

6. Our Proposal's Application

“Intelligent agents are proactive, autonomous, software tools and systems that can determine appropriate actions based on a range of data from multiple sources. Often they can ‘learn’ from experience. They enable systems to become ‘aware’ and respond intelligently to events.”

*David Ley Becta, “Emerging Technologies for Learning,”
ISBN 1 85379 467 8, Vol. 2, pp. 64-79, 2007.*

Although our approach is developed to deal with different application domains, in this Chapter we applied it to a ubiquitous application from the dental clinic cognitive domain. We already illustrated our proposal based on the *Domain Engineering of Ubiquitous Applications* point of view by focusing on the development for reuse. Now, we are interested in the *Ubiquitous Application Engineering* point of view by focusing on the development with reuse.

Our research group has been developing different ubiquitous case studies in the Software Engineering Laboratories at the PUC-Rio and UofT since 2007. In this thesis, we focus on our latest and most extensive case study from the dental clinic cognitive domain. It involves Ph.D. students and the clinic's stakeholders. The dental clinic is an academic dental clinic, which belongs to a dental association in the State of São Paulo, Brazil. Therefore, the clinic's members perform social activities by taking care of the community. Moreover, the clinic also contributes to the dentists' academic life by training them in different dental specialties.

The dental clinic application must deal with different smart-spaces, several content servers, heterogeneous devices, various quality criteria (e.g. privacy, mobility, cost, satisfaction, context-awareness, need for adaptability, e-health and academic issues) and different stakeholders' preferences and daily activities. The main activities are the patient's online registration; the dental appointment scheduling; tasks to improve the patient's treatment (e.g. download of educational videos to help the patient with her/his dental problems and download of x-ray

images to improve the diagnostic capability based on the patient's dental problem); and tasks to improve the clinic's stakeholders satisfaction (e.g. maintain the user's treatment history for further consultation).

Furthermore, all services and contents must be provided anywhere and at any time by using the stakeholders' personal device. In this scenario, the context-aware adaptation and personalization of the services are desired and the privacy issue gets an important position to improve the satisfaction of the clinic's stakeholders, such as: patients, dentists, attendants and other members that perform administrative activities (e.g. manager).

In order to develop the dental clinic ubiquitous application, we conducted meetings with the stakeholders by using brainstorming, open/closed questionnaires, observations and interviews as elicitation techniques, in which we obtained different documents – e.g. the dentist's statements, anamneses dental forms and dental forms of each specialization. We also used scenarios (Leite et al. 1997) to describe the clinic's activities. This strategy simplified our interaction with the stakeholders. From the Patient's viewpoint, a typical context that must be considered based on this case study is:

"Find a dentist to adequately deal with my dental problem, close to my actual location, who respects my privacy policies and whose prices match my social status - thus, the patient's interests must be considered - through the device I am using now - thus, the device heterogeneity must be considered. The request can be placed everywhere and at any time - thus, the service omnipresence and user's mobility must be considered. I am a layperson in software technologies – thus, the software invisibility must be considered. However, I want to know what is going on – thus, the software transparency is required to keep the user aware of what is going on".

We present in detail the *Dental Clinic Ubiquitous Application Engineering* by following the reuse-oriented architecture as well as the life-cycle described in Chapter 5. Finally, Section 6.2 summarizes the Chapter by presenting its remarks.

6.1. Dental Clinic Ubiquitous Application Engineering

There are some issues in the dental clinic cognitive domain that are commonly found in different ubiquitous dental clinic applications, such as: (i) to take care of the patients' dental problems by following specific treatment techniques; (ii) to

protect the patients' personal information from illegal and/or unapproved access; and (iii) to personalize the dental services in order to better achieve the patients' goals. Moreover, there are some particular issues that must be considered when we are developing a specific dental clinic application, such as specific privacy policies and business rules. As previously mentioned, the building blocks of our approach can be specialized – by extending or instantiating them – based on the dental clinic ubiquitous application under analysis. Figure 6.1 shows the specialization of our reuse-oriented architecture for this dental clinic application.

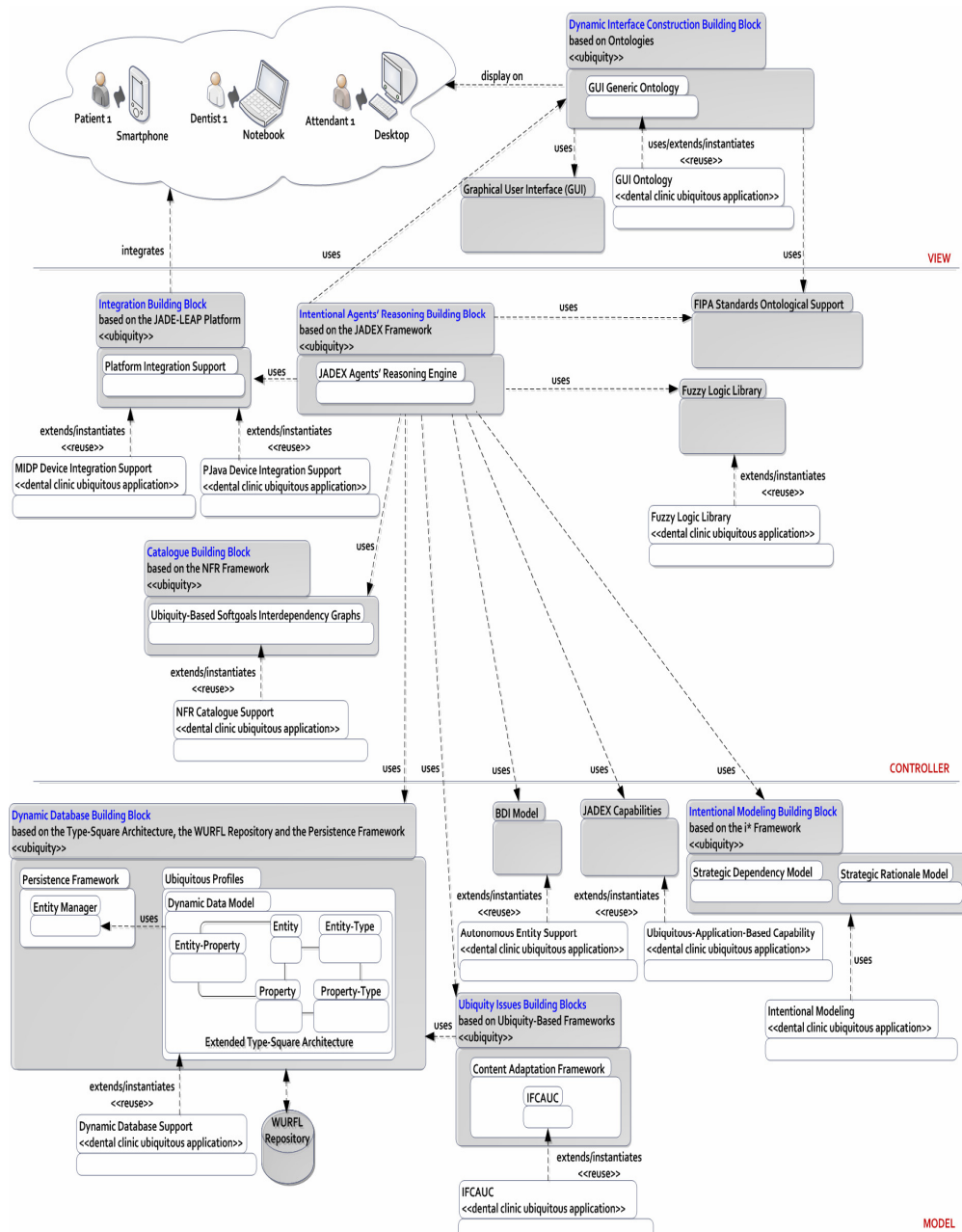


Figure 6.1 - Architecture for the Dental Clinic Ubiquitous Application Engineering

To illustrate a building block specialization according to the dental clinic issues in the *Ubiquitous Application Engineering*, we can consider the development of the GUI Ontology for the dental clinic ubiquitous application – by associating it with the GUI Generic Ontology from the *Dynamic Interface Construction Building Block* – to specifically deal with, for example, the interface elements of dental clinic forms. This specialized GUI Ontology is used to construct dental-clinic-form-based interfaces centered on dental clinic contexts by including the patients’ preferences investigation, their devices’ features determination and other issues of the ubiquitous application under analysis. As follows we describe in details the *Ubiquitous Application Engineering* by using our reuse-based approach.

