Part I

FRAMING THE PROBLEM AND CURRENT STATE-OF-THE-ART

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* angloromanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

Part II

DESIGN ITERATIONS AND CONSTRUCTING A THEORY

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* angloromanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

Part III

GENERALISED MODELS, SOFTWARE ARCHITECTURE AND EVALUATION

In this part of the thesis, the more general concepts and techniques that can be applied to ubiquitous computing are described. These concepts and techniques were extracted from work done during the three design iterations.

10 SOFTWARE ARCHITECTURE

Interaction is an iterative process of listening, thinking, and speaking between two or more actors.

— Chris Crawford, game designer

Parts of this chapter have previously appeared in [71] and [73]

Existing architectural patterns for software like the Model-View-Controller (MVC) model, Document-View and Presentation-Abstract-Control are considered to be inadequate when trying to design software architectures in the ubiquitous computing domain. Ubiquitous computing needs new kinds of mechanisms to meet the flexibility needed to change the purpose, functionality, quality and context of a software system [70].

MVC was first mentioned in Section 2.3.

In this chapter the software architecture used in the three design iterations is described in more detail. It is quite a short chapter, as most of the software architecture issues have already been discussed in the three design iteration implementations in Sections 3.4, 4.4 and 5.4. However, we consider it important that the final software architecture design has its own dedicated chapter, so that it can act as a reference design for future implementations.

We first look at some characteristics of ubicomp middleware, followed by a discussion of the publish/subscribe paradigm and the blackboard architectural pattern. We then look at the Message-Oriented Middleware (MOM) implementation used within the Smart Objects For Intelligent Applications (SOFIA) project, called Smart Space Access Protocol (SSAP). The rest of the chapter is dedicated to the two main implementations of the software architecture as used within the SOFIA project – Smart-M3 and ADK-SIB. These implementations are interoperable with one another through the use of SSAP.

10.1 CHARACTERISTICS OF UBICOMP MIDDLEWARE

There are a number of characteristics, or quality attributes, that are specific to middleware for ubiquitous computing, as defined by Niemelä and Vaskivuo [70]:

- Interoperability

- Scalability
- Reusability
- Maintainability
- Extensibility
- Portability
- Adaptability
- Survivability
- Agility
- Fidelity

Interoperability is defined as the ability for software applications written in different programming languages, running on different platforms with different operating systems, to communicate and interaction with one another over different networks. Scalability is the ability of the system to handle larger numbers of smart objects. Reusability, maintainability and extensibility are characteristics that consider the evolution of software systems. Portability and adaptability are important characteristics for software that has to work in a heterogenous system of devices and networks. Survivability is the ability of a system to timely deliver essential services in the face of attack, failure or accident. Agility is the sensitivity to changes in resource availability. Fidelity is defined to mean to degree to which data presented on a client matches the reference copy at the server.

We focused on a subset of these attributes while working on the software architecture, including interoperability, reusability, maintainability and extensibility. Interoperability was achieved by adhering to the SSAP specification, as described in more detail in Section 10.3. Using ontologies and other Semantic web technologies helped us to improve reusability, while elements of maintainability were tested using the Cognitive Dimensions framework, described in more detail in Chapter 11. Extensibility was achieved by modelling devices and their capabilities in such a way that other devices could easily be added to the system.

Potential scalability was tested by evaluating the performance of the software architecture, as described in Section 11.1.

10.2 PUBLISH/SUBSCRIBE PARADIGM AND THE BLACKBOARD ARCHITECTURAL PATTERN

In publish/subscribe systems, subscribers register their interest in a specific event, and are notified when this event occurs after a publisher publishes the event. The strength of the publish/-subscribe paradigm is that entities are decoupled in time, space and synchronisation [35]. Space decoupling means that the interacting entities do not need to be aware of each other. Time decoupling means that the entities do not need to participate in the interaction at the same time. Synchronisation decoupling means that subscribers can asynchronously be notified when an event occurs. Removing synchronisation dependencies between entities increases scalability.

