# Towards a UbiSafe Environment Using Rules

**Tope Omitola**, University of Cambridge, Computer Laboratory, too20@cam.ac.uk

## Abstract

We describe work towards achieving a reliable and safe ubiquitous computing environment through the control and management of the (feature) interactions of the elements of such environment using rules and an accompanying rule based control system.

## 1 Introduction

Ubiquitous computing depicts a world where several electronic objects are embedded everywhere in the environment and made available to people using a wide variety of user interfaces. Some of the properties of these environments are:

- many of the devices will be of limited storage, processing, display and battery power capabilities,
- the computing environment will be open [1], i.e. there will be a continual entry, exit, and re-entry of devices and applications from disparate sources,
- the resources will vary in their availability, their quality, and their reliability,
- the devices and resources will be aware of their context, including their location, and their relationships with the wider world,
- applications will be built by dynamically composing disparate sub-systems together,
- as the devices, resources, and applications have variable characteristics, they will need to interact to provide overall system behaviour, and therefore some of them will need to have shared access to resources,
- creation of new applications and codes in the form of rules,
- and, there will be a very high potential of feature interactions and unexplored consequences.

Although each component may work very well on its own, composing them together will lead to a high proportion of **feature interactions**, and thereby affect the system's reliability. A **feature** is any part or aspect of a specification having a self-contained functional role, and a **feature interaction** is some way in which a feature or features modify or influence another feature in the overall system's behaviour set. Features might interact by causing their composition to be incomplete, inconsistent, nondeterministic, or un-implementable. Or the presence of a feature might simply change the meaning of another feature with which it interacts. This paper describes tools and techniques to help us build safe and reliable compositions of these components, and to effectively manage and control the interactions of the system units.

## 2 Related Work

Existing work in feature interactions, e.g. [2], amongst others, can be divided into three:

- Software Engineering approaches: A particular technique used elsewhere in the software engineering process is applied to the service creation process with the goal of eliminating interactions. The general trend is to provide a service architecture that constrains designers to provide "safe" arrangement of features. **Limitation**: Difficulty of constraining to a single service architecture in an open system where components come from

different domains and developed from different kinds of service architectures,

- Formal Methods: A wide range of formal reasoning techniques, such as classical, constructive, and modal logics, process algebras, finite and infinite state automata, are employed to detect service level interactions, i.e. interactions that are independent of an actual implementation. **Limitations**: (i) State space explosion, the possible number of states you may need to check may be very large; (ii) Non-compositionality, in an open system, there is non-monotonicity between new features and services and the original features and services. Most work in this area remain firmly within monotonic frameworks,

- On-line techniques: Intended to be applied at run-time in a network or system. They provide a combination of detection and resolution mechanisms. On-line techniques can cope with addition of new services in the system or network. **Limitations**: (i) Processing overhead; (ii) Inability to cope with change in system or network architecture.

# 3 Managing Interactions Using Rules

For a domain of concurrently interacting devices, Rule Based Control is the best way to control them, as the rules are easy to manage and we can ascertain the devices' interactions and their properties. The focus then shifts to designing a language, which we called $\mathbf{R^{FI}}$ (**R**ule Language for Avoiding **F**eature **I**nteraction), that is expressive enough to be used to state rules of behaviour and of interaction of these devices, and the provision of a system, called Rule-Based Control (RBC) system, that processes these rules looking for possible interactions between them (good and bad), informing users of these interactions, and actuating commands on these devices. The system also determines under what conditions a new device, resource, or object will be accepted into the environment. We have made use of the techniques of goals, rules, declarative specifications, the Semantic Web, and Automated Reasoning to automatically detect potential feature interactions, and after resolution, send command and control scripts to control these devices.

# 4 The $\mathbf{R^{FI}}$ Language

## 4.1 Properties

(1) **Declarative**, allows users to express behaviour in terms of what to be done, and not in terms of how to do it; (2) Gives its users good primitives to express **time** and **duration**.