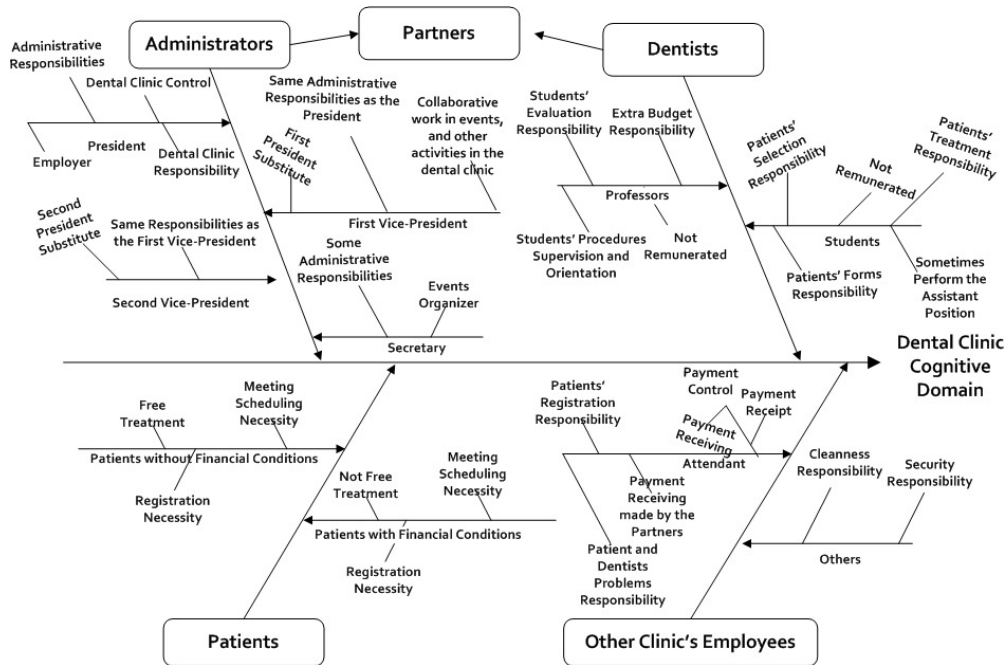
We are particularly interested on the systematic development of an intentional-MAS-driven dental clinic ubiquitous application by reusing the building blocks support sets, which are extended or instantiated from the *Dental Clinic Ubiquitous Application Engineering*.

Next, we present the development of the dental clinic ubiquitous application from the *Early & Late Requirements* disciplines (Section 6.1.1) to the *Test* discipline (Section 6.1.4) by following a model-driven iterative process.

6.1.1. Early Requirements & Late Requirements

In the Early Requirements the proposed guidelines for the systematic development suggest early elicitation and modeling to obtain an overview of the initial settings of the cognitive domain context. Therefore, in the *Dental Clinic Ubiquitous Application Engineering*, the software engineers must compose an adequate view of the dental clinic cognitive domain by considering its main issues. Therefore, they applied different elicitation techniques (e.g. open and close questionnaires, interviews, observation, brainstorming and others). Figure 6.2 briefly shows some details acquired by using some of these elicitation techniques (e.g. Introspection and Observation documented in the first meeting report, Open Questionnaire and Brainstorming documented in the second meeting report, and Close Questionnaire documented in the fifth meeting report), which facilitated the composition of a suitable view about some important requirements of the dental clinic cognitive domain (e.g. dentists are responsible for the patients’ forms and patients must be

registered in the dental clinic to take care of their dental problems). It is important to mention that we are considering a specific domain of dental clinics that also has educational purposes by training the associated dentists in different dental specialties. Therefore, we have the terms “Students” and “Professors” involved in the early elicitation and modeling of the initial settings of the dental clinic cognitive domain.



PUC-Rio - Certificação Digital Nº 0711311/CA

First Meeting

Elicitation Technique: Introspection

In English:

Dra. Mary Silva asked us if we do not want to be registered at the Dental Clinic for a routine dental consult. Thus, we understood the registration, triage, and treatment processes at the APCD Restoring Dentistics specialty, which is the main specialty of the Dra. Mary Silva. The Dra. Mary Silva's treatment was coordinated by the professor Dr. S. Neto. We also had the collaboration of the Dra. P. Rossito, who

Second Meeting

Elicitation Techniques: Brainstorming and Questionnaire

In English:

We requested for the dentist Dra. Mary Silva a description based on the patient scheduling process at the Dental Clinic. In order to facilitate the Dra. Mary's work, we made some questions (i.e. a close questionnaire with questions previously established), such as:

- (1) Can the patient call to the Dental Clinic's secretary in order to schedule a consult by phone?
- (2) Can the patient use an Internet in order to schedule a consult online?

Fifth Meeting

Elicitation Technique: Questionnaire based on the Stakeholders' Intentions

In English:

We requested for the dentist Dra. Mary Silva to answer the questions presented as follow with other dental clinic's stakeholders in order to know the stakeholders' intentionality in some procedures, rules, and forms:

- (1) Why does the Dental Clinic not have online registration? Is it because of security problems? Is it to guarantee that the patient is really interested in the treatment,

Figure 6.2 - Some early elicitation information acquired by using different elicitation techniques

After different iterations with the stakeholders' participation, the software engineers acquired an adequate view about the dental clinic domain, mainly about the activities performed by the stakeholders as briefly presented in Table 6.1.

Table 6.1 - Main activities of the dental clinic cognitive domain focused on Patient and Attendant stakeholders

Activity	Description
Patient's Registration	It represents the process performed between the Patient and the Attendant in order to register the patient at the desired dental clinic. It is necessary that the patient fills the registration form with her/his personal data (e.g. normally with name, address, an identification document, phone number and others).
Dental Meeting Scheduling	It represents the process performed between the Patient and the Attendant in order to schedule a dental meeting with a specialized dentist to take care of the patient's dental problem.
Dental Treatment Payment	It represents the process performed between the Patient and the Attendant in order to pay the dental treatment performed or that will be performed by the dentist to take care of the patient's dental problem.
...	...
Patient's Treatment Canceling	It represents the process performed between the Patient and the Attendant in order to cancel the patient's dental treatment based on different motivations (e.g. patient's need, dentist's need, payment and unapproved behavior).

Based on the early elicitation results, the software engineers modeled the requirements using the i* Framework and/or the NFR Framework. Figure 6.3 illustrates the i* Strategic Dependency model for the Patient and the Attendant actors. In order to simplify the visualization, we optimize the model by showing some essential dependencies based on the patient's registration process. We also perform this kind of optimization in all presented models in this thesis.

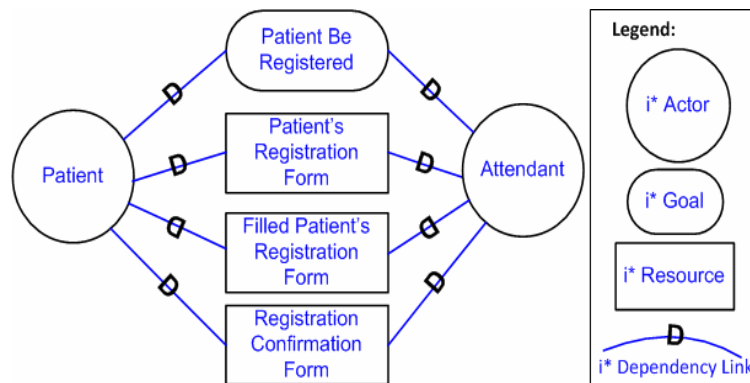


Figure 6.3 - Early Requirements i* SD model for patient's registration process

After the analysis of the obtained results to understand the dental clinic cognitive domain under consideration, the software engineers can perform the next discipline of the process – i.e. the Late Requirements discipline.

In the Late Requirements, the incremental and systematic development suggests the introduction of the application into the modeling obtained from the *Early Requirements* discipline. In order to perform this activity, it is important to compose an adequate view of the application main scenarios as well as to identify its main stakeholders.

Scenario (Leite et al. 1997) is a technique centered on semi-structured description in natural language. This technique is strongly used in the Requirements Engineering community, more specifically during the requirements elicitation process with the stakeholders' interaction. This technique has been used a long time (Carroll 1995). It facilitates the understandability of different activities that are performed by the stakeholders in the organization, in which the software engineers intend to develop the application.

