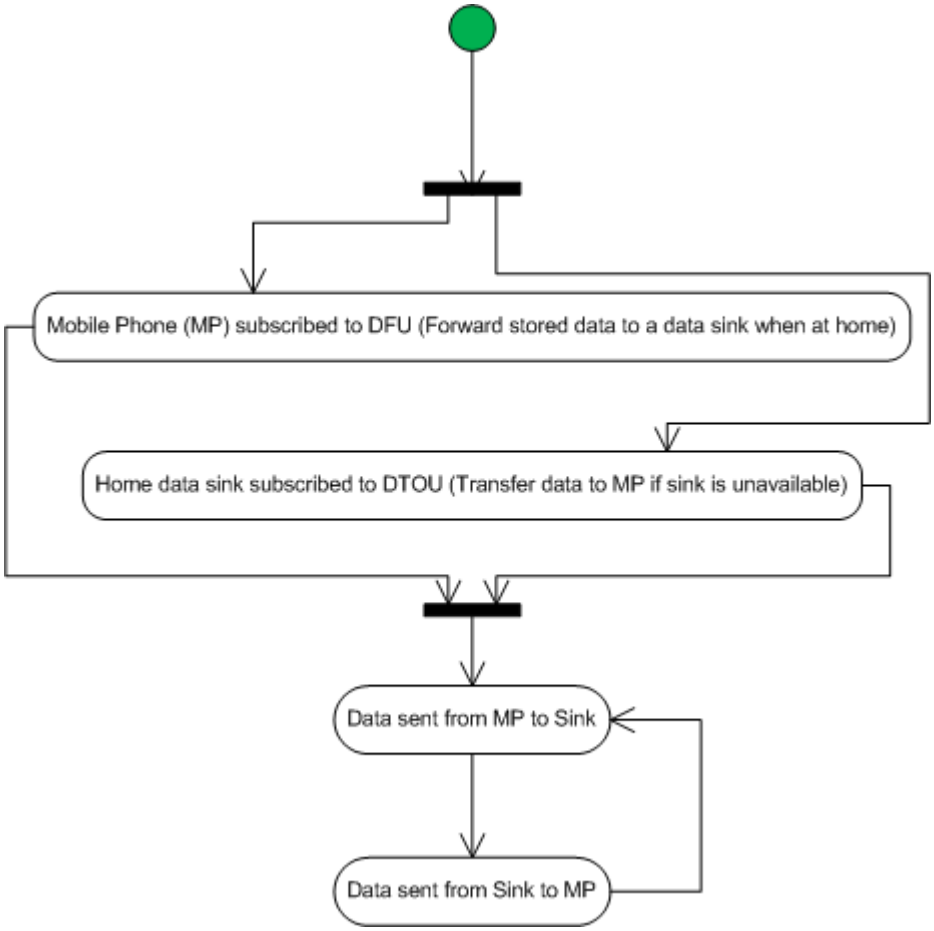


Figure 2.4: AA LI example. A mobile phone is subscribed to the feature DFU and a home data sink is subscribed to feature DTOU. This leads to LI

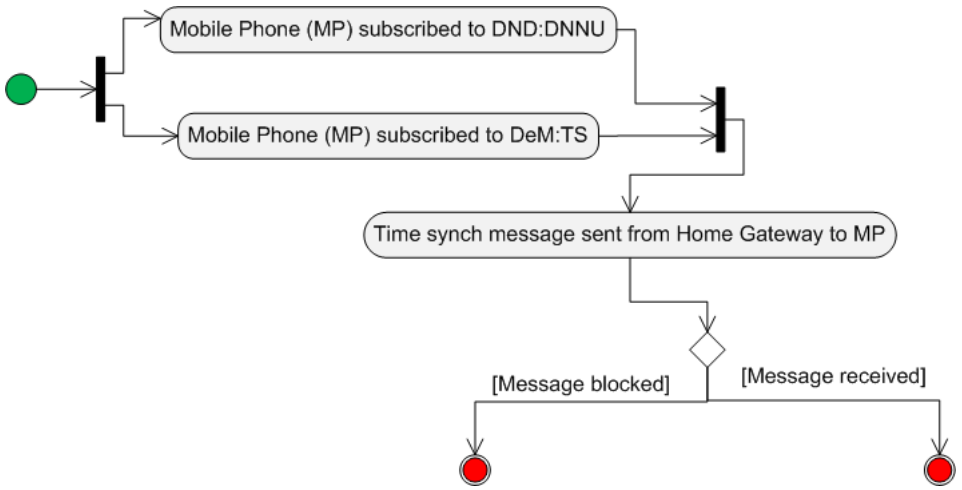


Figure 2.5: AA MAI example. A mobile phone is subscribed to the features DNNU and TS. This leads to MAI.

2.3.5 Missed Trigger Interactions

The definition of a MTI is the operation of a feature that prevents the triggering of another feature.

MTI detection and resolution have escaped previous online and hybrid approaches to resolve FI. The closest result in the literature was made by Kolberg [2004]. He reported that MTI was theoretically detectable, but it was not detected at run time owing to limitations in the underlying system architecture used for his experiments.

2.4 Pairwise and n-way Interactions

The study of feature interaction originally examined telephony call control features. Numerous methods of analysis were developed, some of which focused on pairwise (or 2-way) interactions, while others attempted to elaborate to 3-way or more generally n-way interactions.

3-way or greater interactions are hardly ever observed in telephony, as noted by Calder & Miller [2006]. Furthermore, many n-way interactions can be described using combinations of pairwise interactions. For example, in the contest results reported by Kolberg *et al.* [2000], the entry by Plath & Ryan [2000] shows that only one possible grouping of the given features could allow for a 3-way interaction possibility where pairwise had not previously been found. This was the combination of the features call waiting, terminating call screening and voice mail. However, no combination of these features leads to a 3-way interaction. Outside of telephony, 3-way or greater interactions are also rarely reported. Tsang *et al.* [1997] provides 3 examples of 3-way interactions, however these can still be detected using pairwise analysis. Given these considerations, the work presented in this thesis has concentrated on pairwise analysis.

2.5 Policy conflict

Policy conflict is an area relating to FI. Policies are statements of general principles that govern system wide behaviour in reaction to conditions and events. They differ from the work presented in this thesis in that they apply to a system as a whole, whereas, the work in this thesis concentrates on low-level device rules. Still, an examination of policy conflict analysis is of interest as some of the issues are of concern to both areas.

Chomicki *et al.* [2003] describe an E-C-A based policy description language and a conflict and resolution system for it. For them, a policy is in conflict when its set of actions cannot occur together as defined by the policy author. This is limiting in two ways. Firstly, it says nothing about conflicts that can emerge between multiple policies. Secondly, the form of conflict is similar to MAI described below, but ignores other forms of conflicts that can emerge such as MTI. They describe priority ordering applied to the actions of policies to resolve conflicts. While, the work in this thesis also applies priorities it is important to note that the two systems are not similar. Applying priorities to actions from a single policy implies that the policy author is aware that the actions may conflict with each other. It is therefore better for said authors to compose non-conflicting actions in the first place. On the other hand device rule authors may wish to express priorities for their devices knowing that rules on their device may conflict with other rules, but without knowing about the other conflicting rules.

Authorisation policies are appropriate for an organisation, but make little sense within an AA context where the resources belong to a self-monitored individual. Such works which include those by Masoumzadeh *et al.* [2007] that focus too extensively on detecting and resolving authorization policy conflicts, are of little interest to the focus of this thesis. Obligation policies are little more than E-C-A rules defining what should happen and when. The types of conflicts that they examine are constrained to their authorisation/obligation model and tend to focus exclusively on modality conflicts, which can be considered alternate forms of MAI. Wang & Turner [2008] describe policy conflicts in an area more in line with AA, which is home care. They describe using E-C-A authorisation and obligation policies to perform domestic health monitoring and home automation. Their policy system is reminiscent of that described by Lupu and Sloman and no case is made as to why one would wish to use authorisation and obligation policies in a home care setting. They describe three additional forms of conflict not considered by Lupu and Sloman. Two of these suggest that alternative calculi may be better choices for their rule systems; the EC may be a better choice for concerns regarding *Dependency among Situations* and I*nteractions*

between Actions over Time. It is unclear that the third concern, *Multiple Stakeholders*, is a conflict. Rather it is perhaps a root cause of conflicts, but is not a concern of the work in this thesis. It may be that policy conflict detection models, such as those described in Blair & Turner [2005], are better for organisational domains such as telephony, whereas domains similar to AA may require a different way of thinking about rules than focusing on authorisation and obligation.

Lupu & Sloman [1999] concern themselves with conflicts in policies for distributed systems. They use a management agent approach that concentrates on two types of policies, those that authorise (or forbid) a manager to perform activities on a set of resources and those that oblige the manager to do a set of things. By agent, they mean an automated component that interprets policies. From this point of view the mediator component discussed in this thesis can be seen as an agent that interprets rules. However, this view is perhaps overly liberal. Jennings [2000], based on the work of Wooldridge [1997] describes agents to be computer systems within a particular environment with enough problem solving and environmental reactivity capabilities to accomplish autonomously tasks within the environment. Although advances in agent based programming have been made to the point where they are being deployed in home care settings, examples such as Isern *et al.* [2011] show that the autonomy envisioned by Jennings and Woolridge are not mature yet. The work in this thesis is at a similar level in that rules are programmed to control devices and allow the devices to respond via reified actions determined by the rules.

2.6 The Event Calculus for Conflict Analysis

The EC is related to the Situation Calculus described by McCarthy & Hayes [1969]. The greatest difference between them is that the EC was designed for use with time periods, whereas the Situation Calculus maintains a perspective of global states. Kowalski & Sergot [1986] discussed that they originally designed the EC to circumvent the frame problem of the Situation Calculus. The frame problem refers to the difficulty of handling dynamism in a domain without enumerating conditions unaffected by actions. The solution most commonly used in the EC is called circumscription, which is sometimes also called the rule of inertia. This rule states that a fluent that holds at a given point in time will continue to hold until a point in time when it is explicitly made not to hold. There have also been attempts to rectify the frame problem through extensions to the Situation Calculus, yet some differences remain between the approaches. For instance, Provetti [1996] argued that whilst the extended Situation Calculus and the EC may agree in large part, the EC has a more intuitive terminology for event-driven applications. Given this argument the EC was chosen over the Situation Calculus for investigation in this thesis. The form of the EC used in this thesis was derived from Wilk [2004].

The EC consists of three main concepts: *fluents*, *actions* (or *events*), and *time points*. Fluents are properties of the universe of discourse that can change in time. These properties may either take a form such as "the subject is in the house" or a quantifiable form, for instance the level of ambient sound in a room. A fluent can hold at a given point in time if it was previously initiated by an action and has not been subsequently terminated. Actions occur at points in time and can modify fluents. Time points provide a narrative based structure independent of any particular action.

The EC also provides the predicates listed in Table 2.1. In addition, uniqueness of names was supposed for fluents and actions to neutralise problems that otherwise might arise from identicality. The core of EC applications therefore contain the following:

1. Domain dependent sentences (Σ) describe the effects of actions on fluents at given time points.
2. Domain independent axioms (EC) are the backbone of the EC. These are used to reason about whether or not fluents hold at given time points.
3. A narrative (Δ) provides a list of the events that occur and defines the temporal ordering (such as, given time points τ_1 and τ_2 , τ_1 occurred before τ_2).
4. An initial situation (Δ_0) describes the fluents at the earliest time point.
5. Unique names (Ω) for fluents and actions.
6. A goal (Γ) is queried to determine whether a fluent holds at a given time point. A goal Γ is modelled by $\Sigma \wedge(\Delta\wedge\Delta_0)\wedge$EC $\wedge\Omega$.

Table 2.1: Concepts of the Simple EC

Taxonomy Id	Predicate	Definition
Σ	Initiates(α, β, τ)	Action α initiates fluent β which will begin to hold after time τ
Σ	Terminates(α, β, τ)	Action α terminates fluent β which will cease to hold after time τ
EC	Clipped(τ_1, β, τ_2)	Fluent β is terminated between τ_1 and τ_2
EC	Declipped($\tau_1\beta$, τ_2)	Fluent β is initiated between τ_1 and τ_2
EC	HoldsAt(β, τ)	Fluent β holds at time point τ
Δ	Happens(α, τ)	Action α happens at time point τ
Δ_0	Initallyp(β)	Fluent β holds at the first time point
Δ_0	Initallyn(β)	Fluent β does not hold at the first time point

This form of the EC is used to look for interactions in rules for the work in this thesis. For example, the interaction of rules for OCS and CFB (see figure 2.1) can be described in EC as follows.

1. The initial situation (Δ_0) is:
   - Initallyp(alice_can_make_call).
   - Initallyp(bob_busy).
   - Initallyn(connection).
   - Initallyn(connected _to_charlie).
2. The domain dependent sentences (Σ) are:
   - Terminates(alice_call_bob, connection, t) :- HoldsAt(connected _to_charlie,t). This is the rule for OCS.
   - Initiates(alice_call_bob, connected _to_charlie, t) :- HoldsAt(bob_busy, t). This is the rule for CFB.
   - Initiates(alice_call_bob, connection, t) :- HoldsAt(bob_busy, t).
3. The narrative (Δ) is :
   - Happens(call, t0).
4. The goal (Γ) is:
   - HoldsAt(connection, t1).

Unfortunately, Γ can be satisfied with either true or false. Here the priority of the Initiates and Terminates sentences influences whether the connection is established. Searching for such conflicts in the EC forms of the narratives can lead to the discovery of conflicts amongst rules.

Previously others have applied EC-based approaches to model conflict analysis in distributed systems. Efstratiou *et al.* [2002] modelled obligation policies using the EC and had future plans to detect conflicts amongst them. Bandara *et al.* [2003] developed methods to transform policies and top-down views of system-wide behaviour into EC notation, in order to abduce conflict types. They introduced a considerable number of additional predicates in order to model domain objects and whole system behaviour. Such additions could make it difficult for domain experts to understand the underlying behaviour of the system. Laney *et al.* [2007] adapted requirements, domain information, and specifications into EC format. These were used with a feature composition controller at run-time to handle inconsistencies. Thus, a motivating interest for this thesis is the usage of the EC for conflict detection and resolution for device control.

There are some commonalities, but also some differences, in how this work compares with others in the field. Montangero *et al.* [2008], for instance, held a similar view to Lupu et al., and went as

far as proposing conflict detection and resolution for distributed policy-based systems. The biggest difference between that work and the approach here is the policy-level orientation versus the lower-level rule orientation. As Montangero *et al.* point out this is a difference of the detail at which the system is studied: policies can be seen as top-down control of a system as a whole, whereas the rules discussed here only control a single feature. In their approach, Montangero et al. specified policies in the APPEL language, then translated these into ΔDSTL(X) temporal logic format for conflict detection. An advantage that the work in this thesis has over theirs is that the rules are written in the EC and these are used directly in conflict assessment, thus simplifying the stages of design, development and verification. Such a united approach makes it easier for the end users to understand how the rules in their system operate and problems that can arise from them. Another advantage that this work has over theirs is in how conflicts are described and determined. Montangero et al. defined conflict as being the case when two policies are applicable at the same time and their actions conflict. This is broadly similar to the notion of STI, but leaves no room for handling additional conflict types such as MTI and SAI.

Another approach that involves temporal logics is presented by Calder *et al.* [2009]. In that work, a MATCH Activity Monitoring (MAM) hub is modelled in the PROMELA verification modelling language. These hubs are rule-based pervasive systems, however the rules are less expressive and powerful than the ones presented in this thesis. The MAM rules are simple if->then statements that cannot support complexities such as recursion. Furthermore, the EC-based rule system presented here is advantageous because it provides the writer (and reader) with a sense of the temporal flow of the happenings of actions, rather than simply a listing of ifs and thens. In other words it provides the rule author with a more flowing mental model of what can transpire in the network, and therefore will be more complete and representative of the problems being addressed. Calder *et al.* [2009] presented two forms of verification procedures to look for redundant rules (rules that overlapped, which can be seen to be MAI). The first translated the PROMELA rules into Linear Temporal Logic (LTL) properties which are then fed into the SPIN model checker. Verification by this method is shown to be too slow for use within a volatile network. The second approach fed the LTL model into the miniSAT SAT solver which detects redundancies in a reasonable amount of time. Such an approach, however, has not been shown to detect MTI or STI.

The Horn clause subset of first-order predicate logic is used to define a family of frameworks, known collectively as the EC, in order to represent and reason about actions and their effects in time. The EC was originally designed to reason about database applications, according to Kowalski & Sergot [1986]. Since then, as Miller & Shanahan [1999] pointed out, the EC has been applied to a number of different application areas such as: agent belief modelling, planning, and cognitive robotics. In addition, it is used in work flow management systems by Wilk [2004], by Chen *et al.* [2008] for reasoning about behaviour in smart homes, for analysing policy and system behaviour specifications by Bandara *et al.* [2003], and by Broda *et al.* [2009] for sensor information reasoning and actuation. Charalambides *et al.* [2005] performed policy analysis using the EC which could be useful if one were attempting to resolve organisational policies.

A number of variations of the EC have been used, but the core of the framework (represented as the predicates in Table 2.1) is mainly consistent across formulations, and the predicates used for this thesis are the same as those used by Wilk [2004]. Since the Horn clause subset has been used in the definition of EC, it is relatively straightforward to implement the EC in a declarative programming language such as Prolog.

2.7 Interaction Resolution

Interaction resolution has previously been considered in works such as Marples [2000], Reiff-Marganiec [2002], Wilson [2005], and Nakamura *et al.* [2009]. Each offered alternative approaches, but these can be summarised as reliance on human intervention, reliance on heuristics or reliance on priorities/preferences.

Marples [2000] relied extensively on human intervention to customise approaches at the per-interaction-type level and at the interaction level. These solutions are summarised in table 2.2. The resolution choices offered by Marples boiled down to deciding which rules to disable (not allow to run) at any time. Marples' solution in some circumstances (such as for STI) requires *a priori* knowledge about the interaction which may not be available until run time. Of note as well, is that in some cases of SAI human intervention is required. The view of the work in this thesis however is that device rule conflict resolution should allow for a level of human oversight but must also supply default resolution behaviour until a human has had

time to determine the best resolution strategy to take.

Table 2.2: Summary of resolution types discussed in Marples [2000]

Interaction Type	Resolution strategy
STI	Disable one of the rules. The choice as to which one is determined in advance. For instance for interactions between Call Forwarding Unconditional (CFU) and Do Not Disturb (DND), CFU is allowed to succeed.
SAI	These types of interactions required human intervention. A table of ignorance is maintained because of the beneficence of many of these kinds of interactions. System operators are notified when new interactions are detected and they may add the interaction to the table if the interactions are perceived as benign. No explanation is given for what would happen to offensive interactions.
LI	The rules are disabled.
MTI	No strategy is used since the system could not detect MTI.

Reiff-Marganiec [2002] relied on heuristics to extract a resolution space from a solution space. He distinguished rules that are independent of the semantics of the messages sent between conflicting rules from those that are not. Reiff-Marganiec attempted to determine which features to allow to run, and which to disable like Marples did. The heuristics used included:

- Favour solutions involving the largest number of communicating rules (note that this does not necessarily mean the greatest amount of communication)
- Favour rules with the lowest number of connections
- When all things are equal, choose deterministically rather than randomly

Lupu & Sloman [1999] also provide heuristics to resolve conflicts, for instance to favour negative authorisation over positive ones and to favour more specific policies over less specific ones. They criticise explicit priorities on the grounds that it is difficult for users to assign these without them becoming arbitrary. However, the same criticism can be leveled at the heuristic approach. Ultimately the choice of which heuristics to use represent the priorities of system designers. The results of such heuristic application can only ever be validated against user preference so it is favourable to allow users to specify priorities, but to do so in a way that allows them to consider the consequences of their decisions.

Wilson [2005] describes a priority-oriented resolution strategy. Features are disabled (by not allowing them to receive triggering messages) when interactions are detected. Wilson's solution is somewhat like the one proposed by Marples for handling SAI. Wilson's protocol, however, requires the assignment of priority levels to services. The determination of which feature to grant authority of action to, therefore, can be delayed to a run-time assessment of the priority levels of interacting services, which has the benefit of not relying on *a priori* interaction knowledge. Another interesting point raised by Wilson is the notion of smart services. When particular features are blocked, services can attempt to run alternative features to accomplish their goals in the hope that the alternatives will not interact. Wang & Turner [2008] use a preference system to prioritise policies to resolve conflicts. Their preference system indicates how strongly the policy specifier feels about the given policy, but this is done with disregard to any context within which the policy is used.

Most resolutions disable services, features, rules or actions. Nakamura *et al.* [2009] point out that disabling a feature may be too heavy-handed a resolution for some situations. They, assign priorities to their features, although they discuss the possibility of using heuristics like Reiff-Marganiec did to determine the priorities on the fly. They distinguish between the total disabling of a feature and the suspension of parts of the feature (what they called methods). They expect service developers to assign flags to their methods indicating which of them are mandatory. When interactions occur, lower priority services may suspend non-mandatory methods and continue to run in compromised states. Whilst their idea that complete feature abortion is overly relentless for some situations has intuitive appeal, they did

not show that running a compromised system is indeed a feasible solution. A compromised service may create even greater degrees of negative interaction than one that disables features.

2.8 Sensor Networks

Sensor networks are collections of devices that have one or more of the following capabilities (to some degree): data sensing, data processing, communications, and data storage. Each device may specialise in any of the capabilities or may perform all of them. The capabilities of devices are provided by different types of components including communications transceivers, sensors, and ultra small computing boards. These boards feature power sources, have limited memory, and have input and output connections. Sensors transduce signals from physical phenomena (such as thermal, electrical, or magnetic radiation) into forms that can be used in a digital system. When devices communicate unattached by wires (such as transmission over radio) the network is said to be a WSN. WSNs have been reviewed in a number of articles including Yick *et al.* [2008], Akyildiz *et al.* [2002], Mottola & Picco [2010]. Whilst agreeing for the most part with these descriptions, the view of sensor networks in this thesis follows Römer [2004] in relaxing the wireless communication requirement for the devices (communication may be wireless or wired).

Each device is considered to be a node in a network. There are three fundamental types of nodes: sensor nodes, sink nodes, and gateway nodes. The first two of these are described by Akyildiz *et al.* [2002]. Gateway nodes can be considered as those which transfer information between a sensor network and external networks. A node may have characteristics of one or more of the types (for instance a node could store data and act as a gateway).

The communications infrastructure of a sensor network tends to be *ad hoc* or minimal, as was pointed out by Yick *et al.* [2008]. There are two fundamental approaches to the design of sensor network architectures according to Zhao & Guibas [2004]: Data Collection System (DCS)[2] and CSIP. They describe that signal data in a DCS were transferred from sensors to sink nodes at the edges of the networks, then processed at some later point in time. The authors also discussed how data are processed intelligently in a CSIP and aggregated in-network. Mottola & Picco [2010] describe how these architectural differences arise from alternative goals where, in the former case, designers may only be interested in sensing primary data, whereas in the latter case, the designers may be interested in both sensing and reacting to the data. They considered CSIP superior to DCS because goals could be better achieved through greater device communication, collaboration, and coordination.

The domain of interest, AA in this case, has an impact on design considerations such as the degree to which connectivity is supported, along with system lifetime issues and service quality constraints. Yick *et al.* [2008] describe a number of application domains that were explored using sensor networks. Some of these include health monitoring applications, home automation, military usages, environmental monitoring, commercial projects, and industrial interests. The application area may impact real-world deployment issues (such as device form factor and resource constraints) which are important considerations in the design of the networks. System lifetime issues, such as deployment characteristics, maintenance and longevity of network are considered in chapter 3.1, along with additional AA specific WSN concerns.

2.8.1 WSN Programming Approaches

This thesis is the first to consider sensor network programming models specifically with regard to AA sensor network requirements. Earlier works provided general characteristics for alternative sensor network domains. Coordinating a collection of heterogeneous devices to sense and report findings requires a programming architecture (also known as middleware). Aspects of middleware for WSN are described by Römer *et al.* [2002]. They suggest that the key purpose of WSN middleware is to use power-efficient, robust and scalable techniques to split up sensing and reporting tasks among devices. The authors also discuss how application knowledge needs to be distributed and accessible within the network, and how the network must self-configure and run unattended. Hadim & Mohamed [2006] add security and quality of service concerns to the list of sensor network programming issues.

[2] Zhao & Guibas [2004] defined Collaborative Signal and Information Processing (CSIP) but did not provide a name for traditional sensor networks. Therefore in this thesis the alternative has been labeled DCS.

Various approaches to designing middleware for sensor networks have been documented. A number of these are presented by Römer [2004] including sensor network as database (such as TinyDB discussed by Madden *et al.* [2005]), mobile agent programming, event-oriented programming, distributed virtual machines (VM), and object tracking. These different approaches trade off ease-of-use for expressiveness to varying degrees. Yu *et al.* [2004] describe a virtual machine-based layered architecture for cluster-based middleware. Techniques described by Hadim & Mohamed [2006] include approaches that tightly couple the network protocol stack with the applications, message-oriented middleware (MOM), and macroprogramming the sensor network as a whole. They compare various implementations, focusing on power awareness, openness, scalability, mobility, heterogeneity, and ease-of-use. VM-based and agent-based approaches fared the best in their analysis. Henricksen & Robinson [2006] cited a Linda in a Mobile Environment (LIME)-based tuple-space approach to WSN programming called TinyLime (further described by Curino *et al.* [2005]). They conclude that the implementation does not adequately connect the nodes into a single distributed tuple-space, but that such an approach would be valuable. Costa *et al.* [2007] compared another LIME-based approach called TeenyLIME with TinyLime. The difference between the two, they state, is where the intelligence in the system occurs. In TinyLime the decision making happens in the sink nodes whereas, in TeenyLIME, the intelligence emerges from node-to-node interactions inside the network. Another tuple-space oriented platform is the EQUATOR Universal Platform (EQUIP), discussed by Greenhalgh [2002], but the platform was too heavyweight to be used as middleware for sensor-based applications. Sugihara & Gupta [2008] describe how programming models can be considered low-level and platform-centric if their focus is on the node level, whilst a focus on node grouping or the network level is considered high-level and application-centric. Dedecker *et al.* [2005] describe Ambient-oriented programming (Am-OP) as a class of languages that can deal with connection volatility, ambient resources, device autonomy and the natural concurrency arising from mobile networks. Surveys of programming models have compared various WSN implementations. Horré *et al.* [2007] classified over 40 solutions across different WSN layers (hardware abstraction layer, distributed programming layer, service layer, and management layer) and WSN middleware tiers (sensor operation system tier, sensor network tier, gateway tier, and back end tier for long-term data storage and access). The FACTS middleware (described in detail in Terfloth *et al.* [2006]), for example, was classified as a programming abstraction layer of the sensor network tier, and Open Services Gateway Initiative (OSGi) was classified as a gateway middleware platform.

Currently, no consensus has been reached in the research community as to the best approach to programming sensor networks to meet these various issues. Rule-based middleware for sensor networks (discussed in more detail in the next section) has been used in a number of projects. A rule-based approach was chosen for the work in this thesis because of advantages including, according to Sen & Cardell-Oliver [2006], having simplified programming and concurrency models that make program correctness easier to prove, they are power efficient, and rule-based systems remain sufficiently expressive at high conceptual levels. Furthermore, according to Terfloth *et al.* [2006], thinking about WSN from an event paradigm applies better to sensor networks than thinking about the system using an imperative paradigm. Rule orientation, they argue, is a more natural way to express programs for sensor networks. In addition, Fei & Magill [2008] show that application developers using rule-based middleware are protected from complexities arising from tight real-world integration, network dynamics, and resource limitations. An interesting remaining area of concern is to what degree rules in sensor networks conflict, which will be explored further throughout this thesis.

Middleware programming models have been evaluated using different criteria, although no standard set of criteria has yet been identified against which to judge the solutions. Sugihara & Gupta [2008] measure a number of models against energy efficiency, network scalability, failure resistance, and node collaboration criteria. They describe implementation mechanisms for handling these criteria, but do not provide metrics for comparing their models against the given criteria. Hadim & Mohamed [2006], however, use metrics that are divided into three levels: full, partial, little or no support to compare programming models against a variety of criteria. These include power awareness, openness (system extensibility and modifiability), scalability, mobility, heterogeneity, and ease of use (the degree to which the middleware interface hides the low-level APIs of the devices). Performance metrics are grouped by Pawar *et al.* [2008] into network performance, vertical handover performance (signal detection to actuation delay), and mobile device resource utilisation. Network performance measures the maximum throughput of the network (in bytes), the number of signals transferred over the network during a

monitoring session, and the latency of the network. The vertical handover performance measures the delay between the occurrence of an event and the notification of the event. Mobile device resource utilisation measures the memory and the processor utilisation of a device such as a PDA or phone whilst processing data or acting as a gateway. Application performance is measured by Heinzelman *et al.* [2004]. They assign different levels of QoS for each studied variable to each type of sensor (or sensor grouping), and then review the sensor (or grouping) reporting the data. Design principles identified by Masri & Mammeri [2007] offer many of the previously mentioned points, but also add application knowledge representation, network robustness, and real-world integration. These criteria are divided into three levels: full, partial, or no support. End-to-end latency is a metric used by Dressler *et al.* [2009] to measure the time between when an event occurs and when an action in the network responds to the event.

2.9 Ecologically Valid Assessment and Care

The following introduces key ideas about mobile assessment and care. A number of different but related concepts are used with care at a distance:

- **Health** definition from World Health Organization who [1946]*: "A state of complete physical, mental and social well-being and not merely the absence of disease or infirmity."*
- **Telemedicine** definition from House of Commons Health Committee hou [2005]: *"Electronically mediated interaction between doctor and patient, synchronously or asynchronously."*
- **Telehealthcare** definition from House of Commons Health Committee hou [2005]: *"Electronically mediated interaction between patients and health professionals, often nurses... primarily work to collect diagnostic or other data for doctors, to manage an illness by means of advice, or triage work intended to decide whether a patient warrants admission to hospital."*
- **Telecare** definition from House of Commons Health Committee hou [2005]*: "Highly portable systems for monitoring the health status of people with chronic (eg diabetes and asthma) and degenerative (eg respiratory and cardiovascular) diseases... Such devices measure physiological status and other data, present this data to individual users, and transmit it for review by service providers using either mobile or conventional telephony."*
- **Telepsychiatry** definition from apa [1998]: *"Telepsychiatry is the use of electronic communication and information technologies to provide or support clinical psychiatric care at a distance. This definition includes many communication modalities such as phone, Fax, email, the Internet, still imaging and live interactive 2-way audio-video communication. Live interactive 2-way audio-video communication videoconferencing is the modality addressed in the following report. Videoconferencing has become synonymous with telemedicine involving patient care, distant education, and administration."*
- **m-Health** definition from Istepanian *et al.* [2005]*: "Emerging mobile communications and network technologies for healthcare."*
- **Ambulatory Assessment (AA)** definition from Ebner-Priemer [2010]*: "Ambulatory Assessment comprises the use of field methods to assess the ongoing behavior, physiology, experience and environmental aspects of humans or non-human primates in naturalistic or unconstrained settings. Ambulatory Assessment designates an ecologically relevant assessment perspective that aims at understanding biopsychosocial processes as they naturally unfold in time and in context."*
- **Ecological Momentary Assessment (EMA)** definition from Stone *et al.* [2007]*: "EMA methods are characterized or defined by the repeated collection of real-time data on participants' momentary states in the natural environment."*

The last two terms are of particular interest to this thesis. This work has been informed by the author's research on the Personalised Ambient Monitoring (PAM) project, which fits under the auspices of AA and Ecological Momentary Assessment (EMA). Fahrenberg [2006] discussed a number of related terms.

AA was distinguished from *ambulatory monitoring*, defined as the observation of free-moving patients for illness diagnosis and medication adjustment. Fahrenberg argues that ambulatory monitoring is only one of various goals for AA (such as behaviour observation for the study of developmental psychology), however, ambulatory monitoring is the key AA goal for the purposes of the work described in this thesis. In addition, other related terms such as Context Sensitive Ecological Momentary Assessment (CS-EMA) from Intille [2007], and Experience Sampling Method (ESM) (discussed in Christensen *et al.* [2003]) are subsumed in this thesis by the term AA. There are many similarities between AA and Mobile Healthcare (m-health). Mature projects in the latter category have also informed this thesis. Studies of particular interest include emergency response, as described by Malan *et al.* [2004], assisted living, and geriatric rehabilitation described in Wood *et al.* [2006] and Sixsmith *et al.* [2007]. Similar technologies have been extended into the field of mental health care under the auspices of AA.

AA, according to Ebner-Priemer & Trull [2009], is a better approach to answering particular questions in mental health studies than traditional clinical questionnaire/interview based studies. The traditional self-report approach is reliant upon patient memory to vividly and accurately recall what has transpired over a relatively long period of time (from weeks to months), whereas AA avoids recall issues. The problem with recall, Ebner-Priemer and Trull argue, is that memories are often distorted during their storage and recollection processes. Remembering is constructive and therefore error-prone according to the Research Board of the British Psychological Society BPS-Research-Board [2010]. The environment can influence recall, memories may lack details, and they may have gaps in them.

A number of factors influence the recall of information, such as personal bias towards information associated with positive affect. Furthermore, survey respondents are particularly poor at reporting the frequency of behaviour and the intensity of experience according to Schwarz [2007]. Recollection also misses important types of data such as physiological conditions and contextual information. Another introduction of uncertainty is that subjects are able to cheat self-monitoring protocols. Back-filling, for example, occurs when subjects complete missing items close to a scheduled interview. Such distortions in the measurements of clinical variables reduce the quality of the conclusions. In contrast AA protocols can be used to avoid recollection bias. The concern regarding whether or not researchers can even trust reports from subjects about their current experience is dismissed by Stone *et al.* [2007] where they cite evidence that shows that it is still beneficial to ask subjects about their current or very recent experiences.

AA also provides unadulterated continuous multi-modal data from ecologically valid settings in contrast with laboratory based assessments. There are two fundamental types of lab studies, according to Stone *et al.* [2007], the attempt to model the real world in the lab, and asking people to summarise the real world in the lab. Each of these types of studies has deficiencies in comparison with sampling real world experience and behaviour. In the first case, the data may not be consistent with real world experience owing to the artificial nature of the experiments. The second types of experiments have similar problems as self-reporting, that is they rely on retrospection.

AA is not without its limitations as a method, however. Subjects, for instance, may be burdened by the monitoring technologies and procedures of data capture. Such burden, as described by Collins & Muraven [2007] and Fahrenberg [2006], is of concern to researchers for a number of reasons. Firstly, it impacts on the selection criteria for studies, reducing the numbers of participants. Stone *et al.* [2007] point out that the method may be inappropriate for use with individuals whose physical or mental states are in direct conflict with the technologies being used to perform the studies. Secondly, the burden may keep compliance rates low and may even lead to subjects dropping out of studies. Finally, there are study generalisability concerns because the compliant subjects willing to participate in AA studies may not be wholly representative of the population being studied. Fahrenberg points out three aspects of monitoring burden using AA methods: subject *acceptance* of the technology, subject *compliance* with the correct use of the technology and subject *reactivity* to the technology. Acceptance issues relate to how ready subjects are to use the technology appropriately. Compliance issues cover areas such as recording data at the correct time and complying with the overall study protocols. Whilst lack of compliance can lead to missing data, technological failure may also cause data loss. These two types of failure should be distinguished from each other. Fahrenberg describes three types of reactivity: motivational (changes to a subject's willingness to comply), behavioural (changes to a subject's behaviour resulting from being monitored) and procedural (changes to the subject's routine to facilitate the monitoring). Reactions may involve avoiding particular settings or behaviours, or manipulating the technology.

Performing AA is a motivating factor for the work in this thesis. AA is an interesting area of research that has not had a great deal of attention from the Wireless Sensor Networks (WSNs) community. Sensor networks for AA require a high degree of transparency and personalisation, as well as minimising the number of devices used for monitoring. This is to help lower the subject burden and keep them and their care providers informed about how the subject is being monitored. This thesis is the first to consider conflicts that can arise in rule-based WSNs for AA. More detail is provided in the next chapter regarding the particular devices, features and rules that can be used for AA WSNs.

2.10 Related Work Review

The approach that this thesis presents is related to previous works in the following ways:

- The declarative rules presented in this thesis use A-C form rather than an E-C-A form common to many of the rules and policy-based systems reported above. Device and knowledge management rules are written using the EC. This is the first work to analyse EC-based device rules for the five forms of conflicts described above.
- The approach in this thesis requires direct access to all of the device rules, unlike online FI solutions. This however, is justified because of the high degree of transparency required by AA. In addition, having access to all the rules helps the approach detect all five forms of conflicts described above.
- A pairwise approach is used in this thesis to analyse the device rules as n-wise approaches have not been shown to have significant benefits.
- A priority system is used in this approach to deal with rule conflicts. The two-stage priority system presented here is an attempt to aid users to assign priorities in a more considered way than previous approaches involving only one level of priority.
- The approach presented in this thesis is informed by WSNs with a particular focus on performing AA. AA WSNs require a high degree of transparency as they are used to monitor subjects in their homes. They require a relatively smaller number of (personalisable) device nodes than some forms of WSNs to increase subject acceptance and compliance, as well as to lower subject reactivity. The approach presented here has been designed to suit problems of single subject monitoring using a relatively low number of devices, and therefore may not be useful for searching for conflicts in WSNs involving large numbers of nodes, such as *smartdust* envisioned by Kahn *et al.* [1999].

