Examensarbete

Modelling of Interaction Units

av

Bo Sun

LITH-IDA-EX-- 05/040 --SE

2005-03-29
Examensarbete

Modelling of Interaction Units

av

Bo Sun

LITH-IDA-EX-- 05/040 --SE

2005-03-29

Handledare: Anders Larsson
Examinator: Henrik Eriksson
Abstract

Developing a model of a service system and mobile units including cellphone, PDA, Laptop is an important preliminary step of designing the systems which could provide these units some convenient and entertainment services through common short range communication like blue tooth, wireless LAN, etc.

In this project, an ontology is created to represent this model. Meanwhile, some basic service rules are also programmed and combined with this ontology can be used to simulate some interactions between items inside this model.

The description of this model (ontology) has been made through Protégé\(^1\) and demonstrated by using its graphical interface. The rules have been created by using Jess\(^2\) and implemented with the ontology by using JessTab\(^3\).

\(^1\)http://protege.stanford.edu
\(^3\)http://www.ida.liu.se/~her/JessTab/
Acknowledgement

I would like to thank my supervisor Anders Larsson so much to provide me this topic and patient guidance to me. And I also like to thank my examiner professor Henrik Eriksson for his important help and I appreciate everyone who gave me help and encouragement during my thesis. Thank you!
# Content

**Chapter 1 Introduction** .......................................................... 5  
  1.1 Motivation ........................................................................... 5  
  1.2 Goals .................................................................................. 5  
  1.3 Limitation ........................................................................... 5  
  1.4 Paper Structure ................................................................... 6

**Chapter 2 Background** ............................................................ 7  
  2.1 Ontology .............................................................................. 7  
    2.1.1 Definition ....................................................................... 7  
    2.1.2 Why using an Ontology .................................................. 7  
    2.1.3 Types of Ontology .......................................................... 8  
    2.1.4 The entities of an Ontology .............................................. 8  
  2.2 Protégé ................................................................................ 9  
    2.2.1 What is Protégé ............................................................... 9  
    2.2.2 Why Protégé ................................................................. 11  
      2.2.2.1 RDF ....................................................................... 12  
  2.3 OWL Ontology .................................................................... 13  
  2.4 Jess .................................................................................... 14  
    2.4.1 What is Jess ................................................................. 14  
  2.5 JessTab .............................................................................. 15

**Chapter 3 Related Work** ......................................................... 18  
  3.1 Ambient Intelligence ............................................................ 18  
  3.2 Context-Aware System .......................................................... 18

**Chapter 4 Method Description** ................................................. 20  
  4.1 Modelling .......................................................................... 20  
  4.2 Jess Rules ........................................................................... 22

**Chapter 5 Developing the Ontology** ........................................ 24  
  5.1 Building Initial Ontology ....................................................... 24  
    5.1.1 Determine Domain ........................................................ 24  
    5.1.2 Defining Classes ........................................................... 25  
    5.1.3 Defining Properties ....................................................... 27  
    5.1.4 Defining Restrictions ................................................... 28  
  5.2 Revising Ontology ............................................................... 29  
    5.2.1 Creating Subclasses ....................................................... 29  
    5.2.2 New Properties and Restrictions .................................... 31  
    5.2.3 Inserting Instances ........................................................ 33  
    5.2.4 Final Accomplishment .................................................. 36
<table>
<thead>
<tr>
<th>Chapter 6 Creating Rules</th>
<th>........................................ 39</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Situation Design</td>
<td>........................................ 39</td>
</tr>
<tr>
<td>6.2 Programming Rules</td>
<td>........................................ 40</td>
</tr>
<tr>
<td>Chapter 7 Discussion</td>
<td>........................................ 42</td>
</tr>
<tr>
<td>7.1 Accomplishments</td>
<td>........................................ 42</td>
</tr>
<tr>
<td>7.1.1 Ontology</td>
<td>........................................ 42</td>
</tr>
<tr>
<td>7.1.2 Rules</td>
<td>........................................ 42</td>
</tr>
<tr>
<td>7.2 Limitations</td>
<td>........................................ 43</td>
</tr>
<tr>
<td>Chapter 8 Future Work and Conclusion</td>
<td>........................................ 44</td>
</tr>
<tr>
<td>8.1 Future Work</td>
<td>........................................ 44</td>
</tr>
<tr>
<td>8.2 Conclusion</td>
<td>........................................ 45</td>
</tr>
<tr>
<td>References</td>
<td>........................................ 46</td>
</tr>
<tr>
<td>Appendix A (rules code)</td>
<td>........................................ 48</td>
</tr>
<tr>
<td>Appendix B (example of simulation)</td>
<td>........................................ 54</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Nowadays, users are equipped with more and more mobile devices, such as laptops, personal digital assistant (PDA), and cellphones. Meanwhile, public spaces and homes are equipped with wireless internet access. This has led to new ways of interaction with information system like reading email on a cellphone, book a restaurant visit from a laptop or even using a digital pen and a TV-guide as a remote control for TV-sets and personal video recorder (PVR) [1]. How services are accessed and presented is quite different depending on what device or devices the users are currently using. Services might also have to be split up over a set of devices in order to allow users with adequate interaction possibilities, the users are then presented with a distributed user interface (DUI) [2]. In order to allow system developers to have a sound and efficient way of describing what devices can access what information services, a descriptive model for devices and services is needed as well as rules describing behaviour and requirements. We think that constructing a model of these units (cellphone, PDA, laptop) and their interaction would be necessary and helpful for developing information systems.

1.2 Goals

The major goal of this project is to provide service system designers a model of a service system and different mobile units, and a way of simulating interactions between units and service system by means of the rules created for this model. The goal is accomplished with two tasks:
1. Modelling: Developing an ontology representing mobile units and a service system.
2. Creating Rules: A service system needs a rule engine to manage the interaction between the requests from mobile devices and the service system. Within this work, some rules are created to simulate some situations of interaction.

The description of this model and how the rules work on this model can be seen through Protégé interface.

1.3 Limitation

Since this project is an initial model which is still in research stage, no requirements were asked from companies or particular projects. The ontology created in this project can not take into account complete detailed information and every kind of situation so far. The description of ontology and the rules combined with this
ontology are based on the investigation of actual information, situations and some assumptions. It will be developed and iteratively revised with new requirements asked.

The interaction is not determined to simulate on the real mobile devices. The point of the simulation part is to demonstrate the working situation of the rule policies which will be shown in a simple way.

1.4 Paper Structure
This paper is divided into 8 chapters as below:
Chapter 1 gives an overview of this paper including motivation, goals, limitation, etc. The preliminary knowledge of the technologies used in this project is shown in chapter 2. Chapter 3 gives the work related to this project. Chapter 4 presents method description which will be applied on building the model and creating rules. The detailed design description of developing ontology and rules creation is presented in chapter 5 and chapter 6. Chapter 7 gives the discussion of the whole project. Finally, chapter 8 provides some future work and summarizes this thesis.
Chapter 2

Background

This chapter gives theoretical introduction of the technologies applied in this project. Through this chapter, readers can have a preliminary knowledge of the design thoughts described in chapter 5 and 6 and know about the motivation why these technologies are used to build model and create rules.