Scenarios can be used to describe different contexts. However, they are particularly interesting when they are applied to improve the knowledge in terms of the organization and its stakeholders. It is really important to use this technique when it is necessary, for example, to investigate the stakeholders' behavior/attitudes/intentions in a specific context.

A scenario can also help in the elicitation process, as it can provide a good way, which is closer to the natural language, to describe and represent situations. This description and representation can be presented to the stakeholders in order to improve the requirements investigation, and consequently the design and the implementation of the future application.

As the result of the scenarios specification in the *Dental Clinic Ubiquitous Application Engineering*, the software engineers obtained various scenarios that compose an appropriate overview about the main activities performed by the stakeholders of the dental clinic under analysis. Therefore, it helped the software engineers to describe the patient's registration, treatment payment, TRIAGE scheduling, dental meeting scheduling, temporary patient's treatment suspension, patient's treatment canceling, and other processes according to the dental clinic under analysis. Figure 6.4 presents a detailed scenario based on the patient's registration process into the dental clinic case study.


1. Dental Clinic Registration Scenario	
<p>Description: This scenario describes the dental clinic registration process as the first activity performed by the attendant when a patient wants to use the dental clinic's services.</p>	
<p>Title: Dental Clinic Registration Objective: To describe the Dental Clinic Registration Process Context: FIRST INTERACTION BETWEEN THE PATIENT AND THE DENTAL CLINIC Actors: patient and attendant Resources: procedures form, registration form, anamneses form, payment information form, registration payment receipt, money in cash. Episodes: <ul style="list-style-type: none"> ✓ Patient goes to the Dental Clinic ✓ Patient asks about the Dental Clinic's services ✓ Attendant gives the procedures form or explains these procedures to the patient ✓ If the patient be interested in the Dental Clinic's services, then the attendant gives to her/him the registration form </p>	

Figure 6.4 - Detailed dental clinic scenario for patient's registration process

According to the scenarios' specification for the dental clinic ubiquitous application, the software engineers identified the main stakeholders. Table 6.2 presents some of them.

Table 6.2 - Main stakeholders of the dental clinic case study

Stakeholder	Description
Patient	User of the available dental clinic services to take care of her/his dental problem.
Dentist	Active actor at the dental clinic that performs several important tasks, such as the TRIAGE Process (i.e. a previous selection of patients based on their dental problems, and then the indication of a dentist with adequate specialty to deal with each dental problem), and the patient's dental problem treatment.
Attendant	Active actor at the dental clinic that performs several important tasks, such as patient's registration; registration payment, TRIAGE scheduling, and serious problems reporting.
Professor	Active actor at the dental clinic that performs some tasks such as dentist's supervision, dentist's support, and patient's treatment support.
Manager	Administrative position (e.g. President, Vice-President and Secretary) that solves serious problems in the dental clinic, and contracts professors and attendants.
...	...

These initial specification process of the dental clinic ubiquitous application facilitated the incorporation of this application into the i* SD model (shown in Figure 6.3). Figure 6.5 illustrates this introduction by enriching the patient's registration early modeling with the dependencies among the Patient, the Application and the Attendant. Here, we are following the TROPOS Model-Driven Development by representing the Application as another i* actor.

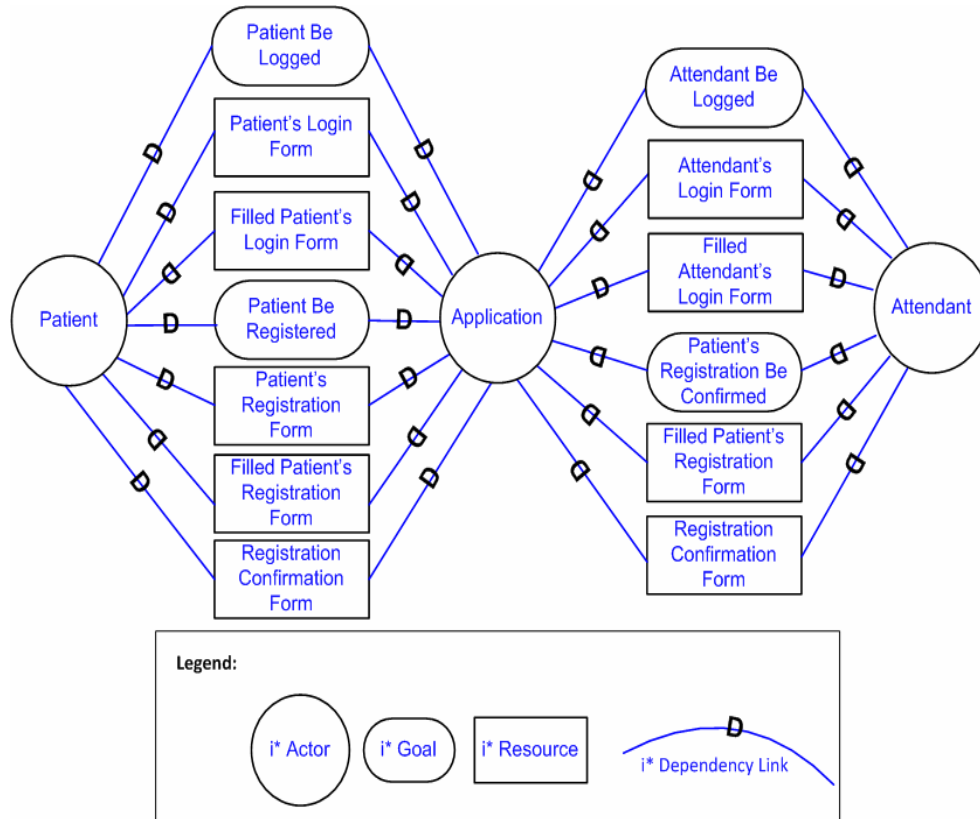


Figure 6.5 - *Late Requirements* i* SD model – Patient/Application/Attendant – for patient's registration process

Moreover, it is important to incorporate the ubiquitous application issues. The resulting model must contain specific ubiquitous issues, such as the ones identified by using some elicitation techniques during the dental clinic's investigation process, and shown in Figure 6.6. Therefore, the software engineers must consider that the dental clinic ubiquitous application under development must deal with device heterogeneity, service omnipresence, content adaptability, profiles awareness, process complexity invisibility, mobility, and other ubiquitous concerns. It is important to remember that it is possible to report on some problems for the Early Requirements discipline.

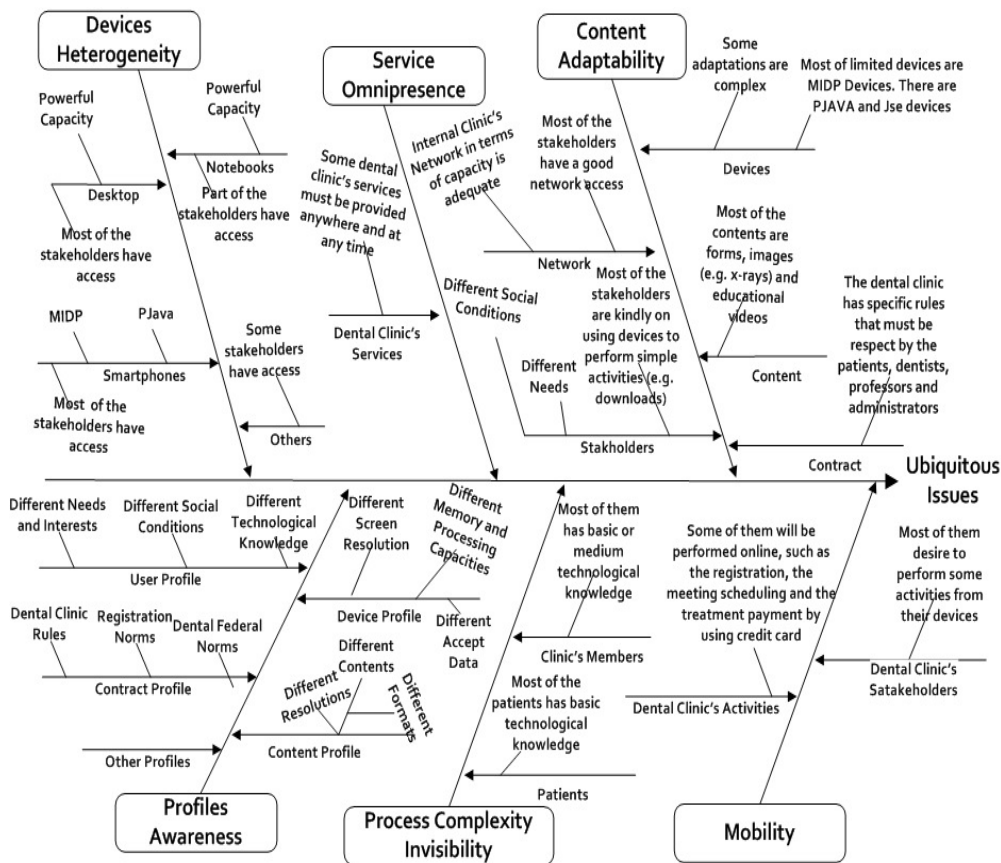


Figure 6.6 - Some late elicitation information acquired by using different elicitation techniques

The *NFR Catalogue Building Block* can also contribute to the ubiquitous issues identification. The *Catalogue Usage Method* offers specific activities to explore, collect, model, operationalize and validate the main non-functional ubiquitous requirements from a baseline called *NFR Catalogue*. At the end of the process, among other conclusions, the software engineers will know, for example, that the ubiquitous applications operate within ever-changing environments, rely on a growing list of heterogeneous devices, assume user mobility and content server distribution, and require adaptability and context awareness.