2.11 Summary

This chapter introduces background material on AA, WSN, FI and EC. This work is motivated by the reliable delivery of AA services in WSNs. The combination of AA and WSN is further considered in chapters 3 and 4. FI problems have been reviewed in a number of areas, however rule-orientated WSN in general and WSN for AA in particular are ignored. These areas offer researchers interesting and novel problems to handle, particularly with respect to the expression of rules and conflict. The EC is introduced as the basis for a solution when hunting for device conflicts. The approach to using the EC for conflict detection is addressed in chapter 5.

Chapter 3

Ambulatory Assessment Sensor Network Features and Rules

3.1 Introduction

This chapter reviews Ambulatory Assessment (AA) projects to determine necessary and sufficient rules for analysis in subsequent chapters. This review includes state of the art in AA, along with considerations of future directions for AA from the literature. The contribution of this chapter is the determination of the features necessary and sufficient for AA sensor networks and to list the rules that can be used to control the behaviours of devices in a sensor network for AA.

3.2 State of the Art Technology

The type of system under consideration is one which can be used to perform micro-longitudinal studies that involve sampling moments in a subject's life. The core service of such systems is sampling events over time and storing them. Sampling has tended to be achieved through momentary assessment by using mobile devices to prompt subjects for answers to questions. A number of additional features are described in the AA literature that can extend this service, including:

- Mechanisms to adjust prompting schedules
- Collecting physiological and environmental data in addition to momentary assessments
- Combining event-based, time-based, subject state-based and contextual triggers
- Adding data-orientated conditional questioning to momentary assessment
- Presenting assessment information to subjects and care providers

The advantages of AA over traditional forms of assessment are discussed in section 2.9. Smyth & Stone [2003] report that reducing the reliance on retrospective recall is the chief benefit of using AA systems. AA systems also offer greater generalisability of the studies in comparison to laboratory studies, owing to their ecological relevance and greater ability to monitor dynamic processes as they unfold in time. Additional benefits, according to Le *et al.* [2006], include higher compliance rates than traditional paper-based surveys, improved survey structures, and real-time provisioning of data access to researchers.

A number of systems exist for performing AA, some of which are reviewed by Ebner-Priemer & Kubiak [2007] and Fischer [2009]. Below is an examination of commonly available systems: Experience Sampling Program (ESP), discussed by Feldman *et al.* [2001], Purdue Momentary Assessment Tool (PMAT), documented in Weiss *et al.* [2004], and MyExperience, discussed in Froehlich *et al.* [2007].

These systems allow researchers to set up diary study protocols on Personal Digital Assistants (PDA) and mobile phones. Only one of the systems (MyExperience) provides appropriate access to external sensors. Even this is limited to the mobile environment. Therefore, Alarm-Net, discussed by Wood *et al.* [2006], is also included in the examination below in order to describe an existing system that captures

and responds to data at home. Alarm-Net, however, does not support the mobile environment and does not have the rich questionnaire facilities of some of the other systems. As such, there was no solution available at the time of writing that combined sensor and questionnaire data collection from mobile and home environments.

3.2.1 Experience Sampling Program (ESP)

ESP, discussed in Feldman *et al.* [2001], was the first open-source software developed for PDAs to provide AA capabilities. The Palm OS based software is used to display surveys and questionnaires, and to record responses and response times. Intel Experience Sampling Program (iESP), discussed by Consolvo & Walker [2003], is a modified version of the software that has gone dormant subsequent to publication; however, many of its features have been subsequently added to ESP.

ESP has a number of interesting features. It has two question display interval modes, branching questions, and probabilistic questioning capabilities. Response times are recorded along with the responses. Alternative response types are available which are used to provide feedback to subjects about study progress. The software supports two question display interval modes: automatic mode, in which the questions are asked at either fixed rate intervals or random intervals, and manual mode, in which subjects can answer questions at their own convenience. Multiple questions can be prompted, either in sequential ordering or using branching logic. Each question in a question set is numbered, and the branching logic uses goto statements (labeled %NEXT) to jump to particular questions. Questions can also be set to have a probability of occurrence. Branching control was added to make sure that the correct following question is displayed based on whether or not the former question is displayed. A limited amount of study feedback was available, but this includes notifications about the completion of studies.

ESP also has a number of limitations as an AA vehicle. Fischer [2009] wrote that the greatest limitation of the software was that it is designed for an outdated platform. There are other limitations, however. Firstly, question sets are fixed in the memory of the device, with no method of automatically updating the questions. Therefore, although the questions can be chosen from sets randomly or in fixed order, the questions themselves are not chosen based on the context of the individual. Other limitations include the inability to support contextual information in the questioning, the inability to adjust the questioning protocol to the context of the subject, and the inability to allow the subject to annotate responses with contextual information. The branching logic is somewhat limited as well. It is possible to generate infinite loops in the questioning, and it is not possible to branch from a question in one question set to one in another question set. Finally, another limitation is that the data are stored within the software and require special software to download them.

3.2.2 PMAT

PMAT, developed by the Military Family Research Institute at Purdue University is Java-based software for Palm OS. It provides a number of additional features that are not available in ESP, as described by Le *et al.* [2006]. These include more advanced prompting schedules and advances in data storage, as well as descriptions of future directions for AA technology.

Advanced scheduling features are available in PMAT. These include being able to set up alternative prompting schedules and question sets for different days, combine signal and event recording in the same study, alert the subject to prompts at fixed or random intervals within a time window, do not disturb capabilities, and randomization at the question set level. However, PMAT is limited in that it cannot be used to set up randomization within a question set. Although it supports question branch logic, the logic is restricted to one level off the main trunk.

PMAT provides some data storage features. For example it saves data to a memory card in addition to the PDA memory, a capability unsupported by ESP. However, the memory card data backup schedule is fixed to back up at midnight every night that a card is present. In addition PMAT data can be exported to Comma Separated Value (CSV) format for analytical purposes.

There are additional features that Le *et al.* [2006] would have liked to have been developed. In particular, these features include on-the-fly scheduling changes, the ability to assign one device to multiple subjects, being able to combine data from multiple sources (such as questionnaire, images and sound), and being able to access data instantly.

3.2.3 MyExperience

MyExperience is open source sampling software for mobile phones and PDAs used to develop studies that incorporate quantitative data from sensor readings and qualitative survey responses, according to to Froehlich *et al.* [2007]. It also supports a variety of sense and response features.

The sensor, trigger and action system is of interest in regards to this thesis's focus on such rules. Survey user interface options and behaviour in MyExperience are defined in XML as collections of actions, conditionally triggered by sensor readings. Sensors can be either hardware-based, such as microphones, or software-based such as device usage information (for example key stroke analysis) or context data (such as calendar entries). A number of sensors are built into MyExperience and researchers can add additional ones as plug-ins. The most recent version of MyExperience supports more than 140 sensor event types from a range of categories including communication, application usage, media capture, user context and environmental sensing. Triggers can be written in C# or the Simkin scripting language to respond to sensor events. Triggers can also respond to sensor metadata to handle user state conditions (such as the first time a subject entered a location or after the user had been in a location for a designated amount of time), or to detect the failure of sensors if they do not respond within a given time frame. Trigger logic controls the execution of actions. Action types include the launching of external applications, database synchronisation, user notification across different user interfaces, screenshot recording, sending text messages, and displaying surveys.

MyExperience also supports interesting data collection and handling features. All data are automatically stored locally in SQL Server 2005 Mobile Edition (SQL Mobile) databases. On-device databases can be synchronised with SQL databases running on remote back end connections if the mobile devices are connected over Evolution-Data Optimized (EVDO), 802.11 wireless or General Packet Radio Service (GPRS), however operations for offline sensing and local data storage are also supported.

A number of MyExperience features handle survey display and response. Survey questions can be specified based on previously collected data using dynamic parameters completed by scripts executed immediately prior to displaying questions. Scripts can also be executed following subject response in order to handle complex branching and response activities. Also, survey prompts can be closed manually by the user or automatically after a given amount of time.

MyExperience has limitations with regards to action chaining and action triggering logic. Actions cannot activate sensors and then chain. Furthermore, whilst multiple actions may be activated by the same trigger, the results of these cannot cause additional actions to be performed. Of greatest concern, however, is that MyExperience has functionality that can conflict, yet no system is in place to detect or correct them. Multiple action interactions, for instance, may arise from the use of multiple triggers. The reliability of MyExperience for long-term monitoring is questionable given the possibility of conflicts amongst its features.

3.2.4 Alarm-Net

Wood *et al.* [2006] discuss a Wireless Sensor Network (WSN) used to perform residential monitoring in assisted living situations called Alarm-Net. This WSN integrates heterogeneous sensors for monitoring physiological and environmental variables. Physiological readings are collected using PDAs, but these are expected to be used only on site. Alarm-Net uses a network protocol to support various devices and individuals to query for current information about the states of the variables. The devices are capable of processing data on-the-fly and caching data. A gateway application is configured to process incoming data from all of the nodes in real time in order to determine the circadian activity patterns of the subjects under observation.

The gateway uses the query protocol and results from activity analysis to perform node-level power and privacy management. Activity analysis is used in power management to determine which sensor nodes should be disabled in order to conserve power based on the probable habits of the subjects. Data aggregation strategies were also incorporated to reduce power consumption in the network. Wood *et al.* [2006] describes an example in which aggregated pulse rate information is reported every second from data collected five times a second. Furthermore, the reporting system allows reports to be generated only when thresholds are exceeded, for instance only reporting a pulse-rate above 130 beats per minute. In addition, users can provide policies to the power manager that directly enable and disable sensors. Therefore, the device states are controlled by commands based on a mixture of analysed activity patterns,

user defined policies, and query reports. Commands are issued with priority values in order to reduce conflicting commands. A command has to have higher priority than a previous one in order for it to override the previous one.

Alarm-Net is a step in the right direction for AA systems. It combines environmental and physiological readings, provides real-time data analysis, and modifies the network on-the-fly based on the activity patterns of the subjects. However, Alarm-Net does not incorporate data from subjects taken outside the residential facility, nor can the system incorporate qualitative data from the subjects. More importantly, however, is that whilst an attempt was made to reduce conflicting commands, the priority system did not solve device conflicts altogether. For instance, missed trigger interaction can still emerge. Consider, for example, what would happen if a high-priority message disabled a device that was programmed to send data to another device. This other device would be starved of information and would possibly not trigger actions as a result of the disabled one.

3.3 Future Directions of Ambulatory Assessment

Intille [2007] presented a vision of Context Sensitive Ecological Momentary Assessment (CS-EMA). He imagined that incoming multimodal data streams are continuously collected, analysed, stored, and trigger alterations to the collection process. Collection processes are automatically personalised to subjects and situations that they experience. Intille proposed that the combination of data on subject location, physical activity, proximity to others, and self reports on psychological state is superior to previous behavioural monitoring systems. One benefit of this, for instance, is to lower the burden subjects feel when being questioned electronically. By converting raw sensor data into meaningful activity labels (such as walking or working) the questioning of subjects can be reduced and contextualised such that only meaningful questions are asked given a particular activity. Intille provides two example scenarios to highlight the types of studies that this form of system could benefit including:

1. Exploring relationships between TV watching and sedentary behaviour using TV viewing data and accelerometer data

2. Exploring relationships between sedentary behaviour, the built environment and driving using car sensors, GPS, accelerometers, and questions based on the subject's given activity (such as walking or sitting)

CS-EMA requires the combination of self-reported qualitative data with physiologically measured quantitative data. Raw data, therefore, must be processed on-the-fly, in order to estimate subject activity. Subjects may still be prompted about their thoughts, feelings, and activities, but these prompts are contextualised from the physical information. Furthermore, questions alter in response to historic data, or if the data deviate from statistical norms.

Another change called for by Intille [2007] is to reduce reactivity by allowing subjects to adjust their questioning frequencies. Previously, Intille *et al.* [2003] showed that sensing devices could be programmed to provide context-sensitive triggering, such that particular questions were asked only when the subject was in a specific state (physiological, emotional, local or behavioural) or when the environment was in a particular context (noise levels, light levels, diurnal patterns, etc.). Another concept from those examples is that features of the subjective sampling system are desirable in addition to the contextual triggering. Such features include complex question and answer flow, question aggregation, allowing subjects to specify when they do not want to be disturbed, precise time-based triggering for certain questions (including the capability for recurrence), randomisation of questioning, and bounded timing limits to querying.

System adaptation is necessary to meet CS-EMA challenges and is one of the motivating aspects of this thesis. The long-term study of people using mobile and environmental devices implies that, as new technologies become available, they must be integrated into existing frameworks. Limitations reported in the literature on existing AA studies support a view that it is important to move from reliance on static timing and event mechanisms (such as sampling every hour), towards studies that are able to dynamically collect data from a variety of intermixed sources. For instance, Collins & Muraven [2007] are concerned with an over-reliance concerning subject self-reporting, and the lack of handling control conditions. They report technological barriers to being able to enhance their self report data with

behavioural and physiological information. This, however, is already technologically feasible as reported by Blum & Magill [2010], given their experiences with the Personalised Ambient Monitoring (PAM) project. They collected data streamed from a variety of sources for offline analysis. The Data Collection System (DCS) approach, unfortunately, was inappropriate for determining control conditions in a live environment. With adaptation, however, comes the concern of minimising device conflicts. Like other complex adaptive systems such as call control systems, conflicts can emerge that degrade the integrity of the network. These conflicts must be guarded against in future AA systems.

3.4 Derived Feature Rules

The previous sections show that the core service of AA is to display survey questions, and to record responses and response times. The features described above in the various projects, including ones derived from their limitations, are shown together in table 3.4.1. Rules that are sufficient or necessary to describe these features, or parts thereof, are used in the following chapters in conflict analysis examples. The ones that are used are described in the *Rule* column of the table and described in greater detail in the next chapter. The notation used is *Group:Rule*, where *Group* is the group that the rule belongs to. Rule groupings are described in the next chapter.

3.4.1 Ambulatory Assessment Features

Project	Feature	Rule
ESP	Alternative question display interval modes (such as fixed rate, random, probabilistic and manual entry)	Device Management (DeM):Data Recording Frequency (DRF)
	Alternative response options	Subject Interaction (SI):Prompt (Prompt)
	Providing subject feedback	SI:Prompt, SI:Alert (Alert)
	Providing notifications to subject (such as end of study)	SI:Alert
PMAT	Prompt scheduling	DeM:DRF, SI:Prompt
	Question set scheduling	DeM:DRF
	Combine signal and event recording	Data Management (DaM):Data Storage Unconditional (DSU)
	Sliding window prompting	DeM:DRF, SI:Prompt
	Do not disturb	Do Not Disturb (DND):Do Not Notify Unconditional (DNNU), DeM:Deactivate Immediate (DI)
	Save data to memory card as well as device memory	DaM:DSU
	Backup schedule (fixed)	DaM:Automatic Data Transfer (ADT)
	Export data to csv	DaM:ADT
PMAT Desired improvements	On-the-fly scheduling changes	Context Detection Service (CDS):Context Triggering System (CTS), State Detection Service (SDS):State Triggering System (STS)

	Allow single device to be used by multiple participants	DaM:Inbound Data Screening (IDS), DaM:Outbound Data Screening (ODS)
	Combine data from multiple sources	DaM:Redirect Data Stream (RDS)
	Instant data access	DaM:ADT
MyExperience	Combine sensor readings with survey responses	DaM:Data Storage Through Processing (DSTP)
	Actions triggered by sensor readings	CDS:CTS, SDS:STS
	Trigger scripting	DeM:DRF
	Sensor metadata triggering	CDS:CTS, SDS:STS
	Notification across various interfaces	CDS:CTS, SDS:STS, SI:Alert
	Automatic SQL storage	DaM:ADT
	Database synchronisation	DaM:ADT, DaM:RDS
	Send text message	DaM:ADT
	Question parameterisation and dynamic completion from data or survey response	CDS:CTS, SDS:STS, SI:Prompt
AlamrNet	Query for variable states	Data Quality Control (DQC):Report Device States All (RDSA)
	Processing data	DaM
	Caching data	DaM
	Circadian pattern recognition	DaM:DSTP
	Node-level power and privacy management	DeM:Power Management (PM)
	Activity analysis	DaM:DSTP
	Data aggregation	DaM:DSTP
	Threshold-based reporting	CDS:CTS, SDS:STS, SI:Alert
Future Directions	Continuous collection, analysis and storage of multi-modal data	DaM, DeM
	Trigger collection procedure changes	CDS:CTS, SDS:STS
	Subject personalisation	SDS:Report Subject State (RSS), SDS:STS
	Combination of data on subject location, physical activity, proximity and self report	CDS:CTS, CDS:Environment Detection (ED),CDS:Report Location (RL), SDS:RSS,SDS:STS
	Contextualised prompting	CDS:CTS, CDS:ED,CDS:RL, SDS:RSS,SDS:STS
	Questions altered in response to historic data or deviations from norms	CDS:CTS, SDS:STS, SI:Prompt
	Context sensitive triggering	CDS:CTS, CDS:ED,CDS:RL

3.5 Summary

The purpose of this chapter is to describe necessary and sufficient rules for AA WSNs. The state of the art in AA projects and their limitations reveal a number of rules that have the potential to conflict when a variety of sensing and information processing devices are interconnected. This will continue to be the case as more complex aspects of CS-EMA are routinely incorporated into AA studies. This chapter has documented features necessary and sufficient for AA sensor networks and provided a list of the rules that can be used to control the behaviours of devices in a sensor network for AA. The rules documented here are described in detail in the next chapter and form the basis for examining rule conflict in latter chapters.

Chapter 4

Ambulatory Assessment Rules and Conflicts

4.1 Introduction

This chapter examines patterns of behaviour that can emerge from the use of a variety of devices and rules. The distinctive contributions of this chapter include fuller descriptions of the rules presented in the previous chapter, and an examination of the Ambulatory Assessment (AA) sensor network rules for examples where they can conflict. It begins by describing an example scenario where device rules conflict. Example sensor network devices are presented in detail. Devices are chosen based on the requirements for AA sensor networks discussed in the previous chapter. A subset of the devices and rules for AA are selected from the literature review of the last chapter and are listed below. These rules are analysed to determine whether or not they can be combined into logical rule sets. They are grouped into two main types of services: those for device control (such as time synchronisation, sensor polling and automated data storage), and those for acquiring and acting upon knowledge about the subjects and their contexts. Rules are listed for each of these types in tables 4.1 and 4.2.

This chapter also presents a series of examples that show conflicts that can emerge from usage patterns across collaborating devices. The collaboration between devices is considered for how it may lead to conflicts. The examples presented below show that conflicts corresponding to each of the interaction types discussed in section 2.3 of chapter 2 can emerge. These are a threat to real-time behaviour monitoring systems. Their detection and avoidance is the subject of the next chapter.

4.2 Example of Conflict: A Brief Visit Home

This example of device conflict involves three sensors used to monitor a subject with bipolar disorder. A *LocationMonitor* node (such as a wearable Global Positioning System (GPS) unit) monitors where the subject is in the world, an *ActivityMon* node monitors activities in which the subject is engaged in (perhaps using a custom application on a mobile phone), and a *HomeMonitor* node collects information from a variety of sensors in the subject's home. Nodes rely on each other to better perform their roles, but such reliance may cause undesirable conflicts.

Device behaviours are controlled through rules on each device. For example, rule 1 of Brief Visit Home (BVH), programmed on the HomeMonitor, stipulates that the HomeMonitor should be active only when the subject is located at home, since the subject lives with other people. Rule 2, for the ActivityMon, defines that when the subject is at home the ActivityMon should be limited to only selecting from relevant home activities. Rule 3, also for the ActivityMon, dictates that if the subject is performing the activity "travelling", then only the start and end locations of the journey should be recorded in order to conserve SLI node resources. Rule 4 (on the activity node) governs the length of the "travelling" activity as distinct from other activities. This value could be learned from user behaviour and could change in time. Here, rule 4 is set such that if the user enters the car, drives, makes a "micro stop" (such as buying milk at the convenience store) and arrives at a "macro stop" (for instance the gym,

or place of work), it is all considered as part of the same travelling activity.

This type of example can expose rule conflicts that may lead to unreliable behaviour. In this particular case Missed Trigger Interaction (MTI) can emerge. This can happen when the subject travels from home, returns home briefly, then sets off again. When the subject initially leaves home, the HomeMonitor is deactivated and the other two nodes enter their travelling states. When the subject returns home briefly (a behaviour not initially considered by the researchers), the HomeMonitor remains off (a trigger to turn on is not sent by the ActivityMon) and does not capture any further abnormal behaviour patterns, resulting in a loss of data. When the subject re-departs, normal system data capture ensues.

Conflicts such as these are often subtle. They are not necessarily obvious to anyone programming individual devices, especially when the programmers are unaware of the behaviours of all of the devices in the network. The rest of this chapter describes a variety of devices, rules, and examples that can be used to reason about the detection and resolution of these types of conflicts.

4.3 Ambulatory Assessment Devices

The examples in the following sections describe AA rules running on a variety of devices. The devices are a representative subset of those presented in the previous chapter. A home gateway device is added for the coordination of some home devices and to bridge communications between the home Local Area Network (LAN) and the Body Sensor Network (BSN). The following is a description of the various example devices. Some of these may be considered stand-alone equipment whereas others may be considered as components used by other devices. The devices may have enough capabilities to perform the activities themselves, or act as slave nodes to other devices that do.

Accelerometer

Accelerometers are sensor components used to measure the magnitude and direction of acceleration of the containing device, which is assumed to be worn or carried by the monitored subject. These are proposed for use in BSN for health monitoring by Jovanov *et al.* [2005] and examined by Amor & James [2009] for use in AA as part of the Personalised Ambient Monitoring (PAM) project. Amor and James show how accelerometer data can be processed to extract information about subject activity.

GPS receiver

GPS receivers calculate their geographic location from data received from multiple satellites that orbit the Earth. These can be used to provide subject location data to a sensor network when worn or carried by a subject. GPS receivers may be sensing components of mobile phones or may be stand-alone devices that communicate with a network over communication protocols such as Bluetooth.

Heart rate monitor

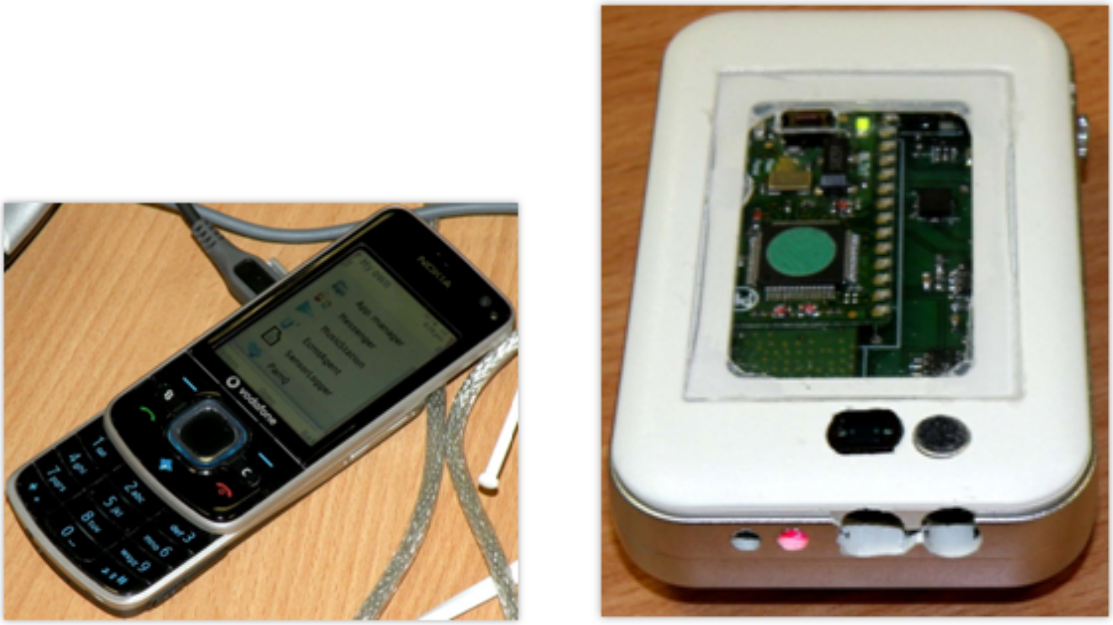
Heart rate monitors can be used to measure the number of heart beats per minute and transmit the data to a sensor network. Heart rate has been described as an important indicator of psychiatric state and is thus a useful measure of the state of subjects. For example, Henry *et al.* [2010] show that bipolar disorder patients have decreased heart rate variability during manic phases of the disease.

Home gateway

A home gateway is a device (typically a PC) that can be used to interconnect home LAN devices. A home gateway can also communicate with mobile gateway devices to share information between LANs and BSNs.

Light meter

Light meters measure the amount of light in the environment. Changes in the amount of light that a subject experiences may influence circadian rhythms and mood disorders, as described by Hill [1992].



(a) Typical Mobile Phone used in the PAM project.

(b) Wearable Device for Recording Accelerometer, Light and Sound Data.

Figure 4.1: Various Devices Used in the PAM Project. Photos provided courtesy of Pawel Prociow.

Mobile phone

For the purposes of this thesis, mobile phones are portable devices with relatively high computing capabilities that can exchange data over a variety of communication protocols. Subjects are assumed to carry their mobile phones with them when they are in transit and away from home. Their mobile phones can act as BSN gateway nodes and communicate with other BSN devices and home gateway nodes. An example mobile phone used in the PAM project which is shown in figure 4.1a.

Power meter

Power meters report the amount of battery life remaining in battery-powered devices, such as mobile phones.

Calendars, clocks and timers

Some devices are able to keep track of the current date and time using internal calendars and clocks. Timers may also be available to devices in order to execute actions after a given amount of time.

Wearable health monitor

Wearable health monitors are BSN nodes that are worn or carried by subjects. These can communicate with other BSN nodes such as mobile phones to report data about subject state or context. An example of a wearable health monitor is the one used in the PAM project which is shown in figure 4.1b. It contains a light meter, sound meter and accelerometer. The device communicates data to a mobile phone over Bluetooth, and exposes an API to control its sensor data capture rates.

4.4 Device Rules

Various aspects of AA Wireless Sensor Networks (WSNs) that incorporate numerous devices across BSNs and LANs are presented in the previous chapter. Such AA WSNs support the collaboration of devices and

are presented in the following sections.

The kinds of rules for this type of system have been grouped into two categories: device control rules and knowledge rules. Device control rules handle the way that devices sense and manage data, handle programming updates, and interact with subjects. Knowledge rules are used in the examination of the data to determine the states and contexts of the subjects, along with being used in the analysis of the quality of the data. A number of rules are presented below for both of these categories. They are grouped into rule sets and include discussions with respect to AA and Wireless Sensor Network (WSN) literature, along with usage examples.

4.5 Device Control Rule Group

Device control rules are used to manage how devices operate to collect and manage data, and interact with users. These rules are at the core of being able to collect, store and provide access to physiological and environmental data, as well as to present information to subjects, and prompt them for responses. Five rule groupings have been identified as shown in table 4.1: Data Management (DaM), Device Management (DeM), Do Not Disturb (DND) and Subject Interaction (SI). Each of the groups is described in the following subsections.

Table 4.1: Device Control Rules

Group	Rule
Data Management (DaM)	
	Automatic Data Transfer (ADT)
	Data Storage Through Processing (DSTP)
	Data Storage Unconditional (DSU)
	Redirect Data Stream (RDS)
	Inbound Data Screening (IDS)
	Outbound Data Screening (ODS)
	Retry Data Transfer On Unavailable (RDTOU)
Device Management (DeM)	
	Deactivate Immediate (DI)
	Data Recording Frequency (DRF)
	Power Management (PM)
	Time Synchronisation (TS)
Do Not Disturb (DND)	
	Do Not Notify Unconditional (DNNU)
Subject Interaction (SI)	
	Alert (Alert)
	Prompt (Prompt)

4.5.1 Data Management (DaM)

DaM rules are used to control the flow of information within a network. The rules may be written from the point of view of the sender of data, or from the receiver's point of view. Each device may use one or more of these rules to react to situations in which it receives data (either as a sensor source or as a recipient from another device). Reactions may involve processing, storing, or forwarding the data. These rules also determine to a large extent how individual devices respond to changes in the network. For instance, a rule may be used to determine where a particular device should send data when the device it normally sends data to becomes unavailable.

Data management rules are described below, including: ADT, DSTP, DSU, IDS, ODS, RDS and RDTOU. These rules may be used in a variety of combinations. For example a location monitor might use ADT to automatically report location data to an activity monitor. An activity monitor, in turn, may use DSTP

to limit the amount of data it stores by aggregating them before their storage.

These rules are selected from Ambulatory Assessment (AA) requirements and through consideration of general network operation. Some AA tools (such as Experience Sampling Program (ESP) and Purdue Momentary Assessment Tool (PMAT) which are described in section 3.2) do not require these types of rules since the collection and storage of their data occurs in hard coded locations on the sampling devices. This situation is suboptimal as it means that device failure leads to data loss, and researchers have to wait until the devices are returned before they have access to the data.

Tools require these types of rules once they extract data from devices at run time. However, this is often done statically with rules unchanging in time. MyExperience, for instance, uses a single rule exclusively, called "opportunistic synchronisation". It automatically forwards data from a database on a device to a central storage database. This is similar to the ADT rule presented here.

Automatic storage may not be appropriate in all circumstances. Alarm-Net (described in section 3.2.4), for example, did not just automatically store data, but, instead uses alternative processing rules to aggregate data and perform circadian rhythm analysis upon it. The rule DSTP is included here to accommodate such data processing. In addition to data processing, systems like Alarm-Net may support security policies, which are also included herein as IDS and ODS.

There are similarities between some data management rules and some call control features from classic Feature Interaction (FI) literature. For example Originating Call Screening (OCS) is used in call control and a similar concept is used here called ODS. There are not, however, exact matches in all cases. For instance, the data management rule DSTP has no counterpart in call control.

4.5.1.1 Automatic Data Transfer (ADT)

ADT transfers data from the device to a predefined recipient when the devices come within communications range. For instance, a mobile phone may automatically upload data via Bluetooth to a PC when they come into proximity of each other. Consider a sequence for a successful data transfer between the subscribing device (the sender) and a recipient device, and a connection failure case. The sender is the device where the rules are activated. The sender attempts to connect to the recipient. If the connection fails (for instance because the two devices are not within communication range), the sender waits a period of time before attempting to connect again. Once a connection is established, the data is uploaded from the sender to the recipient. The rule is terminated when the data transfer is complete. The sender begins in a listening state ready to receive connections. It streams data after a connection is made and then returns to a listening state after all of the data is transmitted.

This rule may be used in many cases, such as scenarios involving data being automatically transferred from a mobile phone to a home gateway. In these scenarios the mobile phone periodically attempts to connect and transfer data to the home gateway. The mobile phone may not be able to connect all the time if the communications protocol being used is temporarily unavailable (such as if the two are communicating over Bluetooth and the distance between the devices is out of range).

4.5.1.2 Data Storage Through Processing (DSTP)

The recipient of new incoming data uses DSTP to process and then store the data. The recipient waits listening for a connection. When one comes in, data is streamed and collected. This data is processed using appropriate algorithms once the streaming has completed. The processed data is stored and the device goes back to listening for more connections. From the sender's viewpoint, it is a simple matter of uploading the data. If a connection is established, the data is uploaded and the connection is terminated upon data transfer completion. Otherwise, if a connection fails to be established, the rule is terminated. The sender might be subscribed to additional rules that allow for alternative connection establishments or reconnection attempts.

Previous projects such as Alarm-net demonstrated the value of processing data in order to reduce network resources, and for determining context and states of subjects. Another usage example, derived from the Personalised Ambient Monitoring (PAM) project, involves a mobile phone that receives data from wearable health monitors, including accelerometer data streamed at rates above 20 readings per seconds. It is more power and memory efficient to store processed data instead of storing each reading.

4.5.1.3 Data Storage Unconditional (DSU)

Using DSU, the receiver of a new incoming data stream stores the raw data, either directly to a file or to a database. A subscriber listens for a connection from another device. The subscriber streams and stores the data when a connection is established, after which it goes back to listening for connections. If the sender fails to connect to the target recipient, the rule terminates. Alternatively, on success it uploads the data and completes the data transfer.

DSU is the basic data management rule available in most AA solutions. Any device with sufficient storage capacity may use this rule. For instance a mobile phone might record to a database all questionnaire responses and all streamed data from additional Body Sensor Network (BSN) devices used in long term studies.

4.5.1.4 Inbound Data Screening (IDS)

IDS blocks data delivery from a predetermined sender. The data recipient is the rule subscriber. It listens for connections, screens the sender against a list of known illegal senders, then listens again for legitimate device connections. The sender attempts a connection which succeeds or fails depending on whether it is screened by the recipient. If the sender is not screened then it transfers the data.

This rule can be used in security policies to make sure that devices are passed data from legitimate sources. Alternatively, it can be used to manipulate data routing, especially if devices are broadcasting data to multiple receivers. For instance, multiple gateways may be used in a home to screen data sent from a subject's mobile phone and prevent the data being duplicated in the home environment.

4.5.1.5 Outbound Data Screening (ODS)

ODS blocks data transfer from a subscribing sender device to a predetermined recipient device. The sender is the rule user and it attempts to connect to a recipient. At that point the subscriber screens the recipient to make sure it can legitimately receive the data. Standard data transfer ensues if a connection is established. If the recipient is screened then the connection is never established. The recipient will not receive a message indicating that screening transpired. Instead, it will listen for connections and stream data upon establishing them.

This rule might be used to maintain network integrity, prevent data flooding, and to handle certain data security concerns. An example of its usage is a scenario involving a mobile phone screening a particular home data gateway in order for the mobile unit to only upload to another device. This may be useful if the former mobile phone is less secure than the receiving device.

4.5.1.6 Redirect Data Stream (RDS)

Using RDS, the receiver of a new incoming data stream redirects data to another device. Here the subscriber device is both a data receiver and sender, and two other devices are shown. The first device sends data to the subscribed device. Assuming a successful connection and data transfer, the subscriber then attempts to send the data to another device. If that connection is successful then the subscriber goes back to listening after completing the data transfer. If the connection to the third device fails, the subscriber waits for a period of time specified in the rules, and retries the connection.

This rule is useful in scenarios involving home gateways, environmental sensors, and mobile devices. In such scenarios, environmental sensors may pass data to home gateways that forward data to mobile phones. Alternatively, the data flow might begin with the mobile phones and flow to the gateways, followed by transfer to the environmental devices.

4.5.1.7 Retry Data Transfer On Unavailable (RDTOU)

RDTOU allows the sender to retry a data transfer. This rule attempts to connect, and terminates on successful data send. If a connection failure occurs then the sender periodically retries the connection. Rules can limit the number of retries to be attempted or the amount of time until stopping retry attempts. The recipient device listens for a connection and downloads data once connected.

4.5.2 Device Management (DeM)

Device management rules are used to control common settings such as the device clock times and frequencies at which sensor readings are captured. The ones chosen for analysis include: Activate Immediate (AI), DRF, PM, and TS. AI controls when a device should be on or off. Each device may have access to a number of sensors, and their data recording frequencies may be controllable, along with the time that the sensors should be activated. In addition, power management can be used to make sure that battery operated devices do not expend energy unnecessarily.

4.5.2.1 Activate Immediate (AI)

AI allows a device such as a sensor to become active immediately. The recipient device is subscribed to the rule. It listens for a connection. If the sender connects and sends an activation command the subscriber will perform the given activity immediately.

4.5.2.2 Data Recording Frequency (DRF)

DRF determines how often a particular device, such as a sensor, will collect data. For example this rule may be used to set an accelerometer to collect data at a rate of 20 readings per second. Setting such data recording frequencies statically had been the mechanism used in most previous systems. Personalisation (the value of which is discussed in the previous chapter), however, requires adjustable frequencies. Logic rules may be used to set frequencies dynamically. The accelerometer might be set to generate 20 readings per second for standard behaviour, for instance, but then raised to a higher level if the subject's heart rate increases above a certain threshold.

The subscriber to the rule listens for a connection. It receives data once a connection is established and it calculates a new recording frequency based on its rules and the data it received. Then it alters the device frequency. The sender conducts a data upload sequence but does not know what happens when it sends the data.