2.1 Ontology

2.1.1 Definition

“Ontology”, this concept could go “as far back as Aristotle’s attempt to classify the things in the world.” [3]. A lot of definitions of this term have been given before in Artificial Intelligence field and lately in semantic web, here, we take a concise but comprehensive one:

“An ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them.” [4] This vocabulary refers to an abstract model (or taxonomy, hierarchy) which consists of identified relevant basic concepts in a domain. “machine-interpretable” means an ontology should be machine processed, which is opposite to natural language.

2.1.2 Why using an ontology

There are 5 major reasons to motivate the development of ontology:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

Compared to the other goals, sharing common understanding of the structure of information among people or software agents is a more common goal of developing an ontology [5]. Here is an example: If the library databases which are located nation-wide share the same bibliography ontology which consists of terms they all share (concepts like books, authors, journals, etc), a national library network

---

4 A particular field of knowledge, such as breast cancer [13].
could be set up to answer readers queries\(^5\) through the internet, it would be much more convenient than through only one single library.

### 2.1.3 Types of Ontology

Before developing an ontology, it’s helpful to start with understanding what types of ontology there are. Ontologies are classified into four types according to different generality levels ([3], [6], [7], [8], [9]):

- **Domain ontologies** capture the knowledge related to a particular type of domain, e.g. domain electronic, medical, mechanic, digital, etc.
- **Generic Ontologies** are related to several domains and not referred to the particular one. Generic ontologies are also defined as super theories [10] or core ontologies [7].
- **Application Ontologies** are the type of ontologies which contain all the necessary knowledge to model a particular domain [11].
- **Representation Ontologies** refer to any particular domain. They provide representational entities without stating what should be represented. A well-known representational ontology is the Frame Ontology [5], which defines concepts such as frames, slots and slot constraints allowing the expression in an object-oriented or frame-based way.

### 2.1.4 The entities of an Ontology

What could be seen in an ontology, what entities should be inserted into an ontology to construct a model [3].

- **Classes** An ontology defines some basic concepts in a domain and relations among them as mentioned in 2.1.1 and these concepts are usually called classes. Defining classes is the major work of building an ontology. For example, class *mankind* represent all the people on the earth. We can also define the subclass which is more specific than a class (This class is also called the superclass of this new subclass). Class *male* and class *female* could be the subclass of class *mankind*.

- **Slots** describe various features and attributes of the concepts. They are also called properties. For example, *mankind* should have property *age* to describe how old a person is.

- **Facets** are the restrictions on these properties. For example, we should add restriction like 0-150 on the property *age* to make it reasonable and logical.

- **Instances** are some specific individuals of a class. For example, *Tom* and *Jane* could be two instances of class *male* and class *female* respectively.

The whole process of designing an ontology could be described by Figure 1.

---

\(^5\) Certainly, this ontology should be accepted by readers as prerequisite.
2.2 Protégé

2.2.1 What is Protégé

In this project, Protégé was used (version Beta 3.0) to develop the ontology. Protégé is a “tool which allows the user to construct a domain ontology, to customize knowledge-acquisition forms, to enter domain knowledge and a platform which can be extended with graphical widgets for tables, diagrams, animation components to access other knowledge-based systems embedded applications and a library which other applications can use to access and display knowledge bases as well” [13]. Some screenshots of Protégé are shown here to give readers a brief impression of Protégé.
Figure 2. The classes interface of Protégé.

Figure 3. Classes Hierarchy
From figure 4, it can be seen that slot age has value type: 

**Integer** and some other facets, such as **single** and **minimum 0, maximum 150** which means a person can have only one age from 0 to 150 years old, which is based on fact.

2.2.2 Why Protégé [12]

First of all, Protégé is designed for system developers and domain experts to develop knowledge-based system[^6] [12]. It is a tool to permit the integration of creating classes and entering instances of classes, and convenient to domain experts to model...
concepts in a specific domain and create applications on it to solve problems regarding to this domain.

Before talking about the second motivation of using Protégé, it might be helpful to introduce some knowledge about RDF/RDFS first.

### 2.2.2.1 RDF

As the ontology definition mentioned in 2.1.1, the ontology must be defined to be machine-interpretable, so that computer could share the information. RDF (Resource Definition Framework) is such an ontology language which is “a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page” [22]. Though this project is not related to sharing information on web, it’s still useful to describe information about mobile units and other concepts in the ontology.

RDFS (RDF Schema) [22] is an extension of RDF which allows the expression of classes, properties, instances, etc. Let’s see an instance definition in RDFS illustrated by Figure 5.

```xml
<rdf_:Male rdf:about="&rdf_;human_Instance_6"
    rdf_:age="24"
    rdf_:name="Tom"
    rdfs:label="human_Instance_6">
    <rdf_:nationality>Sweden</rdf_:nationality>
    <rdf_:nationality>US</rdf_:nationality>
</rdf_:Male>
```

So, why using Protégé? Protégé provides developers with import and export functions to manage RDF or RDFS files in Protégé projects: One side, Protégé can import an existing RDF or RDFS and transform it into a Protégé project, which let developers see what they have done and do modification in graphic way. On the other side, Protégé could export a Protégé project into a RDF or RDFS standard file, which means that the ontology developers could create an ontology without knowing the details of writing RDF files and get a machine-interpretable ontology eventually.
There are some other tools\textsuperscript{7} as well which can develop ontology such as \textbf{Oiled}, \textbf{OntoEdit} and so on. \textbf{Protégé} is the commonest one of these tools.

### 2.3 OWL Ontology

In this project, Protégé OWL plug-in\textsuperscript{8} was used to create the ontology. Ontology Web Language (OWL), is a standard ontology language developed by World Wide Web Consortium at present. Compared to RDF (RDFS), OWL provides more features to describe classes and properties, such as classes relation (disjoints, etc), property characteristics (symmetric, transitive, etc) and so on. Let’s see a property definition in OWL.

\begin{verbatim}
<owl:ObjectProperty rdf:ID="hasNationality">
  <rdfs:domain rdf:resource="#Mankind"/>
  <rdfs:range rdf:resource="#Nation"/>
</owl:ObjectProperty>
\end{verbatim}

Protégé OWL also provides a richer set of restriction operators and allows a reasoner

\textsuperscript{7}http://www.topquadrant.com/documents/TQ1202 Ontology%20Tool%20Survey.pdf

\textsuperscript{8}In OWL project, slot and instance are represented by property and individual respectively.
(racer, etc) for logical model to check whether the classes are mutually consistent or not. OWL interface is illustrated by Figure 7.

![Figure 7. OWL Interface](image)

### 2.4 Jess

A rule engine is needed to manage the interaction between service system and mobile devices. In this project, Jess was chosen as rule language to create rule policies.