As already explained, the proposed *NFR Catalogue* is centered on a repository of reusable models that aggregates the results of our efforts in capturing ubiquitous *NFR* issues in models to be shared in a common baseline with the Ubiquitous Computing community. Therefore, our catalogue is founded on the *NFR Framework* and consists of softgoal hierarchies for ubiquitous applications *NFRs*. These hierarchies represent positive/negative contributions among different softgoals, as well as suggestions for their operationalizations, inspired by intentional Multi-Agent System (MAS) and goal-oriented ideas.

Although our catalogue's operationalizations – in this thesis – are illustrated through the application of MAS, the developers can use the software hierarchies to guide their ubiquitous projects in different paradigms (e.g. object-oriented and component) since softgoal decompositions and interdependencies are independent of implementation details.

Figures 6.7 and 6.8 illustrate the Mobility NFR acquired from our Catalogue by performing the *Explore* activity of the *Catalogue Usage Method*. In this activity the NFR Catalogue is consulted and the Mobility NFR is extracted from the baseline.

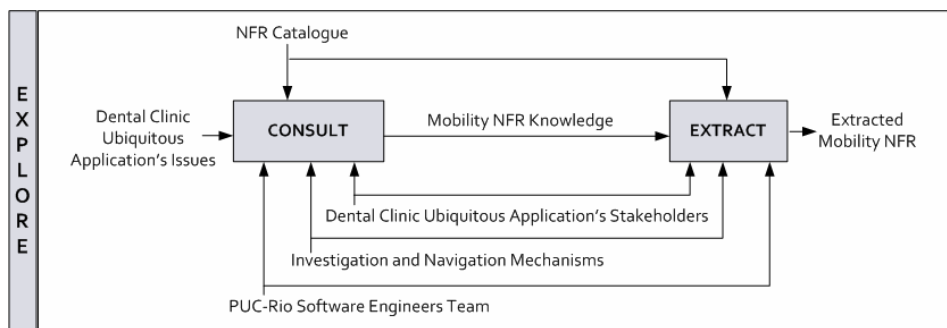


Figure 6.7 - *NFR Catalogue Usage Method* instantiated from the *Dental Clinic Ubiquitous Application Engineering – Explore* activity

It is relevant to consider that Mobility means in this context: *"the state of being in motion, using untethered technology to access data/services from occasionally-connected, portable, networked computer devices."* The mobility can be decomposed in Distribution [Software] (not shown), Versatility [Software], Portability [Software], Traceability [Software], Connectivity [Software], Reliability [Software], Recoverability [Software] and Location Awareness [Software] (not shown). These NFRs are also decomposed in other NFRs (e.g. Versatility [Software] is decomposed in Functional Versatility [Software]). Furthermore, we propose different operationalizations for each specified NFR, such as: "Integration Platform (e.g. JADE-LEAP Platform)" help Operational System Portability [Software] and Platform Portability [Software]). Some details were omitted in order to facilitate the visualization of the Mobility SIG.

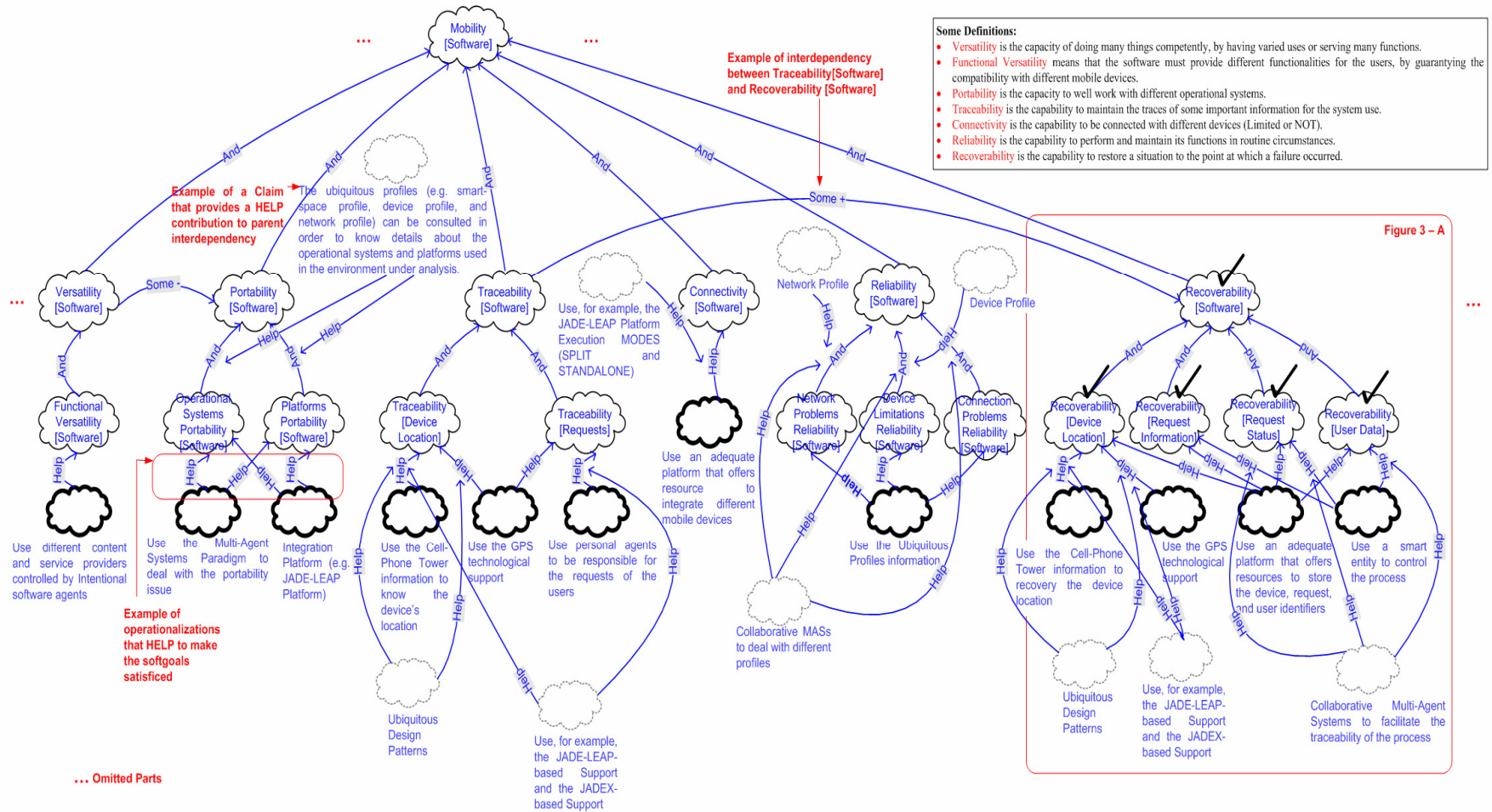


Figure 6.8 - Software Mobility SIG from our NFR Catalogue

Based on the investigation of the ubiquitous issues in the *Dental Clinic Ubiquitous Application Engineering*, Figure 6.9 presents the introduction of this ubiquitous application into the *Late Requirements i* SD model* shown in Figure 6.5. This evolved model focuses on Privacy [Patient], Privacy [Attendant], and Heterogeneity [Device] issues while performing the patient's registration login and registration.