## 4.2 Syntax and Informal Semantics

A **Rule** is of the form:

$$R^{FI} ::= (E_{ID}, (E_{TEMP\_PRE} \bullet Event_{PRE}) \rightarrow (E_{TEMP\_POST} \bullet Event_{POST})) \quad (1)$$

Where $E_{ID}$ is the Identifiers expression, $E_{TEMP\_PRE}, E_{TEMP\_POST}$ are temporal expressions, $Event_{PRE}$ is the Events' Pre-condition expression, $Event_{POST}$ is the Events' Post-condition expression, and
$\bullet ::= \Box \mid \Diamond$ ($\Box$ (at every/any time); $\Diamond$ (sometime))

$E_{ID} ::= (R_{ID}, R_{PRIO}, U_{ID}, U_{PRIO})$
$\mathbf{R_{ID}}$ is the rule identifier. $\mathbf{R_{PRIO}}$ is the rule priority. $\mathbf{U_{ID}}$ is the user identifier. $\mathbf{U_{PRIO}}$ is the user priority.

### 4.2.1 $E_{TEMP\_\{PRE,POST\}}$ Syntax and Semantics

**Alphabets, Terms, and Formulae**

1. the temporal sort, $\mathcal{S}_{\mathcal{T}}$
2. the granularity sort, $\mathcal{S}_{\mathcal{G}}$, denoting the set of granularity temporal domains, such as years, months, weeks, days, minutes, seconds
3. the temporal position operator $\star_{(\phi)}$

4. the granularity, $\nabla^{granularity}$ , and displacement operator, $\nabla_{displacement}$

5. the projection operator, $\Box$, and its dual, $\Diamond$

6. the temporal position operator, $\star_{(\phi)}\mathcal{F}$, which evaluates $\mathcal{F}$ from temporal position $\star_{(\phi)}$

7. the projection operator $\Box\mathcal{F}$ evaluates formula $\mathcal{F}$ and $\mathcal{F}$ is true if $\mathcal{F}$ is true at all related instants, and its dual

#### 4.2.2 $Event_{PRE}$ and $Event_{POST}$ Syntax and Semantics

**Alphabets, Terms, and Formulae**

1. A set $\mathcal{R}$ of Resource (device and object) names, e.g. switch1, light1, etc.

2. A set $\mathcal{A}$ of action names, e.g. "on", "off", "up", "down", etc.

3. A serial conjunction operator, $\otimes$, which composes events together sequentially

4. A serial disjunction operator, $\oplus$, which composes events together but acts like a choice operator

5. A parallel operator || which joins events together and performs them in parallel

6. A set $\mathcal{L}$ of logical connectives: $\mathcal{L} = \{ \wedge, \vee \}$, used to connect composite events

7. The unary operator, $\neg$

8. A pair of port commands $\mathcal{P} = \{ ? , ! \}$, where ? is an input port command, and ! is an output port command

9. The quantification symbols $\mathcal{Q} = \{ \exists, \forall \}$ on $\mathcal{R}$

#### 4.2.3 Examples of Rules

$$
\begin{aligned}
&(rule1, 3, too20, 5),\\
&(\star_{[2006-09-28T11:00]} \Box\, (door!open)) \\
&\rightarrow\ (\star_{[2006-09-28T11:00]} \Box\, \nabla_{15}^{mins}\, (door?close))
\end{aligned}
\tag{2}
$$

If the door is open at 11:00, keep it open for 15 minutes, and close it after 15 minutes.

$$
\begin{aligned}
&(rule2, 2, too20, 5),\\
&(\star_{[2006-09-28T11:00]} \Box \quad switch1!on)\\
&\rightarrow\ (\star_{[2006-09-28T11:00]} \quad \Box\, (\nabla_{60}^{mins}\, switch1?off)\\
&\qquad\qquad || \,(light1?on \otimes \nabla_{60}^{mins}\, light1?off)\,)
\end{aligned}
\tag{3}
$$

This means, if switch1 is on at 11:00 on 28 Sept 2006 until 12:00 on 28 Sept 2006, keep light1 on during the length of that time.