4.5.2.3 Power Management (PM)

Power can be a scarce resource for some nodes in AA Wireless Sensor Networks (WSNs) but not all of them. Home gateway nodes, for example, may be plugged into the mains of the home so have greater access to power.

A variety of power management routines are suggested for the nodes that do have power consumption concerns, as described by Akyildiz *et al.* [2002]. They cite a number of different power saving rules such as lowering communication availability and reducing processing performance. For AA purposes, lowering communication availability can be translated into having the device use DND:DNNU. Lowering notifications allows the device to continue to gather and report data without the overhead of processing, and possibly storing, messages from external sources. Storing data to flash memory is shown by Blum & Magill [2010] to have a high power consumption cost for mobile devices.

Consider, for example, a mobile phone that uses a power management rule. When its power levels get below a certain threshold the power management rules can be activated allowing the device to continue to collect data from internal sensors but not from external sources such as might be sent from other BSN nodes. The loss of their data may be undesirable; however, these would be lost anyway if the mobile phone battery run down. Alternatively, the other nodes in the BSN might store the data and then forward it to the phone once it stopped using the power management rule. A subscriber to the power management rule will continue to stream data until power management is turned on. At that point the connection between the sender and receiver of data will break.

4.5.2.4 Time Synchronisation (TS)

Synchronised time amongst devices in a Wireless Sensor Network (WSN) is paramount for the smooth running of the network. Many rules such as device activation delay and data aggregation depend on a consistent view of time across the nodes. Time synchronisation is not trivial in sensor networks owing to unreliable network characteristics. For instance some of the nodes are mobile and subject to intermittent connectivity. Time synchronisation is discussed extensively by Römer *et al.* [2005].

The Pair-wise Synchronisation algorithm, presented by Van Greunen & Rabaey [2003], is a method of time synchronisation that can be used for AA sensor networks. Pair-wise Synchronisation can be used in single-hop and multi-hop networks to synchronise pairs of nodes. Upon establishing a connection, the initial sender transmits its local time as time stamp t1. The time stamp t2 is calculated by the subscribing device. This is calculated, according to Van Greunen & Rabaey [2003], by adding the time of message transmission (based on the distance between the nodes and the propagation characteristics of the links between them) and the offset between the two clocks to t1. The initial sender then receives a message with t1, t2 and a time stamp t3 from the subscriber. It calculates t4 by adding the message transmission time to t3 and subtracting the offset. This value is then passed to the subscriber.

This rule is useful, for instance, in maintaining synchronisation between the clocks of a home gateway and a mobile phone. The phone may subscribe to the rule and periodically receive time synchronisation messages from the gateway. These devices then go on to synchronise the rest of the devices within their respective BSNs and Local Area Network (LAN)s.

4.5.3 Do Not Disturb (DND)

DND rules are important in AA systems and are comparable to features in call control systems. These types of features began to appear in early AA systems such as in PMAT for subject questionnaire prompting, being disabled temporarily whilst the subject performs an activity such as attending a meeting. In call control scenarios this type of rule allows a subscriber to block incoming messages notifying that a call is coming in.

Sensor network systems can require additional rules to handle the fine granularity of control. Firstly, there is a distinction made between monitoring and notification. It may be the case that a subject does not want to be disrupted by notifications, but finds ongoing ambient monitoring acceptable. Alternatively, both may be intolerable. It may also be the case that some devices ought to refrain from monitoring or notification whilst others may remain enabled. The DNNU rule is chosen for conflict analysis.

4.5.3.1 Do Not Notify Unconditional (DNNU)

DNNU prevents the subscribing device from receiving notifications. The rule subscriber is in a blocked state instead of a listening state. Therefore connection attempts to this device from data senders automatically fail. This may be used, for instance, on a mobile phone if a subject wanted to exclude message passing to the device which might be desirable if the phone is engaged in other network activities such as online gaming. In addition, preventing notification to the device may be used to reduce subject interaction such as alerts about sensor reading threshold. If rules are set up to alert the subject about exceeding these thresholds, then the subject may suspend such alerts temporarily by using DNNU.

4.5.4 Subject Interaction (SI)

AA systems may be required to interact with the subjects being monitored. The original intention of Experience Sampling Method (ESM) systems, such as ESP, is to repeatedly prompt subjects for responses to questionnaires over the courses of their days. Over time, richer prompting schedules involving alternative question flows are built into the systems to support more detailed studies.

WSNs provide mechanisms for conducting ambient monitoring such that there need be very minimal interaction with the subjects. This has lead to naturalistic studies and may also be valuable in AA studies. The aim, therefore of SI rules should be to supply researchers with a range of interaction options.

Two types of SI rules have been identified. The Alert rule allows researchers to notify subjects to the occurrence of events or states. These may pertain to a subject, for instance if their anxiety levels are above a certain threshold, or to the network, for instance if a particular sensor in their home is out of communication with the rest of the system for too long. The Prompt rule allows researchers to solicit feedback from the subjects. These can be brief single promptings or they can be strung together to form into questionnaires.

The rules discussed here allow the question sets and alert messages to be customised to the individual. A number of issues about interfaces used for subject interaction (including accessibility and internationalisation) are beyond the scope of this thesis. Alerts and prompts in this work are based on

standard user interface widgets (such as pop-up alert boxes, text fields, check boxes, sliders and radio buttons) presented on hand-held devices or personal computers. According to Stone *et al.* [2007], this type of monitoring is appropriate for subjects that are visually unimpaired, have adequate fine motor and cognitive skills, and are willing to engage with the technologies. It should be noted however, there is no known theoretical limit to the way the information is presented, so alternative interfaces may be used such as television sets, or special actuators in the environment such as speakers and microphones.

4.5.4.1 Alert

The Alert rule can be used to notify subjects of events or state changes. Some AA systems provide the ability to personalise the alert messages and their output options. MyExperience, for example allows researchers to customise "MessageAction" and "NotificationAction" action types. These display message boxes with customised messages and fonts. In addition the latter action provides options for playing sounds, vibrating the device, forcing the back light of the device on, as well as providing a snooze rule whereby the alert might disappear for a period of time before reappearing.

A notifier device displays alerts to a subject and another one sends data to it. Upon connection and data transmission, the notifier formulates the message, determines the output options and displays the alert. Message formulation uses rules to embed the data into friendly, subject-personalised wording. Output options include personalised rules for whether or not to use alert sounds, what font types to use in the messages and how long to display the alert for.

4.5.4.2 Prompt

One of the key rules of the next generation of AA system is to embed subject context and state sensitivity into the question sampling – what Intille [2007] called Context Sensitive Ecological Momentary Assessment (CS-EMA). The Prompt rule allows researchers to solicit responses to questions from subjects. These can be chained together to form questionnaires for in-moment surveys. For example, a home gateway might determine that there is missing data and send a message to a mobile phone so that the phone can query the subject for annotation describing whether the device failed or whether the subject decided to not comply with the study protocol.

A device is set up to prompt for questionnaire responses. Upon connection and data transmission it formulates a question based on the data, determines the response options and displays the prompt. Question formulation uses rules to determine the correct question to ask in response to the data received. Response options may be used to determine whether responses are free-form or whether they should be selected from a list (either as a single selection or as multiple selections). Response options may also determine the action to perform subsequent to the response, such as displaying a follow on question or displaying a completion message. This therefore allows question chaining and branching. Personalised rules control how the question and response options are displayed in a similar manner to the notification rule.

4.6 Knowledge Rules

Knowledge rules are used to analyse subject state and context, and allow the devices to react to such information. Three groups of the rules have been identified as shown in table 4.2: Context Detection Service (CDS), State Detection Service (SDS) and Symptom Analysis (SyA). These rules are described in the following sections.

Table 4.2: Knowledge Rules.

Group	Rule
Context Detection Service (CDS)	
	Context Triggering System (CTS)
	Environment Detection (ED)
	Report Location (RL)
State Detection Service (SDS)	

	Report Subject State (RSS)
	State Triggering System (STS)
Data Quality Control (DQC)	
	Query for Missing Information (QMI)
	Report Device States All (RDSA)

4.6.1 Context Detection (CDS)

Context detection allows an Ambulatory Assessment (AA) Wireless Sensor Network (WSN) to monitor and adapt to variables describing the environment that the subject is in at any given point in time. These variables include ambient qualities such as light and sound levels, the time of day, and the location of the subject. Three rules are identified pertaining to contextual detection: CTS, which determines the reaction to detected changes of context, ED, which allows for the monitoring of variables pertaining to environment, and RL for handling location data.

4.6.1.1 Context Triggering System (CTS)

The CTS rule allows a subscribing device to respond to changes in contextual information. Actions are associated with changes to contextual data variables. These actions are triggered when the variables are within bounds. The subscriber listens for data connections. The sender reports contextual data upon establishing a connection. The subscriber receives the data and checks to see if it is within triggering bounds. If it is, the actions are performed and the device goes back to listening for additional connections. This type of rule can be useful for responding to changes in the environment. For example, a scenario involves a mobile phone receiving data about how bright it is in the area around a subject. A researcher may want to trigger a questionnaire on the mobile phone to ask a question about how the subject is feeling if the subject starts spending a greater amount of time in the dark than usual.

4.6.1.2 Environment Detection (ED)

ED allows the subscriber to publish environment variable data to recipient devices. The publisher attempts to transmit information to the recipients. The recipient listens for connections and then receives environmental contextual data. The publisher then attempts to connect and if it is successful reports the data.

4.6.1.3 Report Location (RL)

RL sends the location of the subject from one device to another. The location publisher attempts to transmit location information to recipients. The sender attempts to connect. The report location rules terminate if the connection fails. Otherwise, once connected, the sender reports the location then completes the data transfer, upon which the rules terminate. The recipient, meanwhile, listens for data connections. When one is made, the recipient receives the location data and then returns to listening for connections.

The usage of this rule, for example, can be seen in a scenario involving Global Positioning System (GPS) data captured on a mobile phone (either using an internal sensor or received from an external device). A home gateway device may register itself to receive such reports. When the mobile phone reports the location data, the home gateway receives the data and turns the sensors it controls on or off depending on the reported location of the subject in order to improve network efficiency by only recording data when the subject is present.

4.6.2 State Detection (SDS)

State detection provides an AA WSN with capabilities to monitor and adapt to variables describing the physiological, emotional and behavioural state of the subject. Two rules are identified pertaining to state detection: RSS, which reviews the state of the subject and STS for performing specific actions dependent on the states.

4.6.2.1 Report Subject State (RSS)

RSS transmits state variables describing the subject from one device to another. A sender device attempts to connect to a recipient. If the connection fails the report location rules terminate. Otherwise, once connected the sender reports the state data then completes the data transfer, upon which the rules terminate. The recipient, meanwhile, listens for data connections. When one is made, it receives the data and then returns to listening for connections. Other rules can determine what should happen to the data (such as processing, storing, or forwarding it) when they are received.

The usage of this rule, for example, can be seen in a scenario involving a mobile phone that publishes data on how a subject is feeling captured from a questionnaire on the phone. The mobile phone may report the data to a home gateway that alters its recording or actuation depending on the mood of the subject.

4.6.2.2 State Triggering System (STS)

The STS rule allows a subscribing device to respond to changes in state information. Actions are associated with changes to state data variables. Such actions are triggered when the variables are within bounds. The subscriber listens for data connections. The sender reports contextual data upon establishing a connection. The subscriber receives the data and checks to see if it is within triggering bounds. If it is, the actions are performed and the device goes back to listening for additional connections. This type of rule might be useful for responding to changes in subject state. For example a scenario involves triggering an action to change accelerometer data capture frequency depending on physiological variables received by a mobile phone.

4.6.3 Data Quality Control (DQC)

DQC provides rules to maintain the integrity of the data being captured across the network.

4.6.3.1 Query for Missing Information (QMI)

The QMI rule provides a subscribing device with a mechanism to take action if expected data is missing. For instance a scenario involves a home gateway using the rule to activate a questionnaire on a mobile phone to request a response from a user as to why expected environmental data is not being received. The subscriber listens for data connections. The sender reports data upon establishing a connection. If no connection is established for a given period of time then the subscriber will perform a particular action defined in its rules.

4.6.3.2 Report Device States All (RDSA)

The RDSA rule transmits all device state information from one device to another. This data includes variables for all devices connected to the given sending device. Device data is useful for maintaining the network. If a connection fails after the sender attempts to connect, then the report location rules terminate. Otherwise, once connected, the sender reports the device data and completes the data transfer. The recipient, meanwhile, listens for data connections. When one is made, it receives the data and then returns to listening for connections. Other rules may decide how the data can be processed, stored, or forwarded when they are received.

4.7 Conflict Examples

An important research question addressed in this thesis is whether or not WSN rules can conflict with each other. Here it is shown that conflicts can emerge.

Feature Interaction (FI), described in chapter 2, occurs when the operation of one feature is influenced by the operation of another one. Five types of interactions are presented: Shared Trigger Interaction (STI), Sequential Action Interaction (SAI), Looping Interaction (LI), Multiple Action Interaction (MAI) and Missed Trigger Interaction (MTI). Examples of rule conflicts corresponding to each of these types of interactions are described below. These examples show that rule-orientated systems are susceptible to each of the five types of conflict unless precautions are taken.

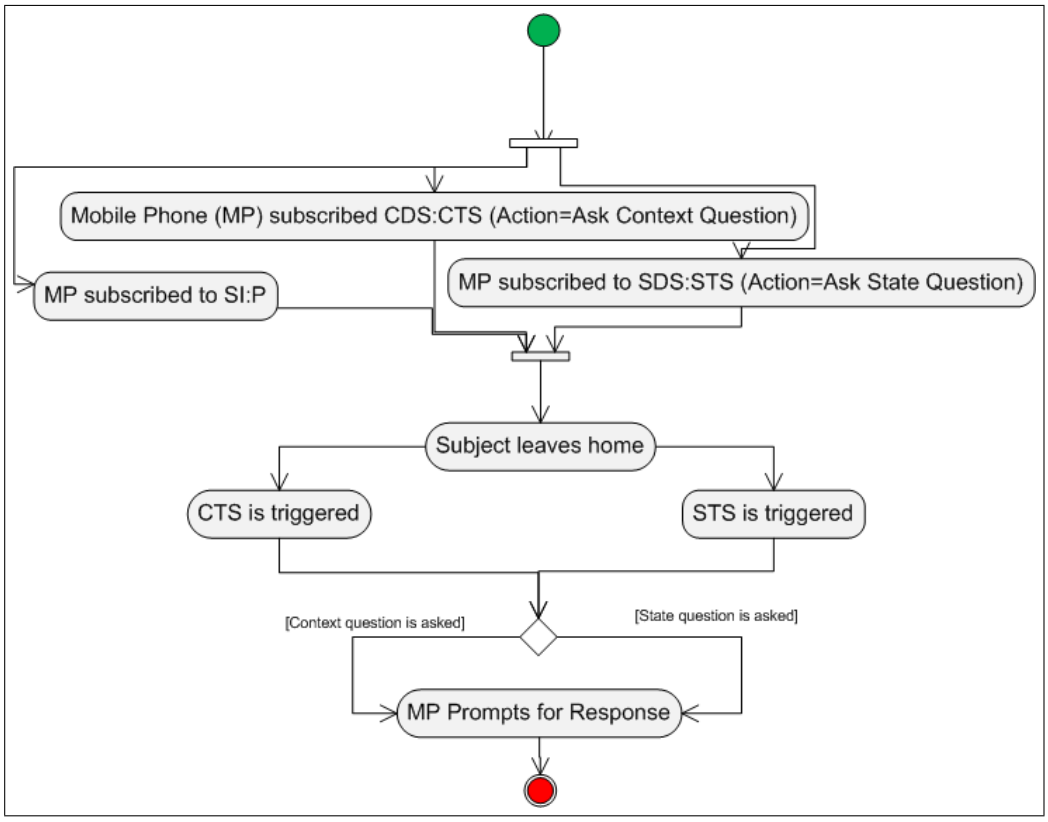


Figure 4.2: CDS:CTS vs SDS:STS Interaction Diagram

The examples describe situations that may occur when performing AA using a sensor network system. They use combinations of rules selected from the device control services and knowledge services, discussed earlier in this chapter. Two or more rules are used per example. Some of the examples show conflicts occurring on a single device and others show conflicts across devices. In addition, some of the examples highlight conflicts that occur within the same rule group (intra-service interaction), while others show conflicts occurring between multiple groups (inter-service interaction). Each description includes rules that conflict, devices upon which the rules operate, and state transitions that lead to the conflicts. UML state machine diagrams are shown for each example. These use the standard diagram format except for some of the MTI examples in which dashed arrows are used to indicate missed trigger transitions.

4.7.1 Shared Trigger Examples

Conflicts caused by multiple rules sharing a trigger occur when actions are performed by multiple rules in response to the same triggering event, and one or more of the actions are different from how they may have behaved if only one rule had responded to the trigger.

4.7.2 Example 1: CDS:CTS vs. SDS:STS

This example is characterised by the use of rules that run on the same device which is shown in figure 4.2. In this case a mobile phone has rules for CDS:CTS, SDS:STS and Subject Interaction (SI):Prompt (Prompt). CTS has rules to trigger the prompting about what activity the subject is engaged in when leaving home. Similarly, rules have been set up using STS to prompt the subject about the emotional state being experienced when the subject's behavioural state changes from sitting to walking. A conflict can occur when the subject walks away from home. Here, both rules are triggered, however only one can prompt for a response to its question.

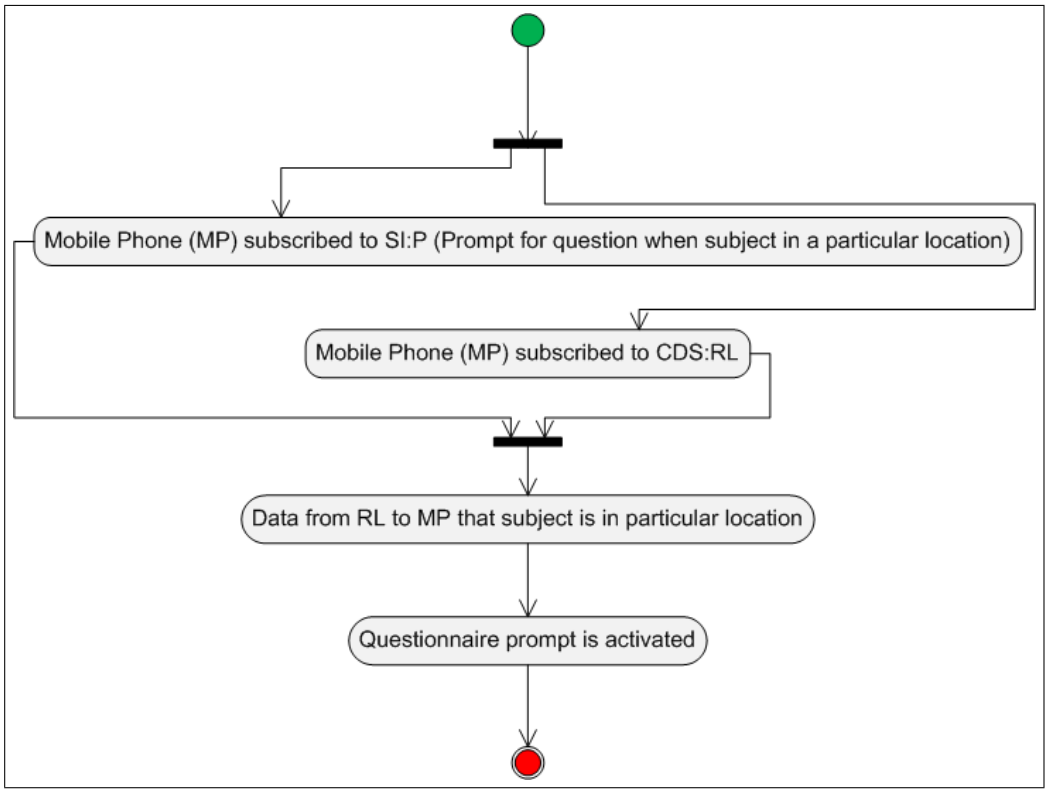


Figure 4.3: SI:Prompt vs CDS:RL Interaction Diagram

4.7.3 Sequential Action Examples

When one rule is triggered in response to the actions of another it is said that a conflict of type SAI has occurred. Side-effects of these types of conflict are not always harmful. The examples described here show a mixture of beneficial and non-beneficial conflicts.

4.7.3.1 Example 1: SI:Prompt vs. CDS:RL

This example is an of example of SAI involving a single device with two rules from different groups running on it. Figure 4.3 shows that a mobile phone has rules for SI:Prompt and CDS:RL. The rules of SI:Prompt indicate that it should prompt for a particular question when the subject is in a particular location. SI:Prompt is activated when CDS:RL reports that the subject is in the correct location. Here the SAI is beneficial since SI:Prompt intentionally relies upon CDS:RL.

4.7.3.2 Example 2: Data Management (DaM):Redirect Data Stream (RDS) vs. DaM:Automatic Data Transfer (ADT)

This is an example of a conflict involving two rules which come from the same group but run on two separate devices. Figure 4.4 shows that the devices involved are a mobile phone, a Wearable Health Monitor (WHM), and a home gateway. The phone has the rule DaM:RDS and the WHM has the rule DaM:ADT. DaM:ADT is programmed to transfer data to a home gateway. SAI occurs because the ADT rule leads to the sequential activation of the RDS rule.

4.7.3.3 Example 3: SDS:STS vs. Device Management (DeM):Data Recording Frequency (DRF)

The trouble determining the quality of SAI is exemplified in this example. The example shown in figure 4.5 looks beneficial. Here, a mobile phone is subscribed to SDS:STS with rules that ensure that if the subject's heart rate exceeds a particular threshold, then the capture rate of the accelerometer in the

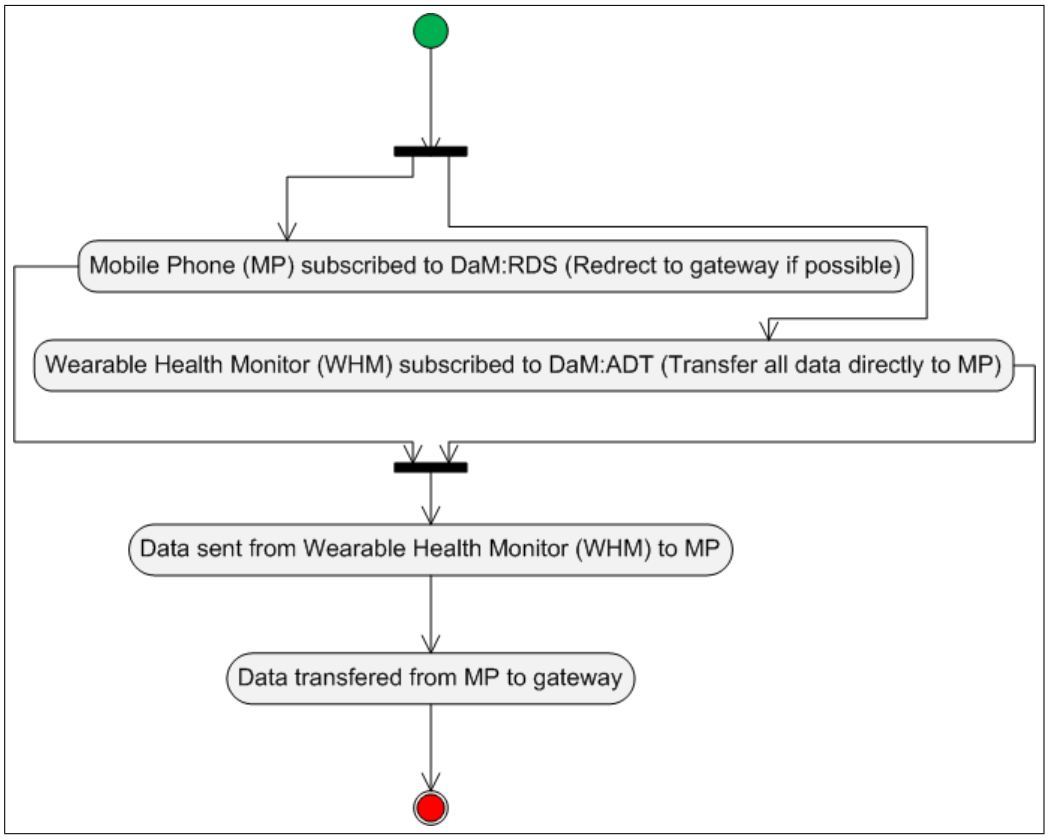


Figure 4.4: DaM:RDS vs DaM:ADT Interaction Diagram

phone is adjusted. In addition the phone is subscribed to DeM:DRF which will perform the adjustment on receiving a signal. STS will signal DRF in a normal working condition. Consider, however, what should happen if the phone also had been set up with CDS:CTS in a manner akin to STI example 1. In that case then it could be possible that a race condition could arise such that it would be uncertain whether the actions of STS would be performed, and therefore DRF would also be uncertain of firing.

4.7.4 Looping Examples

LI occurs when rules are chained such that the calling of one of them leads to the calling of another, and this in turn leads to a sequence whereby the original is called again.

4.7.4.1 Example 1: DaM:ADT vs. DaM:RDS

This example shows how two rules from the same group may be used on different devices and lead to LI. Figure 4.6 shows that a mobile phone has the ADT rule set up to send data to the home gateway. The home gateway has RDS set up to redirect data streams to the mobile phone. This, for instance, could be necessary if a home gateway is partially offline for repairs. This configuration of rules across devices leads to a loop when the mobile phone attempts to transfer data to the home gateway and automatically receives back the sent data.

4.7.4.2 Example 2: DaM:Retry Data Transfer On Unavailable (RDTOU) vs. DaM:Inbound Data Screening (IDS)

Figure 4.7 shows that this example involves rules running on a mobile phone and a gateway. The mobile phone has a rule for RDTOU. The home gateway has the IDS rule and has restricted itself from receiving data connections from the mobile phone. This results in a RDTOU loop because IDS blocks the connection.

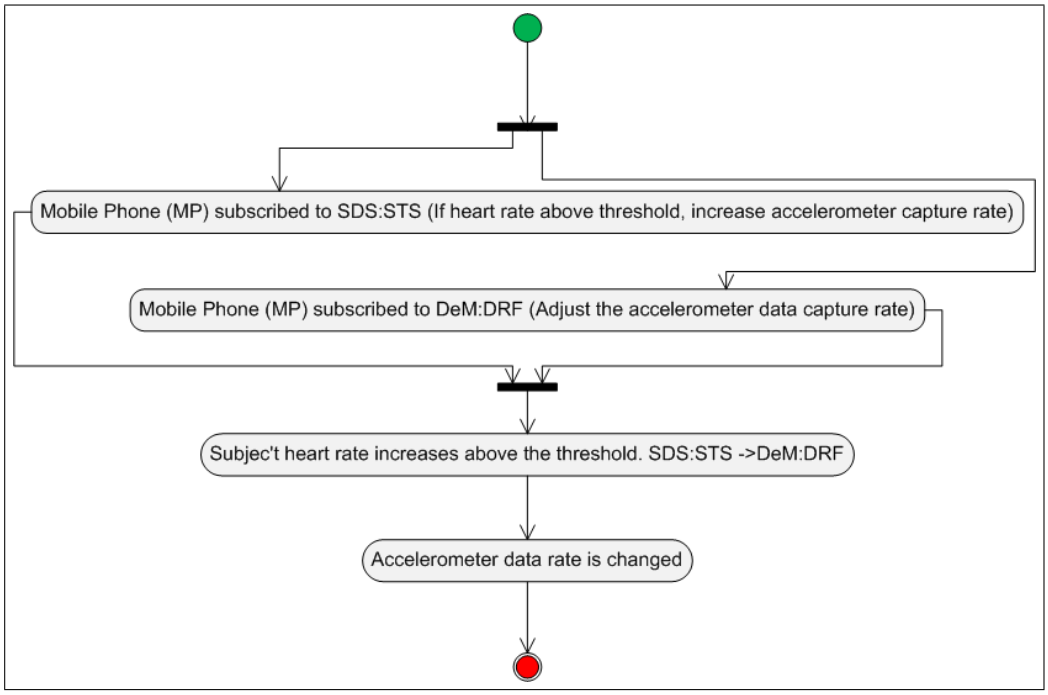


Figure 4.5: SDS:STS vs DeM:DRF Interaction Diagram

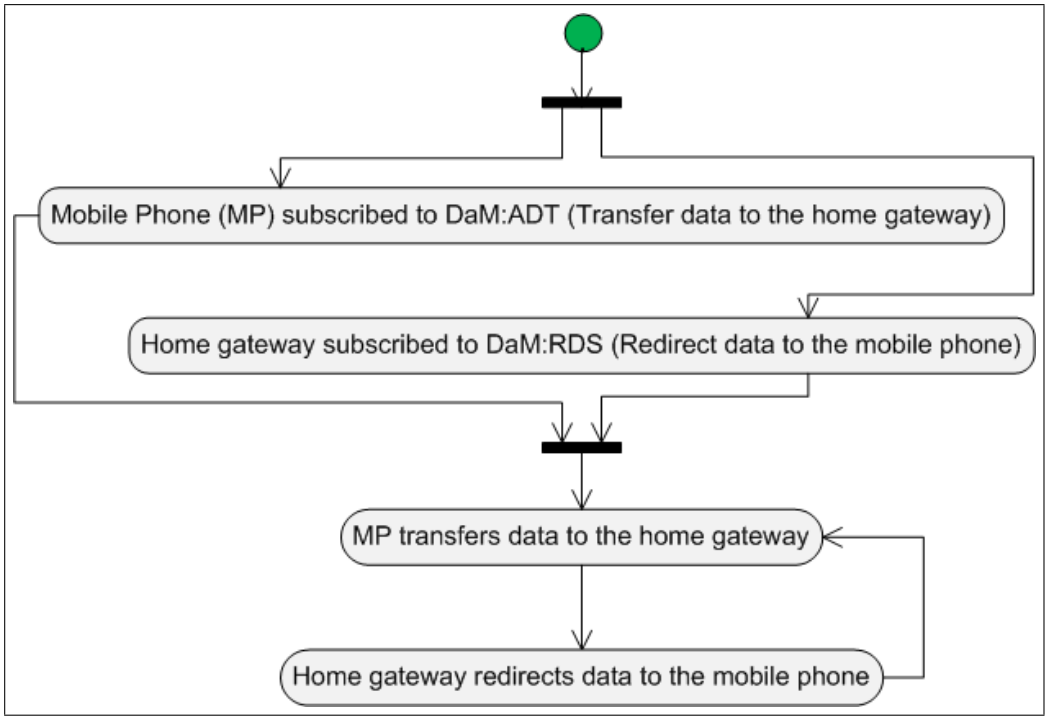


Figure 4.6: DaM:ADT vs DaM:RDSInteraction Diagram

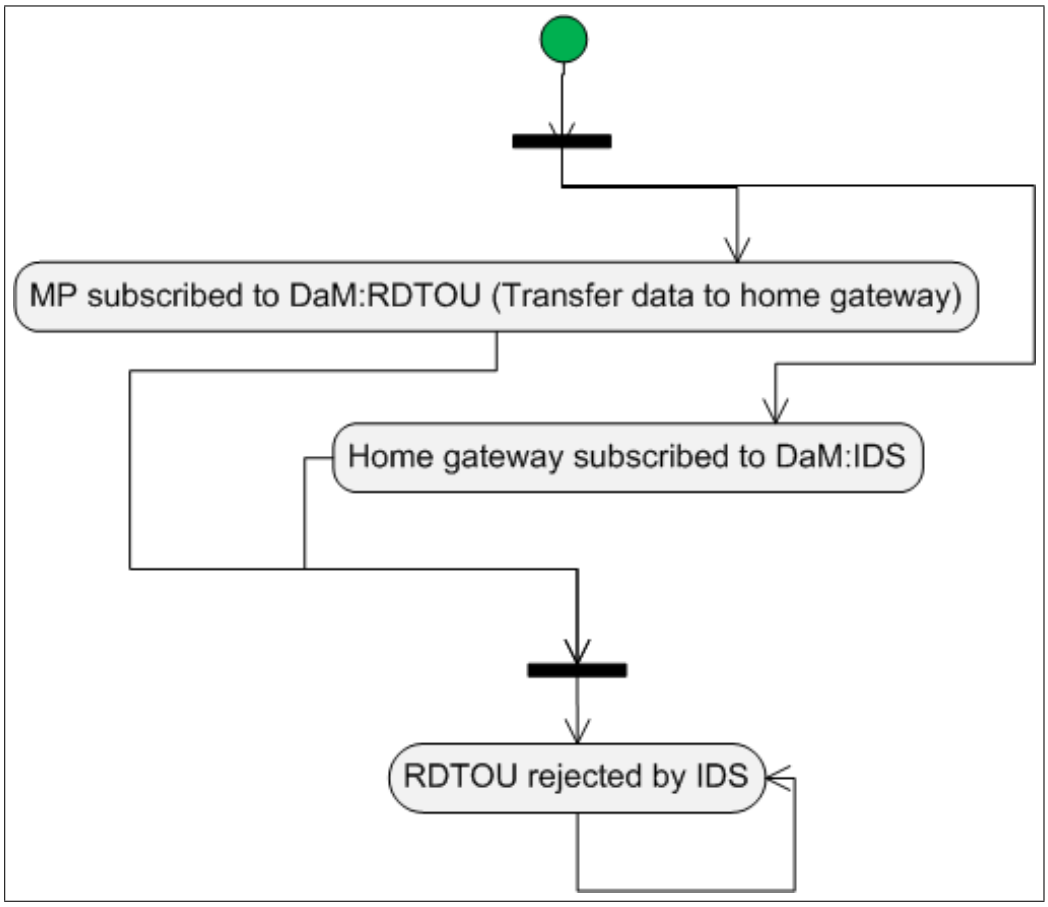


Figure 4.7: DaM:RDTOU vs DaM:IDS Interaction Diagram

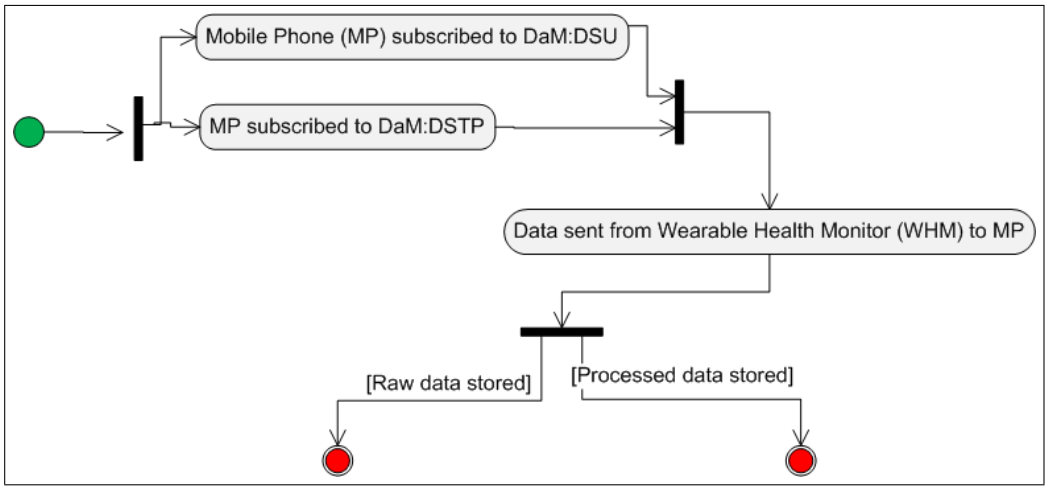


Figure 4.8: DaM:DSU vs DaM:DSTP Interaction Diagram

4.7.5 Multiple Action Examples

MAI occurs when different rules attempt to control the same device at the same time.

4.7.5.1 Example 1: DaM:Data Storage Unconditional (DSU) vs. DaM:Data Storage Through Processing (DSTP)

This example, shown in figure 4.8, is characterised by the use of rules from the same group running at the same time on a single device. Two rules on the mobile phone control what should happen to new data. Both indicate that something should happen to the same data, one to store the raw data, and the other to process the raw data and store the result. This result may be benign, such as if both are storing to different locations. If, however, the storage location is the same for both rules, and metadata are not used to differentiate raw and processed data, then this conflict will corrupt the data store. It is possible that data may be sent to the same location multiple times or, worse, data may be written by the execution of the first rule and then overwritten by the second one. The problems are compounded because race conditions arise as to which data is written first.

4.7.5.2 Example 2: DaM:ADT vs. DaM:Outbound Data Screening (ODS)

This example involves the use of DaM:ADT and DaM:ODS rules running on the same mobile phone. Figure 4.9 shows that ADT establishes that data should be transferred from the phone to the home gateway but ODS has a rule blocking connections from the phone to the home gateway. When new data comes in from a WHM to the mobile phone, one of the two rules is invalidated by the other. Either the data will be transferred or the connection will be blocked. A race condition would decide which rule has precedence.