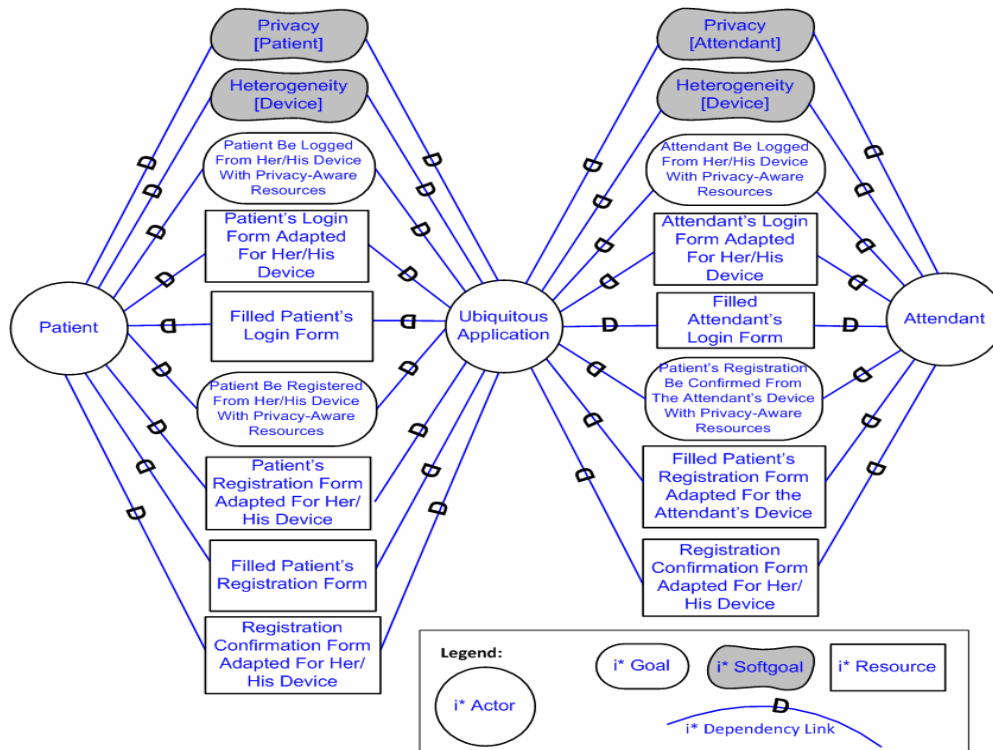


Figure 6.9 - *Late Requirements i* SD model* – Patient/Ubiquitous Application/Attendant – for patient's registration process centered on patient privacy and device heterogeneity issues

After the ubiquitous issues incorporation, the software engineers can refine the model by using a detailed analysis. The goals analysis technique suggested by the *Intentional Modeling Building Block* through the use of the *i** Framework as well as the propagation rules suggested by the *NFR Catalogue Building Block* through the use of the *NFR Framework* can facilitate this refinement. In order to illustrate this process, we consider some specific activities offered by the *Catalogue Usage Method* to, respectively, collect, model and operationalize the ubiquitous non-functional requirements modeling by considering specific issues of the dental clinic ubiquitous application.

In the Collect activity of the *Catalogue Usage Method*, the extracted Mobility NFR, for example, can be picked up or instantiated to better attend the

dental clinic's issues. In the dental clinic case, it is instantiated – extending some details – by obtaining an evolved Mobility SIG and its Frame-Like Notation with some specific adaptations in the Traceability NFR as presented in Figure 6.10.

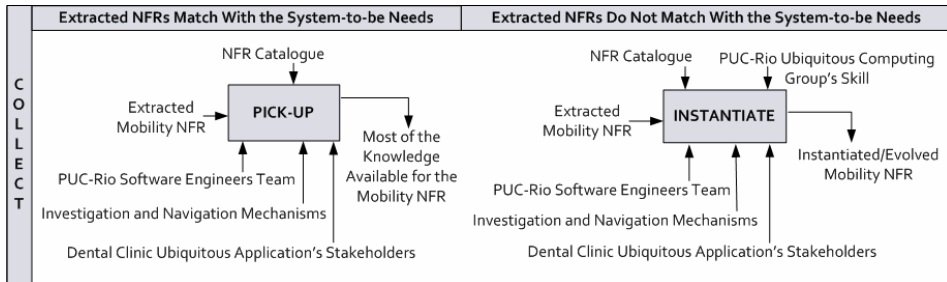


Figure 6.10 - *NFR Catalogue Usage Method* instantiated from the *Dental Clinic Ubiquitous Application Engineering – Collect* activity

In the Model activity, the instantiated Mobility NFR can be decomposed and its interdependencies determined for the SIG notation as well as its decomposition, claim and correlation rule can be specified for the Frame-Like notation as shown in Figure 6.11.

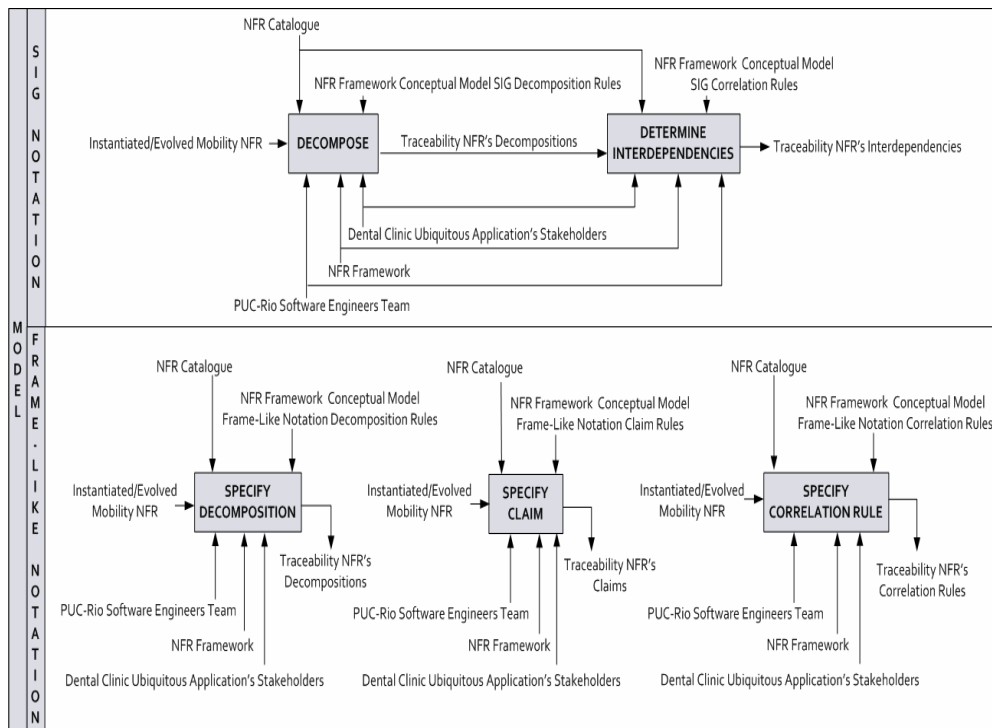


Figure 6.11 - *NFR Catalogue Usage Method* instantiated from the *Dental Clinic Ubiquitous Application Engineering – Model* activity

In the Operationalize activity, based on the Mobility NFR knowledge, it is possible to just select the operationalizations from the baseline or even to specify them based on expertise and skills as illustrated in Figure 6.12.

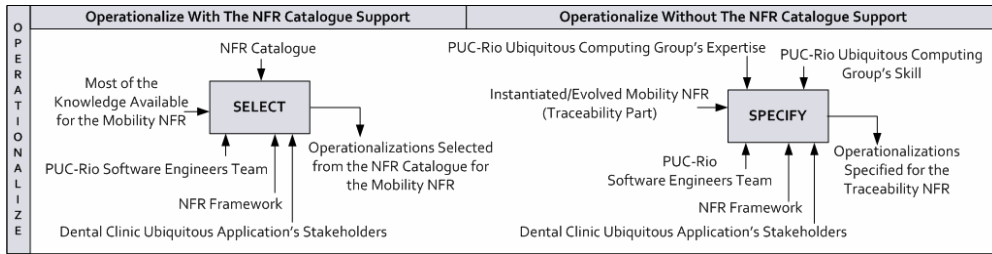


Figure 6.12 - *NFR Catalogue Usage Method* instantiated from the *Dental Clinic Ubiquitous Application Engineering – Operationalize* activity

It is important to consider that the mentioned refinements are performed throughout the incremental and systematic development. Figure 6.13 shows an example of refinement – specified in the final stages of the requirements and design activities – for the Mobility SIG, specifically for the Traceability issue. In this field, software engineers instantiates the original Mobility SIG by also extending the Traceability [Software] in order to maintain traces to directly associate the user’s requests and the responsible agent for dealing with them. This agent is registered with an identifier at the MAS platform. It can avoid problems with device disconnection because of, for example, low battery and lack of signal.