#### 2.4.1 What is Jess [14]

Jess, Java Expert System Shell, is a rule engine and scripting environment written entirely in Sun's Java™. It is a tool for building a type of intelligent software called **Expert Systems**[^9]. The rules created by Jess are fired to take actions based on facts which in this project are items related to instances of classes. Jess rules can be compared to *if* … *then* statements in procedural language. However, since Jess uses **Rete**[^10] algorithm, rules are executed whenever their *if* parts (their

[^9]: An Expert System is a set of *rules* that can be repeatedly applied to a collection of *facts* about the world.
left-hand-sides or LHSs) are satisfied, which makes Jess less deterministic than typical procedural language. This makes Jess rules running faster than normal if ... then statements. An example and description of Jess rules are shown below.

```jess
Jess> (defrule allowed-person
"If a person is older than 22, print his (her) name."
?c <- (object (age ?x&:(> ?x 22)))
=>
(printout t (slot-get ?c name) " is older than 22." crlf))
```

This rule has two parts, separated by the "=>" symbol (which you can read as "then"). The first part consists of the LHS: “?c <- (object (age ?x&:(> ?x 22)))”. The second part consists of the corresponding RHS (right-hand-sides) action: “(printout t (slot-get ?c name) " is older than 22." crlf)”. This rule means whenever a person (instance) whose age (property) is more than 22, print his (her) name (property). Readers can know more details about Jess rules in chapter 6.

### 2.5 JessTab

Since we want to make the Protégé ontology and Jess rules work simultaneously to simulate some interactions under different circumstances, a tool is needed to achieve this goal, **JessTab**[^1] is such a plug-in for Protégé which provides a console window to make Protégé interact with Jess when Protégé is running.

[^1]: [http://www.ida.liu.se/~her/JessTab/](http://www.ida.liu.se/~her/JessTab/)
Through JessTab, Protégé instances can be mapped into Jess facts and we could create rules which would directly operate on Protégé knowledge base (an aggregation of instances). Let’s see how the example mentioned in 2.4.1 works on the Protégé.
Figure 9. First, map all the instances of class mankind to Jess facts, then input the rule, finally, fire the rule and get one result.
Chapter 3
Related Work
This chapter gives an overview of some accomplished research work and projects which have some common features this project could share with.

3.1 Ambient Intelligence
“Ambient intelligence is an exciting new information technology paradigm in which people are empowered through a digital environment that is aware of their presence and context and is sensitive, adaptive, and response to their needs.” [15] There have been some projects and systems constructing ambient intelligence environment, such as Philips Nebula [16], iDorm [15], and others. Here, iDorm is exemplified to detail the character of ambient intelligence environment.

The iDorm [15] is an intelligent dormitory which creates an ambient-intelligence environment by using embedded agents. It works for it as a test bed in ubiquitous-computing [17][18] environment. The devices which contain embedded agents include:

a. The static iDorm embedded agent receives sensors (such as temperature sensor, smoke detector, etc) data containing user’s learned behaviour through wire network, computes appropriate control actions and send them back to iDorm effectors across the network.

b. A mobile service robot performs a servant-gadget for delivering various objects of interest to the iDorm users such as food, drink, and medicine.

c. A portable iPAQ (poket PC) interface or a portable mobile-phone interface monitors and controls the iDorm through wireless network (Bluetooth).

These agents communicate and interact with each other through wireless network: The static iDorm embedded agent could send task request to the robot which inform back its current status. The iPAQ could get the current status information of iDorm and send control action to the static iDorm embedded agent.

3.2 Context-Aware System [21]
“Pervasive-computing research shows that a user’s environment is dynamic, adaptive, and interactive. It contains sensors, wireless devices, and personal and service agents operating autonomously with the aid of context.” Context-aware system is such a multi-agent middleware to “make the environment more easily personalized at runtime and adapted and managed at provisioning time” [21] by using a negotiation protocol and ontology model. The agents inside this system include: user and service agents, context management agent, inference agent, inference service.

12 http://www.webopedia.com/TERM/P/pervasive_computing.html
ontology agent, context provider agents. The functions which these agents could provide are categorized into three modules:

a. **Context representation module** (access, storage and query) “gathers context from the surrounding area and provides it in an ontology-based representation that facilitates interpretation and inference.” [21]

b. **Context communication module** (discovery, delivery and negotiation) “governs communication between agents using a registration process …” [21]

c. **Context management module** (interpretation, engagement and inference) “provides methods for querying and storing context information.” [21]

Based on context-level negotiation, these agents work together to take care of requests between users and services (email, printing, entertainment, etc).

A few research projects have been developed by applying context-aware theories. Two examples are given here:

- **FLAME2008** [19]: provides a platform which offers mobile users personalized context- and situation-aware service in mass events like the Olympic Games 2008. In FLAME 2008, “Context values of all users are gathered automatically by sensors, and whenever a user significantly changes his context, his situation is derived dynamically by the inference engine. Finally, the service matching provides as a result all offers which fit to the user’s situation and profile, grouped into categories. This updates the actual set of recommended services at the user’s PDA or smart phone. All this is done using inference based on a modular ontology.” [19]

- **Context Studio** [20]: is “an application personalization tool for semi-automated context-based adaptation, was further developed for the end users of small-screen mobile devices.” [20] In this application, the context ontology vocabulary hierarchy is transformed into a folder-file model representation which allows easy navigation and is viewed through a graphical user interface. The context and action values are classified into concept folders according to their types and could be updated and presented in the UI at runtime.

All intelligent systems mentioned above provide their services or activities to users based on different types of mechanism (rules, methods, etc).
Chapter 4

Method Description

This chapter summarizes the method of building the ontology and creating the rules.

4.1 Modelling

Modelling procedure is a gradual and iterative process. Before actually starting to build the ontology, an initial process should be prepared.

1. Define domain of ontology
   First, what type this ontology belongs to, what scope this ontology could cover, who are going to use and develop this model and what useful information it could provide them.

2. List concepts
   Second, what items could be included into this ontology (they will be revised and expanded during the design procedure), what properties these concepts could have to relate them.

3. Defining classes and Arrange classes hierarchy
   Then, after having listed all the terms needed for this ontology, we have to determine classes definition and what their hierarchy look like (also check classes consistency) according to axioms and some common senses. For example, there are classes Mankind, Male, Female and both male and female belong to mankind.

4. Defining properties and their restrictions
   The fourth part we should think about is what properties could describe a specific class and the internal structures of classes. In Protégé OWL, properties represent relationships between two individuals [23]. Object properties which have Instance and Class value type and Datatype properties which have Boolean, String, Integer, etc value type are two major OWL property types. Most of the properties used in this project are Object properties. For example, a Mankind could have hasName, hasAge, hasNationality, etc properties. Then we have to set up the value type (string, integer or instance, etc), allowed value number (cardinality) of a property and other restrictions like Universal restriction (\(\forall\)) which makes a given property only have relationships to a specific class [23], for example, Universal Restriction can be added on Object property nationality to class Nation (\(\forall hasNationality Nation\)). Figure 10 shows this situation.
Afterwards, another kind of restriction range would be very convenient to add on some object properties which are instance type to create individuals later. For example, if range Nation is added to property hasNationality, see Figure 11 and Figure 12:

![Figure 11. hasNationality has range (allowed class) Nation](image-url)
5. **Insert Individuals**

Finally, some individuals should be created for classes. The purpose of inserting individuals in this project is to provide corresponding entities for simulating of Jess rules according to different situations. So what we should think about in this part is to investigate practical entities (for creating individuals) and their features (for filling in the related property values) about each class.