There are three variants of publish/subscribe systems:

- Topic-based – Entities subscribe to individual topics, usually with some form of hierarchical addressing to organise the topics
- Content-based – Consumers subscribe to selective events by specifying filters, using some kind of subscription language
- Type-based – Events are filtered according to their type

In the SOFIA project, Knowledge Processors (KPs) communicate with a message broker using the blackboard architectural pattern, where the message broker uses a triple store as a common knowledge base. Communication between KPs occurs through the insertion and removal of triples into or from the triple store. Given a set of smart devices, the blackboard may be used to share information between these devices, rather than have the devices explicitly send messages to one another. If this information is also stored according to some ontological representation, it becomes possible to share information between devices that do not share the same representation model, and focus on the semantics of that information [78]. The Semantic Information Broker (SIB) is the information store of the smart space, and contains the blackboard, ontologies, reasoner and required service interfaces for the KPs or agents.

This blackboard approach is complemented by a publish/-subscribe component, that allows KPs to subscribe to specific triples in the triple store. The KPs are then notified when these

triples are added, removed or updated in the triple store. Communication between the KPs and SIB occurs using SSAP, which is the focus of the next section.

10.3 SMART SPACE ACCESS PROTOCOL (SSAP)

MOM is used to send messages between components in a distributed system. Commercial options include Java Message Service (JMS), Microsoft Message Queuing (MSMQ) and IBM's WebSphere framework. Advanced Message Queuing Protocol (AMQP) is an emerging standard, of which RabbitMQ[1] is a popular implementation. ZeroMQ[2], also written as ØMQ , was created to be simpler and faster than the AMQP standard, and does not require a dedicated message broker. Other message protocols include Extensible Messaging and Presence Protocol (XMPP), Message Queue Telemetry Transport (MQTT) and Streaming Text Oriented Messaging Protocol (STOMP).

XMPP is an Internet Engineering Task Force (IETF) standard.

In the SOFIA software architecture, KPs communicate with the SIB through SSAP messages [54] over TCP/IP. SSAP consists of a number of operations to insert, update and subscribe to information in the SIB. These operations are encoded using Extensible Markup Language (XML).

For operations initiated by a KP, the KP sends a request message and the SIB responds with a corresponding confirmation message. For SIB initiated operations, the SIB sends an indication message. The KP does not respond to SIB initiated operations, as indication messages contain non-essential information. Every session must start with a join operation, and a leave operation ends a session.

To insert information into the triple store, an insert operation is used by the KP, where the triples are encoded in RDF/XML. A SIB confirmation message indicate whether the operation was successful or not. Similarly, a remove operation is used to remove information from the triple store. An update operation removes information from the triple store and inserts new information as an atomic operation.

To query the triple store, a template consisting of a list of triples is used, where each triple may have a wildcard as its subject, predicate or object. The result of the query is a list of all triples that match the template. All triples in the triple store that match any of the triples in the list are returned.

1 http://www.rabbitmq.com/
2 http://www.zeromq.org/

A subscribe operation creates a persistent query that is stored in the SIB and is re-evaluated automatically after each change to the contents of the triple store. An unsubscribe operation will terminate a persistent query. The publish/subscribe mechanism used is closest in scope to the content-based variant described in the previous section.

SSAP is supported by both the SIB implementations used in our work, such that software developed for the one implementation is also interoperable with the other implementation. We now focus in more detail on these two implementations: Smart-M3 and ADK-SIB.

The Smart-M3 implementation was used during the first design iteration, while the ADK-SIB implementation was used during the second and third design iteration.