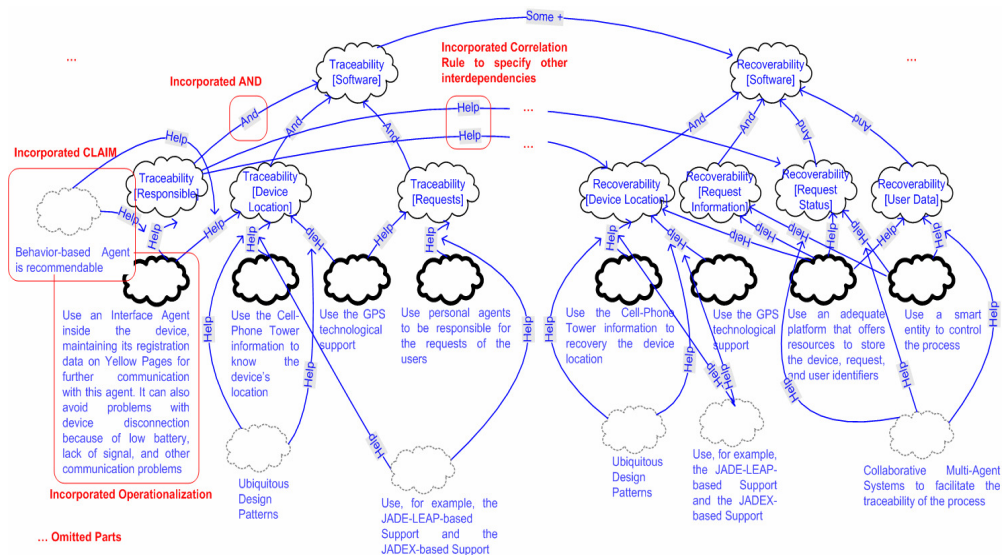


Figure 6.13 - Evolved part of the *Software Mobility SIG*

Figure 6.14 emphasizes the *NFR Catalogue Building Block's* support in the *Dental Clinic Ubiquitous Application Engineering*. The reuse is centered on the Mobility SIG by illustrating the Traceability [Responsible] as an extended NFR; the interdependencies between the Traceability [Responsible] and both Recoverability [Device Location] and Recoverability [Request Status]; and an extended operationalization for the Traceability [Responsible].

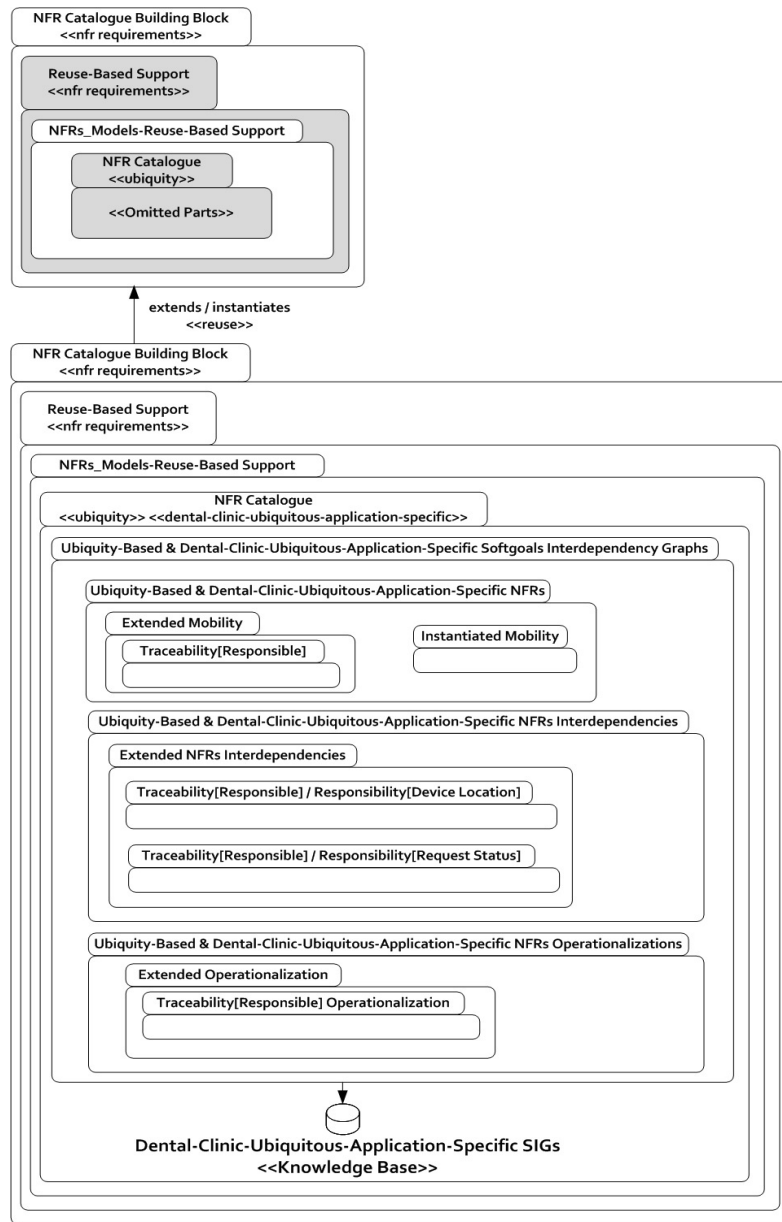


Figure 6.14 - *NFR Catalogue Building Block* in the *Dental Clinic Ubiquitous Application Engineering*

Only to illustrate a more complete *Late Requirements i* SD model*, which was obtained after some iterations of the proposed incremental and systematic development, Figure 6.15 zooms in privacy issues (e.g. accountability and dependability issues) and invisibility issues as well as on the use of a Mobile Agent to deal with the mobility issues (e.g. Distribution) in ever-changing contexts. The model also presents different software agents (e.g. Interface Agent, Personal Agent, AMS Agent, DF Agent, Mobile Agent and Dental Clinic's Manager Agent) used to perform the registration process centered on the dental clinic's way-of-working.