### 4.2 Jess Rules

After building the model, the next topic is to create rules. In this part, there are several instructions to guide the procedure of rules creation.

- Since we want to apply rules on simulating the interaction between service system and mobile units, the rule policies should be thought about from two sides: service system and mobile units. The problems in this section are:
  - From system side: what requirements should be put on the properties of each service item. From mobile units side, what property values each mobile
device should send to the system. These problems care about what could be put into *if* conditions (LHS).

- Certainly, we also have to think about what actions could be done after rules fired. This problem cares about *then* actions (RHS).

- Since we want to simulate the procedure of interaction, the corresponding rules could be fired in real time according to different situations.

- Is there a criterion of creating rules to let future developers develop new rules efficiently? If so, what does it look like?
Chapter 5

Developing the Ontology

We have talked about a preliminary process of creating the ontology. This chapter will give the details of modelling procedure.

Since ontology design is an iteratively refined process, we will describe the designing procedure from two stages.

5.1 Building Initial Ontology

During the first stage, an initial model of the project is built. This model will be revised on the second stage afterwards.

5.1.1 Determine Domain

- **User Group**
  The project is designed specifically to provide a model for the companies and some academic departments who want to design and produce some kind of service systems. Through this model, designers could have some initial thoughts of designing a service system, especially the rule engine of the system.

- **Domain**
  The domain of this model is oriented to mobile units which include Cellphone, PDA (Personal Digital Assistant), Laptop (see Figure 13). It also integrates the knowledge about communication schemes (Bluetooth, infrared, etc), operating system, etc, so this ontology belongs to Application Ontology.
5.1.2 Defining Classes

According to Figure 13, the major classes could be firstly defined with ServiceSystem and Device which has subclasses Laptop, PDA, Cellphone. It is a model between mobile units and service system, some other classes are needed to represent the features of mobile devices to connect them together, especially for creating the rules later on. So class Features which consists of subclasses Communication, OperatingSystem, Pad, Screen is defined. Figure 14 shows classes hierarchy.
Class **Communication** represents the wireless communication schemes (Wi-Fi\(^{13}\), BT\(^{14}\), Infrared) by which mobile units interact with service system. Class **Pad** represents the platforms (Keypad, Keyboard, Touchpad) on which users could operate devices. Class **Screen** represents some parameters related to the screen of mobile devices, like displaying pixels, etc. Through some practical investigation and actual information, we think these classes could be the necessary factors which deal

\(^{13}\) Wireless Network
\(^{14}\) Bluetooth
with the interaction between mobile devices and service system.

5.1.3 Defining Properties

After determining classes, properties need to be defined to describe the relations among classes. On one hand, one Device needs different Features to describe its related parameters sent to ServiceSystem. On the other hand, ServiceSystem provides corresponding services to this Device according to inspecting these parameters. So in order to make individuals from Device have relationships with individuals from various Features and have services from ServiceSystem, Object properties below are defined: hasCommunication hasOperatingSystem hasPad hasScreen hasService to Device. See Figure 15.

![Diagram showing relationships between Communication, OperatingSystem, Pad, Screen, ServiceSystem, and Device with properties such as hasCommunication, hasOperatingSystem, hasPad, hasScreen, and hasService.](image-url)
5.1.4 Defining Restrictions

From Figure 15, it can be obviously seen that all properties of Device are defined as Instance type and ranged specifically to each subclass of Features and class ServiceSystem. Let’s take a look at an example: property hasCommunication.

![Figure 16. Property: hasCommunication](image)

Why ranges of Features are defined to properties of Device is because the specification of data sent to a service system should have a standard. Then cardinality restrictions are defined to properties of Device. Some known facts about mobile devices:

1. Each mobile device has only one operating system while running.
2. Some mobile devices have been equipped with more than one pad, for example, Sony PEG-UX50 (PDA) has both Touchpad and Keyboard.
3. Some mobile devices have more than one communication ports, for example, Acer TravelMate 661xvi (Laptop) has connectivity through Wi-Fi, Bluetooth and Infrared. But in this project, it is assumed that a device can connect service system through only one communication way at one time.
4. One single device can run only one service application provided by ServiceSystem. Cardinality restrictions to properties of Device are illustrated through Conditions Widget, see Figure 17.
5.2 Revising Ontology

After the initial ontology has been created, next work is to revise it more specifically to apply on this project, especially for creating the rules.

5.2.1 Creating Subclasses

- **Cellphone**: With the development of mobile technology, there emerged two kinds of cellphone: smartphone and PDAphone. Besides basic functions of ordinary cellphones, they provide more advanced, powerful capabilities to users, such as wireless e-mail, personal information management, remote data transfer between phone and computers, remote control of computers and other electronic equipments, etc. Compared to typical cellphone, the two categories of cellphone (1) have larger, high pixels display (2). incorporate more powerful processors and run completely high-level operating system which provides platform for application developer. The distinctions between them are: PDAphone inherit PDA features with using Touchpad as input interface and having larger display while smartphone use standard cellphone keypad. Based on description above, two subclasses of Cellphone are created: Smartphone and PDAphone. Actually, most of current cellphones which can make use of services provided by service system could be categorized into these two types.

- **OperatingSystem**: Since operating systems running on Laptop are more powerful than those for PDA and Cellphone, OperatingSystem should be categorized into two subclasses: OS_Cellphone_PDA and OS_Laptop.

- **Screen**: Normally, several concepts could describe features of screen on mobile devices. Through investigation so far, service requirements for screen mainly care about the screen size (display pixels). In this project, a subclass of Screen is created: ScreenSize.

- **ServiceSystem**: A service system can provide different types of service, and each
type has a few applications. Three categories of services are assumed in this project: **TV-APP**, **Game**, **OtherService**. Since games running on laptop are more delicate than those running on PDA or cellphone, two subclasses are created for **Game**: **Type1** (for PDA and Cellphone) and **Type2** (for Laptop). New classes hierarchy is shown below:
5.2.2 New Properties and Restrictions

- **Device**: In this project, it is assumed that all devices have color display. A datatype property of **Device** is defined: **hasColorScreen** which has Boolean value type and **true** hasValue (⊇) restriction. It is shown below:
PDAPhone: Since both PDA phone and PDA (some of which also have keyboard) use touch screen for input, a datatype property is defined to PDAPhone and PDA: hasTouchpad which has Boolean value type and true hasValue(∈) restriction. It is shown through Figure 20.