# 5 The RBC System

## 5.1 Properties

(1) A non-monotonic deductive system that elicits possible feature interactions; (2) Policies of incoming electronic objects in ontological form (using, for example, RDF [3]); (3) A Rule language to specify intended behaviour of electronic objects interactions.

## 5.2 System Components

Fig 1. shows the system architecture.

1. Registration Resource Module: **Registers** resources; **Generates** Horn clauses of events' sinks and sources in resources; **Notifies** the Event Service of events' sinks and sources

2. Rules Processor: Converts rules into Horn clauses inserting them into the rule base

3. Rule Base, which consists of

   - Rules in Horn clause form
   - A Logic Inference Engine: Runs inferencing and reasoning algorithms on rules. The outputs of these reasoning algorithms are rules which have the following characteristics:
     - feature interaction-free
     - conflict-free

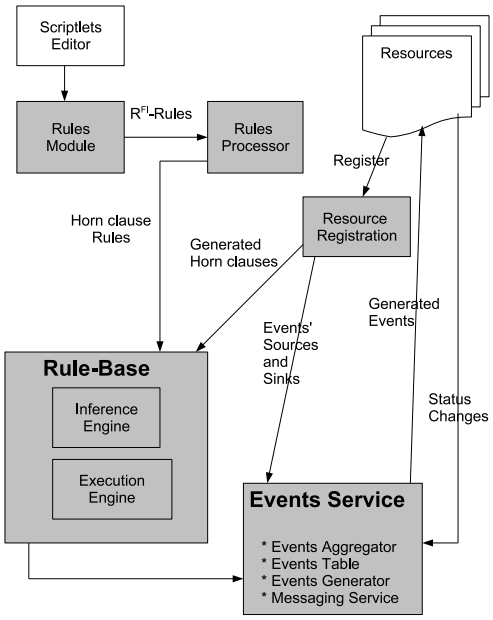


Figure 1: System Architecture

– satisfy safety and liveness properties

Users are notified of satisfiability of submitted rules.

4. The Execution Engine. Activates feature interaction-free, conflict-free rules turning them into commands that are sent to the Event Service.

   In our rule-based system, the stored knowledge (i.e. the inference engine) is orthogonal from the control mechanism (i.e. the execution engine), an example of good separation of concerns.

5. Events Service: Is **notified** of events originating from resources; **Receives** commands from the Execution Engine and generates events for resources; **Provides** the asynchronous messaging capability of the system.

### 5.3 Types of Reasoning performed on Rules

Conflict Detection and Resolution algorithms which solve the following problems: a rule is an event structure: $E.S. =< \tau, I, R, A \cup V_i >$
$\tau$ = Priority Assignment, I = time interval (in milliseconds), R = resource,
A = action, $V_i$ = action attributes.
A **feature interaction problem**, is defined as:
For $e.s._1 \in E.S._1 \wedge e.s._2 \in E.S._2$ .
$e.s._1 = [\tau_1, i_1, r_X, a_1] \wedge e.s._2 = [\tau_2, i_2, r_X, a_2]$ and a truth-value binary relation R on $i_1 \times i_2 \wedge R \in \{during, equal, overlaps, starts, ends\}$, there is a **feature interaction** if $i_1 R i_2 \wedge a_1 \neq a_2$ given the same resource $r_X$.
We also performed **safety** and **liveness** reasoning on rules.

# 6 Conclusion

In this paper, we described a mechanism that helps to increase the reliability and safety of ubiquitous computing environments through the control of interactions of their constituent elements. We described the system we have built which made use of ontologies, goals, rules, and a non-monotonic deductive system to detect, resolve, and control the (feature) interactions amongst the elements of such ubiquitous computing environments.

# References

[1] C. Hewitt and P. de Jong: *Open Systems*. Technical Report, AIM 691, A.I. Laboratory, M.I.T. Dec. 1982

[2] P. Dini, R. Boutaba, L. Logrippo: *Feature Interactions in Telecommunications Systems IV*. pub. IOS Press (Amsterdam), 1997

[3] Resource Description Framework: http://www.w3.org/RDF/