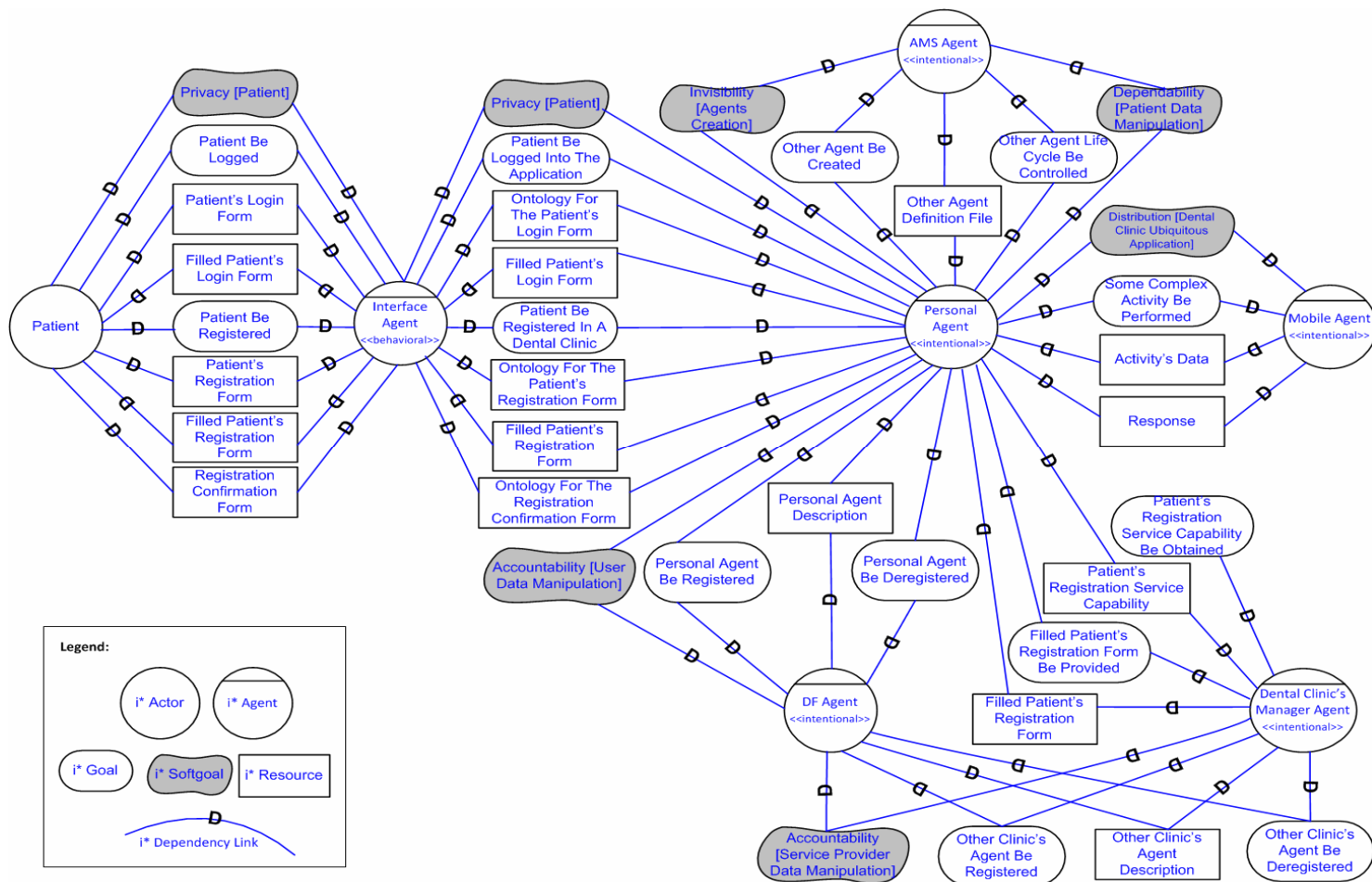


Figure 6.15 - More complete i* SD model centered on privacy, invisibility and mobility issues

Summarizing the main goal of each agent, we have: (i) the Interface Agent that concerns with the communication between the patient and its intentional personal agent; (ii) the Personal Agent as the patient's personal assistant to intermediate the communication between the patient and the dental clinic application, by including the offered services and contents; (iii) the AMS Agent to control the agents' life-cycle from its creation to its death; (iv) the DF Agent to register and deregister the agents of the MAS platform at the Yellow Pages; (v) the Mobile Agent to perform some complex activities in different server by also contributing to the distribution issue; and (vi) the Dental Clinic's Manager Agent that deals with the dental clinic's capabilities to support different services.

6.1.2. Architectural Design & Detailed Design

In the Architectural Design the software engineers specify the system's global architecture in terms of sub-systems interconnected through data and control flows (i.e. dependencies). The dental clinic could be composed of different departments, and each of them could be modeled as a sub-application of the main application. Moreover, the dental clinic application could interact with other dental clinic applications, which have different ways-of-working. Again, they could be modeled as different sub-applications. In order to exemplify the reuse in the system's global architecture specification, we illustrate a model reuse by combining the dental clinic application's modeling and the IFCAUC modeling. The result is the dental clinic application's modeling enriched by an intentional-MAS-driven content adaptation modeling (i.e. its intentional agents and their dependencies). The content adaptation proposed by the IFCAUC is based on a conceptual model, extensively investigated by our research group to deal with this concern in ever-changing contexts. Figure 6.16 illustrates the resulting model by incorporating a Mobile Agent and an Adapter Agent with a specific purpose (i.e. content adaptability) into the previous model. The former agent is used to migrate from one server to another by performing the content adaptation in a dedicated server. The latter agent is used to collaboratively adapt the desire content according to the ubiquitous profiles and other personalization-based information by using different adaptation techniques.

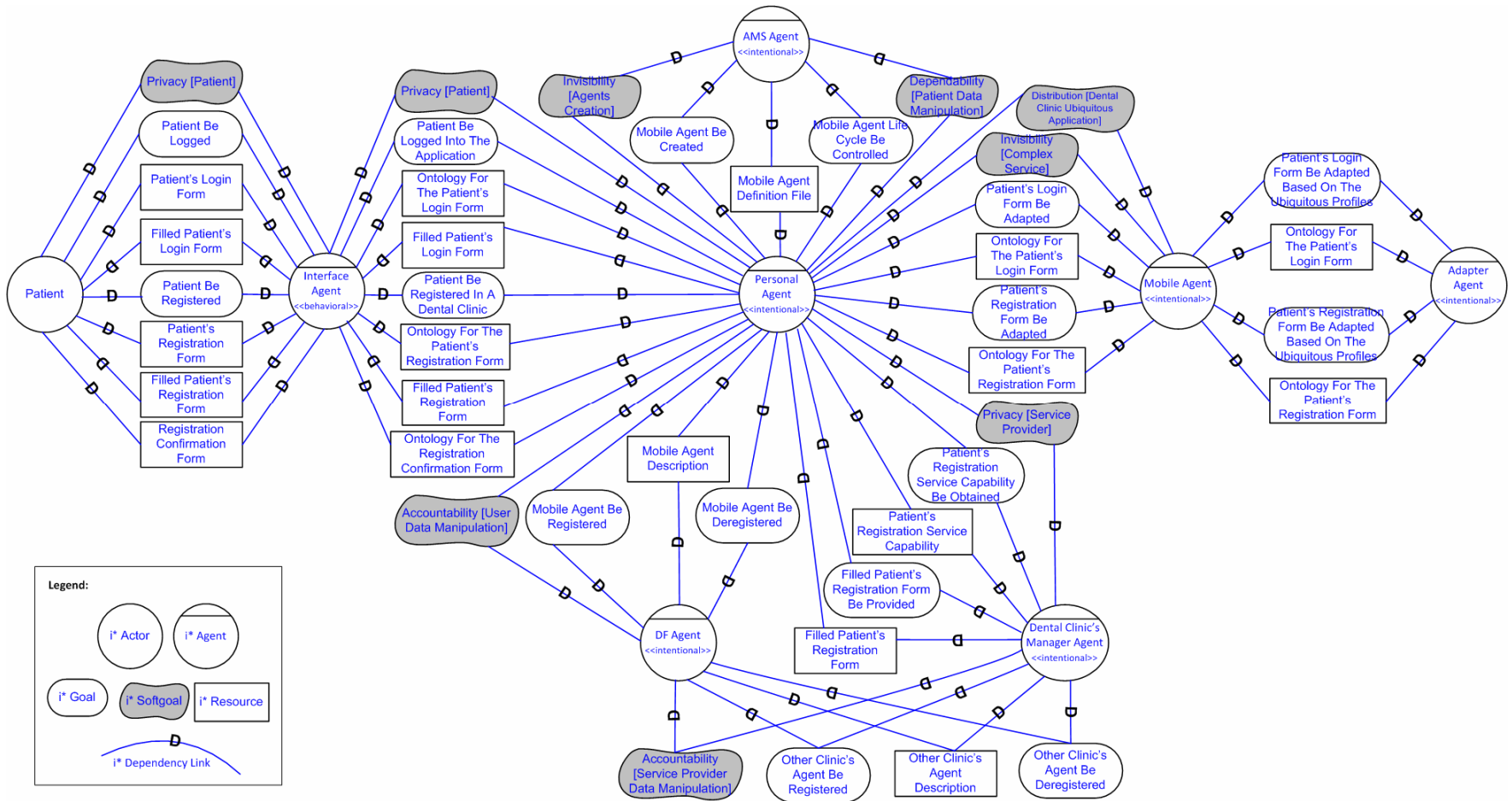


Figure 6.16 - More complete i* SD model centered on privacy, invisibility, mobility and adaptability issues

In this discipline it is also necessary to specify the agents' capabilities by defining and modeling them. Figure 6.17 illustrates the modeling of an intentional Mobile Agent, capable to perform content adaptations in a dedicated server. This particular ability is modeled as a Mobility Capability based on the *Intentional Agents' Reasoning Building Block*, specifically centered on the JADEx Capability support. The Mobility Capability as well as other modeled capabilities compose the agents' capabilities support in the dental clinic case study.