![Figure 20. New Property for PDA and PDAPhone](image)

- **SmartPhone** and **Laptop**: It's obvious that the laptop (some of which have touch screen) use keyboard for input. And as it is mentioned above that the smart phone use keypad for input, datatype properties: hasKeyPad and hasKeyboard defined to SmartPhone and Laptop respectively.
- **OS_Cellphone_PDA**: Most current games which cellphone and PDA can download and other services require their operating systems to support JAVA. So a datatype property: hasjava which has Boolean value type and true hasvalue (∈) restriction.
- **ScreenSize**: As it is mentioned previously, in this project, ScreenSize represents the display pixels of mobile devices. Display pixels could be represented by Height * Width (dot-matrix). So two datatype properties are defined to ScreenSize: hasHeight and hasWidth which have Integer value type, See Figure 21:
5.2.3 Inserting Instances

The last major work of accomplish this ontology is to create instances (individuals) of the classes. Since all object properties of Device (hasCommunication, hasOperatingSystem, hasPad, hasScreen, hasService) are ranged to each subclass of Features respectively and ServiceSystem, which means that these properties value of individuals of Device will filled in with the instances from subclasses of Features and ServiceSystem, the procedure of inserting instances starts with creating individuals for Features and ServiceSystem.

- Features
  - Communication: Bluetooth, infrared and Wi-Fi (wireless network) are three main wireless technology which could be equipped with mobile devices. Under individuals tab, three instances of Communication are created: BT, Infrared, Wi-Fi (Figure 22).
OperatingSystem

- **OS_Cellphone_PDA**: Symbian, window mobile, palm and linux are the only platforms which can provide operating systems for cellphone or PDA so far. So instances: Symbian, WindowsMobile, Palm and Linux are created for OS_Cellphone_PDA.

- **OS_Laptop**: Most personal laptops are installed with windows OS in different versions. Besides, there are several other OS installed in users’ laptops, such as mac os (mainly for apple laptop), unix and so on. Instances of laptop OS created for this project are the commonest ones: Windows2000, WindowsXP, Mac OS, Unix.

Pad: Key-pad, key-board, touch-pad are only three interfaces so far which mobile devices can use for input. Instances of Pad consist of: Keypad, Keyboard, Touchpad.

Screen

- **ScreenSize**: Instances created for ScreenSize in this project are associated with the instances of Device. SS1 – SS8 are created for Cellphone and PDA, SS9 is created for laptop (In general, most current laptops have 1024 * 768 display pixels, see Figure 23).
Figure 23. Instances of ScreenSize

- **ServiceSystem**: In this project, a few service instances are assumed to each subclass of ServiceSystem. Game1, Game2, Game3 are created for Type1. Game01, Game02, Game03 are created for Type2. Component1, Component2, Component3 are created for TV-APP (see Figure 24). Print is created for OtherService.

Figure 24. Instances of ServiceSystem

- **Device**: According to the definition of subclasses of Device, some instances are created for each subclass and their properties\(^{15}\) are filled in by adding instances

\(^{15}\) Property hasService acts as a part of rules creating and performs a role of simulating what service has been running on the device. So the value for it will be inserted by taking actions of rules.
from Features. Let’s take a look at one example (Figure 25)

![Figure 25. An Instance of PDA: SONY PEG-UX50](image)

### 5.2.4 Final Accomplishment

First, from description in 5.2.3, classes Communication and Pad could be defined as Enumerated class which contains and only contains individuals created previously. They could be defined as a NECESSARY & SUFFICIENT condition (Figure 26).
Second, write some comments for classes, properties and instances to explain what these terms represent or supplement some information. Figure 27 gives an example.

Finally, check consistency of this ontology [23]: Invoke the reasoner (Racer is used in this project) first, then click on the button Classify taxonomy on the Protégé-OWL toolbar, afterwards, inferred hierarchy\(^\text{16}\) window will be pop open next to the existing asserted hierarchy\(^\text{17}\) window. This situation is shown below: Figure 28

\(^{16}\) It is manually constructed [23].
\(^{17}\) It is automatically computed by reasoner [23].
As a result of using reasoner, this ontology is logically consistent.

So far, the work of building model is completed. Since an ontology can’t contain all the possible information about the domain [4], this ontology is just an initial model for this domain. Part of information in it is specialized and assumed for creating rules. It would be revised and developed by developers for other applications.
Chapter 6

Creating Rules

Rule policies of a service system mainly take care of this situation: what service options could be provided to a mobile unit which is requesting service. See Figure 29.

![Figure 29. Interaction Procedure between service system and a mobile unit](image)

The whole role policies consist of a series of rules which deal with the relation between each specific service and mobile devices. So the topic in this section describes mostly about the rule for each single service.

6.1 Situation Design

Situation design talks about what requirements a single service ask to allow service request, viz. what should be put into the LHS (if conditions). For example, service TV-APP (still see Figure 13), requires there are 3 mobile units sending requests simultaneously which include a laptop for TV screen, a cellphone (or a smart phone) for remote control, a PDA (or a PDA phone) for other components. The requirements for each unit are:
### Requirements

<table>
<thead>
<tr>
<th>Units</th>
<th>connection</th>
<th>minimum display</th>
<th>pad</th>
<th>operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>Bluetooth or Wi-Fi</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cellphone</td>
<td>200 * 160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDA</td>
<td>Bluetooth or Wi-Fi</td>
<td>320 * 300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The blank items of each unit in this chart indicate no particular requirements for these properties of units.

This work could be done through adding comments to instances of `ServiceSystem` created already (see Figure 30).

![Figure 30. Requirements for TV-APP](image)

### 6.2 Programming Rules

After the situations for each service have been set up, rules for these services could be designed. For example, according to requirements for TV-APP, rule for it could be divided into 4 subsections, Let’s take a look at one of them:

```jess
Jess> (defrule service-TV1
"What kind of situation does system can provide TV application in"
?a <- (object (is-a Laptop) (hasCommunication ?com1) (hasService nil))
    (object (is-a Communication) (OBJECT ~?com1) (:NAME "Infrared"))
?b <- (object (is-a Cellphone) (hasScreen ?scr1) (hasService nil))
```

18 Since one device can operate only one service at one time, the value of `hasService` must be null before getting any service.
(object (is-a ScreenSize) (OBJECT ?scr1) (hasHeight ?h1&:(> ?h1 200))
(hasWidth ?w1&:(> ?w1 160)))
?c <- (object (is-a PDA) (hasCommunication ?com2) (hasScreen ?scr2) (hasService
nil))
  (object (is-a Communication) (OBJECT ~?com2) (:NAME "Infrared")))
  (object (is-a ScreenSize) (OBJECT ?scr2) (hasHeight ?h2&:(> ?h2 320))
(hasWidth ?w2&:(> ?w2 300)))
?add1 <- (object (is-a TV-APP) (:NAME "Component1"))
?add2 <- (object (is-a TV-APP) (:NAME "Component2"))
?add3 <- (object (is-a TV-APP) (:NAME "Component3"))
  =>
  (printout t "(Laptop) " (slot-get ?a :NAME) " can get TV screen; (Cellphone) "
(slot-get ?b :NAME) " can get TV remote; (PDA) " (slot-get ?c :NAME) " can get other
items of TV." crlf)
)

The LHS (separated by =>) of this rule describes the conditions which **TV-APP**
requires, and the RHS of this rule provides what actions will be taken if all the LHS
conditions have been satisfied. The last sentence of this rule, “(printout t … crlf)”
performs a role of displaying result, especially for simulation of displaying service
menu (Figure 29). Readers can find all the rules code so far on Appendix A.
Chapter 7

Discussion
This chapter will discuss what tasks could be accomplished in this project and why some positive results could not be seen so far.