4.7.5.3 Example 3: Do Not Disturb (DND):Do Not Notify Unconditional (DNNU) vs. DeM:Time Synchronisation (TS)

Figure 4.10 shows that this example uses rules from two different groups running on the same mobile phone. DND:DNNU and DeM:TS are set up on the mobile phone. If a time synchronisation message is sent from the home gateway to the mobile phone, the execution of one rule will invalidate the other. Either the data will be received or the connection will be blocked. A race condition would decide which rule has precedence.

4.7.6 Missed Trigger Examples

MTI occurs when one rule operates such that it prevents the triggering of the operation of a second one.

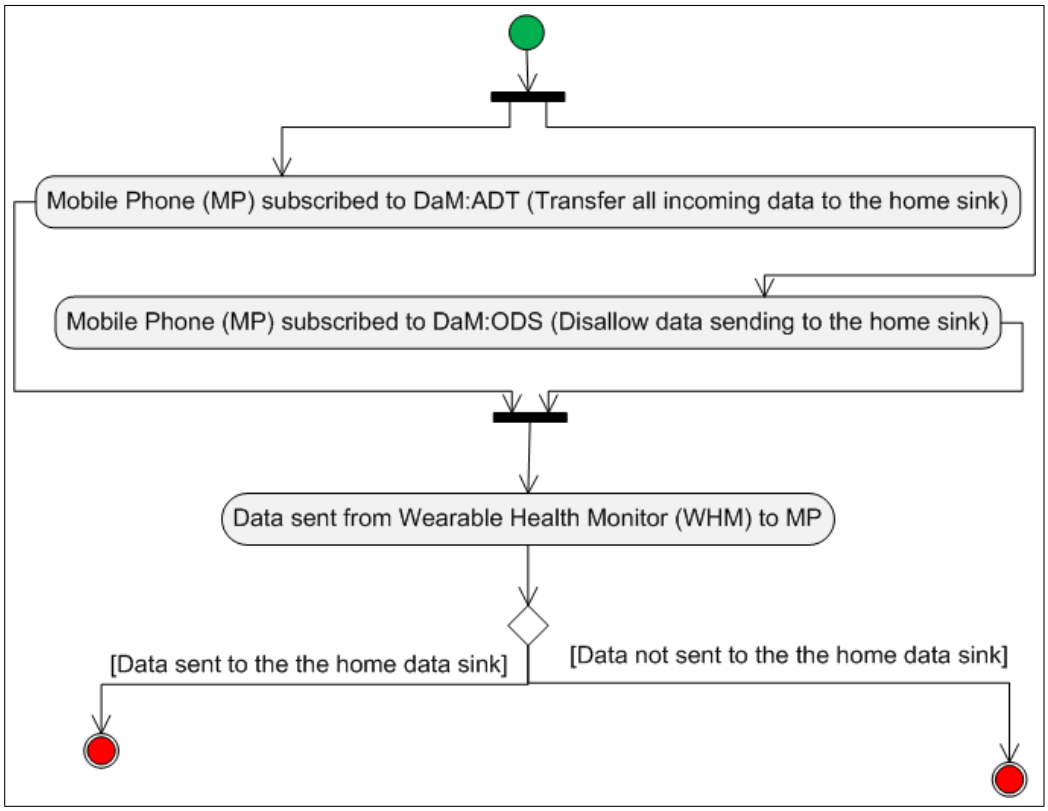


Figure 4.9: DaM:ADT vs DaM:ODS Interaction Diagram

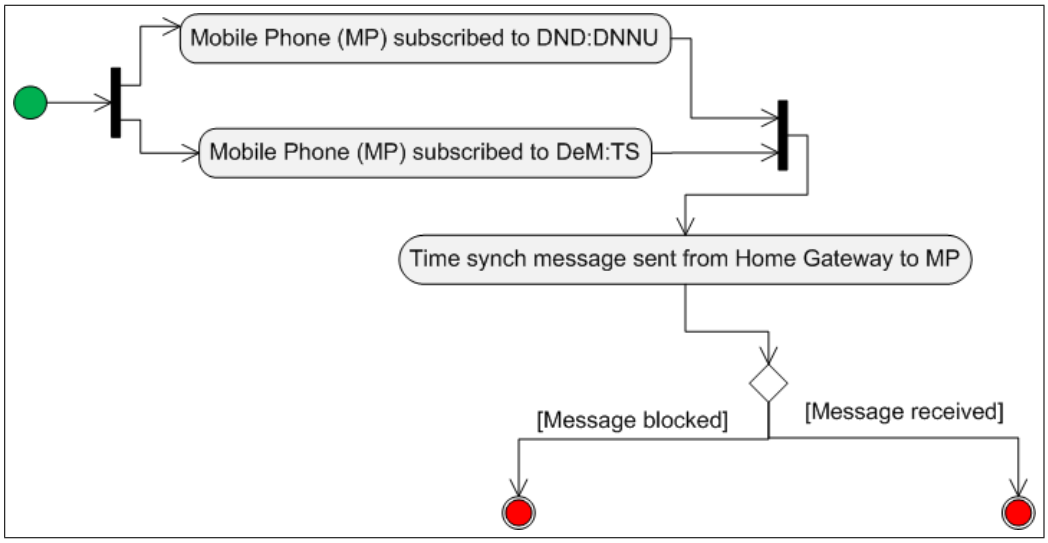


Figure 4.10: DND:DNNU vs DeM:TS Interaction Diagram

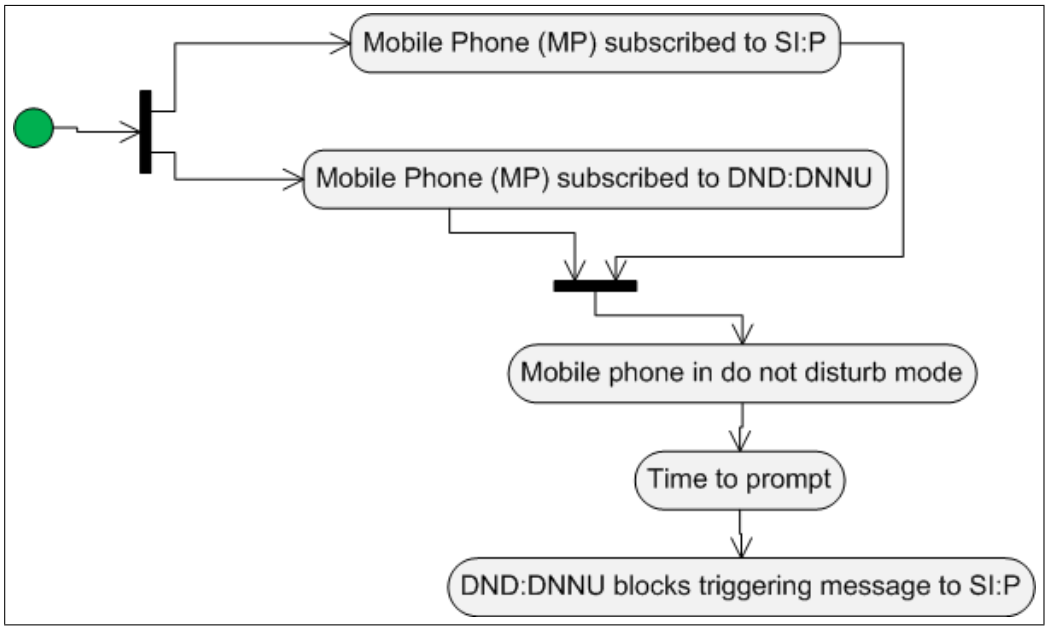


Figure 4.11: SI:Prompt vs DND:DNNU Interaction Diagram

4.7.6.1 Example 1: SI:Prompt vs. DND:DNNU

This example shows that the use of rules from different groups operating on the same device can lead to MTI. Figure 4.11 shows that a mobile phone has rules for SI:Prompt and DND:DNNU. When DNNU is active on the mobile phone it prevents SI:Prompt from being triggered.

4.7.6.2 Example 2: SDS:STS vs. CDS:CTS vs SI:Prompt

In this example MTI occurs such that a rule is allowed to operate, but then prevents itself from being called a second time. Figure 4.12 shows that a mobile phone has rules for SDS:STS, CDS:CTS and SI:Prompt. The sequence shown depicts that STS calls SI:Prompt when the subject enters a particular state. CTS also calls SI:Prompt whilst the subject is in the middle of completing the questionnaire from the first prompt. This blocks the Prompt rule called by CTS since only one prompt can be displayed at a time.

4.7.6.3 Example 3: CDS:CTS vs. DeM:Activate Immediate (AI)

This example presents the Brief Visit Home (BVH) example discussed in section 4.2. It shows that MTI can arise when identical rules are used on multiple devices. Figure 4.13 shows that MTI occurs when the CTS rules contain delays in its transmission of an activation message to the home gateway.

4.8 Example Discussion

The examples presented in this chapter show that the use of multiple rules within and across devices can cause a variety of conflicts.

These examples present devices individually controlled by rules that may interact in such a way as to make them susceptible to each of the five types of interactions discussed in chapter 2. Conflicts in such systems arise from conflicts within the same service, or from different services. Wilson [2005] has previously shown that multiple types of interactions can emerge by the use of intra-service features. It is shown here that multiple types of conflicts can also be caused by rules across groupings. For example, STI example 1 and MTI example 2 use the same rules which all come from different groups.

The examples show that problems can arise regardless of whether all of the rules are operating on a single device or spread across devices. For example, MTI example 1 uses multiple rules on the same device whereas MTI example 3 used rules on different devices.

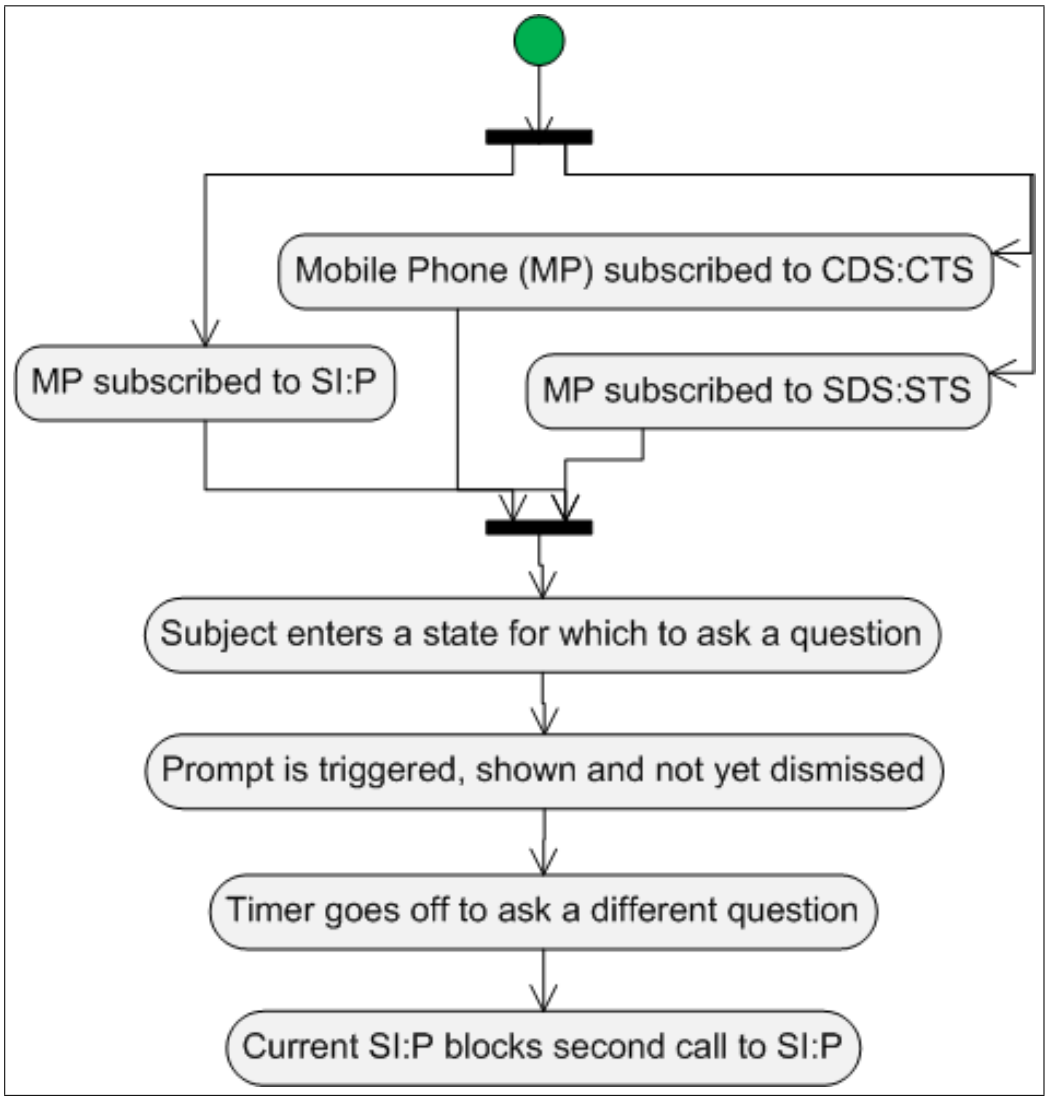


Figure 4.12: SDS:STS vs CDS:CTS Interaction Diagram

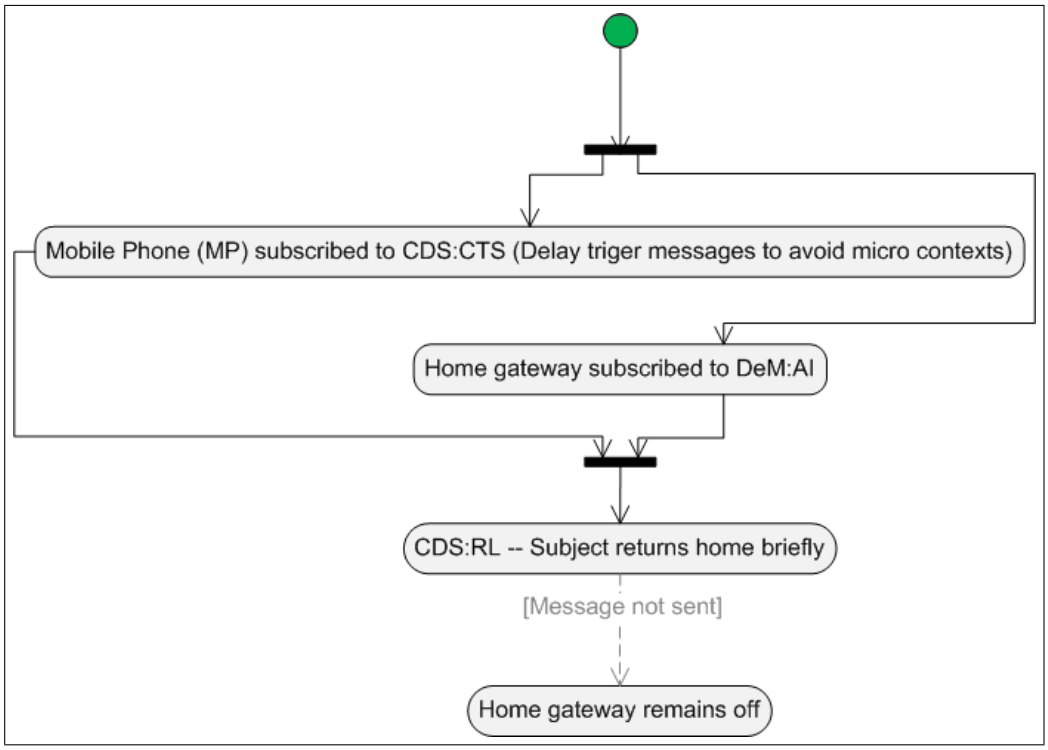


Figure 4.13: CDS:CTS vs DeM:AI Interaction Diagram

4.9 Summary

This chapter makes two important contributions to the thesis. It describes device and knowledge management rules that are distributed across multiple nodes in a sensor network. It also shows that the use of such programmable rule sets can lead to conflicts. This chapter lists sample devices for AA along with network services and rules. These are used in a number of examples that described how five types of conflict may emerge through their usage. Examples are given that exemplify inter-service and intra-service conflict for multiple rules running on a single device or on multiple devices. The examples also show that multiple types of conflicts may be caused by the same rules. Rule conflict is a problem that can afflict rule based interacting devices. The following chapters build on this one to provide an approach to detecting and resolving such interactions in a live environment. They describe the approach and the test infrastructure, and provide data that shows how the examples do lead to problems if left unresolved.

Chapter 5

Detecting and Avoiding Low-Level Rule Conflict

5.1 Introduction

Personalising devices and networks to monitor subjects *in situ* exposes the need to use adaptable programming approaches. Reliability concerns, however, are shown in the previous chapter to arise because rules can conflict. Such conflict may impact on system functionality and decisions affecting subject lives.

This chapter presents a novel conflict detection and resolution system for sensor network systems called CoLlaborative Information Processing Protocol and Extended Runtime (CLIPPER). It builds on the rules from previous chapters and attempts to resolve the five conflict types previously discussed. A mixture of techniques are presented for analysis, conflict detection, and resolution. Subsequent chapters present the evaluation of these methods. The distinctive contribution of this chapter is the description of an approach to detecting and resolving five forms of rule conflict within Ambulatory Assessment (AA) sensor networks.

This chapter begins with a review of concerns regarding approaches to conflict detection and resolution. This is followed by the description of novel detection strategies and resolution techniques. The chapter is concluded with a description of how the approaches can be automated for run time analysis and resolution.

5.2 Goals, Assumptions, and Choices

A variety of issues were considered in the development of the conflict detection and resolution system. Some of these included: fitting analysis into the development life cycle, the degree of user involvement in the analysis process, the level of detail that rules should operate on and how to handle various data types.

The development life cycle is discussed by Calder *et al.* [2003a] and Kolberg [2004]. Calder *et al.* [2003a] describe how Feature Interaction (FI) detection and resolution may be appropriately handled at different stages of the system life cycle. They highlight four types of FI analysis: software engineering techniques, formal methods, online approaches, and hybrid approaches. In certain cases it may be possible to perform offline analysis at design time to analyse system specifications using software engineering techniques or formal methods. This, however, requires a considerable amount of knowledge about the users, the devices, the services, and the run-time environments. All of this information may not be available to any single development team member during development, or at all. Therefore, run time analysis and resolution may also be required. Consider, for example, the Brief Visit Home (BVH) Missed Trigger Interaction (MTI) example 3 in section 4.7.6.3. The example demonstrates that rules may belong to different groups and run on various devices.

Kolberg [2004] uses an offline approach to minimise interactions before deployment given the information at hand, then online techniques post-deployment to further maintain the integrity of the network as it changes over time. CLIPPER may be used for offline analysis. It may also be used post-deployment

for analysis when device rules are changed (such as if a rule is inserted, modified or removed). This allows CLIPPER users to make decisions based on run-time environment information when changes occur. CLIPPER requires direct access to the rules, which online techniques in telephony Feature Interaction have attempted to avoid because telephony networks involve multiple providers that are unable or unwilling to share information with each other. However, the form of AA sensor network that CLIPPER was developed for requires a high degree of transparency regarding the stored data about the subjects and the device rules that are used to collect and maintain the data.

The main users of AA systems are the subjects of analysis. For these users, CLIPPER follows the notion from Wilson [2005] that users should not be involved in detecting and resolving interactions. Device administrator are another type of user in charge of deploying devices into subject environments. CLIPPER provides details of the analysis and resolution process to device administrators, in order that they can make informed decisions to minimise conflicts amongst the devices.

The approach embodied by CLIPPER can be seen to combine software engineering elements with formal methods along the lines described in Turner [1993]. Device rules and conflict rules are formally described in the Event Calculus (EC). CLIPPER analyses low-level device behaviour rules by examining execution sequences of combinations of declarative sensor network device rules. CLIPPER can be run offline purely as a formal analytical engine for the rules. However, the software engineering side of CLIPPER also allows these rules to result in action statements that can be reified into actual device behaviour when the rules are in concordance. Rule formats are presented below for device programming, as well as conflict detection programming. No assumptions are made as to the semantic typing of the data for use with these rules, since the rules are to be used to exchange, process and store a variety of data types (such as sensor readings, questionnaire responses and data aggregates). CLIPPER uses mediators to ensure that devices communicate with each other in a stable way. Mediators are similar in nature to the notion of feature managers discussed by Marples [2000], described in chapter 2.

5.3 Conflict Detection Approach Characteristics

This section describes an approach to using the EC to perform rule conflict detection in CLIPPER. It describes techniques and details specific to the detection of detecting each of the five types of conflicts discussed in chapter 2. Here, conflict means rules that contain Σ, EC or Δ clauses that are reached or ignored as a consequence of querying another rule within the same executing logic program.

CLIPPER finds instances of different forms of conflict amongst collections of device rules. It does so by triggering devices rules and passing messages to them. Rule execution sequences are analysed to determine whether they lead to conflicts. It ignores the contents of the messages and the semantic meanings of the rules. CLIPPER requires that the syntactic structure of testable rules and the rule clauses are presented in the EC format.

CLIPPER resolves goals in the following manner. It begins by loading the device rules and then proceeds to check for conflicts between pairs of rules (including checking rules against themselves). Checking a pair of rules involves two phases: initialisation and detection.

The initialisation phase resets the Prolog environment by removing all assertions from it. It then adds a number of time points (establishing a linear order amongst them) and initialises a fluent that represents a message that can be sent to the rules.

The detection phase involves passing device rules, time points and messages to conflict detection rules. The conflict detection rules are then used to evaluate whether or not the device rules are concordant or conflict, and to record evaluation results.

5.3.1 Device Rule Notation

The rules describing device operation are written using the form of the EC presented in chapter 2. These rules are written using Prolog rule syntax and are composed of a head followed by conditions containing initial situation (Δ_0) sentences, domain dependent sentences (Σ), and narrative (Δ) sentences. The rule heads are limited to the form *rule_name(Trigger,T)*, where *rule_name* is the name of the device rule, *Trigger* is the triggering message and T is the initial time point. Although *Trigger* and T are the only arguments of the rule, it is possible that additional variables are used within the rule. These variables may be instantiated by assertions from other rules or the triggering environment. This allows the rules

to be written in a generic format with variables that can be instantiated as appropriate for the device and its context, with a similar outcome as Blair & Turner [2005] policy variables.

Δ_0 sentences can be used to describe initial state constraints. These terms may be useful in cases where rules are only activated when they are in particular states. For instance, the activation of screening in Inbound Data Screening (IDS) requires that the sender (Alice) be on the screening list of the recipient (Bob). This fact can be asserted into the initial situation as:

- *Initially_p(bob_screen(alice)).*

Σ sentences describe actions that may be taken by rules. For example processing data can occur as a result of Data Storage Through Processing (DSTP) (see page 45 for DSTP description). This can be described in a Σ sentence as:

- *Initiates(receive_data,process(data),T2).*

Δ sentences describe the activity sequences of the rules. A sender attempting to connect to a recipient device, for example, is an activity of IDS. This can be described in the EC terminology as:

- *Happens(sender_attempts_connection,T1).*

5.4 CLIPPER Analysis

The analysis components of CLIPPER is a Prolog program that is spread across four core files. Other files are needed as well to describe the conflict rules and device rules being analysed. These components are extensible and can be used as the basis for programs to analyse types of rules other than the ones presented here, by modifying the conflict rules and device rules to match the domain of interest. The following sections describe each of the core files, along with the conflict rules and the device rule files.

The core analysis files consist of detection.pl, utility.pl, eventCalc.pl and rules.pl. The detection.pl file contains the starting goal term, which loads the other necessary files as part of its resolution. The source code for these files is presented in Appendix A.

The goal resolution flow is shown in figure 5.1. It shows that after the files are loaded the groups of rules are analysed until no permutations remain. To analyse the groups, CLIPPER does some initialisation and sets up the rules with the correct message and time values. It then uses the given conflict rule to check for conflicts between the device rules, and then writes the output. When no more permutations of rules remain, a completion message is written and the goal is satisfied.

Detection

The predicates of detection.pl are shown in listing A.1. These are used for the main goal which is "analyseConflicts". The "singleCheck" predicate is of particular interest since it is the one that is repeatedly

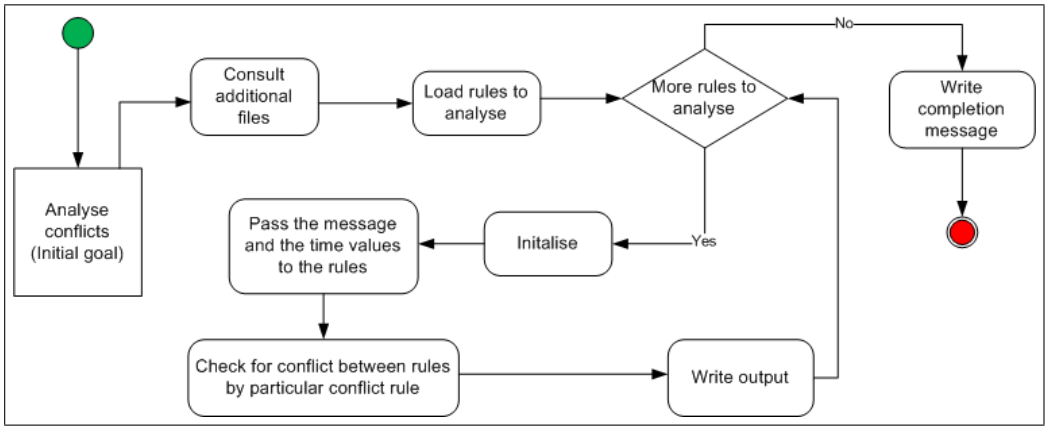


Figure 5.1: Goal resolution flow diagram for the rule interaction analytical framework.

called with the various rules. This predicate initialises the world of discourse and device rules. It then calls the conflict detection rule and passes in to it the device rules, a message, and the starting time points for the device rules. The conflict rule used in the listing is for Sequential Action Interaction (SAI), and some others are commented out to show that they can be used instead.

Utility

The predicates of utility.pl are shown in listings A.2. These predicates are used for two main purposes: to permutate through lists of device rules and conflict rules, and to control output strings. The output of CLIPPER can be written to the screen and/or to file.

EventCalc

The core EC predicates are presented in listings A.3. These present a simple form of the EC. The "initEC" rule may be of particular interest as it is not directly part of the EC core rule set, but rather insures that a consistent environment for the EC is maintained. Device rules assert EC sentences when they are called. The "initEC" predicate ensures that these are retracted prior to conflict detection for particular sets of device rules.

Rules

The rules.pl file, described in listing A.4 shows an example of device rules that will be analysed. The listing shows that the Looping Interaction (LI) analysis rules are being used, but any other rules can be used by consulting the appropriate file and adding the rules to the "Rules" list.

5.4.1 Conflict Detection Rule Notation

The conflict detection rules are written using EC goal (Γ) sentences along with additional clauses for ordering test sequences and writing output. The Γ sentences are used to ensure that particular fluents hold at given points in time. The fluents that should hold are dependent on the nature of the interactions being analysed. MTI analysis, for instance, uses Γ sentences to check whether the contents of a message continue to exist when they are passed between rules. For example, a Γ sentence can be written as:

- *holdsAt(message(Content),T2).*

In order to perform conflict detection, CLIPPER requires conflict rules and device rules in addition to the core rules described above. Various rules are presented in chapter 5. These are presented in a simplified format for explanatory purposes. Listing A.5 depicts the expanded form of the conflict detection rules.

5.5 Conflict Detection Algorithms

Five forms of conflict analysis are chosen for consideration as discussed in the previous chapter. The following sub-sections describe algorithms for each of these using the conflict detection rule notation.

5.5.1 Missed Trigger Detection

Detecting MTI can be accomplished by testing rules sequentially to ensure that a common fluent holds before being passed to each of the tested rules. The fluent can be considered as a type of triggering message that should remain in a consistent state between rules. Such an approach need not make any assumptions about the contents of the message, nor about the actions that should be performed by the rules, nor about what the rules do upon receiving a message.

Figure 5.2 depicts MTI detection. The analytical framework evaluates a MTI concordance rule with arguments that consist of a pair of feature description rules, time points for the start times of each of the rules, and the message fluent. The fluent initially holds prior to being passed to the first rule. Rules conflict if the fluent becomes clipped prior to the execution of a rule.

Feature description rules satisfy two qualities: *successfulness* and *destructiveness*. Regarding successfulness, conditions of a rule may or may not allow it to complete all of its predicates. If it does complete

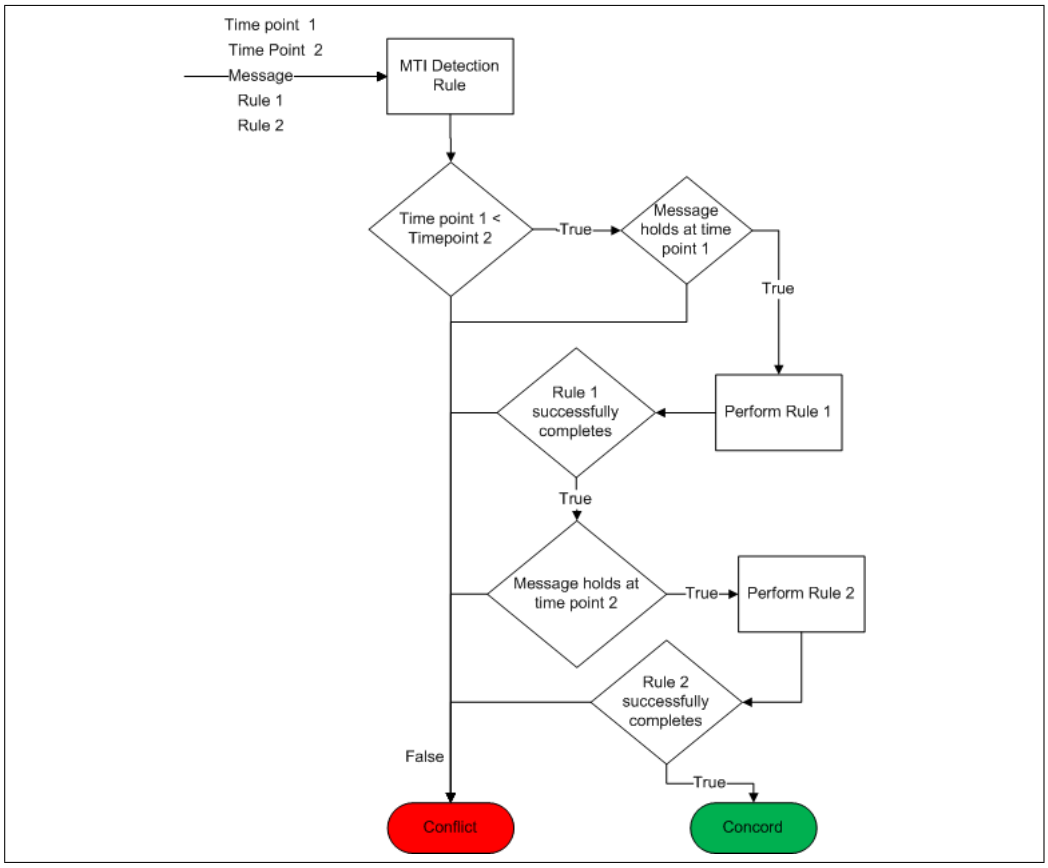


Figure 5.2: MTI Detection algorithm diagram

them then the rule is said to be successful, otherwise it is considered a failure. As for destructiveness, a rule may or may not alter a given message by destroying the message or modifying it in such a way as to invalidate it. Modifying a message is seen as terminating it at a given time point and initiating a new message at either the same time point or at a later one. A destructive rule may be described in the EC terminology by saying that it terminates the message at a given time point, and a passive rule as one that does not. Rules can therefore be described as belonging to one of the four combinations of these types: Passive-Failure, Passive-Successful, Destructive-Failure, and Destructive-Successful. Feature rules conflict by MTI when the initial rule is a destructive one.

The MTI detection algorithm has two phases, an initialisation phase followed by a detection phase. The initialisation phase resets the universe of discourse to a basic state whereby there are four distinct points in time. This phase also asserts that some action occurs at time point zero and generates a message at that point. The detection phase determines whether or not the message will trigger an action at time point three following its passage from the first rule at time point one to the second at time point two. If the message will trigger an action the rules are said to concord, otherwise they are said to conflict. The detection algorithm is described in listing 5.1. This code listing, along with the rest in this chapter, shows Prolog code that has been simplified for explanatory purposes. Such simplification removes user output information and reduces the instructions that portray inert events (such as writing output to file).

An example of potentially conflicting rules is given in listing 5.2. Both of these rules (R1 and R2) receive variables for the triggering message fluent and the time points for when they are respectively executed. R1 is destructive because it terminates the triggering message, whereas R2 is passive because it does not terminate the triggering message. The results of analysing these rules for MTI are shown in table 5.1. The table shows that if R1 is used prior to a second usage of the rule or the use of R2 then MTI occurs. Two instances of R2, on the other hand will concord if they are used together, as will R2 called before R1.

Table 5.1: MTI detection results for two abstract rules (R1 and R2).

Rule 1	Rule 2	Result
R1	R1	MTI
R1	R2	MTI
R2	R1	Concordance
R2	R2	Concordance

Listing 5.1: Rule describing MTI conflict detection algorithm.

```
1  % Are Rule1 and Rule2 MTI free using a Message at times T1 and T2?
2  % Rule1 and Rule2 conflict if the first one sets a message into a state such
3  % that the second one does not operate correctly.
4  mti(Rule1,Rule2, Message, T1,T2) :-
5          % Make sure that T1 < T2
6          ( before(T1,T2);!, fail ),
7          % Make sure the message holds when the first rule receives the message.
8          ( holdsAt(message(Message),T1); !, fail ),
9          % Perform the first rule and make sure it succeeds.
10         ( Rule1; !, fail ),
11         % Make sure that the message holds at the time the first rule completed.
12         % If the message does not hold there has been a conflict.
13         ( holdsAt(message(Message),T2); !, fail ),
14         % If the second rule succeeds then the rules concord.
15         ( Rule2; !, fail ).
```

Listing 5.2: Abstract feature rules for MTI testing.

```
1  r1(M,T1) :-
2          T2 is T1 + 1,
3          assert(happens(action,T1)),
4          assert(initiates(action,trigger(M),T1)),
5          assert(terminates(action,message(M),T1)),
6          assert(terminates(action,trigger(M),T2)).
7
8  r2(M,T1) :-
9          T2 is T1 + 1,
10         assert(happens(action,T1)),
11         assert(initiates(action,trigger(M),T1)),
12         assert(terminates(action,trigger(M),T2)).
```

5.5.2 Shared Trigger Detection

Shared Trigger Interaction (STI) occurs when multiple actions are performed in response to the same triggering event, and the behaviour of one or more of the actions is different from the behaviour arising from a single rule response to the trigger. Testing for STI can be accomplished by querying a feature rule, then resetting the query environment, querying a second rule, then querying the first rule again. If there are differences in the Σ sentences between the first and second instances of the first rule then the first and second rule conflict by STI.

The analytical framework can perform STI detection as shown in figure 5.3. It loads an STI detection rule along with arguments that consist of a pair of feature rules, but ignores the time points and the message fluent arguments. The fluent initially holds prior to being passed to the first rule and the second rule. Rules conflict if a check of the initiated actions from the first instance of the first rule does not match the second instance's initiated actions.

The initialisation phase is performed twice for STI detection. Both initialisations perform the same activities. They ensure that all EC assertions (such as those for Σ sentences) are retracted, insert time points, and set Δ_0 to include an initially set message fluent.

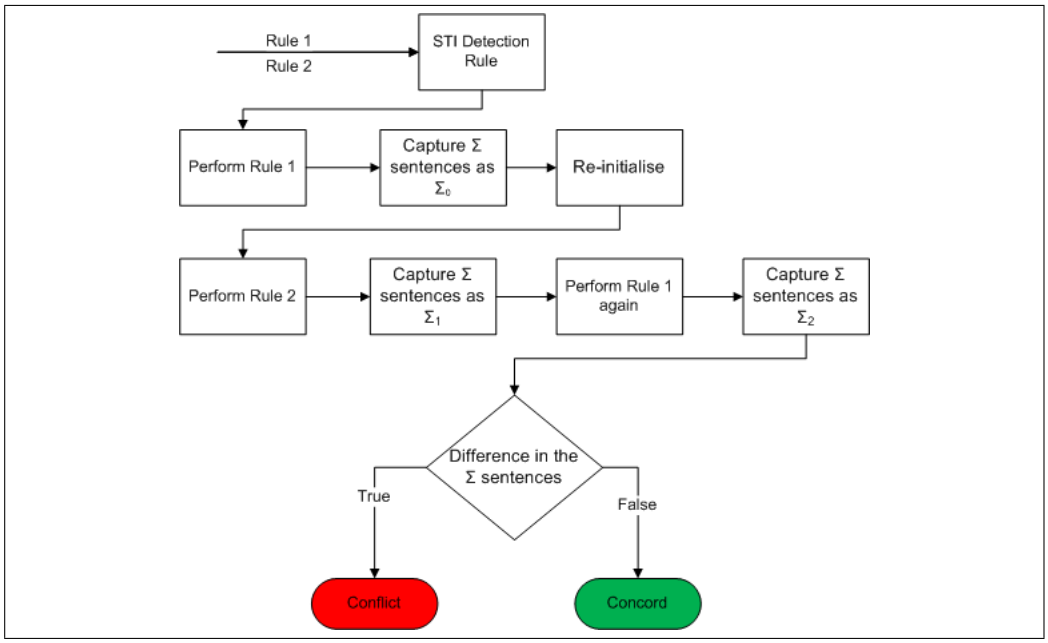


Figure 5.3: STI Detection algorithm diagram

The STI detection phase, described in listing 5.3, involves the following steps:

1. Query the first rule
   (a) Capture its Σ sentences as Σ_0
2. Re-initialise
3. Query the second rule
   (a) Capture its Σ sentences as Σ_1
4. Re-query the first rule
   (a) Capture its Σ sentences as Σ_2
5. Check for differences in the Σ sentences. The rules conflict if there are differences
   (a) Concordance is defined as Σ_0 - Σ_2 - Σ_1 = 0.

