Self-managed Cells for Ubiquitous Systems

Naranker Dulay[1], Emil Lupu[1], Morris Sloman[1],
Joe Sventek[2], Nagwa Badr[2], and Stephen Heeps[2]

[1] Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2AZ, United Kingdom
{n.dulay, e.c.lupu, m.sloman}@imperial.ac.uk
[2] Department of Computing Science, University of Glasgow,
17 Lilybank Gardens, Glasgow G12 8RZ, United Kingdom
{joe, nagwa, heeps}@dcs.gla.ac.uk

Abstract. Amongst the challenges of ubiquitous computing is the need to provide management support for personal wireless devices and sensors. In this extended abstract we introduce a policy-based architecture that supports management at varying levels based on the concept of a self-managed cell. Cells include policy-driven agents that support context-based and trust-based access control and system adaptation. Cells can also organize themselves through federation and nesting.

1 Introduction

Advances in ubiquitous computing infrastructures have the potential to dramatically broaden the role of computing in the everyday lives of people with a greater proliferation of personal wireless devices, and more significantly with wireless computing devices starting to be embedded in the environment: in buildings, in roads, in vehicles, in the landscape, in home appliances, in clothing, on packaging of consumer goods in shops; even as implants in plants, animals and humans. The challenges of ubiquitous computing will not only be about building such ubiquitous environments, they will also be about managing the resources and omnipresent information which ubiquitous systems will need to discover, capture, process and publish behind the scenes. This information will be ephemeral, mobile, fragmented and voluminous with no predictable flows between producers or users of the information.

1.1 Ubiquitous Systems Management

Existing architectures for network and systems management are aimed at large-scale corporate environments, telecommunications networks and Internet service providers and do not cater for ubiquitous environments, although specific techniques for monitoring, event correlation, service discovery, quality of service and policy-based management can be used to some degree. For ubiquitous systems, architectures are needed that can scale down to small devices with local decision-making. The limitations of small devices, e.g. memory size, CPU speed, battery life, screen size, network range and changing connectivity; require new techniques for optimizing resource usage and tailoring information within tight deadlines. Management will also need to be performed according to measures of context and trust and tailored to the individual preferences and circumstances of users. Flexible techniques will be needed to filter information and perform access control, as well as defining and enforcing privacy. Users will expect management functions to be invisible and carried out automatically.

We are developing a policy-based architecture that supports management at varying levels of granularity, using the concept of a self-managed cell (or simply a cell). A cell consists of a set of hardware and software components that represent an administrative domain. Cells are able to function autonomously and thus capable of self-management. A cell could represent the resources available in a PDA, a body area network of physiological sensors and controllers. At the enterprise level, a cell could represent the resources and application components relating to a set of collaborating partners forming a virtual organisation spanning multiple countries. In each case, cells include and evolve the required management services, appropriate to the scale and environment of the cell. These management services interact with each other through asynchronous events exchanged over an event bus. In essence, a cell is a "closed-loop" system where changes of state in the managed objects and resources trigger adaptation that in turn affects the state of the system. In ubiquitous environments, the cells would also typically include management components that provide service discovery and contextual management.

A cell includes a policy-driven agent that supports context-based and trust-based access control and system adaptation for one or more ubiquitous devices. Cells can load additional management functions and organise themselves into larger management cells through federation and nesting. Potentially, each ubiquitous device that a user carries, and each device situated in the environment, is capable of being a self-managed cell and running a management agent that carries out management functions and policies. In practice, we envisage that some devices (e.g. sensors) will be too primitive to run their own management agent, but will be capable of being managed by an external cell, such as a mobile phone, over a wireless link, such as bluetooth. This extended abstract introduces the architecture of self-managed cells.

2 Self-managed Cells

Each self-managed cell consists of a number of core management components: the cell watchdog, the event service, the discovery service, the policy service, and the domain service. Cells can also load components for context and trust management as well as monitoring and intrusion detection. Proxies are required to interact with the various communication interfaces of devices and managed components, for example to enable cell policies to perform actions on device-specific management interfaces, and to convert low-level signals to cell events. The following outlines the core services of each self-managed cell.

2.1 Cell Watchdog

When a cell is first instantiated, it starts up the cell watchdog. This is a special service that is responsible for loading and instantiating the core components of the cell, typically from local storage (e.g. a memory card), or from a remote cell. The cell watchdog is also responsible for cleanly removing and restarting core components when a core component fails, or if a core component needs to be updated. Essentially the cell watchdog has the responsibility to ensure the survivability of the core management components, and ideally should be in firmware and always alive.

2.2 Event Service

Management systems are essentially event-driven, as changes of states need to be notified to several, potentially unknown management services. Examples of events include: the discovery of a new device, a change in context (e.g. battery level low), an intrusion alert. The event service provides at-most-once, persistent publish/subscribe delivery and is used for both intra-cell and inter-cell management. The event service supports event correlation for flexibility.

2.3 Discovery Service

The discovery service is responsible for detecting the presence of devices that come into wireless range. These may be primitive devices that are managed by the cell, devices that are managed by others cells, or devices that are not currently managed by any cell. Once a device is discovered, the discovery service communicates with the device to get further attributes (e.g. type, profile, services provided) and generates a “new-device” event for other management components. The discovery service needs to distinguish between transient failures, which are common in wireless communications, and when some device is really no longer available (e.g. out of range or switched off).