7.1 Accomplishments

7.1.1 Ontology
In this project, a model of mobile units is presented by using ontology. As an application ontology, this model has been created tightly related with the procedure of creating rules. A few properties of Device (hasCommunication, hasPad, etc) and the instances created for classes of Features (Communication, Pad, etc) were considered as the parameters of rule conditions. So it wasn’t thought about separately to model and create rules in this project. However, it still could be a reference model for service system designer or normal readers to learn about some basic concepts and the relations between them. In the future, this model could be developed or reclassified according to different application requirements.

7.1.2 Rules
Jess rules specify a set of actions which could be performed for a given situation. It’s the merit of Jess to be applied in this project to simulate the interactions.

After loading the rules which have been already created into Protégé and firing them through JessTab, the procedure of simulating interaction could be described briefly below:
1. Mapping all the instances of the ontology into Jess facts. This step could be considered in the actual environment that the service requests including the corresponding parameters about the mobile devices are sent to the service system.
2. Checking the information of facts according to the conditions of rules. Since the rules created in this project are specifically designed for each service, the data of mobile devices will be verified according to each service requirements.
3. Providing service. Based on the verifying results, what mobile units could get what services will be printed out and the corresponding service instance will be inserted into the satisfied Device instances.

Through the description of simulation above, the basic procedure (service request, verifying received data, providing service) of interaction between mobile devices and service system could be accomplished by this project.
7.2 Limitations

- **Ontology:** Since modelling and creating rules were considered together during this project, some concepts of ontology could not be defined according to actual situation for Jess restrictions. For example, normally a laptop has more than one communication interface (Bluetooth, infrared, Wi-Fi), but property `hasCommunication` has to be restricted with single value because Jess rules can’t detect more than one value and read them all.

- **Rules:** First, since the whole project was not initially designed to apply on real devices, rules created in this project could not be transformed into real devices or system and perform any interaction in the practical environment. Second, since the interactions was implemented on a single machine, it could not simulate the actual situations completely because the communication part was ignored in this project. Once Jess rules will be applied on some prototype of service system, it has to combine some other platforms which could process the data received from communication channels first and then transform it into Jess formats to do next work. At this point, Jess rules are suitable to be developed under research environment instead of practical situations. Finally, since rules created in this project could not finish some tasks, like detecting how many values a property has as described above, is it the only way to use Jess rules simulating the interactions on this model created by ontology has not been proved so far.
Chapter 8

Future Work and Conclusion

This chapter presents some future work which might be done to revise and develop this project, and finally the whole project will be summarized.

8.1 Future Work

The main contribution of this project is to use Jess rules simulating the interactions between mobile units and service system which are modeled with an ontology. Correspondently, Future work is supposed to think about both for ontology and for rules.

For ontology

Since it’s an initial model of mobile units, some improvements could be done by developers to make it shared well and easily in practical circumstances, such as, modify current information, add new classes or subclasses and so on.

For rules

Compared to the model, more developments could be predicted for rules.

- First, since Jesstab is a Protégé plug-in oriented to standard file, there are some incompatible problems when it is configured to OWL file. For example, function (set-kb-save <definition-type>|* <definition-name>|* TRUE|FALSE) which save Jess definitions together with Protégé project can’t be applied in this project. These problems might be fixed through modifying JessTab API for Protégé.

- Second, function deffuntion can not be applied in this project because of incompatible problem. It will make rules look more simple and readable by using deffuntion. For example: If several rules perform the same actions (print, insert instances), these actions could be written into a function by using deffuntion. Just input function name and related parameters when creating the actions of these rules.

- Third, in this project, most properties in this model are defined with cardinality restriction, which require these properties have only single value. However, if a new property will not be restricted with value number, how to detect how many values this new property has, read them and do some actions are not found yet.

- Finally, another future task could be thought about. The way to create rules in this project is specifically related with the requirements of each service. It’s awkward and inconvenient for rule developers to create new rules. There should

\[19\] For example, define a new class Brand representing the corporations which produce mobile devices, like SONY, SIEMENS, NOKIA, etc.
be a more general way to do it so that developers don’t have to know about too much specific information before creating rules. For example, a new class Requirement could be defined, and properties and instances could be also defined according to each service and their requirements. Rule developers don’t need to know the details of these service requirements, just create rules via reading information from the instances of Requirement and taking corresponding actions.

8.2 Conclusion

This project provides a model of mobile devices and a service system. The description of this model (ontology) combined with some rules would be helpful to design a service system, especially the rule engine.

This model is created by Protégé – OWL, through which a clear hierarchy of defined classes with related information (properties, instances, comments) is illustrated. It aims to give designers of service system a rudimental prototype about different mobile devices, their features, and internal structures, etc.

Jess rules created for this model gives a way to simulate the interaction between mobile units and service system. They provide some basic thoughts of how to design rule policies for the rule engine of a service system. Even though these rules couldn’t applied on real situations so far, they still could be employed into some research prototypes of service system to test how they are going to work in actual circumstances.

This project shows how developers and system designers can take advantage of such a model in both designing as well as deploying information system for mobile devices. The rule engine also gives developers a high level way of defining requirements as well as behavior of the system as a whole. The project also demonstrates how services can be split up into sub-services (components), to allow application supporting distributed user interfaces to be developed.
References


Appendix A

User Manual: Copy all the codes below and paste them into a new text file saved as for example: “D:\mobile\rule_list.txt”. Then copy the first line of the file except semicolon “(batch “D:\mobile\rule_list.txt”)” to the command line of Jess widget when running Protégé.

; (batch "D:\mobile\rule_list.txt")

(mapclass Device)
(mapclass Features)
(mapclass ServiceSystem)

; 1(1).
(defrule service-TV1 "What kind of situation does system can provide TV application in"
    ?a <- (object (is-a Laptop) (hasCommunication ?com1) (hasService nil))
    (object (is-a Communication) (OBJECT ~?com1) (:NAME "Infrared"))
    ?b <- (object (is-a Cellphone) (hasScreen ?scr1) (hasService nil))
    (object (is-a ScreenSize) (OBJECT ?scr1) (hasHeight ?h1&:(> ?h1 200))
    (hasWidth ?w1&:(> ?w1 160)))
    ?c <- (object (is-a PDA) (hasCommunication ?com2) (hasScreen ?scr2) (hasService nil))
    (object (is-a Communication) (OBJECT ~?com2) (:NAME "Infrared"))
    (object (is-a ScreenSize) (OBJECT ?scr2) (hasHeight ?h2&:(> ?h2 320))
    (hasWidth ?w2&:(> ?w2 300)))

    ?add1 <- (object (is-a TV-APP) (:NAME "Component1"))
    ?add2 <- (object (is-a TV-APP) (:NAME "Component2"))
    ?add3 <- (object (is-a TV-APP) (:NAME "Component3"))

    =>
    (slot-set ?a hasService ?add1)
    (slot-set ?b hasService ?add2)
    (slot-set ?c hasService ?add3)