Listing 5.4 provides an abstract example of two potentially conflicting rules which receive the same time point for activation, but the message fluent they receive may or may not be active at that time point. The first rule (R1) initiates an action in response to a trigger only if the triggering message holds at the given time point. The second rule (R2) always terminates the triggering message regardless of the state of the trigger.

Table 5.2 shows the results of analysing these rules for STI. The table shows an interesting pattern of detection. Here, two instances of the same rule conflict, but instances of different rules do not.

Listing 5.3: Rule describing STI conflict detection algorithm.

```
% Are Rule1 and Rule2 STI free using a Message at time T1 (T2 is unused)?
% Rules A & B conflict by STI when actions are performed by them both in response to the same
% triggering event, and the list of actions is different from how it would be if
% only one feature had responded to the trigger.
sti(Rule1,Rule2, _, _, _) :-
        % Perform the first rule
```

```
7         Rule1,
8         domainDependentSentences(F1),
9         % Reset world
10        resetWorld,
11        % Perform the second rule
12        Rule2,
13        domainDependentSentences(F2),
14        % Perform the first rule
15        Rule1,
16        domainDependentSentences(F3),
17        subtract(F3,F2, Difference1),
18        subtract(F1,Difference1, Difference2),
19        length(Difference2,Len),
20
21        %Write result of check of rule conccordance
22        (
23                (
24                        Len=0,
25                        addConcord(sti, Rule1, Rule2),
26                        write(' Yes.\n')
27                );
28                (
29                        addFi(sti, Rule1, Rule2), write(' No.\n')
30                )
31        ).
32
33 domainDependentSentences(Res) :-
34        findall([B,C],initiates(_,B,C),Find1),
35        findall([B,C],terminates(_,B,C),Find2),
36        append(Find1, Find2, Find3),
37        subtract(Find3,[[null, null]], Res).
```

Listing 5.4: Abstract rules that conflict with themselves but not each other.

```
1 % Initiate an action in response to a trigger if it's triggering message holds
2 r1(Trigger,T) :-
3         T2 is T+1,
4         assert(happens(Trigger,T)),
5         (
6                 (
7                         holdsAt(message(Trigger), T),
8                         assert(initiates(Trigger,action,T))
9                 );
10                assert(happens(ignore,T2))
11        ).
12
13 % Terminate a triggering message
14 r2(Trigger,T) :-
15        assert(happens(Trigger,T)),
16        assert(terminates(Trigger,message(Trigger),T)).
```

5.5.3 Multiple Action Detection

The occurrence of Multiple Action Interaction (MAI) is caused by the attempted control of a single device by multiple rules at the same time. The given framework does not distinguish device types and has no knowledge of where the rules are triggered. Adding such information, though possible, will yield little benefit as it amounts to a declaration that the rules are triggered on the same device. If such a declaration is desirable, then one only needs to assume that the rules are triggered on the same device. Given that assumption, MAI can be detected using the STI detection rules.

Table 5.2: STI detection results for two abstract rules (R1 and R2).

Rule 1	Rule 2	Result
R1	R1	STI
R1	R2	Concordance
R2	R1	Concordance
R2	R2	STI

Table 5.3: MAI detection results for two abstract rules (R1 and R2).

Rule 1	Rule 2	Result
R1	R1	MAI
R1	R2	MAI
R2	R1	MAI
R2	R2	MAI

Listing 5.7 shows two abstract rules used for an example of MAI detection. These rules are presumed to run on the same device. Table 5.3 shows that these rules always conflict. In contrast, table 5.4 shows the result of using the rules from the STI example above (listing 5.4). These rules only conflict with themselves, but not with each other.

5.5.4 Sequential Action Detection

SAI can be detected by testing to determine if a feature rule performs an action that leads to the performance of actions by a second rule. This can be accomplished by triggering rules sequentially within the framework and checking for α sentences that describe actions that will be performed as a result of the firing of the two rules.

The analytical framework performs the procedures shown in figure 5.4 and described in listing 5.6. It uses the standard initialisation phase and then loads the SAI detection rule. This begins by ensuring the correct ordering of the time points. It then performs the first feature rule, stores its α sentences, and then re-initialises the world. Then it performs the second feature rule and stores its α sentences whereupon it re-runs the first rule and subtracts the second rule's actions from its actions. The remaining α sentences are compared with the actions from the initial run of the first rule. If they are the same, the second rule results in no additional actions, therefore the rules concord; otherwise they conflict by SAI .

Listing 5.5 shows two abstract rules were used to show an example of the algorithm. The first rule (R1) checks if the triggering message still holds and then depicts that some event happens that terminates the triggering message. The second rule (R2) also checks if the triggering message still holds, but then it only describes that some event occurs afterward, without explaining any actions that may come of the event.

The results of testing the rules in the analytical framework using the SAI detection rule are shown in table 5.5. Sequential actions were detected when either R1 or R2 followed a previous call of R1.

Listing 5.5: Abstract rules used to explain SAI.

```
1 r1(Trigger,T) :-
2         (
3                 holdsAt(message(Trigger), T),
```

Table 5.4: MAI detection results for the abstract rules (R3 and R4) from the STI example.

Rule 1	Rule 2	Result
R3	R3	MAI
R3	R4	Concordance
R4	R3	Concordance
R4	R4	MAI

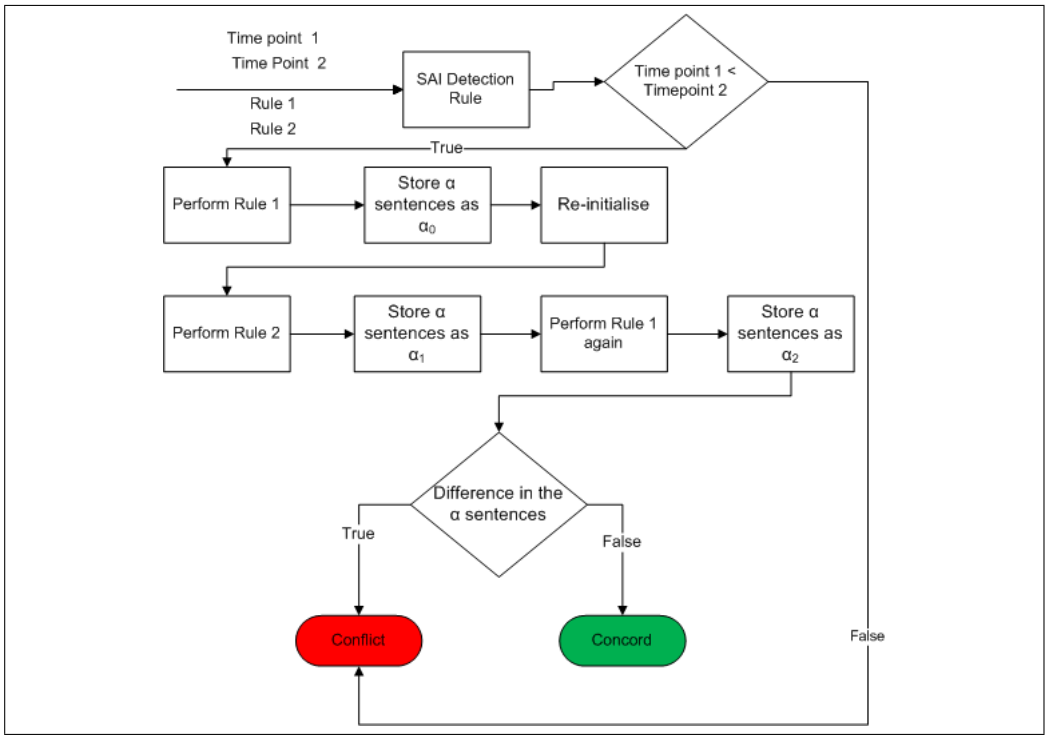


Figure 5.4: SAI Detection algorithm diagram

Table 5.5: SAI detection results for two abstract rules.

Rule 1	Rule 2	Result
R1	R1	SAI
R1	R2	Concordance
R2	R1	SAI
R2	R2	Concordance

```
4                   assert(happens(event1,T)),
5                   assert(terminates(event1,message(Trigger),T))
6           );
7           true.
8
9   r2(Trigger,T) :-
10          (
11                  holdsAt(message(Trigger), T),
12                  assert(happens(event2,T))
13          );
14          true.
```

Listing 5.6: Detection rule describing SAI conflict.

```
1   % Are Rule1 and Rule2 SAI free using a Message at times T1 and T2?
2   % Rules A & B conflict if a rule 2 occurs before rule 1 and alters
3   % what happens in rule 1.
4   sai(Rule1,Rule2, _, T1,T2) :-
5           % Make sure that T1 < T2
6           before(T1,T2),
7           % Determine what happens in rule 1 if rule 2 is not fired first
8           Rule1,
9           findall([B,C],happens(B,C),F1),
```

```
10          % Reset world
11          resetWorld,
12          % Perform the second rule
13          Rule2,
14          findall([B,C],happens(B,C),F2),
15          % Determine what happens in rule 1 after rule 2 is fired
16          Rule1,
17          findall([B,C],happens(B,C),F3),
18          subtract(F3,F2, Difference1),
19          (
20                      % A conflict occurs if what happens in rule 1 after rule 2 is
21                      % different than what happens in rule 1 before rule 2
22                      (
23                              F1\=Difference1,
24                              addFi(sai, Rule1, Rule2)
25                      );
26                      (
27                              addConcord(sai, Rule1, Rule2)
28                      )
29          ).
```

5.5.5 Loop Detection

LI occurs when one rule triggers another which in turn causes the first one to be re-triggered. LI, therefore, is a special case of SAI that can be defined as SAI leading to the triggering of the first rule's actions. This can be detected by performing SAI checks on the rules and examining the output for cases where two rules have SAI regardless of whether they are the first or second rule.

For SAI to be detected a rule must trigger a change in behaviour of another at a given point in time. The time points and the rules are called by the framework which then checks for differences in the way that a rule executes, whether it runs before or after another rule. If its operation is the same in both cases then SAI is not detected.

Table 5.6 presents three cases of LI being detected between two abstract rules. The rules are listed in listing 5.7. Here loops emerge when both rules are instances of the same type or instances of different types.

Listing 5.7: Abstract rules used to explain MAI and LI detection.

```
1  r1(Trigger,T) :-
2          (
3                  holdsAt(message(Trigger), T),
4                  assert(happens(event1,T)),
5                  assert(terminates(event1,message(Trigger),T))
6          );
7          true.
8
9  r2(Trigger,T) :-
10         (
11                 holdsAt(message(Trigger), T),
12                 assert(happens(event2,T)),
13                 assert(terminates(event2,message(Trigger),T))
14         );
15         true.
```

5.6 Resolution Strategies

Having shown that it is possible to detect conflicts, it is now important to consider what may be done about them. This section describes strategies that may be applied to deal with the conflicts. It also

Table 5.6: LI detection results for sample abstract rules.

Rule 1	Rule 2	Result	LI Case
R1	R1	SAI	LI(i)
R1	R2	SAI	LI(ii)
R2	R1	SAI	LI(ii)
R2	R2	SAI	LI(iii)

describes how the decisions of which to use can be reached and how these can be applied to cases of the five interaction types previously discussed.

The approach to resolution in this thesis focuses on the use of priorities rather than human intervention or heuristics for the reasons described in chapter 2 specify priorities, but to do so in a way that allows them to consider the consequences of their decisions.

5.6.1 Device Priorities

CLIPPER uses a rule-based device priority system to handle conflict resolution. The resolution rules determine which of the conflicting rules have precedence, and which are to be temporarily disabled in order to avoid the interaction.

Device and network-wide priorities are combined to determine rule precedence. Device-level and system-wide priorities are described as facts that are taken into account during resolution. These priorities are guiding principles for each device.

Device priorities have a considerable benefit in handling the heterogeneity of sensor networks. For example, the priorities of a gateway device that is drawing power from the home and has a high-speed Internet connection will be considerably different to those of a mobile phone with limited storage capacity and bandwidth concerns. The mobile phone may prioritise data processing, whereas the gateway may prioritise data integrity since it has ample power and data storage facilities. If the MAI interacting rules Data Storage Unconditional (DSU) and DSTP are present on each device then the appropriate resolution for the wearable device is to disable DSU, but the appropriate resolution for the gateway is to disable DSTP. For the resolution system to decide, the mobile phone administrator could set a device level priority for data integrity to a high value and set DSTP to a higher priority over DSU with respect to data integrity. The gateway administrator can do similar for the gateway priorities but giving a higher priority to DSU. The outcome would therefore depend on the device in question and its priorities.

In this approach the resolution system always disables rules with the lowest priority. Multiple rules with the same lowest priority (including no priority) are all disabled. This allows a system to gracefully handle conditions that can occur where priorities are not present or do not pertain to the interacting rules. In some cases, such as the LI example 2 in section 4.7.4.2, interactions may occur between rules across devices which results in multiple priorities of the same order. In that example the mobile phone has transference of data as its highest priority and the home gateway has security as its highest priority.

The priority system of CLIPPER works by consulting the priority files associated with each device, the core conflict analysis and resolution files and the conflict detection report file. Variables from each of the conflicts found in the conflict report are passed in to the resolution system's resolve rule. The variables include the type of conflict (such as MAI) and each device and device rule that conflicted. Each rule is checked to see if a priority has been applied to them. If both have priorities they are checked to see which has precedence. The rule with the lowest precedence is disabled as are rules that have no priorities applied to them. If two rules have equal precedence then they are both disabled.

5.7 CLIPPER Simulation Architecture

A mediator-based approach is used in this research to simulate how network devices can behave when influenced by conflict detection. This approach uses a mediator to analyse device rules and load appropriate actions into devices.

CLIPPER, uses object-oriented programming to automate conflict analysis. Automated CLIPPER-based applications run on physical devices using key concepts defined for devices, mediators, their rule

engines, and actions performed by the devices. The devices are controlled though logic rules and facts. Actions, rules, and facts are loaded and unloaded into devices and knowledge bases at run-time. This allows networks of devices running CLIPPER applications to change and react to new information. The Java source code for CLIPPER is provided at http://code.google.com/p/clipper-cd/.

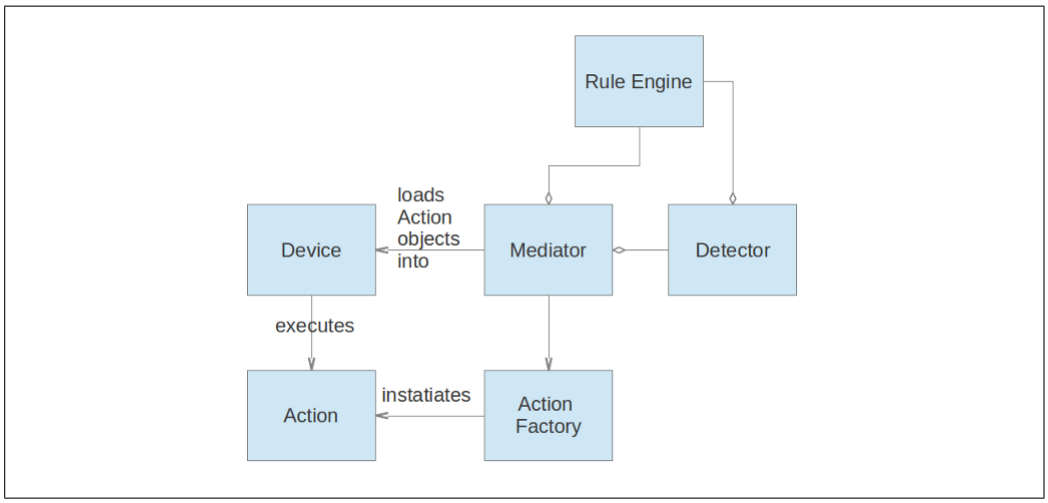


Figure 5.5: Overview of CLIPPER class architecture showing relationships among main classes.

Figure 5.5 shows an overview of the main architectural elements of simulation applications which include:

- *Device* classes represent physical devices such as mobile phones and gateways. Multiple *Devices* can be added to an application. Devices are provided with a path to files containing rules specifying their behaviour. *Devices* are loaded with *Actions* by *Mediators* and execute them in sequence.
- A *Mediator* loads device rules into a *Rule Engine,* passes the rules to a conflict *Detector,* and loads *Action*s into *Devices.*
- *Rule Engine*s query declarative logic facts and rules given goals.
- *Detectors* perform conflict detection by loading the device rules files and CLIPPER core analysis logic files into a *Rule Engine* and query it for a conflict goal.
- *Action* classes implement EC actions (or events) from device rules into *Action* objects. For instance, a rule for Automatic Data Transfer (ADT) might have a *happens* sentence which includes an action for *attempt_connection.* An *Attempt_connection* action class is required to handle the connection attempt over a particular network protocol. Actions are executed by their containing *Devices.*
- *Action Factories* reify *Action* classes given a complementary EC rule name supplied by a *Mediator.*

In CLIPPER, device objects observe and react to stimuli input into a physical device. Device objects are programmed to watch for input into the physical device. They register with mediators and report to them when new data enter the system.

New devices can be added to the application by registering with a mediator. Adding new devices to the system requires three steps:

1. Identify existing action classes, and/or program new ones and register them with the action factory.
2. Register the device with a mediator.
3. Add rules regarding the device and actions it should perform to the rule set.

Mediators ensure that rule conflict is checked on device registration and removal. Mediators do so by passing the device rule file paths to their detectors. Detectors detect conflict by preparing a temporary

directory with the CLIPPER core analysis logic files, a logic file that it generates containing the files to consult, and a generated rule listing the device rules to analyse. Detectors load the generated file into a rule engine which interface with SWI-Prolog using the JPL library. The rule engine uses Prolog to consult the analysis and device rules files and then perform conflict detection as described in 5.4. Conflicts are written as facts to a detection report file describing the type of conflict (such as MTI), and the pair of conflicting rules.

Mediators control the system-level logic as to what system-wide actions should take place in response to input. The mediators also control inter-device communication. When devices are updated (such as when new readings are sensed) they notify their mediators of occurrences of events. Mediators then make calls to rule engines to determine actions to be carried out. These actions are loaded into devices for execution. Conflict resolution should be performed by a mediator after it has checked for conflict. It should only load the actions of non-conflicting rules, or rules that have higher priority over rules of lower priority that conflict. To do so, it can use the device priority logic system described above with the conflict detection report file and device priorities files to ascertain the appropriate actions to load into devices, ignoring low priority conflicting ones.

Rule engine objects query knowledge base facts and rules. Satisfied queries return the names and parameters of actions to perform, along with the identities of devices to perform the actions. These queries return from the rule engine to the calling mediators. The mediators use action factory objects to generate action objects which are loaded into the appropriate devices. Enqueued actions are executed by devices. Rules, facts and actions can be loaded and unloaded into the rule set and action factory at run-time. This allows actions to load new facts and rules into the rule set.

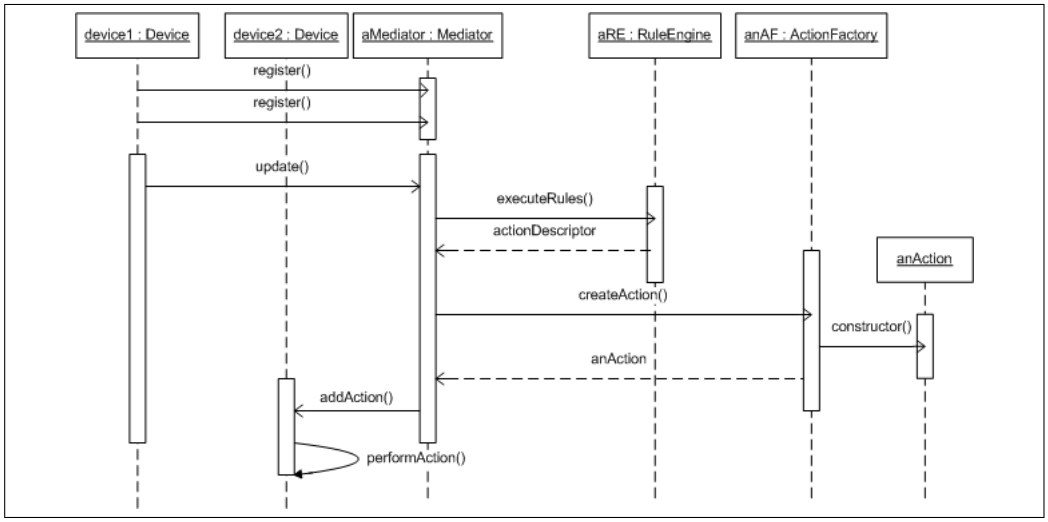


Figure 5.6: Standard CLIPPER usage sequence.

A standard usage sequence is depicted in figure 5.6. In the figure two devices register with a mediator. Some time later one of the devices sends an update message to the mediator. The mediator's rule engine is called to determine the actions that should be performed. A given action is then reified by the action factory. The action is passed back to the mediator which loads the action into the device that should perform it. The action is then performed by the device, ending the sequence. In this diagram, the number of devices shown, along with the decision to have the first one generate the update message and the second one perform the action are arbitrary.

An example using such a sequence involves a mobile phone and a wearable GPS unit. The phone can act both as a mediator and as a device. When the wearable GPS unit sends new data to the mobile phone, it can call rules to determine actions to take based on the given location. An action such as one to ask a question about the subject's current activity can then be returned by the rule engine to the mediator on the phone which may then be loaded into a questionnaire application on the phone.

5.8 Summary

This chapter contributes a novel approach to rule conflict detection and resolution. Such conflicts are shown to emerge when various devices are programmed independently but rely on each other. Design goals, assumptions and decisions regarding the solution are discussed along with the presentation of a novel solution that can be used to detect and resolve five forms of conflict.

Key concepts are presented here including the device rule and conflict rule notations based on the EC, logic-based detection algorithms for MTI, STI, MAI, SAI and LI, and resolution strategies derived from device priorities. These concepts are used in the next chapter, which describes an assessment and analysis of the approach.

Chapter 6

Approach Assessment

6.1 Introduction

Previous chapters of this thesis describe how the use of rules can lead to conflicts within devices and across device networks. Conflicts that emerge from device rules to perform Ambulatory Assessment (AA) are presented in chapter 4. Chapter 5 describes conflict detection and resolution strategies for sensor network systems. This Approach Assessment chapter presents an evaluation of the methods from chapter 5 against the examples discussed in chapter 4.

This chapter contributes evaluations that show that conflicts can indeed arise in device control rules, and that the analysis engine can reliably detect and resolve the five forms of rule conflict previously discussed. Results from two methods of evaluation are presented, including a reference testbed system used to show the emergence of conflicts, and the results of an assessment of the analysis approach.

This chapter begins with a description of the results of running CoLlaborative Information Processing Protocol and Extended Runtime (CLIPPER) within a test harness to show that conflicts can emerge. That is followed by the presentation of the results from using the analysis engine to determine resolution against the various examples described in chapter 4. The chapter concludes with a discussion of the results.

6.2 Testbed Configuration and Testing

The following discusses a CLIPPER-based testbed example that shows the emergence of conflicts. The example illustrates conflict that can arise between rules. This is shown using an environmental monitor (Env_Mon) reporting data to a gateway. The environmental monitor uses the Data Management (DaM):Automatic Data Transfer (ADT) rule to report all data to the gateway. For different tests the gateway subscribes to either, or both, the DaM:Data Storage Unconditional (DSU) and DaM:Data Storage Through Processing (DSTP) rules (the use of both causes an MAI as described in chapter 4). Figure 6.1 shows a class diagram depicting the classes used in the example. A home gateway is the concrete device class which registers with a mediator containing a rule engine. The rule engine can call out to a rule manager capable of testing for Multiple Action Interaction (MAI). The available action classes that are shown are derived from the Δ predicates of the DaM:ADT, DaM:DSU, and DaM:DSTP rules.

Three tests show the emergence of conflict. In the first test the gateway subscribes only to DaM:DSU, in the second it subscribes only to DaM:DSTP[1], and in the third it subscribes to both, but the mediator does not use conflict detection or correction. In all cases, the following run time behaviour takes place:

1. The mediator is instantiated with a new rule engine.

2. The actions shown in figure 6.1 are added to the action factory.

[1]For these tests the storage and processing actions are simplified to eliminate possible interference resulting from more complex schemes, and to clearly show the conflict. The storage actions wrote the type and value to the same file, and the processing actions annotated the raw data with a "processed" message. The conflict can be avoided if multiple files are used for storage (different ones for each rule), butt that is not a long term solution, as it is not possible to enforce such a division, and it could have deleterious effects for any down-stream actions that expect data to be found in a particular location.

Table 6.1: Sample Stored Data from First Test

Reading Type	Reading Value
light	3
light	13
sound	19
light	7

3. The gateway and environmental monitor devices are instantiated, their rules are loaded into them, and registered with the mediator.

4. A virtual environmental monitor device is programmed to capture 20 light meter readings per second and one sound reading per second.

5. Each test runs for 60 seconds.

The rules files used in the tests along with the resulting data files can be downloaded from `http://www.cs.stir.ac.uk/~jmb/clipper/data/mai/00000000/`.

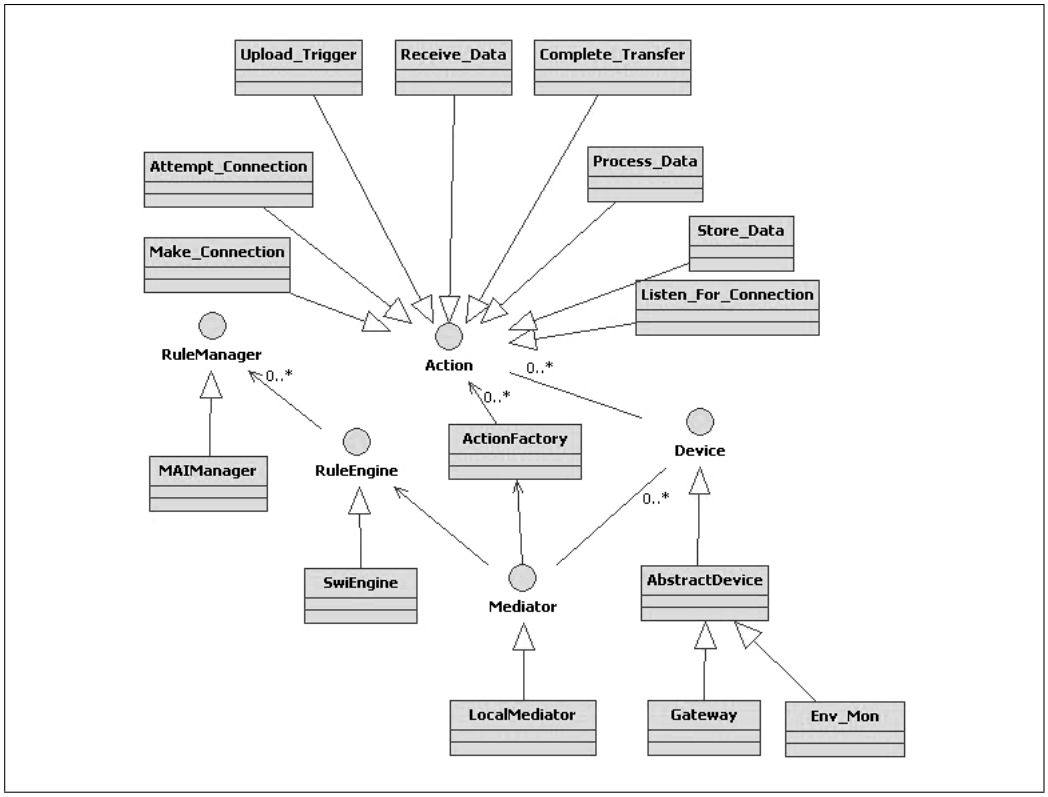


Figure 6.1: CLIPPER class overview diagram for MAI interaction online analysis example.

6.2.0.1 A well behaved system

A normal pattern of behaviour is observed as a result of the first two tests. Manually comparing the log files with the data storage files shows that the environmental monitor generates more than 1000 readings per test. The data is stored correctly by the gateway as per the given rules: in the first test the raw data is stored, and in the second test the processed data is stored. Samples of the data for test 1 and 2 are shown in tables 6.1 and 6.2 respectively.

Table 6.2: Sample Stored Data from Second Test

Reading Type	Reading Value
light	3 – PROCESSED
light	14 – PROCESSED
light	2 – PROCESSED
sound	18 – PROCESSED

Table 6.3: Sample Stored Data from Third Test

Reading Type	Reading Value
light	8
light	3 – PROCESSED
sound:	14
light	10 – PROCESSED

6.2.0.2 Conflict

The resulting data from the third test shows that an undesirable rule conflict takes place (as expected). More than 1000 readings are generated and stored. Half of them, however, are stored with raw values and the other half are stored with processed values. This situation is clearly undesirable since the non-homogeneity of the data will invalidate further analysis. A sample of the data for this test is shown in table 6.3.

6.3 Detection & Resolution Examples

A number of examples are described in chapter 4. The following subsections describe the rules for each of these cases and the results from testing each one using the conflict analysis and resolution engine. Each subsection contains a results table. Each table lists the rules used in the test along with the conflict detection results and resolution proposals. A summary of the results is shown in table 6.15.

The resolution proposals are based on a set of priorities configured for the resolution system, shown in listing B.1. These are set by the author based on likely settings for all of the rules used in the examples. Each of the aforementioned tables contains a resolution column. Entries are only included in this column for conflict results (they are left blank for concordance results), and entries mean that the given rules should be either enabled (E) or disabled (D).

The choice of whether to enable or disable a rule depends on the type of conflict that is detected. If the type is Missed Trigger Interaction (MTI), then the rule should be enabled, otherwise the rule should be disabled. This follows from the notion that it would not make sense to disable a rule that is already being blocked by another rule. A rule will, however, only be enabled if it has a higher priority than the blocking rule. This allows, for instance, a high priority security rule to quite rightly cause a low priority user notification message to miss its trigger.

Possible resolution entry values include a named rule, “either”, or “both”. The term “either” means that either of the given rules may be disabled. Following the principle, “When all things are equal, choose deterministically rather than randomly” from Reiff-Marganiec [2002], always the first, or always the second rule should be disabled. The term “both” means that both of the rules should be disabled.

6.3.1 Shared Trigger Example 1

The rules used for Shared Trigger Interaction (STI) example 1 can be found in listing B.2. These rules are Context Detection Service (CDS):Context Triggering System (CTS) and State Detection Service (SDS):State Triggering System (STS).

The CDS:CTS rule has Δ sentences that describe its activities. It also has Σ sentences that will be activated if the message fluent holds at the subsequent time point to rule activation. This is a destructive rule because the message fluent is terminated at the activation time point by this rule.

Table 6.4: STI Detection results for STI example 1.

Rule 1	Rule 2	Result	Resolution (D)
CDS:CTS	CDS:CTS	STI	Both
CDS:CTS	SDS:STS A	STI	SDS:STS(A)
CDS:CTS	SDS:STS B	Concordance	
SDS:STS A	CDS:CTS	STI	SDS:STS(A)
SDS:STS A	SDS:STS A	STI	Both
SDS:STS A	SDS:STS B	Concordance	
SDS:STS B	CDS:CTS	Concordance	
SDS:STS B	SDS:STS A	Concordance	
SDS:STS B	SDS:STS B	Concordance	

Table 6.5: SAI Detection results for SAI example 1.

Rule 1	Rule 2	Result	Resolution (D)
CDS:RL	CDS:RL	Concordance	
CDS:RL	SI:Prompt	SAI	SI:Prompt
SI:Prompt	CDS:RL	Concordance	
SI:Prompt	SI:Prompt	SAI	SI:Prompt

Two forms of the rule SDS:STS are presented. Normally, only one form of a rule is used, however here two are given to highlight possible differences in the way Σ sentences may be written. The first form of the rule (form A) depicts an active response to changes in data. In the interests of space, the rule depicted here calls the CDS:CTS rule since the two rules are similar in nature. The second form of the rule (form B) uses the same Δ sentences as the other two rules, but it is passive and does not contain any Σ sentences.

Table 6.4 shows the results of analysing these rules for STI. The table shows that form B of SDS:STS is in concordance with the other rules, but the other rules conflict with themselves and each other. Rows two and four are of particular interest as they show that the rules conflict with each other.

For rows two and four, the resolution system determines that SDS:STS should be disabled. This results from the relative priorities of CDS:CTS and SDS:STS (the former having a higher data collection priority). The choice of priority between these two rules is arbitrary, and would require a domain expert to make this choice in a concrete setting. Further testing with the priorities shows that inverting the data collection priorities leads to the recommended disabling of CDS:CTS. Setting both data collection priorities to the same value results in the recommendation of disabling both rules.

6.3.2 Sequential Action Example 1

Sequential Action Interaction (SAI) example 1 involves the rules presented in listing B.3 which lists the rules CDS:Report Location (RL) and Subject Interaction (SI):Prompt (Prompt). Table 6.5 shows two occurrences of SAI.

The first of these (CDS:RL followed by SI:Prompt) is of particular interest since it shows that the expected conflict from the example occurs. The detection of this does not mean that there is necessarily a problem, it means that caution should be taken when using these rules together. This is because it is possible that CDS:RL can become disabled owing to the resolution of some other conflict. If that is the case, then SI:Prompt would also be disabled in this instance, which may interfere with data collection protocols. Should one wish to resolve this conflict for certain, then the resolution of this conflict should to be to disable SI:Prompt.

6.3.3 Sequential Action Example 2

The rules used in SAI example 2 are shown in listing B.4. The rules DaM:ADT and DaM:Redirect Data Stream (RDS) are described. The results of testing the rules in the analytical framework using the SAI detection rule are shown in table 6.7.

Table 6.6: SAI Detection results for SAI example 3.

Rule 1	Rule 2	Result	Resolution (D)
DeM:DRF	DeM:DRF	SAI	Either
DeM:DRF	SDS:STS A	SAI	SDS:STS A
DeM:DRF	SDS:STS B	SAI	SDS:STS B
SDS:STS A	DeM:DRF	SAI	SDS:STS A
SDS:STS A	SDS:STS A	SAI	Both
SDS:STS A	SDS:STS B	SAI	SDS:STS A
SDS:STS B	DeM:DRF	SAI	SDS:STS B
SDS:STS B	SDS:STS A	SAI	SDS:STS A
SDS:STS B	SDS:STS B	SAI	Both

Table 6.7: SAI Detection results for SAI example 2 and LI example 1.

Rule 1	Rule 2	Result	LI Case	Resolution (D)
ADT	ADT	SAI	LI(i)	Either
ADT	RDS	SAI	LI(ii)	RDS
RDS	ADT	SAI	LI(ii)	ADT
RDS	RDS	Concordance	No	

Sequential actions are detected between two instances of DaM:ADT, as well as between DaM:ADT and DaM:RDS. DaM:RDS does not cause SAI when two instances of it are used. The system is validated against this example since it is expected that a SAI between DaM:ADT followed by DaM:RDS would be detected. The resolution of this conflict is to disable DaM:RDS.

6.3.4 Sequential Action Example 3

SAI example 3 contains rules for Device Management (DeM):Data Recording Frequency (DRF) and SDS:STS listed in listing B.5. The same two forms of the rule SDS:STS presented in STI example 1 are presented here as well.

The results of testing the rules in the analytical framework using the SAI detection rule are shown in table 6.6. This shows that SAI is detected in all rule pairings. Rows 4 and 7 confirm that SAI is detected. In these cases the resolution system determines that SDS:STS should be disabled. This occurs because there is a preference for battery life over data collection.

6.3.5 Looping Example 1

Looping Interaction (LI) is detected when examining DaM:ADT and DaM:RDS as shown in table 6.7. The same rules are used as in Sequential Action example 2 (listing B.4).

The table presents two cases of SAI in which LI is also found. The first row depicts DaM:ADT leading to an SAI with another instance of itself. This may result in an infinite call loop.