10.4 SMART-M3 ARCHITECTURE

The M3 (multi-device, multi-vendor, multi-domain) architecture is an interoperability platform based on a blackboard architectural model that implements the ideas of space-based computing [54]. It consists of two main components: a SIB that acts as a common, semantic-oriented store of information and device capabilities, and KPs, virtual and physical smart objects that interact with one another through the SIB. Various SIB implementations exist that conform to the M3 specification of which Smart-M3, developed by Nokia, was the first open source reference implementation released in 2009[3]. RDF Information Base System (RIBS), developed by VTT, is a C-based implementation of M3 targeted for devices with low processing power, but requires a large amount of memory [34].

10.5 ADK-SIB

The SIB implementation used during the second and the third design iteration is called ADK-SIB (Application Development Kit SIB) and was developed within the SOFIA project. The ADK-SIB is a Jena-based[4] SIB written in Java and runs on the Open Services Gateway initiative (OSGi) framework.

Reasoning in the standard ADK-SIB is implemented using the Jena Ontology API, but only basic reasoning with symmetric properties and transitive properties is supported. Our main contribution to improve the ADK-SIB implementation was to implement support for OWL 2 RL/RDF Rules reasoning, as

3 http://sourceforge.net/projects/smart-m3/
4 http://jena.sourceforge.net/

well as SPARQL Inferencing Notation (SPIN) rules using the TopBraid SPIN API[5].

The ADK-SIB and SPIN API was first mentioned in Section 4.4.1.

When the SIB starts up, we first load the ontology, written in Web Ontology Language (OWL) 2, from a specified web address into our *asserted* model. We then load the OWL 2 RL specification, specified as SPIN rules, from another OWL file. We also load any custom SPIN functions into a third model. We then build a union model of the three models and store all the asserted triples in a hashmap to improve lookup efficiency. Finally the TopSPIN reasoning engine performs inferencing across the union model, and all the inferences are stored in the *inferred* model.

Whenever a new triple is added, removed or updated, the inferred model is cleared and inferencing is performed using the reasoning engine. This means that no inferencing needs to be performed when a query is run.

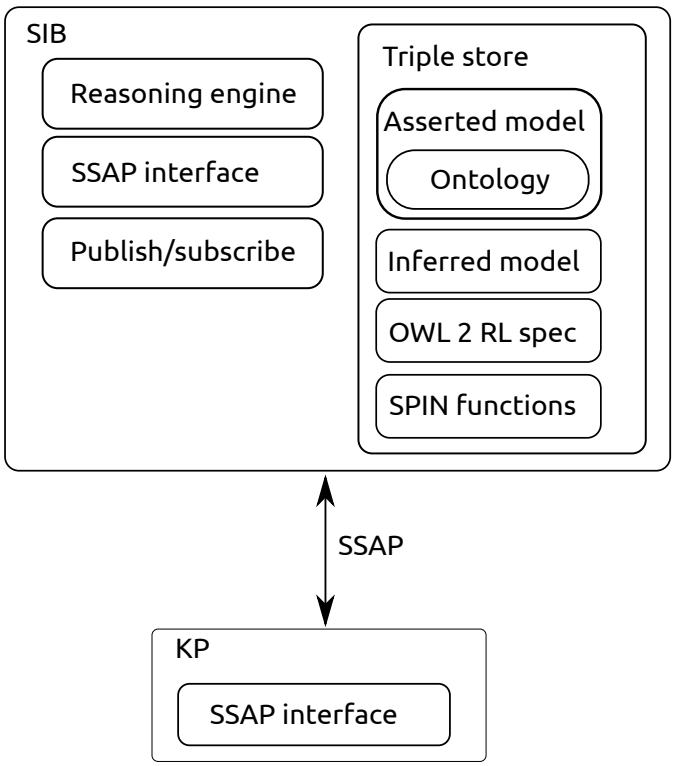

Figure 53: Our software architecture

The final software architecture is shown in Figure 53. The system performance of the software architecture was evaluated during the Smart Home pilot of the second design iteration, and this evaluation is described in the next chapter. A valida-

5 http://topbraid.org/spin/api/

tion of the entire system, including the ontology, using the Cognitive Dimensions framework is also described.

Part IV

APPENDIX