    (printout t "1.TV: (Laptop) " (slot-get ?a :NAME) " can get TV screen; (Cellphone) "
    (slot-get ?b :NAME) " can get TV remote; (PDA) " (slot-get ?c :NAME) " can get other items of
    TV." crlf)
)

; 1(2).
(defrule service-TV2 "What kind of situation does system can provide TV application in"
    ?a <- (object (is-a Laptop) (hasCommunication ?com1) (hasService nil))
    (object (is-a Communication) (OBJECT ~?com1) (:NAME "Infrared"))
    ?b <- (object (is-a Cellphone) (hasScreen ?scr1) (hasService nil))
    (object (is-a ScreenSize) (OBJECT ?scr1) (hasHeight ?h1&:(> ?h1 200))
    (hasWidth ?w1&:(> ?w1 160)))

    ?add1 <- (object (is-a TV-APP) (:NAME "Component1"))
    ?add2 <- (object (is-a TV-APP) (:NAME "Component2"))
    ?add3 <- (object (is-a TV-APP) (:NAME "Component3"))

    =>
    (slot-set ?a hasService ?add1)
    (slot-set ?b hasService ?add2)
    (slot-set ?c hasService ?add3)

    (printout t "2.TV: (Laptop) " (slot-get ?a :NAME) " can get TV screen; (Cellphone) "
    (slot-get ?b :NAME) " can get TV remote; (PDA) " (slot-get ?c :NAME) " can get other items of
    TV." crlf)
)
(object (is-a ScreenSize) (OBJECT ?scr1) (hasHeight ?h1&:(> ?h1 200))
(hasWidth ?w1&:(> ?w1 160)))

?c <- (object (is-a PDA) (hasCommunication ?com2) (hasScreen ?scr2) (hasService nil))

(object (is-a Communication) (OBJECT ~?com2) (:NAME "Infrared"))
(object (is-a ScreenSize) (OBJECT ?scr2) (hasHeight ?h2&:(> ?h2 320))
(hasWidth ?w2&:(> ?w2 300)))

?add1 <- (object (is-a TV-APP) (:NAME "Component1"))
?add2 <- (object (is-a TV-APP) (:NAME "Component2"))
?add3 <- (object (is-a TV-APP) (:NAME "Component3"))

=>
(slot-set ?a hasService ?add1)
(slot-set ?b hasService ?add2)
(slot-set ?c hasService ?add3)

(printout t "1.TV: (Laptop) " (slot-get ?a :NAME) " can get TV screen; (Cellphone) "
(slot-get ?b :NAME) " can get TV remote; (PDA phone) " (slot-get ?c :NAME) " can get other
items of TV." crlf)
)

; 1(3).
(defrule service-TV3 "What kind of situation does system can provide TV application in"
?a <- (object (is-a Laptop) (hasCommunication ?com1) (hasService nil))

(object (is-a Communication) (OBJECT ~?com1) (:NAME "Infrared"))
?b <- (object (is-a Smartphone) (hasScreen ?scr1) (hasService nil))

(object (is-a ScreenSize) (OBJECT ?scr1) (hasHeight ?h1&:(> ?h1 200))
(hasWidth ?w1&:(> ?w1 160)))

?c <- (object (is-a PDA) (hasCommunication ?com2) (hasScreen ?scr2) (hasService nil))

(object (is-a Communication) (OBJECT ~?com2) (:NAME "Infrared"))
(object (is-a ScreenSize) (OBJECT ?scr2) (hasHeight ?h2&:(> ?h2 320))
(hasWidth ?w2&:(> ?w2 300)))

?add1 <- (object (is-a TV-APP) (:NAME "Component1"))
?add2 <- (object (is-a TV-APP) (:NAME "Component2"))
?add3 <- (object (is-a TV-APP) (:NAME "Component3"))

=>
(slot-set ?a hasService ?add1)
(slot-set ?b hasService ?add2)
(slot-set ?c hasService ?add3)

(printout t "1.TV: (Laptop) " (slot-get ?a :NAME) " can get TV screen; (Cellphone) "
(slot-get ?b :NAME) " can get TV remote; (PDA phone) " (slot-get ?c :NAME) " can get other
items of TV." crlf)
)
(defrule service-TV4 "What kind of situation does system can provide TV application in"
  ?a <- (object (is-a Laptop) (hasCommunication ?com1) (hasService nil))
  (object (is-a Communication) (OBJECT ~?com1) (:NAME "Infrared"))
  ?b <- (object (is-a Smartphone) (hasScreen ?scr1) (hasService nil))
  (object (is-a ScreenSize) (OBJECT ?scr1) (hasHeight ?h1&:(> ?h1 200))
  (hasWidth ?w1&:(> ?w1 160)))
  ?c <- (object (is-a PDATelephone) (hasCommunication ?com2) (hasScreen ?scr2))
  (hasService nil)
  (object (is-a Communication) (OBJECT ~?com2) (:NAME "Infrared"))
  (object (is-a ScreenSize) (OBJECT ?scr2) (hasHeight ?h2&:(> ?h2 320))
  (hasWidth ?w2&:(> ?w2 300)))
  ?add1 <- (object (is-a TV-APP) (:NAME "Component1"))
  ?add2 <- (object (is-a TV-APP) (:NAME "Component2"))
  ?add3 <- (object (is-a TV-APP) (:NAME "Component3"))
  =>
  (slot-set ?a hasService ?add1)
  (slot-set ?b hasService ?add2)
  (slot-set ?c hasService ?add3)
  (printout t "1.TV: (Laptop) " (slot-get ?a :NAME) " can get TV screen; (Smartphone)"
  " (slot-get ?b :NAME) " can get TV remote; (PDA phone) " (slot-get ?c :NAME) " can get other"
  "items of TV." crlf))

; 2.
(defrule service-print "What kind of situation does system can provide print service in"
  ?c <- (object (hasCommunication ?com) (hasService nil))
  (object (is-a Communication) (OBJECT ?com) (:NAME "BT"))
  ?p <- (object (is-a OtherService) (:NAME "Print"))
  =>
  (slot-set ?c hasService ?p)
  (printout t "2.Other Service: " (slot-get ?c :NAME) " can get print service." crlf))

; 3(1).
(defrule service-game1_1 "What kind of situation does system can provide Game 1 service in"
  ?c <- (or (object (is-a Cellphone) (hasScreen ?s) (hasService nil))
  (object (is-a Smartphone) (hasScreen ?s) (hasService nil))
  )
  (object (is-a ScreenSize) (OBJECT ?s) (hasHeight ?h&:(> ?h 200))
  (hasWidth ?w&:(> ?w 160))))
; 3(2).
(defrule service-game1_2 "What kind of situation does system can provide Game 1 service in"
  ?c <- (object (is-a ~Laptop) (hasScreen ?s) (hasPad ?p) (hasService nil))
    (object (is-a ScreenSize) (OBJECT ?s) (hasHeight ?h&:(> ?h 200))
      (hasWidth ?w&:(> ?w 160)))
    (object (is-a Pad) (OBJECT ?p) (:NAME "Keyboard")))
  ?g1 <- (object (is-a Type1) (:NAME "Game1"))
  =>
    (slot-set ?c hasService ?g1)
    (printout t "Game 1: " (slot-get ?c :NAME) " can download Game 1." crlf)
)