The second loop that is detected occurs because SAI is detected between DaM:ADT and DaM:RDS in both orderings of rule calls. The detection of this second case of LI validates the system against the example. The resolution for this loop is to disable either the ADT rule or the RDS rule depending on which rule is detected in the loop first.

6.3.6 Looping Example 2

Table 6.8 shows the analysis of the second LI example. The rules for DaM:Inbound Data Screening (IDS) and DaM:Retry Data Transfer On Unavailable (RDTOU) are listed in listing B.6.

The table shows that a loop is detected when instances of DaM:RDTOU are used as both the first and second rule. This loop results from DaM:IDS blocking the connection between devices, even though DaM:IDS does not lead to SAI with DaM:RDTOU. The resolution system determines that the solution to the loop is to block both instances of DaM:RDTOU. This exposes a subtle limitation in the resolution

Table 6.8: SAI Detection results for LI example 2.

Rule 1	Rule 2	Result	LI Case	Resolution (D)
IDS	IDS	Concordance	No	
IDS	RDTOU	Concordance	No	
RDTOU	IDS	SAI	No	Both
RDTOU	RDTOU	SAI	Yes	Both

Table 6.9: MAI Detection results for MAI example 1.

Rule 1	Rule 2	Result (D)	Resolution
DaM:DSU	DaM:DSU	MAI	Both
DaM:DSU	DaM:DSTP	MAI	DaM:DSTP
DaM:DSTP	DaM:DSU	MAI	DaM:DSTP
DaM:DSTP	DaM:DSTP	MAI	Either

system since it would only be necessary to block one instance DaM:RDTOU to prevent a loop. Blocking the first instance would result in an inherent block of the second (the same as blocking both), but blocking just the second instance would terminate the loop without the need to block the first one.

6.3.7 Multiple Action Example 1

The rules for the first MAI example are given in listing B.7. It presents rules for DaM:DSU and DaM:DSTP which are presumed to be running on the same device.

Table 6.9 shows the results of the analysis. All four combinations of rules result in MAI, including the middle two rows, which validates the system against the example.

The resolution system determines that DaM:DSTP should be disabled in both of the middle rows. This results from a preference for data integrity over device battery life. Regarding battery life, a case can be made that DaM:DSTP is superior to DaM:DSU because, saving data to flash memory and transmitting it wirelessly are large power drains, as shown by Blum & Magill [2010]. Storing (or transmitting) the results of a discrete amount of processed data should therefore increase battery life.

6.3.8 Multiple Action Example 2

This example involves checking if MAI is detected between DaM:ADT and DaM:Outbound Data Screening (ODS). The rules are listed in listing B.8. Two forms of DaM:ODS are presented in the listing to see the detection behaviour when the device is on the screening list versus when it is not.

The results of this example analysis are shown in table 6.10. ODS (A) is the unscreened rule and ODS (B) is the screened one. The system is validated against this example as shown in rows two and three because DaM:ADT conflicts with the screened instance but not the unscreened one (MAI is only detected in row three).

The resolution system determines that DaM:ADT should be disabled based on the importance of security to the system.

6.3.9 Multiple Action Example 3

Listing B.9 describes conflicting rules from MAI example 3. The conflict between the rules DeM:Time Synchronisation (TS) and Do Not Disturb (DND):Do Not Notify Unconditional (DNNU) shown in row two of table 6.11 validates the system against the example. The resolution system determines that the DNNU rule should be disabled in this case. This results from the value of data integrity over user interaction.

6.3.10 Missed Trigger Example 1

Table 6.12 shows the results from running MTI analysis against the rules in listing B.10. The rule DND:DNNU causes MTI with SI:Prompt as expected in the example. A consequence of such a conflict

Table 6.10: MAI Detection results for MAI example 2.

Rule 1	Rule 2	Result	Resolution (D)
DaM:ADT	DaM:ADT	MAI	Both
DaM:ADT	DaM:ODS (A)	Concordance	
DaM:ADT	DaM:ODS (B)	MAI	DaM:ADT
DaM:ODS(A)	DaM:ADT	Concordance	
DaM:ODS (A)	DaM:ODS (A)	MAI	Both
DaM:ODS (A)	DaM:ODS (B)	MAI	DaM:ODS (A)
DaM:ODS(B)	DaM:ADT	Concordance	
DaM:ODS (B)	DaM:ODS (A)	Concordance	
DaM:ODS (B)	DaM:ODS (B)	MAI	Both

Table 6.11: MAI Detection results for MAI example 3.

Rule 1	Rule 2	Result	Resolution (D)
DeM:TS	DeM:TS	MAI	Both
DeM:TS	DNNU	MAI	DNNU
DNNU	DeM:TS	Concordance	
DNNU	DNNU	MAI	Both

would be that rules that relied upon for subject prompting responses would also be missed.

The resolution system recommends enabling DNNU in this case which means that the MTI would not be avoided. This is because it has a higher priority than SI:Prompt. This priority setting generally makes sense as users tend to want their notification preferences respected. For instance, a user might be in a meeting and therefore not want notifications. Quite rightly, any prompt should miss its triggers in this setting. It is possible, however, for an unusual data collection protocol to assign SI:Prompt higher priority than DNNU. In such situations, the resolution system recommends enabling SI:Prompt and thereby avoids MTI.

6.3.11 Missed Trigger Example 2

This example uses the rules in listing B.11, which include the ones from shared trigger example 1 as well as SI:Prompt. The last row of table 6.13 shows that SI:Prompt causes MTI with a second call to it, as is predicted in the example. The resolution system determines that both the SI:Prompt rules should be enabled, ensuring that one prompt does not inhibit the triggering of the other.

6.3.12 Missed Trigger Example 3

This example validates the Brief Visit Home example at the beginning of chapter 4. The example expected MTI between CDS:CTS and DeM:Activate Immediate (AI) if the latter followed the former with a delay in it. The rules are shown in listing B.12.

Row two of table 6.14 shows the detection of this conflict, thereby validating the system against the example. The resolution system determines that DeM:AI should be enabled thereby avoiding the MTI.

Table 6.12: Detection results for MTI example 1.

Rule 1	Rule 2	Result	Resolution (E)
DNNU	DNNU	MTI	Both
DNNU	SI:Prompt	MTI	DNNU
SI:Prompt	DNNU	MTI	DNNU
SI:Prompt	SI:Prompt	MTI	Both

Table 6.13: MTI Detection results for MTI example 2.

Rule 1	Rule 2	Result	Resolution (E)
CDS:CTS	CDS:CTS	MTI	Both
CDS:CTS	SDS:STS A	MTI	CDS:CTS
CDS:CTS	SDS:STS B	MTI	CDS:CTS
CDS:CTS	SI:Prompt	MTI	CDS:CTS
SDS:STS A	CDS:CTS	MTI	CDS:CTS
SDS:STS A	SDS:STS A	MTI	Both
SDS:STS A	SDS:STS B	MTI	SDS:STS B
SDS:STS A	SI:Prompt	MTI	SDS:STS A
SDS:STS B	CDS:CTS	Concordance	
SDS:STS B	SDS:STS A	Concordance	
SDS:STS B	SDS:STS B	Concordance	
SDS:STS B	SI:Prompt	Concordance	
SI:Prompt	CDS:CTS	MTI	CDS:CTS
SI:Prompt	SDS:STS A	MTI	SDS:STS A
SI:Prompt	SDS:STS B	MTI	SDS:STS B
SI:Prompt	SI:Prompt	MTI	Both

Table 6.14: Detection results for MTI example 3.

Rule 1	Rule 2	Result	Resolution (E)
CDS:CTS	CDS:CTS	MTI	Both
CDS:CTS	DeM:AI	MTI	DeM:AI
DeM:AI	CDS:CTS	Concordance	
DeM:AI	DeM:AI	Concordance	

6.4 Analysis and Discussion

The previous sections contain two key points. Firstly, the results from the testbed indicate that conflicts occur in the networks under consideration. Conflicts must be detected to ensure that the recorded data remains reliable. Secondly, as can be seen in table 6.15, the example tests showed that the conflict detection and resolution analysis reported as expected, lending weight to the justification of the approach.

The example tables also show a number of results for cases that are not specifically looked for. These have been included to give an overall impression of how well the various rules work together. Certain rules, such as those in table 6.6, showed a high degree of conflict whereas those in table 6.4 has a higher degree of concordance.

The rules were analysed to determine which were the most and least conflict prone. In total, all of the 17 rules presented above were assessed against each other as both the first and second rules in the analysis approach. Out of a total of 867 tests (all of the rules were tested for STI, SAI and MTI), 410 conflicts were detected. Many of the rule combinations may only rarely occur (if at all), but it is useful to look at such a wide spread of cases in order to identify patterns that can help author rules to minimise conflicts.

6.4.1 Shared Trigger

A high number of shared trigger conflicts were discovered amongst the rules, so an investigation is used to understand how best to write the rules in order to maximise concordance. Figure 6.2 shows that the rule with the largest number of STI conflicts is the screened form of DaM:ODS and the fewest is shared by CDS:RL and DeM:AI. The range of STI conflicts is from 0 to 23, with a median of 22 and inter-quartile range of 18.

The rules are categorised in table 6.16. Five categories are identified:

1. Those showing high numbers of conflicts in both the first and second rules

Table 6.15: Example Analysis Results

Study Title	Involved Rule	Detected?	Resolved?
STI 1	CDS:CTS , SDS:STS A	Yes	Yes
SAI 1	CDS:RL, SI:Prompt	Yes	Yes, but may not be wise to do so.
SAI 2	DaM:ADT, DaM:RDS	Yes	Yes
SAI 3	SDS:STS (A and B), DeM:DRF	Yes	Yes
LI 1	DaM:ADT, DaM:RDS	Yes	Yes
LI 2	DaM:RDTOU, DaM:RDTOU	Yes	Yes, but may be too heavy-handed.
MAI 1	DaM:DSU, DaM:DSTP	Yes	Yes
MAI 2	DaM:ADT, DaM:ODS	Yes	Yes
MAI 3	DND:DNNU, DeM:TS	Yes	Yes
MTI 1	DND:DNNU, SI:Prompt	Yes	Yes
MTI 2	SI:Prompt, SI:Prompt	Yes	Yes
MTI 3	CDS:CTS, DeM:AI	Yes	Yes

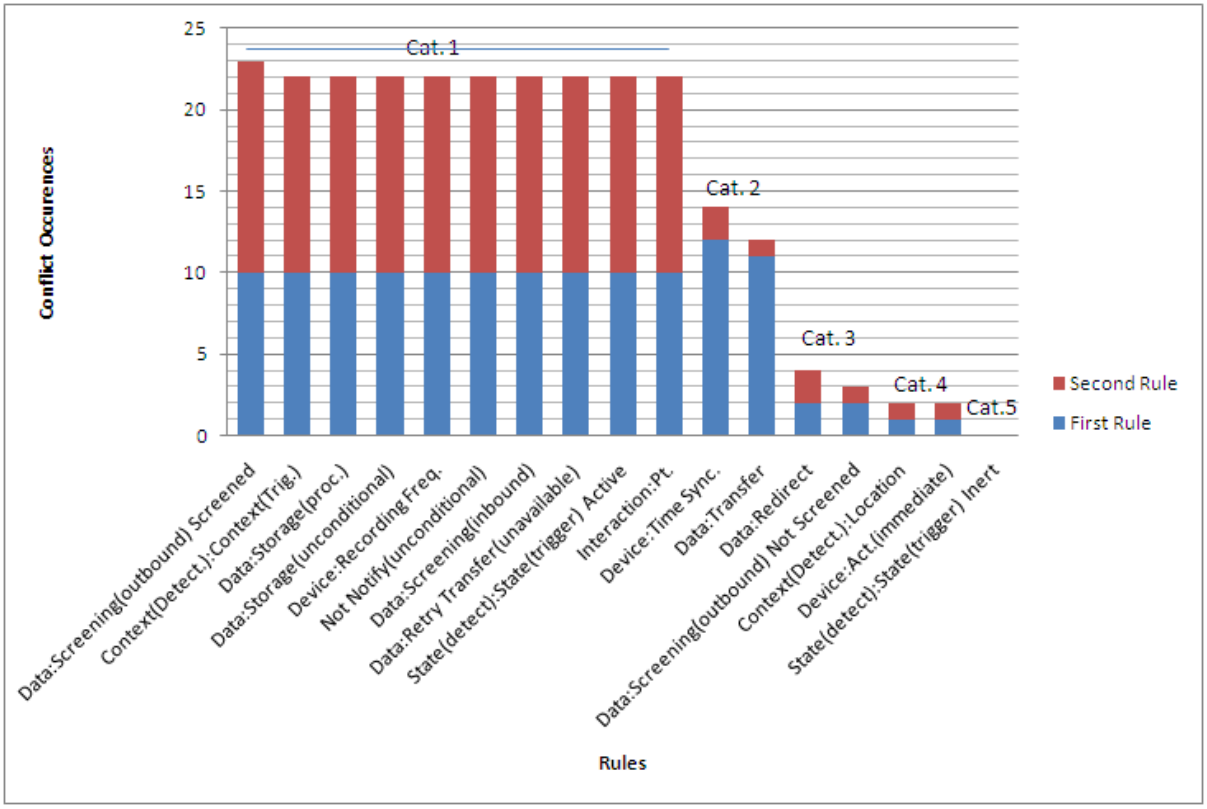


Figure 6.2: The number of STI conflicts for 17 rules. The median is 22 and IqR is 18.

2. Those that have a high number of conflicts in the first rule only
3. Those that conflict only with themselves and one other,
4. Those that conflict only with themselves
5. Those that do not conflict

Ten rules fit category 1. Those rules conflict only with each other when they are the first rules but not with the rules in the other categories. When the rules in category 1 are second rules, however, they conflict with themselves as well as those in category 2. When category 2 rules are first rules, they conflict with all of the rules in category 1 as well as with themselves (but not each other). One of the category 2 rules (DeM:TS) also conflicts with a category 3 rule (DaM:RDS). Some of the rules such as DaM:ODS Screened are inherently conflict prone. This rule is designed to prevent the operation of other rules for security reasons, so it is understandable that it alters the patterns of behaviour for so many other rules when they share a trigger. Further analysis was conducted to understand the high number of conflicts involving other rules.

Analysis of rule categories 1 and 2 reveal that the high number of conflicts is caused by clauses that alters the states of the triggering message fluents. The difference between the rules in categories 1 and 2 is that those rules in the former category always alters the states of the triggering message fluents, whereas those in the latter category have branching conditions that alter the states only under certain circumstances.

The reason that the category 5 rule does not conflict with the others is that it is an inert rule that does not initiate the triggering of any other activity. It entirely lacked Σ sentences and therefore the rule may never take part in a situation involving shared triggers. Since the detection algorithm does not detect shared triggers, the behaviour is correct. A naïve response to this might be to develop systems in which the rules never initiate or terminate fluents in order to fully avoid shared triggers. A Data Collection System (DCS) would be an example of such an approach. When the goal, however, is to have automated responses to sensed data, then shared triggers may emerge.

Table 6.16: STI conflict review

Cat.	Rule	# Conf. First Rule	# Conf. Second Rule	Total Conf.
1	DaM:ODS Screened	10	13	23
1	CDS:CTS	10	12	22
1	DaM:DSTP	10	12	22
1	DaM:DSU	10	12	22
1	DeM:DRF	10	12	22
1	DND:DNNU	10	12	22
1	DaM:IDS	10	12	22
1	DaM:RDTOU	10	12	22
1	SDS:STS Active	10	12	22
1	SI:Prompt	10	12	22
2	DeM:TS	12	2	14
2	DaM:ADT	11	1	12
3	DaM:RDS	2	2	4
3	DaM:ODS Not Screened	2	1	3
4	CDS:RL	1	1	2
4	DeM:AI	1	1	2
5	SDS:STS Inert	0	0	0

The ideal then is to minimise the conflicts such as can be seen in categories 3 and 4. Those cases only conflict with other instances of themselves. For example, if two rules to activate a device do share a trigger, only one of them would be able to activate it because of the binary nature of device activation. These rules lend themselves therefore to the notion that rules work best when they avoid altering their triggering message fluents.

6.4.2 Sequential Action

Figure 6.3 shows that DaM:ADT has the largest number of SAI conflicts and the unscreened form of DaM:ODS has the fewest. The median number of SAI conflicts is 15 with a range between 0 and 24, and an inter-quartile range value of 13. From these, 8 categories of conflicts are identified for analysis, as shown in table 6.17:

1. Those that have a medium or high number of conflicts as the first rules and as the second rules
2. Those that have a high number of conflicts as the first rules but medium as the second rules
3. Those that have a medium number of conflicts as both the first and second rules
4. Those that have high number of conflicts as the first rules but none as the second rules
5. Those that have no conflicts as the first rules, but a medium number as the seconds
6. Those that conflict only with one other rule
7. Those that do not conflict at all

The low impact rules of categories 6 and 7 provide insights into minimising conflicts. DaM:ODS Not Screened is the only rule does not conflict at all (category 7), because the firing of this rule does not change the actions that occur for the other rules. Unscreened rules fire as normal. Furthermore, the other rules do not impact on what happened to it.

Another interesting rule is DaM:RDS. It only conflicts with the rule DaM:ADT (both as first and second rule) since the two rules shared a common fluent. The modification of the state of the fluent allows or disallows actions to transpire. From the examination of these rules it is apparent that rules in general will not conflict if they limit their shared fluents, thereby limiting the impact on what happens to each other.

Table 6.17: SAI conflict review.

Cat.	Rule	# Conf. First Rule	# Conf. Second Rule	Total Conf.
1	DaM:ADT	12	12	24
1	CDS:CTS	7	12	19
1	DaM:DSTP	7	12	19
1	DaM:DSU	7	12	19
1	DeM:DRF	7	12	19
1	SDS:STS Active	7	12	19
2	SI:Prompt	12	6	18
2	DaM:RDTOU	10	6	16
3	SDS:STS Inert	7	8	15
4	DeM:AI	12	0	12
4	CDS:RL	11	0	11
4	DeM:TS	11	0	0
5	DaM:ODS Screened	0	6	6
5	DND:DNNU	0	6	6
5	DaM:IDS	0	6	6
6	DaM:RDS	1	1	2
7	DaM:ODS Not Screened	0	0	0

Examining the three category 5 rules reveals that they each conflict with the rules: CDS:RL, DaM:ADT, DeM:TS, DeM:AI, DaM:RDTOU, and SI:Prompt.

The category 5 rules prevent the normal functioning of other rules. For instance one of the conflicts that is detected is that DND:DNNU would prevent SI:Prompt from firing as normal. This is a very common pattern for devices such as mobile phones that are put into a silent mode, thereby preventing prompting. Some AA cases, however, require strict adherence to prompting schedules. Such common behaviour may be overlooked by protocol developers and it is important to tune any priority system in order to decide which rule should have priority.

The category 4 rules conflict with ones that terminate a common fluent. The timing of the termination is important. A conflict will emerge if it occurs earlier than the category 4 rule performing a check on it for branching logic. These rules also do not initiate or terminate any common fluents themselves, therefore they do not cause any conflicts. Minimising actions on common fluents can help reduce these types of conflicts.

The only category 3 rule is SDS:STS Inert, and it exposes 8 false positive instances. Conflicts are reported when there are two instances of this rule because the rule only contained Δ sentences without any branching statements. This is clearly an instance of a false positive. It arises because the rule is written to include two instances of the action "listen_for_connection" (the initial and terminal actions) and the SAI conflict rule filters one of them out. When the terminal action is removed from the rule and the conflict check is re-performed, the two instances are in concordance. This new form of the rule, furthermore, concords with all other rules when it is used in the second instance, reducing it to a category 5 rule. The removal of repeated actions is not a long-term solution for conflict detection, however, because it would be possible for rule authors to include such repetitions. Instead, future work should adjust the algorithm to filter out such cases.

Category 2 rules conflict in the same manner as category 4 rules when they act as first instances, but they conflict in the same manner as category 5 rules when they are second instances. The category 1 rules also conflict in the same way as category 4 rules when they act as first instances. They conflict with the rules as category 5 rules do when they are second instances, and include 6 additions. Inspection of the 6 additions reveal that they are false positive results for the same reason as the category 3 ones are.

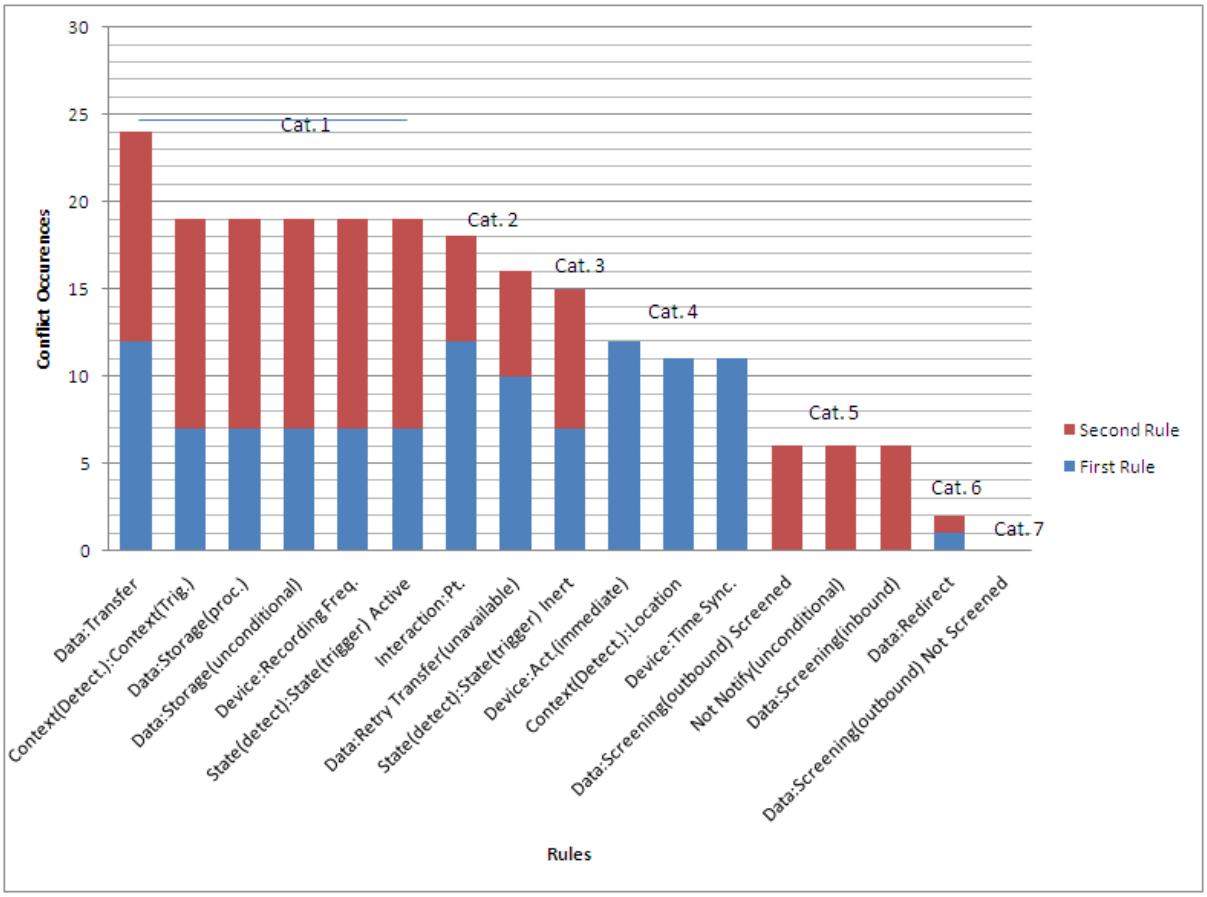


Figure 6.3: The number of SAI conflicts for 17 rules. The mean is 13.06. SD is 7.

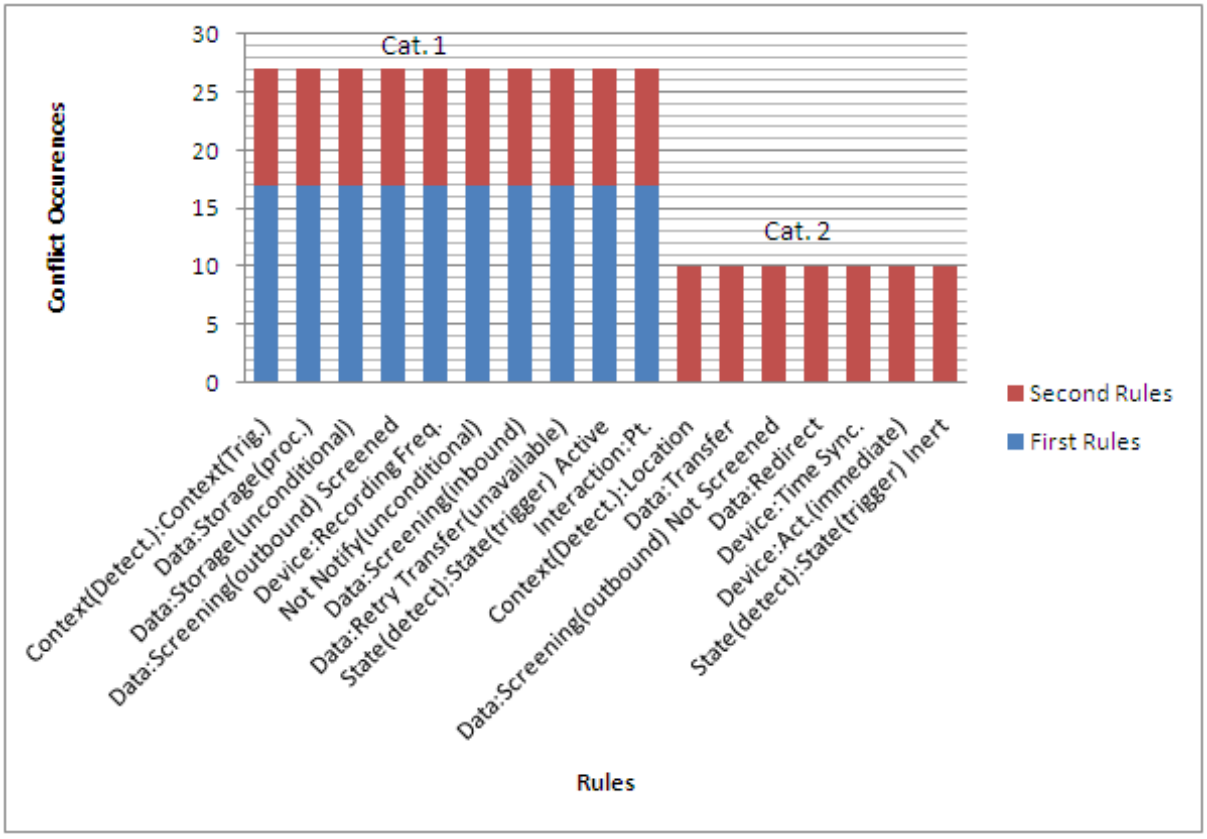


Figure 6.4: The number of MTI conflicts for 17 rules.

6.4.3 Missed Trigger

Looking at MTI in figure 6.4 and table 6.18 shows that there are two categories of rules. The first category contained 10 of the rules, and these are conflict prone as both the first and second rules. The other 7 are only involved in conflicts as second rules with those of the first category. The median number of MTI conflicts is 27 with a range between 10 and 27, and an inter-quartile range of 17. The 10 rules of category 1 are destructive whereas the others are passive. Destructive rules should be minimised in order to reduce missed trigger conflicts. Given these forms of the rules, however, the conflict detection algorithm detected results correctly.

6.4.4 Timing Analysis

Additional testing was performed to determine conflict detection algorithm timing. These tests used the SWI-Prolog "time" statistics feature, and were run on a 2.53 GHz Intel Core 2 Duo CPU with 1.85 GB of RAM. These tests confirm that the conflict algorithms have a complexity of $O(n^2)$.

Analysis shows that conflict detection for 17 rules takes on average under a tenth of a second per conflict type. The rule set is doubled six times (to a total of 1088 rules) in order to view execution times for larger data sets. For each doubling of the number of rules the analysis time increased by around a factor of four. The analysis times vary depending on the conflict type being searched for, owing to the complexity of the analysis rules. In addition, machine-dependent factors (such as other tasks performed by the CPU in the background) influence the timings. To deal with such irregularity, multiple tests were run and average values are reported. Designers can apply a heuristic such that doubling the number of rules may quadruple the time it takes to perform conflict analysis.

Figure 6.5 presents data for timing tests along with a chart of the data points. The figure shows that MTI detection performed the best, while MAI and STI tended to perform a little slower. The increase in the time it takes to complete the conflict analysis for double the number of rules ranged between 2.33

Table 6.18: MTI conflict review.

Cat.	Rule	# Conf. First Rule	# Conf. Second Rule	Total Conf.
1	CDS:CTS	17	10	27
1	DaM:DSTP	17	10	27
1	DaM:DSU	17	10	27
1	DaM:ODS Screened	17	10	27
1	DeM:DRF	17	10	27
1	DND:DNNU	17	10	27
1	DaM:IDS	17	10	27
1	DaM:RDTOU	17	10	27
1	SDS:STS Active	17	10	27
1	SI:Prompt	17	10	27
2	CDS:RL	0	10	10
2	DaM:ADT	0	10	10
2	DaM:ODS Not Screened	0	10	10
2	DaM:RDS	0	10	10
2	DeM:TS	0	10	10
2	DeM:AI	0	10	10
2	SDS:STS Inert	0	10	10

and 6.26, with a median value of 4.08. A chart of the frequency of the increases is shown in figure 6.6. This shows that the combined values of timing increases for the five conflict algorithms tended to cluster around 4. The following are the individual ranges and median values of increases per conflict type:

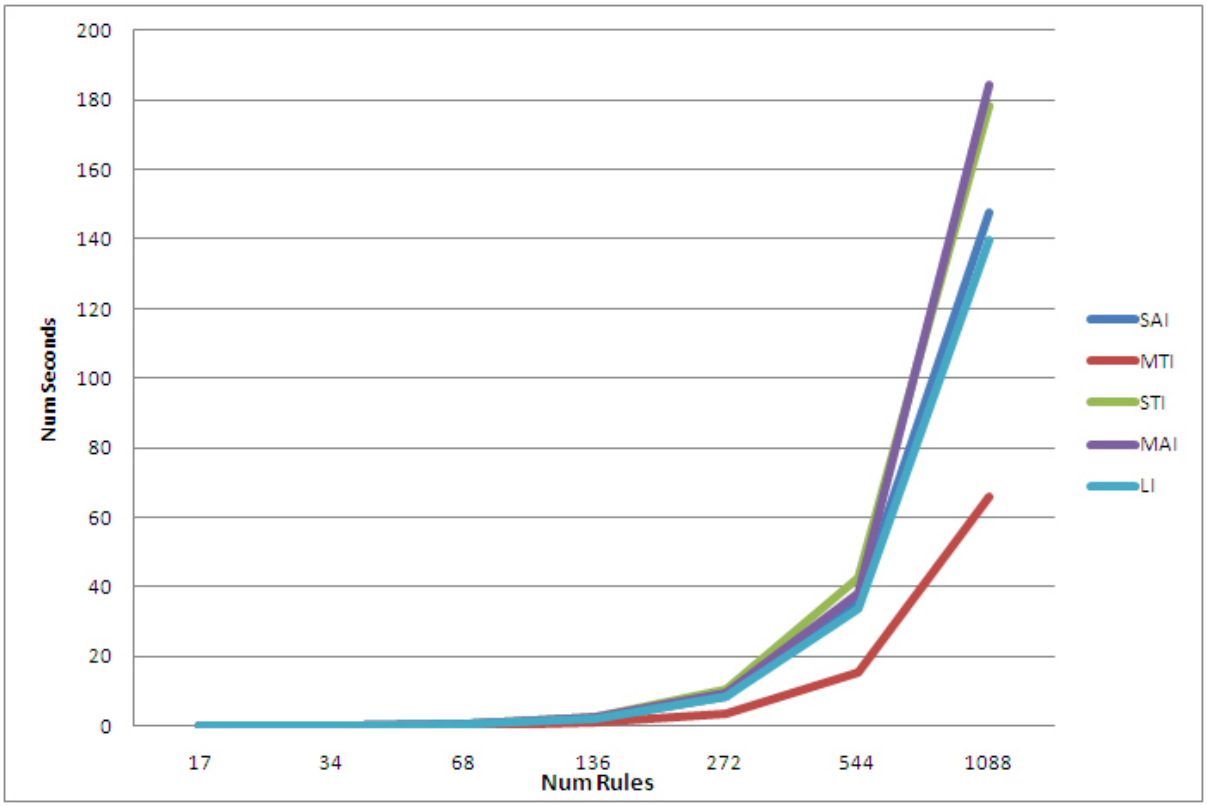
- SAI:
  - Range: 3.33 – 4.35
  - Median: 4.02
- MTI
  - Range: 2.33 – 4.25
  - Median: 3.89
- STI
  - Range: 3.63 – 5.33
  - Median: 4.17
- MAI
  - Range: 3.79 – 4.84
  - Median:4.05
- LI
  - Range: 3.67 – 6.26
  - Median: 3.95

6.5 Evaluation of the Approach

The following discusses advantages and limitations of the Event Calculus (EC)-based rule programming paradigm, and detection and resolution approaches that are presented earlier.

# Rules	SAI Avg. Time (Sec.)	MTI Avg. Time (Sec.)	STI Avg. Time (Sec.)	MAI Avg. Time (Sec.)	LI Avg. Time (Sec.)
17	0.045	0.03	0.03	0.035	0.025
34	0.15	0.07	0.16	0.155	0.095
68	0.505	0.25	0.58	0.625	0.595
136	1.99	0.96	2.51	2.54	2.185
272	8.655	3.79	10.28	9.62	8.31
544	36.02	15.495	42.815	38.075	34.005
1088	147.58	65.915	178.4	184.15	139.905

(a) Timing test results.



(b) Timing results chart.

Figure 6.5: Timing Analysis.

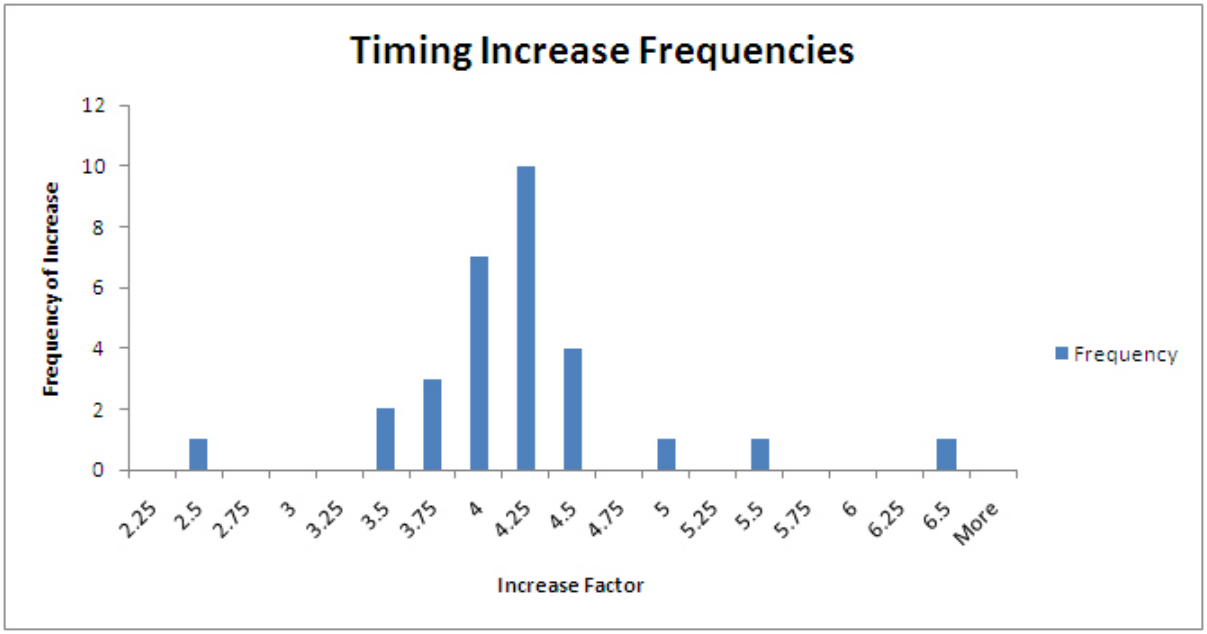


Figure 6.6: Timing increase frequencies for doubling the number of rules

6.5.1 Advantages

The approach presented in this work is one of bottom-up device programming that allows multiple providers to offer services and to form *ad hoc* reliable networks. Furthermore, the network is expected to change in time with devices coming and going, owing to replaced technologies, new discoveries, and device reliability issues (such as battery life). This work has significant advantages.