2.4 Policy Service

The policy service is responsible for the execution of policies. Policies are rules that govern the choices in behaviour of the cell. Two kinds of policy are currently supported. Obligation policies (event-condition-action rules), which define what actions to carry out when specific events occur, and authorisation policies which define what actions are permitted or not permitted, for what or for whom, and under what conditions. Policies can be added, removed, enabled or disabled to change the behaviour of a cell. See cell policy language (section 3).

2.5 Domain Service

The domain service provides a means of hierarchically grouping references to objects (c.f a filesystem). Objects include devices, services (including core services), policies, neighbouring cells. For example, when a new device is discovered, a reference to it, is normally added to the domain `/dev` as well as to application-specific domains, for example, `/music/headset/bluetooth`. Domains are also used to define authorisation policies in the cell policy language, e.g. objects within the subject domain `/players/mp3` are permitted to perform the action `play` on objects in the target domain `/headsets`.

2.6 Context Service and Trust Service

In addition to the core components, cells can also load a context service and a trust service. These allow context and trust information to be defined, gathered and combined, and used in evaluating policy constraints. Changes in context and trust can raise events that trigger obligation policies that cause adaptation.

3 Cell Policy Language

Central to the management of cells is the Cell policy language and interpreter. The language is loosely based on the Ponder policy language developed at Imperial College London. All primitive policies are encapsulated into one composite type called the *relationship*. There are no roles, groups, or management structures. There are no domain scope expressions. Subjects can be based on credential verification as well as domain membership. The language includes explicit support for domain creation/removal as well as enabling/disabling of policies. Composite event can be defined. There are explicit rules for authorisation conflict resolution based on explicit relationship ordering rules. The syntax is also cleaner and less cluttered than Ponder and is suitable for interactive execution.

3.1 Relationships

Relationships encapsulate one or more policies. Currently obligation (event-condition-action) policies and authorisation policies are supported. Relationships can also encapsulate other relationships. Relationships are created, enabled, disabled, removing as a whole, e.g. policies cannot be added to a running relationship, other than by disabling and removing the relationship, and replacing it with a new relationship with the additional policy. The policies act as an atomic unit, for example, disabling an individual authorisation may lead to unexpected results. The policy service includes a multi-threaded interpreter for concurrently executing obligation policies. The following examples illustrate the Cell policy language.

Example 1. Authorisation policy. Members of the family domain are allowed to play games on the pda but only at home or in the car.

```
context home_car: location=home or location=car

auth+ /family -> home_car ? /pda/games.play
```

Example 2. Authorisation policy. Doctors who can present a credential issued by the British Medical Association (BMA) can issue commands to the cell's medical devices in an emergency in the UK.

```
credential medic:role=Doctor and issuer=BMA and issueyear>2005
context UK_emergency: location=UK and condition=wounded

auth+ -> medic and UK_emergency ? /medical/devices.commands
```

Example 3. Obligation policy. On discovering a new bluetooth headset add it to the sound/output/bluetooth domain.

```
on HeadsetDetect (X) -> X.type=bluetooth ?
                        /sound/output/bluetooth.add(X)
```

Example 4. Obligation policy. After 20 failures to enter a PIN, disable the Mobile Phone policy and enable the Stolen mobile phone policy

```
event Stolen: count (PIN_failure, 20)

on Stolen () -> /policy/mobile/normal.disable (),
                /policy/mobile/stolen.enable ()
```

4 Inter-cell Interactions and Self-organisation

Although self-managed cells provide the management capability for supporting configuration and adaptation within a device, there is a need to support management across multiple cells. The cell architecture supports two forms of inter-cell organisation:

- *Federated* to support peer-to-peer interactions between cells in order to collaborate and share resources, for example police, ambulance and fire workers collaborating and sharing resources at car-accident. Management relationships between federated cells are often transient, but can be longer-lived.
- *Nested,* where several cell nest within an enclosing cell and nested cells are not visible to cells external to the enclosing cell i.e. any management interaction is via the enclosing cell. Cells can move and out of enclosing cells, for example, the cell of a patient returning home, may nest in the home cell, and be governed by the policies of the home cell.

We model cell-cell interactions through relationships. Each cell defines its own relationships with respect to other cells. When a new cell is discovered it is subject to a similar procedure as devices. However for cells, additional actions and protocols are supported including exchange of policies, event registrations, and domain membership details. These protocols allow cells to share management information and resources and self-organise through federation and nesting.

5 Current Status and Future Work

We are currently developing Java-based implementations of the cell architecture to run on Series 60 Nokia phones, HP iPaq PDAs and laptops over bluetooth, wi-fi, and GPRS. We are also experimenting with body sensor nodes with Zigbee wireless capability that communicate by low-power radio with the iPaq. A simulator to test larger cells and more easily simulate repetitive events or devices coming into and out of range is being developed.

There are many issues still to be resolved, such as making sure the protocols optimise the use of battery power; how to make sure a device is ‘owned’ by the appropriate cell and not taken over; how to present management information and policies to end-users and elicit policy settings; investigating the best design patterns for inter-cell management; how to specify and implement privacy policies that allow users to control access to personal information, and what mechanisms to use to anonymise personal information and prevent tracking.

Acknowledgments

The authors wish to thank the UK Engineering and Physical Sciences Research Council for their support of this research through grants GR/S68040/01 and GR/S68033/01.