; 4.
(defrule service-game2 "The units including PDA and PDA phone which have touchpad could download Game 2"
  ?c <- (or (object (is-a PDA) (hasService nil))
            (object (is-a PDAPhone) (hasService nil)))
  ?g2 <- (object (is-a Type1) (:NAME "Game2"))
  =>
    (slot-set ?c hasService ?g2)
    (printout t "Game 2: " (slot-get ?c :NAME) " can download Game 2." crlf)
)

; 5.
(defrule service-game3 "Game 3 is an interacting game which requires two units have windows mobile operating system and bluetooth communication"
  ?a <- (object (is-a ~Laptop) (OBJECT ?x) (hasCommunication ?com1)
         (hasOperatingSystem ?o1) (hasScreen ?s1) (hasService nil))
    (object (is-a Communication) (OBJECT ?com1) (:NAME "BT"))
    (object (is-a OS_Cellphone_PDA) (OBJECT ?o1) (:NAME "WindowsMobile"))
    (object (is-a ScreenSize) (OBJECT ?s1) (hasHeight ?h1&:(> ?h1 120))
      (hasWidth ?w1&:(> ?w1 100)))
  ?b <- (object (is-a ~Laptop) (OBJECT ~?x) (hasCommunication ?com2)
(hasOperatingSystem ?o2) (hasScreen ?s2) (hasService nil)
  (object (is-a Communication) (OBJECT ?com2) (:NAME "BT"))
  (object (is-a OS_Celphone_PDA) (OBJECT ?o2) (:NAME "WindowsMobile"))
  (object (is-a ScreenSize) (OBJECT ?s2) (hasHeight ?h2:&:(> ?h2 120))
  (hasWidth ?w2:&:(> ?w2 100)))

?g3 <- (object (is-a Type1) (:NAME "Game3"))
=>
  (slot-set ?a hasService ?g3)
  (slot-set ?b hasService ?g3)
  (printout t "Game 3: " (slot-get ?a :NAME) " and " (slot-get ?b :NAME) " can play Game 3." crlf)
)

; 6.
(defrule service-game01 "Game 01 is a simple downloading game running on Windows OS"
  ?c <- (object (is-a Laptop) (hasOperatingSystem ?o) (hasService nil))
  (or (object (is-a Communication) (OBJECT ?com1) (:NAME "BT"))
    (object (is-a Communication) (OBJECT ?com1) (:NAME "Wi-Fi"))
  )
  (or (object (is-a OS_Laptop) (OBJECT ?o1) (:NAME "Windows2000"))
    (object (is-a OS_Laptop) (OBJECT ?o1) (:NAME "WindowsXP"))
  )
  ?g01 <- (object (is-a Type2) (:NAME "Game01"))
=>
  (slot-set ?c hasService ?g01)
  (printout t "Game 01: " (slot-get ?c :NAME) " can download Game 01." crlf)
)

; 7.
(defrule service-game02 "Game 02 is an interacting Windows game requiring high system support"
  ?a <- (object (is-a Laptop) (OBJECT ?x) (hasCommunication ?com1) (hasOperatingSystem ?o1) (hasService nil))
  (or (object (is-a Communication) (OBJECT ?com1) (:NAME "BT"))
    (object (is-a Communication) (OBJECT ?com1) (:NAME "Wi-Fi"))
  )
  (or (object (is-a OS_Laptop) (OBJECT ?o1) (:NAME "Windows2000"))
    (object (is-a OS_Laptop) (OBJECT ?o1) (:NAME "WindowsXP"))
  )
  ?b <- (object (is-a Laptop) (OBJECT ~?x) (hasCommunication ?com2) (hasOperatingSystem ?o2) (hasService nil))
  (or (object (is-a Communication) (OBJECT ?com2) (:NAME "BT"))
    (object (is-a Communication) (OBJECT ?com2) (:NAME "Wi-Fi"))
  )
(or (object (is-a OS_Laptop) (OBJECT ?o2) (:NAME "Windows2000"))
   (object (is-a OS_Laptop) (OBJECT ?o2) (:NAME "WindowsXP")))

?g02 <- (object (is-a Type2) (:NAME "Game02"))
=>
   (slot-set ?a hasService ?g02)
   (slot-set ?b hasService ?g02)
   (printout t "Game 02: " (slot-get ?a :NAME) " and " (slot-get ?b :NAME) " can play Game 02." crlf)
)

; 8.
(defrule service-game03 "Game 03 is a game for Macintosh OS"
   ?c <- (object (is-a Laptop) (hasOperatingSystem ?o) (hasService nil))
   (object (is-a OS_Laptop) (OBJECT ?o) (:NAME "Macintosh"))

?g03 <- (object (is-a Type2) (:NAME "Game03"))
=>
   (slot-set ?c hasService ?g03)
   (printout t "Game 03: " (slot-get ?c :NAME) " can download Game 03." crlf)
)
Appendix B

Here is an example of the procedure of simulating interactions between service system and mobile units by using created rules.

1. Request and Service Options
   First, type (batch “C:\Documents and Settings\x04bosu\thesis\rule_display.txt”) into console window of JessTab, see Figure 31.

   ![Figure 31. Load Jess Rules](image)

   This command parses and evaluates the given file (rule_display.txt) as Jess code [14], return TRUE if successful.
From Figure 32, all rules have been loaded successfully into “Rule Engine”. There is one function “(mapclass Device)” inside the file (rule_display.txt) which maps all instances of Device into Jess facts. It could be imagined that the requests from the mobile units around service system have been sent to the “Rule Engine” of the system (Figure 33).
Second, type **run**, the rule engine will be activated to inspect the information received from units and take actions.
Figure 34. Result of Interaction

Figure 34 simulates displaying what service options transmitted back to mobile devices.

2. Service Provided
Some actions should be added to rules to simulate the interaction if a unit selects one service item of service menu (the last two steps of Figure 29). It could be shown by using example mentioned in 6.2, TV-APP:

```
defrule service-TV1 "What kind of situation does system can provide TV application in"
    ...
    =>
    (slot-set ?a hasService ?add1)
    (slot-set ?b hasService ?add2)
    (slot-set ?c hasService ?add3)
    ...
```

(slot-set ?a hasService ?add1) means the value of hasService is set with component1 (TV screen) for laptop if all conditions are satisfied. This procedure could be illustrated by Figure 35.
Figure 35. Toshiba_Portege_M200 get TV screen
Modelling of Interaction Units

Developing a model of a service system and mobile units including cellphone, PDA, Laptop is an important preliminary step of designing the systems which could provide these units some convenient and entertainment services through common short range communication like blue tooth, wireless LAN, etc.

In this project, an ontology is created to represent this model. Meanwhile, some basic service rules are also programmed and combined with this ontology can be used to simulate some interactions between items inside this model.

The description of this model (ontology) has been made through Protégé and demonstrated by using its graphical interface. The rules have been created by using Jess and implemented with the ontology by using JessTab.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Författarens Bo Sun]