A significant contribution of this work is an approach that detects and resolve all five forms of conflict that are looked for; something the author is unaware of other systems accomplishing. In contrast, as is shown in chapter 2, works developed from Marples [2000] are not able to detect MTI and Event-Condition-Action (E-C-A) based authorisation/obligation policy work chiefly tends to focus on detecting and resolving MAI-like conflicts.

The rules presented in this work provide greater support for AA protocol designers than previous systems such as Experience Sampling Program (ESP) and Purdue Momentary Assessment Tool (PMAT), including:

- Studies can mix event-based, time-based, subject state-based and contextual triggers
- Subjects may be prompted for questionnaire assessment in dynamic response to the concurrent collection of on-body and environmental data
- Thorough data handling and provision mechanisms
- Rule-based presentation of information
- Automatic conflict detection and resolution

These advantages may greatly aid the uptake of AA. Larsen [2007], for instance, pointed out the primacy of research questions over data collection and analysis. The above items provide researchers with greater abilities to pursue their questions, and allow them to be flexible in their approaches to their collection and analysis methods. Furthermore, these points lead strongly towards realising Intille [2007]'s CS-EMA+10 vision of of AA future developments, including supporting the data collection procedures he outlined for longitudinal studies such as data-reactive subject prompting.

6.5.2 Limitations

This thesis supports areas to do with reliable behaviour monitoring. As such it is able to show that conflicts can emerge in Wireless Sensor Network (WSN) comprising relatively small sets of nodes. It provides a mechanism for describing rules and conducting conflict detection and resolution for the particular type of WSN under investigation. The study is limited to aspects relating to service description access, network scale, focus on particular types of conflict, run-time behaviour, example extrapolation, heavy-handedness of resolution, and timing issues.

An important issue for the approach presented here is that it requires access to rules for all of the devices that are participating in the network. The approach assumes that the mediator will be trusted by all participants and vendors, and that they will be amenable to writing rules and actions in the formats presented herein. Some other approaches, such as that of Kolberg [2004], are able to perform a degree of analysis without such requirements. The requirements are justified for care systems, however, because transparency is vital to the form of clinical analysis that these technologies are subject to.

A limitation of the conflict analysis approach is that it compares all of the rules against each other for each of the conflict types, and therefore is of $O(n^2)$ complexity. This type of approach works with a number of devices and rules that is appropriate for care networks, which is on the order of a dozen or so devices that have access to a couple of dozen rules. Under these conditions, the algorithms complete within acceptable time constraints. For every doubling of the number of rules used, however, there is an approximate four-fold increase in the time it takes to perform conflict analysis. Such an approach, therefore, will not be appropriate for systems that interconnect large numbers of sensors that use vast numbers of rules to guide their behaviours.

System testing is constrained to run-time simulation rather than the run-time analysis of a real-world WSN. This has the advantage of allowing different virtual devices to inter-operate without particular constraints, such as limited support for the Prolog and Java languages on mobile devices. The testing is also limited by the selection of the rules and priorities. The rules are selected from a process that involved reviewing literature on AA and Feature Interaction (FI), as well as from experience working on the Personalised Ambient Monitoring (PAM) project. This selection process is not exhaustive, but rather is intended to be representative of rules that may be used in a number of care settings. The rules are translated by the author from literature-based descriptions into their diagrammatic and EC based forms. In only a very few cases (such as for DeM:TS) did the descriptions in the literature include working algorithms. Interpretations of the descriptions are necessary. The author intended to represent the rules accurately in their EC rule forms, but at times alternative interpretations are possible (such as having multiple forms of the rule SDS:STS). In contrast, other approaches such as Calder *et al.* [2009] are able to analyse rules directly from log files of running systems.

System testing is also constrained to the examination of the analysis and resolution system as discrete elements. This separation of concerns is chosen in order to focus on the nature of the rule conflicts specifically. Further real-world testing of WSN for care networks that have conflict analysis built into them is a strong recommendation for future work. It is felt, however, that the current level of testing showes that conflicts in the rules can be detected and that appropriate resolution steps can be triggered in response.

The priorities are also interpreted by the author. Alternative priorities are conceivable, and such alternatives can be beneficial as part of the personalisation of any system for real-world subjects. The system presented herein is able to accept other conflict detection rules, but no assertions are made regarding how well any other conflict rules will perform. Some of these, however, may work better than others. Their success will be based on how well their algorithms match the concerned conflict types and on their focus on the comparison of rules with each other. The resolution approach may also not be appropriate for all types of applications. As noted, for instance, in the Looping 2 example results, both conflicting rules may be affected by the resolution system, when only one of the two may need to be. Future work may concentrate on developing a more subtle resolution approach.

The results of the study are impacted by the detection of false positives in one of the conflict detection algorithms, as well as by the number of examples used in the testing. Eight false positive results are detected in SAI analysis. The detection of these results arises in a very unusual circumstance, and the other conflict analysis results are not impacted by this. The algorithm for detecting SAI may be improved in future work to eliminate this cause of false positive results by filtering cases where multiple rules share the same α sentences. No false negative results are detected in the examples, however this is limited by

Table 6.19: MTI detection results for abstract rules (macroA, macroB, and passive).

Rule 1	Rule 2	Result
macroA	macroA	MTI
macroA	macroB	MTI
macroA	passive	MTI
macroB	macroA	MTI
macroB	macroB	MTI
macroB	passive	MTI
passive	macroA	Concordance
passive	macroB	Concordance
passive	passive	Concordance

the number or type of examples chosen for testing. The examples are chosen as reasonable instances of conflict that fit within previously established FI testing constraints. Analysis of the SAI and STI algorithms revealed them to be orientated to reporting worst-case results. These algorithms are limited because they are not capable of determining the quality (goodness or badness) of the type of conflict. As such the numbers of conflicts that are detected in chapter 6 should be viewed with discretion since, firstly, some of these conflicts may only arise very rarely (if at all) in a running system and, secondly, that these conflicts may actually be seen to be beneficial by some designers and users of such systems.

The MTI algorithm is sensitive to the ordering of events. A question arises regarding the information that is known to CLIPPER when it checks rules, since it resets its known Σ, Δ and Δ_0 predicates prior to its analysis (although any other state information is left within the system). The potential problem is that a rule may need to be called multiple times before a conflict occurs, owing to stated assertions that it can make. Such a limitation can be overcome in CLIPPER in at least two ways. Firstly, a recursive call to the given rule can allow it to be called multiple times, allowing it to assert state information that is not reset prior to checks, and thereby alter its behaviour. For example, consider the "state_destro" rule in listing B.13. On its first call, the "perform_activity" fluent checks to see if it holds. It will not, because nothing will initiate it, so the second block of the rule will be entered. In this block the fluent will be initiated and then the rule will be re-called. The subsequent call allows the first block to proceed, which will then lead to the termination of the message fluent. This process makes this a state-oriented destructive rule. A second related method to assert state into the system is by composing rules that call other rules (what is called here a macro rule). Two such macro rules are presented in listing B.13 [2]. It should be noted that the results are the same regardless of the ordering of the rule calls within the macro rule when testing "macroA" and "macroB" for MTI against the "passive" rule, as shown in table 6.19. It should be further noted that the testing of the example rules did not involve the pre-population of such state information and as such is not of concern for this process.

6.6 Possible Applications of Approach

The approach described in this thesis can be used for other applications besides ambulatory assessment. The base set of rules may be appropriate for other sensor network monitoring tasks such as environmental monitoring or multi-sensory robotics.

Alternatively, the rule set may be extended to support additional features. For instance, this approach could be used in a call control environment that requires a response to personalisable and dynamic rules. Such a situation can potentially arise in a loosely federated system of services across various Voice Over IP solutions. In this scenario it may be interesting to add a "teenline" feature that restricts outbound calls from a subscriber's phone during particular times of day. To add such a rule, begin by modelling its activities and sequences. Then axiomatise the activity model into EC notation. For the "teenline" example, activities may include events for attempting to connect and checking connection, and there may be fluents for the particular restricted time of day.

[2] There are instances of macro rule usage in chapter 6 in order to test of the screening rules using alternative states (that a device is or is not screened)

Additional conflict rules may also be written to extend the approach. Beginning with a diagram of the algorithm type is useful. The algorithm should be based on firing rules sequentially or in parallel and examining the asserted EC predicates. There is potential to look for conditions such as resource starvation, race conditions, deadlocks, and so forth.

6.7 Rule Authoring Advice

Rules are composed by converting a starting notion of the rule, possibly as a textual description, into its EC format. Use the predicates from Table 2.1 in the following ways to compose a rule. Begin with a unique name for the rule and ensure it is of arity 2, with the two arguments being a triggering message and an initial time point. The next predicates are, optionally, any additional time points that the rule will require for sequencing the events that happen in time. Multiple events may share the same time point within happens predicates. Next assert the events that happen in the rule. Finally, assert predicates to describe how the events initiate and terminate fluents. A *holdsAt* predicate may be used along with parentheses to describe conditions under which initiations and terminations may differ depending on fluent states.

When composing rules consider the rule writing heuristics described in this chapter. In particular the fluent sharing with other known rules should be kept to a minimum and the rule should be kept passive (it should not alter the triggering message) unless there is a particular reason for doing so. Once a rule is written, it may be immediately analysed to see if it conflicts with other known rules, or alternatively action classes may be encoded in order to implement the rule in the run-time environment. Some rules are inherently designed to conflict. Security rules, for instance, can prevent the firing of other rules. The main trouble is with rules that can be daisy chained. Preventing a rule from firing can have unintended consequences down the line. Sampling protocol designers should maintain a degree of awareness that rules may not always fire as expected for such reasons. They should take precautions to protect their protocols.

Rule authors are encouraged to write passive rules whenever possible. An underlying message passing system should handle message destruction, rather than leaving it to high level application protocols. They are also encouraged to limit the amount of fluent sharing between rules. The locking mechanism of Wilson [2005] is one such attempt. Future work to develop a similar approach for the mobile device realm may help to alleviate some of these difficulties. Some cases, however, will remain when fluents will be shared. If this is the case then it will be important that rule authors are aware of termination timings and take precautions to prevent conflicts that can emerge from these.

6.8 Summary

This chapter has made two contributions to the thesis. Firstly, it presents evaluations of the analysis approach against examples discussed in chapter 4. Secondly, it explains why a number of conflicts occur between rules that highlight the dangers of blindly programming devices independently of each other.

The examples are of particular interest because they are able to show that no false negative results are found. The results of each of these tests are as expected. The study is limited to some degree by the number of examples. Future work may expand these to include additional examples.

The three core conflict algorithms were also tested to ascertain false positive results. None are found for STI nor MTI. A small number of false positive results were discovered for SAI detection. The cause of this has to do with an unexpected circumstance in which multiple rules share the same α sentences. Future work may involve an investigation into refining the SAI detection to filter out such cases.

In addition, four rule writing heuristics are presented from the analysis of the conflicts: conflict by design, minimise destructive rules, limit shared fluents, be aware of fluent termination timings.

The next chapter presents a short review of the thesis and a discussion of future work.

Chapter 7

Conclusions

7.1 Introduction

An approach to handling device conflicts in rule-based ambulatory assessment sensor networks has been considered in this thesis. This chapter reviews the work of this thesis as a whole. Section 7.1 provides a summary of the work. Section 7.2 evaluates the research objectives and questions. Section 7.3 provides a discussion of future work.

7.2 Summary of work

This thesis presents a study into conflicts that emerge amongst low-level sensor device rules when such devices are formed into networks to perform complex tasks such as Ambulatory Assessment (AA). Background information about rule-based programming, conflict detection and resolution, Event Calculus (EC), Wireless Sensor Network (WSN) programming and AA are provided in chapter 2.

Rules to control the behaviours of devices in a sensor network for AA are determined from literature described in chapter 3 and categorised in chapter 4. This includes rules that are necessary and sufficient for AA Wireless Sensor Networks (WSNs). EC-based forms of these rules are presented that can be analysed for conflict. Chapter 3 reviews AA projects to determine necessary and sufficient rules for analysis in subsequent chapters. This review includes state of the art in AA, along with considerations of future directions for AA from the literature. Along with descriptions of device and knowledge management rules for AA, chapter 4 shows that the use of the rules together can lead to conflicts. Examples describe how five types of conflict may emerge through their usage. Examples are given that exemplify inter-service and intra-service conflict for multiple rules running on a single device or on multiple devices. The examples also show that multiple types of conflicts may be caused by the same rules.

Chapter 5 describes an approach for rule conflict detection and resolution. Key concepts are presented for EC logic-based detection algorithms for Missed Trigger Interaction (MTI), Shared Trigger Interaction (STI), Multiple Action Interaction (MAI), Sequential Action Interaction (SAI) and Looping Interaction (LI), and resolution strategies derived from device priorities. Chapter 6 presents evaluations of the approach against examples discussed in chapter 4 and explains why a number of conflicts occur between rules. It also describes four rule writing heuristics that can be used to minimise rule conflict.

7.3 Thesis Achievements

This section reviews the research questions and objectives presented in chapter 1 and describes what has been achieved in the thesis. The research questions are:

- What rules can be used to control the behaviours of devices in a sensor network for AA?
- Can examples of rule conflict be found for AA sensor networks?
- How can AA sensor network rule conflicts be detected and resolved?

The corresponding research objectives are:

- To determine and describe rules necessary and sufficient for AA sensor networks
- To examine AA sensor network examples for rule conflict
- To demonstrate and evaluate an approach to detecting and resolving rule conflicts within AA sensor networks

The points are addressed here with a view to show the distinctive contribution this work makes towards advancing the state of scientific knowledge.

7.3.1 What rules can be used to control the behaviours of devices in a sensor network for AA?

This thesis presents an investigation into the state of the art and future directions for AA, as described in chapter 3. It has shown numerous rules can be used to describe the features that are present in these works. The rules are categorised in chapter 4 into groups for device control and knowledge management. This work is the first to formally describe rules necessary and sufficient for AA and does so using an approach based on the EC logic language.

Chapter 3 presents a review of state of the art AA projects. Rules were derived from these examples. AA has grown out of asking subjects questions in ambulatory settings. However, there are additional features now to support sensor data, processing and response. Underlying these additional features are important device and knowledge management capabilities from time synchronisation and data storage to context and state detection. Rules have been documented in this thesis that formally describe these capabilities. These rules are necessary and sufficient for the future of AA.

7.3.2 Can examples of rule conflict be found for AA sensor networks?

This thesis presents a number of examples of AA sensor network rule conflicts that can emerge. The examples are found from a search for five types of conflict described in Feature Interaction (FI) literature. The examples are presented in chapter 4. Examples are given that exemplify inter-service and intra-service conflict for multiple rules running on a single device or on multiple devices. The examples also show that multiple types of conflicts may be caused by the same rules. Rule conflict is a problem that can afflict rule based interacting devices.

The examples describe conflicts of pairs of rules. For example, to test the MTI detection rules, the example describes a rule that delays a message that would interfere with another device. Using similar examples 410 conflicts were detected within 867 tests across 17 rules.

Investigation into each of the types of conflict revealed different categorisations of rules. Many but not all the rule conflicts included rules as both the first and second rules in the analysis. Looking at the patterns of conflict showed how rules can be categorised by where they conflict and by how much.

7.3.3 How can AA sensor network rule conflicts be detected and resolved?

This thesis contributes a novel method to detecting and resolving AA sensor network rule conflicts. Chapter 4 describes CoLlaborative Information Processing Protocol and Extended Runtime (CLIPPER) which is assessed in chapter 6.

In brief, this work contributes methods that work as follows. Instances of different forms of conflict amongst collections of device rules are detected by triggering devices rules and passing messages to them. Rule execution sequences are analysed to determine whether they lead to conflict. Device rules are checked for conflicts between pairs of rules (including checking rules against themselves). Checking a pair of rules involves two phases: initialisation and detection. Device rules, time points and messages are passed to conflict detection rules. The conflict detection rules are used to evaluate whether or not the device rules are concordant or conflict, and to record evaluation results. Priority files associated with each device, the core conflict analysis and resolution files, and the conflict detection report file are consulted to determine the correct resolution for the conflicts. Variables from each of the conflicts found in the conflict report are passed in to the resolution system's resolve rule. The variables include the type

of conflict (such as MAI) and each device and device rule that conflicted. Each rule is checked to see if a priority has been applied to it. If both have priorities they are checked to see which has precedence. The rule with the lowest precedence is disabled as are rules that have no priorities applied to them. If two rules have equal precedence then they are both disabled.

A significant contribution of this work is an approach that detects and resolves all five forms of conflict that are looked for; something the author is unaware of other systems accomplishing. In contrast, as is shown in chapter 2, works developed from Marples [2000] are not able to detect MTI. Event-Condition-Action (E-C-A) based authorisation/obligation policy works chiefly tend to focus on detecting and resolving MAI-like conflicts.

7.4 Additional Areas of Research

A number of areas of future research have been identified. These include: simplifying rule development by developing rule writing tools, developing systems of abduction and model finding from automated Δ predicates, testing the approach within a larger-scale future version of the PAM network architecture to provide a more responsive system, adding additional device types and rules, and improving the SAI algorithm.

Device rule writing and action class development are still manual processes. It will be beneficial to develop rule writing tools with alerts built-in to help guide users and point out the heuristics mentioned above. A visual editor could be designed that allows users to specify their rules graphically and auto-generate the Δ and Σ predicates from these.

Another way to simplify rule development may be to use alternative forms of reasoning such as abduction or model finding. The approach described in this thesis is deductive, however it has previously been shown by Mueller [2006] and others that EC descriptions may be used for other forms of reasoning. For instance, an abductive planner could be added to auto-generate rules given initial and terminal fluents. It may also be interesting to attempt to perform rule generation by collecting system log files, such as were used by Calder *et al.* [2009], and generating rules from discovered fluents and events.

Other important future work involves testing the approach within a larger-scale future version of the Personalised Ambient Monitoring (PAM) network architecture. This testing should involve additional device types and it will be of interest to test the conflict detection system against these. Such testing should unite the separated parts of the analysis and resolution system to test it as a whole. A larger scale system should be rolled out for use with more patients and involve the approaches discussed here to maintain the device network and provide services. Such an infrastructure and deployment can allow for a greater amount of user studies. Such user studies would be of importance for reviewing priorities in particular. Alternative priorities to those presented in this thesis should be determined and reviewed.

Future work may concentrate on developing a more subtle resolution approach. As described in chapter 2, Nakamura *et al.* [2009] suggested that disabling rules as a whole may be too heavy handed. It would be of interest to see whether the resolution system could be refined to disable only some actions from within a rule but in such a way as to maintain the integrity of a sensor network.

The system presented herein is able to accept other conflict detection rules, but no assertions are made regarding how well any other conflict rules will perform. Some of these, however, may work better than others. Their success will be based on how well their algorithms match the concerned conflict types and on their focus on the comparison of rules with each other.

A small number of false positive SAI detection results are noted in chapter 6, owing to multiple rules sharing the same α sentences. Future work could involve refining the SAI detection algorithm to filter out such cases. Such filtering can be added to the rule writing tool discussed above.

System testing is constrained to run-time simulation rather than the run-time analysis of a real-world WSN. This had the advantage of allowing different virtual devices to inter-operate without particular constraints, such as limited support for the Prolog and Java languages on mobile devices. The testing is also limited by the selection of the rules and priorities. The rules are selected from a process that involved reviewing literature on AA and FI, as well as from experience working on the PAM project. This selection process is not exhaustive, but rather is intended to be representative of rules that may be used in a number of care settings. The rules are translated by the author from literature-based descriptions into their diagrammatic and EC based forms. In only a very few cases (such as for Device Management (DeM):Time Synchronisation (TS)) did the descriptions in the literature include working

algorithms. Interpretations of the descriptions are necessary. The author intended to represent the rules accurately in their EC rule forms, but at times alternative interpretations are possible (such as having multiple forms of the rule State Detection Service (SDS):State Triggering System (STS)). In contrast, other approaches such as Calder *et al.* [2009] are able to analyse rules directly from log files of running systems. Although intriguing, such an approach begs the question as to where the rules originated from to begin with. In addition, Calder *et al.*'s work examined a particular conflict type (rule redundancy) that is not specifically addressed by the work in this thesis. The thesis concentrates on detecting and correcting five types of conflict, although there are some conceptual parallels between the notion of redundancy (in action) and that of MAI. These five are selected from previous literature in FI and chosen because of their likelihood to affect such types of networks.

7.5 Summary

This chapter concludes the thesis with a review of the work as a whole, a reflection on its contributions in respect to the research questions and provides a description of future work. The thesis describes rules to control the behaviours of devices in a sensor network determined from literature into AA.These have been described by the author in a EC-based format for conflict analysis. The author has contributed an approach to detecting and resolving conflicts amongst the rules. Examples are given describing how five types of conflict may emerge through their usage. The thesis presents an evaluation of the approach showing numerous conflicts that can be found amongst AA sensor network rules. This chapter has shown that the three main research questions have been answered and that future work can continue to investigate issues arising from this work.

References

1946 (June). *Preamble to the Constitution of the World Health Organization as adopted by the International Health Conference.* World Health Organization.

1998. *Telepsychiatry Via Videoconferencing.*Committee on Telemedical Services, for The American Psychiatric Association.

2005 (April). *The Use of New Medical Technologies within the NHS.* Oral and written evidence, Ordered by The House of Commons, vol. 2. The House of Commons Health Committee.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. 2002. Wireless sensor networks: a survey. *Computer Networks*, **38**(4), 393–422.

Amor, JD, & James, CJ. 2009. Personalized ambient monitoring: accelerometry for activity level classification. *Pages 866–870 of: 4th European Conference of the International Federation for Medical and Biological Engineering.* Springer.

Bandara, A.K., Lupu, E.C., & Russo, A. 2003. Using Event Calculus to Formalise Policy Specification and Analysis. *Pages 26–39 of: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks.* IEEE Computer Society.

Blair, L., & Turner, K. 2005. Handling policy conflicts in call control. *Feature Interactions in Telecommunications And Software Systems VIII*, **1**, 39–57.

Blum, J., & Magill, E. 2010. The Design and Evaluation of Personalised Ambient Mental Health Monitors. *In: Proceedings of the 7th Annual IEEE Consumer Communications and Networking Conference.* IEEE.

Bowen, T.F., Dworack, F.S., Chow, C.H., Griffeth, N., Herman, G.E., & Lin, Y.J.If. 1989. The feature interaction problem in telecommunications systems. *Pages 59–62 of: Software Engineering for Telecommunication Switching Systems, 1989. SETSS 89., Seventh International Conference on.*

BPS-Research-Board. 2010. *Guidelines on Memory and the Law: Recommendations from the Scientific Study of Human Memory.* Tech. rept. The British Psychological Society.

Broda, K., Clark, K., Miller, R., & Russo, A. 2009. SAGE: A Logical Agent-Based Environment Monitoring and Control System. *Pages 112–117 of: Proceedings of the 3rd European Conference on Ambient Intelligence (AmI'09).* Springer.

Calder, M., & Miller, A. 2006. Feature interaction detection by pairwise analysis of LTL properties - A case study. *Formal Methods in System Design*, **28**(3), 213–261.

Calder, M., Kolberg, M., Magill, E.H., & Reiff-Marganiec, S. 2003a. Feature interaction: a critical review and considered forecast. *Computer Networks*, **41**(1), 115–141.

Calder, M., Kolberg, M., Magill, E., Marples, D., & Reiff-Marganiec, S. 2003b. Hybrid solutions to the feature interaction problem. *Feature interactions in telecommunications and software systems VII*, **1**, 295.

Calder, M., Gray, P., & Unsworth, C. 2009. Tightly coupled verification of pervasive systems. *Electronic Communications of the EASST*, **22**(0), 1–16 (Open Access).

Cameron, E.J., Griffeth, N., Lin, Y.J., Nilson, M.E., Schnure, W.K., & Velthuijsen, H. 1994. A feature interaction benchmark for IN and beyond. *Feature Interactions in Telecommunications Systems*, **1**, 1–23.

Charalambides, M., Flegkas, P., Pavlou, G., Bandara, A.K., Lupu, E.C., Russo, A., Dulav, N., Sloman, M., & Rubio-Loyola, J. 2005. Policy conflict analysis for quality of service management. *Pages 99–108 of: Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*. IEEE.

Chen, L., Nugent, C., Mulvenna, M., Finlay, D., Hong, X., & Poland, M. 2008. Using event calculus for behaviour reasoning and assistance in a smart home. *Smart Homes and Health Telematics*, **1**, 81–89.

Chomicki, J., Lobo, J., & Naqvi, S. 2003. Conflict resolution using logic programming. *Knowledge and Data Engineering, IEEE Transactions on*, **15**(2), 244–249.

Christensen, T.C., Barrett, L.F., Bliss-Moreau, E., Lebo, K., & Kaschub, C. 2003. A Practical Guide to Experience-Sampling Procedures. *Journal of Happiness Studies*, **4**(1), 53–78.

Collins, R.L., & Muraven, M. 2007. *The science of real-time data capture: Self-reports in health research*. Oxford University Press, Inc. Chap. Ecological momentary assessment of alcohol consumption, pages 189–203.

Colmerauer, A., & Roussel, P. 1996. The birth of Prolog. *Pages 331–367 of: History of programming languages—II*. ACM.

Consolvo, S., & Walker, M. 2003. Using the experience sampling method to evaluate ubicomp applications. *IEEE Pervasive Computing*, **1**, 24–31.

Costa, P., Mottola, L., Murphy, A.L., & Picco, G.P. 2007. Programming wireless sensor networks with the TeenyLime middleware. *Lecture Notes in Computer Science*, **4834**, 429–449.

Crespo, R.G., Carvalho, M., & Logrippo, L. 2007. Distributed resolution of feature interactions for internet applications. *Computer Networks*, **51**(2), 382–397.

Curino, C., Giani, M., Giorgetta, M., Giusti, A., Murphy, A.L., & Picco, G.P. 2005. Mobile data collection in sensor networks: The TinyLime middleware. *Pervasive and Mobile Computing*, **1**(4), 446–469.

Das Gupta, R., & Guest, J. F. 2002. Annual cost of bipolar disorder to UK society. *The British Journal of Psychiatry*, **180**(3), 227–233.

Dedecker, Jessie, Van Cutsem, Tom, Mostinckx, Stijn, D'Hondt, Theo, & De Meuter, Wolfgang. 2005. Ambient-oriented programming. *Pages 31–40 of: OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM.

Dressler, F., Dietrich, I., German, R., & Krüger, B. 2009. A rule-based system for programming self-organized sensor and actor networks. *Computer Networks*, **53**(10), 1737–1750.

Ebner-Priemer, U. 2010. *Society for Ambulatory Assessment (SAA) – Understanding Behavior in Context*. last accessed on 26 May, 2010.

Ebner-Priemer, U.W., & Kubiak, T. 2007. Psychological and Psychophysiological Ambulatory Monitoring: A Review of Hardware and Software Solutions. *European Journal of Psychological Assessment*, **23**(4), 214–226.

Ebner-Priemer, U.W., & Trull, T.J. 2009. Ambulatory Assessment - An Innovative and Promising Approach for Clinical Psychology. *European Psychologist*, **14**, 109–119.

Efstratiou, C., Friday, A., Davies, N., & Cheverst, K. 2002. Utilising the event calculus for policy driven adaptation on mobile systems. *Proc. Third International Workshop on Policies for Distributed Systems and Networks, 2002.*, **1**, 13–24.

Fahrenberg, J. 2006. *Assessment in daily life. A Review of Computer-assisted Methodologies and Applications in Psychology and Psychophysiology, years 2000–2005.* Retrieved Aug, 2009.

Fei, X., & Magill, E. 2008. Rule Execution and Event Distribution Middleware for PROSEN-WSN. *Pages 580–585 of: Sensor Technologies and Applications, 2008. SENSORCOMM'08. Second International Conference on.*

Feldman, Barrett, L., & Barrett, D.J. 2001. An Introduction to Computerized Experience Sampling in Psychology. *Social Science Computer Review*, **19**(2), 175–185.

Fischer, J.E. 2009. *Experience-Sampling Tools: a Critical Review.*

Froehlich, J., Chen, M.Y., Consolvo, S., Harrison, B., & Landay, J.A. 2007. MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. *In: Proceedings of the 5th international conference on Mobile systems, applications and services.* ACM.

Gonzalez, Avelino J., & Dankel, Douglas D. 1993. *The Engineering of Knowledge-Based Systems Theory and Practice.* Prentice-Hall, Inc.

Greenhalgh, C. 2002. EQUIP: a software platform for distributed interactive systems. *The Equator Project, University of Nottingham, Nottingham*, **1**, .

Hadim, S., & Mohamed, N. 2006. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, **7**(3), 1–1.

Heinzelman, WB, Murphy, AL, Carvalho, HS, & Perillo, MA. 2004. Middleware to support sensor network applications. *IEEE network*, **18**(1), 6–14.

Henricksen, K., & Robinson, R. 2006. A survey of middleware for sensor networks: state-of-the-art and future directions. *Pages 60–65 of: Proceedings of the international workshop on Middleware for sensor networks.* ACM New York, NY, USA.

Henry, B.L., Minassian, A., Paulus, M.P., Geyer, M.A., & Perry, W. 2010. Heart rate variability in bipolar mania and schizophrenia. *Journal of psychiatric research*, **44**(3), 168–176.

Hill, M.A. 1992. Light, circadian rhythms, and mood disorders: A review. *Annals of Clinical Psychiatry*, **4**(2), 131–146.

Horré, W., Matthys, N., Michiels, S., Joosen, W., & Verbaeten, P. 2007. *A survey of middleware for wireless sensor networks.* Tech. rept. 498. Department of Computer Science, K.U.Leuven.

Intille, S., Tapia, E., Rondoni, J., Beaudin, J., Kukla, C., Agarwal, S., Bao, L., & Larson, K. 2003. Tools for studying behavior and technology in natural settings. *Pages 157–174 of: UbiComp 2003: Ubiquitous Computing.* Springer.

Intille, S.S. 2007. *The Science of Real-time Data Capture: Self-reports in Health Research.* Oxford University Press, USA. Chap. Technological Innovations Enabling Automatic, Context-Sensitive Ecological Momentary Assessment, pages 308–337.

Isern, D., Moreno, A., Sánchez, D., Hajnal, Á., Pedone, G., & Varga, L.Z. 2011. Agent-based execution of personalised home care treatments. *Applied Intelligence*, **34**(2), 155–180.

Istepanian, R.S.H., Laxminarayan, S., & Pattichis, C.S. 2005. *M-health: Emerging mobile health systems.* Springer-Verlag New York Inc.

Jackson, Peter. 1999. *Introduction to Expert Systems.* Third edn. Addison Wesley Longman Limited.

Jennings, N.R. 2000. On agent-based software engineering. *Artificial intelligence*, **117**(2), 277–296.

Jovanov, E., Milenkovic, A., Otto, C., & De Groen, P.C. 2005. A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, **2**(1), 6.

Kahn, J.M., Katz, R.H., & Pister, K.S.J. 1999. Next century challenges: mobile networking for "Smart Dust". *Pages 271–278 of: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM.

Kolberg, M. 2004. *Service interaction management in a deregulated market environment*. Ph.D. thesis, University of Strathclyde.

Kolberg, M., Magill, E.H., Marples, D., & Reiff, S. 2000. Results of the Second Feature Interaction Contest. *Feature Interactions in Telecommunications and Software Systems VI*, **1**, 311–325.

Kowalski, R. 1974. Predicate logic as a programming language. *Information processing*, **74**, 569–574.

Kowalski, R., & Sergot, M. 1986. A logic-based calculus of events. *New generation computing*, **4**(1), 67–95.

Laney, R., Tun, T. T., Jackson, M., & Nuseibeh, B. 2007. Composing Features by Managing Inconsistent Requirements. *In:* du Bousquet, L. (ed), *Feature Interactions in Software and Communication Systems IX*.

Larsen, R. J. 2007. *The Science of Real-time Data Capture: Self-reports in Health Research*. Oxford University Press, Inc. Chap. Personality, Mood States, and Daily Health, pages 251–267.

Le, B., Choi, H.N., & Beal, D.J. 2006. Pocket-sized psychology studies: Exploring daily diary software for Palm Pilots. *Behavior research methods*, **38**(2), 325.

Lupu, E.C., & Sloman, M. 1999. Conflicts in policy-based distributed systems management. *Software Engineering, IEEE Transactions on*, **25**(6), 852–869.

Madden, Samuel R., Franklin, Michael J., Hellerstein, Joseph M., & Hong, Wei. 2005. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst*, **30**, 122–173.

Malan, D., Fulford-Jones, T., Welsh, M., & Moulton, S. 2004. CodeBlue: An adhoc sensor network infrastructure for emergency medical care. *In: Proc International Workshop on Wearable and Implantable Body Sensor Networks*.

Marples, D. 2000. *Detection and Resolution of Feature Interactions in Telecommunications Systems During Runtime*. Ph.D. thesis, University of Strathclyde.

Masoumzadeh, A., Amini, M., & Jalili, R. 2007. Conflict detection and resolution in context-aware authorization. *Pages 505–511 of: Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 1. IEEE.

Masri, W., & Mammeri, Z. 2007. Middleware for wireless sensor networks: A comparative analysis. *Pages 349–356 of: Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*.

McCarthy, J., & Hayes, P. J. 1969. Some Philosophical Problems from the Standpoint of Artifcial Intelligence. *Machine Intelligence*, **4**, 463–502.

Miller, R., & Shanahan, M. 1999. The Event Calculus in Classical Logic – Alternative Axiomatisations. *Computer and Information Science*, **4**, 16.

Montangero, C., Reiff-Marganiec, S., & Semini, L. 2008. Logic-based conflict detection for distributed policies. *Fundamenta Informaticae*, **89**(4), 511–538.

Mottola, L., & Picco, G.P. 2010. *Programming wireless sensor networks: Fundamental concepts and state of the art*. To appear in ACM Computing Surveys.

Mueller, E.T. 2006. *Commonsense reasoning*. Morgan Kaufmann.

Nakamura, M., Igaki, H., Yoshimura, Y., & Ikegami, K. 2009. Considering Online Feature Interaction Detection and Resolution for Integrated Services in Home Network System. *Pages 191–206 of: 10th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2009).*

Negnevitsky, Michael. 2002. *Artificial Intelligence A Guide to Intelligent Systems.* Pearson Education Limited.

Pawar, Pravin, van Beijnum, Bert-Jan, van Sinderen, Marten, Aggarwal, Akshai, Hermens, Hermie, Konstantas, Dimitri, & Maret, Pierre. 2008. Performance Evaluation of the Context-Aware Handover Mechanism for the Multi-Homed Nomadic Mobile Services. *Elsevier Computer Communication journal*, **1**(July), 1.

Plath, M., & Ryan, MD. 2000. *Entry for FIW00 Feature Interaction Contest.* Tech. rept. School of Computer Science, University of Birmingham.

Provetti, A. 1996. Hypothetical reasoning: from situation calculus to event calculus. *Computational Intelligence Journal*, **12**(3), 478–498.

Reiff-Marganiec, S. 2002 (May). *Runtime Resolution of Feature Interactions in Evolving Telecommunications Systems.* Ph.D. thesis, University of Glasgow.

Römer, K., Blum, P., & Meier, L. 2005. *Handbook of Sensor Networks: Algorithms and Architectures.* John Wiley & Sons. Chap. Time synchronization and calibration in wireless sensor networks, pages 199–237.

Römer, Kay. 2004. Programming Paradigms and Middleware for Sensor Networks. *Pages 49–54 of: GI/ITG Workshop on Sensor Networks.*

Römer, Kay, Kasten, Oliver, & Mattern, Friedemann. 2002. Middleware Challenges for Wireless Sensor Networks. *Mobile Computing and Communications Review*, **6**(4), 59–61.

Schwarz, N. 2007. *The Science of Real-time Data Capture: Self-reports in Health Research.* Oxford University Press, Inc. Chap. Retrospective and concurrent self-reports: The rationale for real-time data capture, pages 11–26.

Sen, S., & Cardell-Oliver, R. 2006. A rule-based language for programming wireless sensor actuator networks using frequence and communication. *Proceedings of Third Workshop on Embedded Networked Sensors (EMNETS) 2006*, **1**, To Appear.

Sixsmith, A., Hine, N., Neild, I., Clarke, N., Brown, S., & Garner, P. 2007. Monitoring the Well-being of Older People. *Topics in Geriatric Rehabilitation*, **23**(1), 9.

Smyth, J.M., & Stone, A.A. 2003. Ecological momentary assessment research in behavioral medicine. *Journal of Happiness Studies*, **4**(1), 35–52.

Society, The Royal. 2006. *Digital healthcare: The impact of information and communication technologies on health and healthcare.* Tech. rept. Royal Society.

Stone, A.A., Shiffman, S., Atienza, A.A., & Nebeling, L. 2007. *The Science of Real-time Data Capture: Self-reports in Health Research.* Oxford University Press, Inc. Chap. Historical roots and rationale of ecological momentary assessment (EMA), pages 3–10.

Sugihara, Ryo, & Gupta, Rajesh K. 2008. Programming models for sensor networks: A survey. *ACM Trans. Sen. Netw.*, **4**(2), 1–29.

Terfloth, Kirsten, Wittenburg, Georg, & Schiller, Jochen. 2006 (Jan.). FACTS - A Rule-Based Middleware Architecture for Wireless Sensor Networks. *In: Proceedings of the First International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE '06).*

Tsang, S., Magill, E.H., & Kelly, B. 1997. The feature interaction problem in networked multimedia services – present and future. *BT Technology Journal*, **15**(1), 235–246.

Turner, K. 1993. An engineering approach to formal methods. *Proc. Protocol Specification, Testing and Verification XIII, pages.*, **1**, 357–380.

Turner, K. J., & Campbell, G. A. 2009. Goals for Telecare Networks. *Pages 270–275 of:* Obaid, Abdel (ed), *Proc. 9th Int. Conf. on New Technologies for Distributed Systems.*

Van Greunen, J., & Rabaey, J. 2003. Lightweight time synchronization for sensor networks. *Pages 11–19 of: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications.* ACM.

Wang, F., & Turner, K.J. 2008. Policy conflicts in home care systems. *Proc. 9th. Int. Conf. on Feature Interactions in Software and Communication Systems*, **1**.

Weiss, H.M., Beal, D.J., Lucy, S.L., & MacDermid, S.M. 2004. *Constructing EMA studies with PMAT: The purdue momentary assessment tool user's manual.* Tech. rept. Purdue University.

Wilk, J. 2004. *Dynamic Workflow Pulling The Strings.* Ph.D. thesis, Imperial College London.

Wilson, M., Kolberg, M., & Magill, E.H. 2007. Considering side effects in service interactions in home automation-an online approach. *Feature Interactions in Software and Communication Systems*, **9**, 172–187.

Wilson, M. E. J. 2005 (September). *An Online Environmental Approach to Service Interaction Management in Home Automation.* Ph.D. thesis, University of Stirling.

Wood, A., Virone, G., Doan, T., Cao, Q., Selavo, L., Wu, Y., Fang, L., He, Z., Lin, S., & Stankovic, J. 2006. *ALARM-NET: Wireless Sensor Networks for Assisted-Living and Residential Monitoring.* Technical Report CS–2006–11. University of Virginia Computer Science Department.

Wooldridge, M. 1997. Agent-based software engineering. *Pages 26–37 of: Software Engineering. IEE Proceedings-[see also Software, IEE Proceedings]*, vol. 144. IET.

Yatham, L.N., Lecrubier, Y., Fieve, R.R., Davis, K.H., Harris, S.D., & Krishnan, A.A. 2004. Quality of life in patients with bipolar I depression: data from 920 patients. *Bipolar disorders*, **6**(5), 379–385.

Yick, J., Mukherjee, B., & Ghosal, D. 2008. Wireless sensor network survey. *Computer Networks*, **52**(12), 2292–2330.

Yu, Yang, Krishnamachari, Bhaskar, & Prasanna, Viktor K. 2004. Issues in Designing Middleware for Wireless Sensor Networks. *IEEE Network*, **18**, 15–21.

Zhao, F., & Guibas, L.J. 2004. *Wireless sensor networks: an information processing approach.* Morgan Kaufmann Pub.

Appendix A

Analysis Source Code Listings

The following presents the source code listings for the CoLlaborative Information Processing Protocol and Extended Runtime (CLIPPER) core analysis files described in section 5.4.

Listing A.1: The detection.pl predicates.

```
1  % ***** Routines for the analysing rule interaction *****
2
3  % Initialises the world state
4  initialise(T1, T2, Some_message) :-
5          initEC,
6
7          %Timepoints
8          T0 is 0,
9          T1 is T0+1,
10         T2 is T1+1,
11         assert(timepoint(T0)),
12         assert(timepoint(T1)),
13         assert(timepoint(T2)),
14
15         %Action and Message
16         Some_message = true,
17         assert(initially_p(message(Some_message))).
18
19 % Main entry point for testing for interactions
20 analyseConflicts(Type) :-
21         assert(fiType(Type)),
22         write('****** Checking for conflict in rules.pl file using conflict rule: '), write(Type), writeln(' ******')
               ,
23         getList(Rules),
24         not(singleConflictCheckList(Rules)),!,
25         writeln('****** Completed conflict search. ******').
26
27 % Checks whether or not the rules E1 and E2 conflict or concord
28 singleCheck(E1, E2) :-
29         initialise(T1, T2, M),
30         fiType(Type), !,
31
32         % Pass the message and the time values to the rules
33         %MTI
34         (
35                 (
36                         Type=mti,
37                         aRule(E1,M,T1),
38                         aRule(E2,M,T2)
39                 );
```

```
40          %STI or MAI
41                  (
42                          (
43                                  Type=sti;
44                                  Type=mai
45                          ),
46                          aRule(E1,M,T1),
47                          aRule(E2,M,T1)
48                  );
49          %SAI or LI
50                  (
51                          Type=sai,
52                          aRule(E1,M,T2),
53                          aRule(E2,M,T1)
54                  )
55          ),
56
57          %Get rule names for documentation and write them to the output
58          current_predicate(X,E1), current_predicate(Y,E2),
59          write('Does Rule1 ('), write(X),write(') concord with '), write('Rule2 ('), write(Y),write(')?\n'), !,
60
61          % Check for conflicts
62          ( % ??MAI, LI??
63                  (
64                          Type=mti,
65                          mti(E1, E2, M, T1, T2)
66                  );
67                  (
68                          (
69                                  Type=sti;
70                                  Type=mai
71                          ),
72                          sti(E1, E2, M, T1, T2) % can be used for MAI as well
73                  );
74                  (
75                          Type=sai,
76                          sai(E1, E2, M, T1, T2)
77                  )
78          ).
```

Listing A.2: The utility.pl predicates.

```
1  %***** Utitlity Predicates *****
2
3  % Delete item X from a list.
4  del(X,[X|T],T).
5  del(X,[Y|T],[Y|T1]) :-
6          del(X,T,T1).
7
8  % X1 is a member of a list of rules with arity 2. M is the first arg.
9  aRule(X1,List,M,T1) :-
10         member(X1,List),
11         aRule(X1,M,T1).
12
13 % X1 is a predicate with arity 2. M is the first arg and T1 is the second arg.
14 aRule(X1,M,T1) :-
15         arg(1,X1,M),
16         arg(2,X1,T1).
17
18 % Gets a sorted list of rules
```

```
19 getList(L) :-
20         getRules(L1),
21         msort(L1,L).
22
23 % Permutates through a list checking each item against itself and the remainder of the items
24 singleConflictCheckList(L1) :-
25         getList(L2),
26         member(X,L1),
27         singleConflictCheckList(X,L2),
28         del(X,L1,L3),
29         compare(L3).
30
31 % Compares an item to another list of items
32 singleConflictCheckList(Item,List) :-
33         member(X,List),
34         singleCheck(Item,X),
35         del(X,List,L2),!,
36         singleConflictCheckList(Item, L2).
37
38 getConflictRules(Rules) :-
39         current_predicate(no_msg_mod_conflict,X),
40         %Rules = [X].
41         current_predicate(always_conflict,Y),
42         current_predicate(never_conflict,Z),
43         Rules = [X, Y, Z].
44
45 % Gets a sorted list of conflict rules
46 getCList(L) :-
47         getConflictRules(L1),
48         msort(L1,L).
49
50 % C1 is a predicate with arity 5. The rest are the argument values
51 init_conflict_rule(C1, E1, E2, M, T1, T2, T3) :-
52         arg(1,C1,E1),
53         arg(2,C1,E2),
54         arg(3,C1,M),
55         arg(4,C1,T1),
56         arg(5,C1,T2),
57         arg(6,C1,T3).
58
59 % ***** Routines for writing the output file *****
60
61 % Initialise the XML file
62 initOutputFile(ID) :-
63         path(X),
64         open(X, write, Stream),
65         write(Stream, 'detectionReport('),
66         write(Stream, ID),
67         write(Stream, ').'),
68         nl(Stream),
69         close(Stream),!.
70
71 closeOutputFile(Result) :-
72         path(X),
73         open(X, append, Stream),
74         write(Stream, 'completedDetectionSearch('),
75         write(Stream, Result),
76         write(Stream, ').'),
77         close(Stream),!.
78
```

```
% Result is expected to either be fi or concord
writeResult(Result, TestType, R1, R2) :-
        path(X),
        open(X, append, Stream),
        write(Stream, Result),
        write(Stream, '(type('),
        write(Stream, TestType),
        write(Stream, '),'),
        writeRule(R1,Stream),
        write(Stream,','),
        writeRule(R2,Stream),
        write(Stream,').'),
        nl(Stream),
        close(Stream),!.

writeRule(X, Stream) :-
        write(Stream, 'rule('),
        write(Stream, X),
        write(Stream,')').

writeResults(Id) :-
        initOutputFile(Id),
        ( (
                        detection_result(fi,B,C,D),
                        writeResult(fi,B,C,D),fail
                );true
        ),
        ( (
                        detection_result(concord,B,C,D),
                        writeResult(concord,B,C,D),fail
                );true
        ),
        closeOutputFile(success).

addFi(Type, R1, R2) :-
        assert(detection_result(fi, Type, R1, R2)).

addConcord(Type, R1, R2) :-
        assert(detection_result(concord, Type, R1, R2)).
```

Listing A.3: The eventcalc.pl predicates.

```
% ***** Predicates of the Event Calculus *****

%A fluent holds at some time t if it was initially true and has not been
holdsAt(F, T) :-
        initially_p(F),
        \+ clipped(0, F, T).

% A fluent holds at some time t2 if an action happens before t3 which initiates
% the fluent and the fluent is not terminated during the action (clipped).
holdsAt(F, T2) :-
        happens(A, T1),
        initiates(A, F, T1),
        before(T1,T2),
        \+ clipped(T1, F, T2).

% A fluent does not hold at some time t if it initially did not hold and was
% not initiated (declipped)
notHoldsAt(F,T) :-
```

```
19          initially_n(F),
20          \+ declipped(0,F,T).
21
22  % A fluent does not hold at some time T if an action happens before T which
23  % terminates the fluent and the fluent is not initiated during the action.
24  notHoldsAt(F,T) :-
25          happens(A,T1),
26          terminates(A,F,T1),
27          before(T2,T),
28          \+ declipped(T1,F,T2).
29
30  %***Clipped(t1,ß,t2) Fluent ß is terminated between t1 and t2
31  clipped(T1,F,T2) :-
32          happens(A,T),
33          before(T1,T),
34          before(T,T2),
35          terminates(A,F,T).
36
37  %***Declipped(t1,ß,t2) Fluent ß is initiated between t1 and t2
38  declipped(T1,F,T2) :-
39          happens(A,T),
40          before(T1,T),
41          before(T,T2),
42          initiates(A,F,T).
43
44  %***Time 1 is before Time 2
45  before(T1,T2) :-
46          timepoint(T1),
47          timepoint(T2),
48          T1 < T2.
49
50  %*** Initialise scenario variables
51  initEC :-
52          retractall(initiates(_,_,_)),!,
53          retractall(terminates(_,_,_)),!,
54          retractall(initially_p(_)),!,
55          retractall(initially_n(_)),!,
56          retractall(timepoint(_)),!,
57          retractall(happens(_,_)),!,
58
59          assert(initiates(null,null,null)),
60          assert(terminates(null,null,null)),
61          assert(initially_p(null)),
62          assert(initially_n(null)),
63          assert(timepoint(0)),
64          assert(initihappens(null,null)).
```

Listing A.4: The rules.pl predicates.

```
1   % ***** Setup rules to use during analysis *****
2
3   :- consult(csip_rules).
4
5   getRules(Rules) :-
6
7   %To run rules
8           %current_predicate(dam_adt,A),
9           %current_predicate(dam_rds,B),
10
11          Rules = [A,B].
```

```
12
13 % consult all the components
14 doConsults :-
15         consult('C:/jmb/thesis/classic_thesis_lyx_jmb/sourceCode/collaborationErrorDescription/
               csiplFeatures2/eventCalc'),
16         consult('C:/jmb/thesis/classic_thesis_lyx_jmb/sourceCode/collaborationErrorDescription/
               csiplFeatures2/conflict_rules'),
17         consult('C:/jmb/thesis/classic_thesis_lyx_jmb/sourceCode/collaborationErrorDescription/
               csiplFeatures2/utility'),
18         consult('C:/jmb/thesis/classic_thesis_lyx_jmb/sourceCode/collaborationErrorDescription/
               csiplFeatures2/detection'),
19         consult('C:/jmb/thesis/classic_thesis_lyx_jmb/sourceCode/collaborationErrorDescription/
               csiplFeatures2/outputFileWriter').
20
21 path(X) :-
22         X='C:/detectionReport.pl'.
23
24 :- doConsults.
```

Listing A.5: The conflictRules.pl predicates.

```
1  % ***** Examination of conflict using event calculus *****
2
3  % Are Rule1 and Rule2 MTI free using a Message at times T1 and T2?
4  % Rules A & B conflict if the first one sets a message into a state such that
5  % the second one does not operate correctly.
6  mti(Rule1,Rule2, Message, T1,T2) :-
7          % Make sure that T1 < T2
8          (before(T1,T2);write(' Received timepoints are out of order. Conflict check cannot complete.\n'), !,
               fail ),
9          % Check to make sure that the message holds at the time the first rule receives the message
10         (
11                 holdsAt(message(Message),T1);
12                 write(' Message ('), write(Message), write(') does not hold at time: '), write(T1), write('\n'),
                       !, fail
13         ),
14         % Perform the first rule
15         (
16                 Rule1;
17                 %
18                 writeln(' Rule 1 failed.\n No.'), !, fail
19         ),
20         % Check to make sure that the message holds at the time the first rule completed
21         % If the message does not hold there has been a conflict.
22         (
23                 holdsAt(message(Message),T2);
24                 addFi(mti, Rule1, Rule2),
25                 write(' Message ('), write(Message), write(') does not hold at time: '), write(T2), write('\n
                       No.\n'), !, fail
26         ),
27         % Perform the second rule or report failure.
28         (Rule2; writeln(' Rule 2 failed.\n No.'), !, fail),
29
30         % If no conflict then we expect that the message was triggered.
31         % If all is well then notify the user that the rules are in concordance
32         %(if not then the notification was written out in mti)
33         %(
34                 addConcord(mti, Rule1, Rule2),
35                 write(' Yes.\n').%holdsAt(trigger(Message),T3), write(' Yes.\n'),!;
36                 %write(' Message('), write(Message), write(') not triggered.\n No\n'), !, fail
```

```
37          %).
38
39  % Are Rule1 and Rule2 STI free using a Message at time T1 (T2 is unused)?
40  % Rules A & B conflict by STI when actions are performed by them both in response to the same
41  % triggering event, and the list of actions is different from how it would be if
42  % only one feature had responded to the trigger.
43  % Algorithm for checking for STI:
44  %  Perform first rule
45  %   A = does action get performed
46  %  Reset world
47  %  Perform second rule
48  %  Perform first rule
49  %   B = does action get performed
50  %  Rules concord if A == B
51  sti(Rule1,Rule2, _, _, _) :-
52          % Perform the first rule
53          Rule1,
54          domainDependentSentences(F1),
55          % Reset world
56          resetWorld,
57          % Perform the second rule
58          Rule2,
59          domainDependentSentences(F2),
60          % Perform the first rule
61          Rule1,
62          domainDependentSentences(F3),
63          subtract(F3,F2, Difference1),
64          subtract(F1,Difference1, Difference2),
65          length(Difference2,Len),
66          write('F1: '), writeln(F1),
67          write('F2: '), writeln(F2),
68          write('F3: '), writeln(F3),
69          write('sub(F3,F2): '), writeln(Difference1),
70          write('sub(F1,Dif): '), writeln(Difference2),
71          write('Len: '), writeln(Len),
72          %Write result of check of rule conccordance
73          (
74                  (
75                          Len=0,
76                          addConcord(sti, Rule1, Rule2),
77                          write(' Yes.\n')
78                  );
79                  (
80                          addFi(sti, Rule1, Rule2), write(' No.\n')
81                  )
82          ).
83
84  domainDependentSentences(Res) :-
85          findall([B,C],initiates(_,B,C),Find1),
86          findall([B,C],terminates(_,B,C),Find2),
87          append(Find1, Find2, Find3),
88          subtract(Find3,[[null, null]], Res).
89
90  actionSentences(Find1) :-
91          findall([B,C],happens(B,C),Find1).
92
93  resetWorld :-
94          initEC,
95          T0 is 0,
96          T1 is T0+1,
```

```
97          T2 is T1+1,
98          assert(timepoint(T0)),
99          assert(timepoint(T1)),
100         assert(timepoint(T2)),
101         assert(initially_p(message(true))).
102
103 % Are Rule1 and Rule2 SAI free using a Message at times T1 and T2?
104 % Rules A & B conflict if the first one sets a message into a state such that
105 % the second one is caused to operate.
106 sai(Rule1,Rule2, _, T1,T2) :-
107         % Make sure that T1 < T2
108         (before(T1,T2);write(' Received timepoints are out of order. Conflict check cannot complete.\n'), !,
                fail ),
109         % Perform the first rule
110         Rule1,
111         actionSentences(F1),
112         % Reset world
113         resetWorld,
114         % Perform the second rule
115         Rule2,
116         actionSentences(F2),
117
118         Rule1,
119         actionSentences(F3),
120         subtract(F3,F2, Difference1),
121
122         %write('F1: '), writeln(F1),
123         %write('F2: '), writeln(F2),
124         %write('F3: '), writeln(F3),
125         %write('Difference1 = sub(F3,F2): '), writeln(Difference1),
126         %writeln('Does F1 == Difference1?'),
127
128         (  (F1\=Difference1,
129                         addFi(sai, Rule1, Rule2), write(' No.\n'));
130                         (
131                                 addConcord(sai, Rule1, Rule2),
132                                 write(' Yes.\n')
133                         ) ).
```

Appendix B

Test Feature Rules

The following presents the code listings for the facts and rules tested in this thesis.

Listing B.1: Priority rules used in the examples.

```
% Device priorities. The lower the number the higher the priority (our first priority is...)

% Priority/3 (X,Y,Z) Device X has priority Y at level Z.
priority(wearable,data_transfer,1).

priority(mob_phone,security,1).
priority(mob_phone,data_integrity,2).
priority(mob_phone,battery_life,3).
priority(mob_phone,data_collection,4).
priority(mob_phone,user_interaction,5).

priority(gateway,security,1).
priority(gateway,data_integrity,2).

% Priority/4 (A,B,C,D) For device A, the priority B has rule C at priority level D.
priority(wearable,data_integrity,dam_adt,1).
priority(mob_phone,data_integrity,dem_ts,1).
priority(mob_phone,data_integrity,dem_drf,2).
priority(mob_phone,data_integrity,dam_dsu,3).
priority(mob_phone,data_integrity,dem_ai,4).
priority(mob_phone,data_integrity,dam_adt,5).
priority(mob_phone,data_integrity,dam_rds,6).
priority(mob_phone,data_integrity,dam_rdtou,7).
priority(mob_phone,data_integrity,dam_dstp,8).
priority(mob_phone,battery_life,dam_dstp,1).
priority(mob_phone,battery_life,dem_drf,2).
priority(mob_phone,security,dam_ods_B,1).
priority(mob_phone,security,dam_ids,2).
priority(mob_phone,data_collection,cds_cts,1).
priority(mob_phone,data_collection,sds_sts_inert,2).
priority(mob_phone,data_collection,sds_sts_active,3).
priority(mob_phone,data_collection,cds_rl,4).
priority(mob_phone,user_interaction,dnd_dnnu,2).
priority(mob_phone,user_interaction,si_p,1).
```

Listing B.2: Feature rules used in STI example 1.

```
% respond to changes upon receiving contextual information
cds_cts(Trigger,T) :-
        T2 is T+1,
        assert(happens(listen_for_connection,T)),
```

```
5           assert(happens(make_connection,T)),
6           assert(happens(receive_data,T)),
7           assert(happens(checks_data,T)),
8           assert(happens(listen_for_connection,T2)),
9           ((
10                  holdsAt(message(Trigger), T2),
11                  assert(initiates(checks_data,prompt(Trigger),T)),
12                  assert(terminates(checks_data,message(Trigger),T))
13          );
14          assert(terminates(checks_data,message(Trigger),T))).
15
16  % Form A: responds to changes upon receiving state information
17  sds_sts_active(Trigger,T) :-
18          cds_cts(Trigger,T).
19
20  % Form B: doesn't respond to changes upon receiving state information
21  sds_sts_inert(_,T) :-
22          T2 is T+1,
23          assert(happens(listen_for_connection,T)),
24          assert(happens(make_connection,T)),
25          assert(happens(receive_data,T)),
26          assert(happens(check_data,T)),
27          assert(happens(listen_for_connection,T2)).
```

Listing B.3: Feature rules used in SAI example 1.

```
1   % Display a prompt when triggered
2   si_p(Trigger,T) :-
3           assert(happens(listen_for_connection,T)),
4           (
5                   (
6                           holdsAt(message(Trigger), T),
7                           assert(happens(make_connection,T)),
8                           assert(happens(receive_data,T)),
9                           assert(initiates(receive_data,prompt,T)),
10                          assert(happens(formulate_quesition,T)),
11                          assert(happens(determine_response_options,T)),
12                          assert(happens(determine_display_options,T)),
13                          assert(happens(display_prompt,T))
14                  );
15                  (
16                          assert(happens(connection_failure,T)),
17                          assert(initiates(connection_failure,fail_connection,T))
18                  )
19          ),
20          assert(terminates(display_prompt,message(Trigger),T)).
21
22  % sends data about the location of the subject
23  cds_rl(Trigger,T) :-
24          assert(happens(attempt_connection,T)),
25          (
26                  (
27                          holdsAt(message(Trigger), T),
28                          assert(happens(make_connection,T)),
29                          assert(happens(report_location,T)),
30                          assert(initiates(report_location,complete_transfer,T))
31                  );
32                  (
33                          assert(happens(connection_failure,T)),
34                          assert(initiates(connection_failure,fail_transfer,T))
```

```
35                  )
36          ).
```

Listing B.4: Feature rules used in SAI example 2.

```
1  dam_adt(Trigger,T) :-
2          T2 is T+1,
3          T3 is T+2,
4          T4 is T+3,
5          assert(timepoint(T2)),
6          assert(timepoint(T3)),
7          assert(timepoint(T4)),
8          assert(happens(attempt_connection,T)),
9          (
10                 (
11                         holdsAt(disconnect, T3)
12                 );
13                 (
14                         (
15                                 holdsAt(message(Trigger), T2),
16                                 assert(happens(make_connection,T2)),
17                                 assert(initiates(make_connection,connected,T2)),
18                                 assert(happens(upload_data,T3)),
19                                 assert(initiates(upload_data,data,T3)),
20                                 assert(terminates(make_connection,message(Trigger),T2))
21                         );
22                         (
23                                 assert(happens(connection_fail,T3)),
24                                 assert(initiates(connection_fail,dam_adt(Trigger,T2),T2))
25                         )
26                 )
27         ).
28
29
30         % Receiver of new incoming data transfers the raw data to another device
31         dam_rds(Trigger,T) :-
32                 T1 is T+1,
33                 T2 is T+2,
34                 T3 is T+3,
35                 T4 is T+4,
36                 T5 is T+5,
37                 T6 is T+6,
38                 assert(timepoint(T1)),
39                 assert(timepoint(T2)),
40                 assert(timepoint(T3)),
41                 assert(timepoint(T4)),
42                 assert(timepoint(T5)),
43                 assert(timepoint(T6)),
44                 assert(happens(listening,T)),
45                 (
46                         (
47                                 holdsAt(connected,T1),
48                                 assert(happens(download_data,T2)),
49                                 assert(terminates(download_data,message(Trigger), T2)),
50                                 assert(happens(attempt_connection,T4)),
51                                 assert(happens(upload_data,T5)),
52                                 assert(happens(complete_transfer,T6))
53                         );
54                         (
55                                 assert(happens(close_connection,T2)),
```

```
56                              assert(terminates(close_connection,message(Trigger), T2)),
57                              assert(initiates(close_connection,disconnect,T2))
58                      )
59              ).
```

Listing B.5: Feature rules used in SAI example 3 and LI example 1.

```
1  % Form A: responds to changes upon receiving state information
2  sds_sts_active(Trigger,T) :-
3          T2 is T+1,
4                  assert(happens(listen_for_connection,T)),
5                  assert(happens(make_connection,T)),
6                  assert(happens(receive_data,T)),
7                  assert(happens(checks_data,T)),
8                  assert(happens(listen_for_connection,T2)),
9                  ((
10                         holdsAt(message(Trigger), T2),
11                         assert(initiates(check_data,prompt(Trigger),T)),
12                         assert(terminates(check_data,message(Trigger),T))
13                 );
14         assert(terminates(checks_data,message(Trigger),T))).
15
16 % Form B: doesn't respond to changes upon receiving state information
17 sds_sts_inert(_,T) :-
18         T2 is T+1,
19         assert(happens(listen_for_connection,T)),
20         assert(happens(make_connection,T)),
21         assert(happens(receive_data,T)),
22         assert(happens(check_data,T)),
23         assert(happens(listen_for_connection,T2)).
24
25 % Data Recording Frequency
26 dem_drf(Trigger,T) :-
27         T2 is T+1,
28         assert(happens(listen_for_connection,T)),
29         assert(happens(make_connection,T)),
30         assert(happens(receive_data,T)),
31         assert(happens(calculate_frequency,T)),
32         assert(terminates(receive_data,message(Trigger),T)),
33         assert(happens(listen_for_connection,T2)).
```

Listing B.6: Feature rules used in LI example 2.

```
1  % Retry Data Transfer On Unavailable
2  rdtou(Trigger,T1) :-
3          T2 is T1+1,
4          assert(timepoint(T2)),
5          assert(happens(attempt_connect,T1)),
6          (
7                  holdsAt(message(Trigger),T1),
8                  assert(happens(connect,T1)),
9                  assert(happens(upload_data,T1)),
10                 assert(happens(transfer_complete,T1)),
11                 assert(terminates(connect,message(Trigger),T1))
12         );
13         (
14                 assert(happens(connect_fail,T2)),
15                 (
16                         assert(happens(rdtou(Trigger,T2),T2))
17                 )
```

```
18          ).
19
20  % Inbound Data Screening
21  ids(Data,T1) :-
22          assert(happens(sender_attempts_connection,T1)),
23          assert(happens(recipient_screens_sender,T1)),
24          assert(terminates(recipient_screens_sender,message(Data),T1)),
25          assert(initiates(recipient_screens_sender,sender_connection_failure_date,T1)).
```

Listing B.7: Feature rules used in MAI example 1.

```
1   % The receiver of a incoming data stores them.
2   dam_dsu(Trigger,T) :-
3           T2 is T+1,
4           assert(happens(listen_for_connection,T)),
5           assert(happens(make_connection,T)),
6           assert(happens(receive_data,T)),
7           assert(happens(listen_for_connection,T2)),
8           ((
9                   holdsAt(message(Trigger), T2),
10                  assert(initiates(receive_data,store(Trigger),T)),
11                  assert(terminates(receive_data,message(Trigger),T))
12          );
13          assert(terminates(checks_data,message(Trigger),T))).
14
15  % the receiver of incoming data processes them then stores them
16  dam_dstp(Trigger,T) :-
17          T2 is T+1,
18          assert(happens(listen_for_connection,T)),
19          assert(happens(make_connection,T)),
20          assert(happens(receive_data,T)),
21          assert(happens(process_data,T)),
22          assert(happens(listen_for_connection,T2)),
23          ((
24                  holdsAt(message(Trigger), T2),
25                  assert(initiates(process_data,prompt(Trigger),T)),
26                  assert(terminates(process_data,message(Trigger),T))
27          );
28          assert(terminates(process_data,message(Trigger),T))).
```

Listing B.8: Feature rules used in MAI example 2.

```
1   dam_adt(Trigger,T) :-
2           T2 is T+1,
3           T3 is T+2,
4           T4 is T+3,
5           assert(timepoint(T2)),
6           assert(timepoint(T3)),
7           assert(timepoint(T4)),
8           assert(happens(attempt_connection,T)),
9           (
10                  (
11                          holdsAt(disconnect, T3)
12                  );
13                  (
14                          (
15                                  holdsAt(message(Trigger), T2),
16                                  assert(happens(make_connection,T2)),
17                                  assert(initiates(make_connection,connected,T2)),
18                                  assert(happens(upload_data,T3)),
```

```
19                                      assert(initiates(upload_data,data,T3)),
20                                      assert(terminates(make_connection,message(Trigger),T2))
21                              );
22                              (
23                                      assert(happens(connection_fail,T3)),
24                                      assert(initiates(connection_fail,dam_adt(Trigger,T2),T2))
25                              )
26                      )
27              ).
28
29      % outbound screening
30      dam_ods(Trigger,T) :-
31              assert(happens(attempt_connection,T)),
32              assert(happens(screen_recipient,T)),
33              (
34                      (
35                              holdsAt(recipient, T),
36                              assert(terminates(screen_recipient,message(Trigger),T)),
37                              assert(initiates(screen_recipient,deny,T))
38                              %writeln('ods:recipient_holds')
39                      );
40                      (
41                              assert(initiates(screen_recipient,proceed,T))
42                              %writeln('ods:recipient_not_holds')
43                      )
44              ).
45
46      dam_ods_screened(Trigger,T) :-
47              assert(initially_p(recipient)),
48              dam_ods(Trigger,T).
49
50      dam_ods_not_screened(Trigger,T) :-
51              assert(initially_n(recipient)),
52              dam_ods(Trigger,T).
```

Listing B.9: Feature rules used in MAI example 3.

```
1       % Suppress prompts when triggered
2       dnd_dnnu(Trigger,T) :-
3               (
4                       (
5                               holdsAt(message(Trigger), T),
6                               assert(initially_p(block))
7                       );
8                       (
9                               assert(initially_n(block))
10                      )
11              ),
12              assert(happens(attempt_connection,T)),
13              assert(happens(block_recipient,T)),
14              (
15                      (
16                              holdsAt(block, T),
17                              assert(terminates(block_recipient,message(Trigger),T)),
18                              assert(initiates(block_recipient,deny,T))
19                      );
20                      (
21                              assert(initiates(block_recipient,proceed,T))
22                      )
23              ).
```

```
24
25 % Time Synchronisation
26 dem_ts(Trigger,T1) :-
27
28         T2 is T1+1,
29         T3 is T1+2,
30         T4 is T1+3,
31         T5 is T1+4,
32
33         assert(timepoint(T2)),
34         assert(timepoint(T3)),
35         assert(timepoint(T4)),
36         assert(timepoint(T5)),
37
38         assert(initially_p(t_offset)),
39         assert(initially_n(clock_dif)),
40
41         assert(happens(listening,T1)),
42         (
43                 (
44                         holdsAt(message(Trigger), T2),  % whether or not the connection occurs
45                         assert(happens(record(Trigger, TS1),T2)),
46                         assert(happens(calculate(t_offset, TS1, TS2),T3)),
47                         assert(happens(upload(TS1, TS2, T3),T4)),
48                         assert(happens(receive(clock_dif),T5)),
49                         assert(initiates(receive_clock_dif,clock_dif,T5)),
50                         assert(terminates(record(Trigger, TS1),message(Trigger),T3))
51                 );
52                         assert(initiates(notholdsAt(message(Trigger), T2),blocked,T4))
53         ).
```

Listing B.10: Feature rules used in MTI example 1.

```
1 % Display a prompt when triggered
2 si_p(Trigger,T) :-
3         assert(happens(listen_for_connection,T)),
4         (
5                 (
6                         holdsAt(message(Trigger), T),
7                         assert(happens(make_connection,T)),
8                         assert(happens(receive_data,T)),
9                         assert(initiates(receive_data,prompt,T)),
10                        assert(happens(formulate_quesition,T)),
11                        assert(happens(determine_response_options,T)),
12                        assert(happens(determine_display_options,T)),
13                        assert(happens(display_prompt,T))
14                );
15                (
16                        assert(happens(connection_failure,T)),
17                        assert(initiates(connection_failure,fail_connection,T))
18                )
19        ),
20        assert(terminates(display_prompt,message(Trigger),T)).
21
22 % Suppress prompts when triggered
23 dnd_dnnu(Trigger,T) :-
24        (
25                (
26                        holdsAt(message(Trigger), T),
27                        assert(initially_p(block))
```

```
28                  );
29                  (
30                          assert(initially_n(block))
31                  )
32          ),
33          assert(happens(attempt_connection,T)),
34          assert(happens(block_recipient,T)),
35          (
36                  (
37                          holdsAt(block, T),
38                          assert(terminates(block_recipient,message(Trigger),T)),
39                          assert(initiates(block_recipient,deny,T))
40                  );
41                  (
42                          assert(initiates(block_recipient,proceed,T))
43                  )
44          ).
```

Listing B.11: Feature rules used in MTI example 2.

```
1  % respond to changes upon receiving contextual information
2  cds_cts(Trigger,T) :-
3          T2 is T+1,
4          assert(happens(listen_for_connection,T)),
5          assert(happens(make_connection,T)),
6          assert(happens(receive_data,T)),
7          assert(happens(checks_data,T)),
8          assert(happens(listen_for_connection,T2)),
9          ((
10                 holdsAt(message(Trigger), T2),
11                 assert(initiates(checks_data,prompt(Trigger),T)),
12                 assert(terminates(checks_data,message(Trigger),T))
13         );
14         assert(terminates(checks_data,message(Trigger),T))).
15
16 % Form A: responds to changes upon receiving state information
17 sds_sts_active(Trigger,T) :-
18         cds_cts(Trigger,T).
19
20 % Form B: doesn't respond to changes upon receiving state information
21 sds_sts_inert(_,T) :-
22         T2 is T+1,
23         assert(happens(listen_for_connection,T)),
24         assert(happens(make_connection,T)),
25         assert(happens(receive_data,T)),
26         assert(happens(check_data,T)),
27         assert(happens(listen_for_connection,T2)).
28
29 si_p(Trigger,T) :-
30         assert(happens(listen_for_connection,T)),
31         (
32                 (
33                         holdsAt(message(Trigger), T),
34                         assert(happens(make_connection,T)),
35                         assert(happens(receive_data,T)),
36                         assert(initiates(receive_data,prompt,T)),
37                         assert(happens(formulate_quesition,T)),
38                         assert(happens(determine_response_options,T)),
39                         assert(happens(determine_display_options,T)),
40                         assert(happens(display_prompt,T))
```

```
41                  );
42                  (
43                          assert(happens(connection_failure,T)),
44                          assert(initiates(connection_failure,fail_connection,T))
45                  )
46          ),
47          assert(terminates(display_prompt,message(Trigger),T)).
```

Listing B.12: Feature rules used in MTI example 3.

```
1  % respond to changes upon receiving contextual information
2  cds_cts(Trigger,T) :-
3          T2 is T+1,
4          assert(happens(listen_for_connection,T)),
5          assert(happens(make_connection,T)),
6          assert(happens(receive_data,T)),
7          assert(happens(checks_data,T)),
8          assert(happens(listen_for_connection,T2)),
9          ((
10                 holdsAt(message(Trigger), T2),
11                 assert(initiates(checks_data,prompt(Trigger),T)),
12                 assert(terminates(checks_data,message(Trigger),T))
13         );
14         assert(terminates(checks_data,message(Trigger),T))).
15
16 % Activate immediate
17 dem_ai(Trigger,T) :-
18         assert(happens(listen_for_connection,T)),
19         (
20                 (
21                         holdsAt(message(Trigger), T),
22                         assert(happens(make_connection,T)),
23                         assert(happens(receive_activate_command,T)),
24                         assert(initiates(receive_activate_command,perform_activity,T))
25                 );
26                 true
27         ).
```

Listing B.13: Rules used to describe how state may affect CLIPPER.

```
1  destro(Trigger,T) :-      % Pure destructive rule
2          assert(happens(make_connection,T)),
3          assert(terminates(make_connection,message(Trigger),T)).
4
5  state_destro(Trigger,T) :-   % State-preserving destructive rule
6          (
7                  holdsAt(perform_activity, 2),
8                  assert(terminates(make_connection,message(Trigger),T))
9          );
10         (
11                 assert(happens(make_connection,1)),
12                 assert(initiates(make_connection,perform_activity,1)),
13                 state_destro(Trigger,T)
14         ).
15
16 % Macro rule that calls multiple rules before clipper is reset
17 macroA(Trigger,T) :-
18                 passive(Trigger,T),
19                 destro(Trigger,T).
20
```

```
21 % Macro rule that calls multiple rules before clipper is reset
22 macroB(Trigger,T) :-
23                     destro(Trigger,T),
24                     passive(Trigger,T).
25
26
27 passive(_,_) :-        % Pure passive rule
28           assert(terminates(make_connection,perform_activity,1)),
29           true.
```