Based on the Mobility Capability and considering the patient's registration process in the dental clinic under development, there are various dental clinic forms that must be exchanged between the dental application and the patient from heterogeneous devices. Therefore, it is necessary a mechanism to adapt these forms according to the devices features and other ubiquitous profiles. In this field, as previously mentioned, we suggest the use of intentional mobile agents, which can migrate from the application server to a dedicated one with specialized intentional adapter agents. On one hand, and among other contributions, it avoids the application server overload and improves the agents' autonomy, adaptability and collaboration in dealing with ever-changing conditions. On the other hand, it demands complex protocols and privacy-based resources to transfer users' data.

In the dental clinic application, the software engineers – centered on the application's requirements – decided to use a dedicated server to perform content adaptations. Thus, there are some tasks that must be performed to complete the main task *Adapt The Patient's Login Form In A Dedicated Server*: (i) *Move To Dedicated Server*; (ii) *Adapt With Collaborative Agent*; (iii) *Move Back*; (iv) *Send The Ontology For The Patient's Login Form To The Personal Agent*; (v) *Update The Knowledge Base By Improving Different Beliefs*; (vi) *Kill Itself*, and (vii) all sub-tasks of these tasks. It is possible to observe that the proposed mechanism to adapt different forms based on ubiquitous profiles focuses on the Mobility Capability, ontologies and collaborative intentional MAS. Therefore, in this field the software engineers reused the support proposed by the *Intentional Agents' Reasoning Building Block*, the *Dynamic Interface Construction Building Block* and the *Ubiquity Issue Building Block* (i.e. IFCAUC).

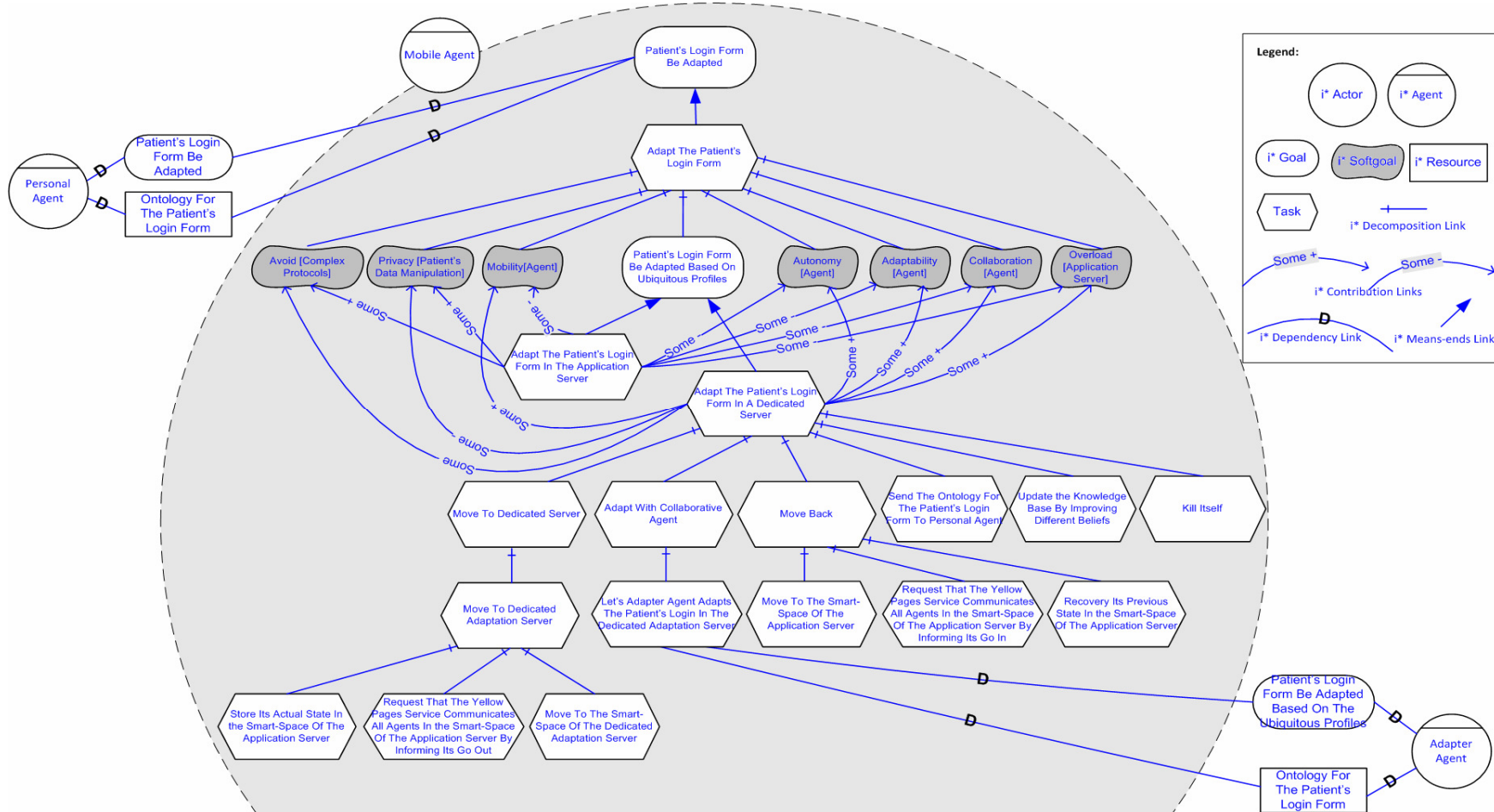


Figure 6.17 - i* SR model for the intentional *Mobile Agent – Mobility Capability*

It is relevant to mention that for each softgoal (e.g. Privacy [Patient's Data Manipulation]), the modeling also details their related softgoals as well as the impact of the alternatives (i.e. tasks chosen at runtime by the agents to achieve the delegated functional goals by considering the context under analysis) on those softgoals. However, it is not shown in the presented i^* models because either it compromises the visualization or they are distributed in various i^* models. In order to illustrate some related softgoals, Figure 6.18 shows the Privacy [Patient] centered on the Privacy [Patient Data Manipulation], which is decomposed in Dependability [Patient Data Manipulation], Accountability [Patient Data Manipulation], Security [Patient Data Manipulation] and their correlated softgoals. The impacts based on the horizontal view were omitted not to compromise the presentation.

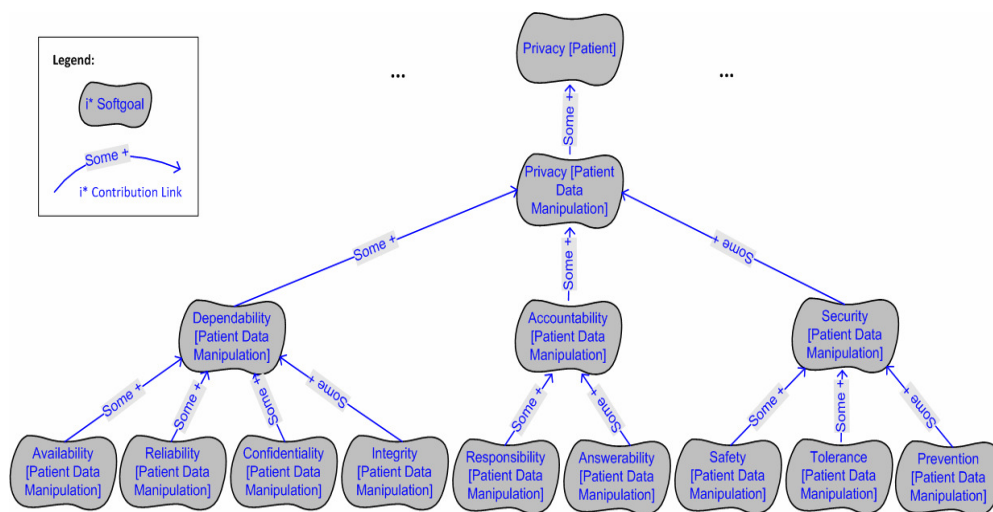


Figure 6.18 – *Privacy [Patient]* decomposition

Figure 6.19 briefly present the Personal Agent i^* Strategic Rationale (SR) model by zooming in some goals, softgoals (e.g. Privacy [Patient's Data Manipulation], Awareness [Ubiquitous Context], Adaptability [Ubiquitous Application], Privacy [Service Provider], Cognitive Capacity [Agent], Autonomy [Agent], Adaptability [Agent]) and tasks of the patient's registration process. It models two alternative tasks to achieve the goal *Patient's Be Registered In A Dental Clinic With Intentional MAS*: (1) *Register The Patient In A Dental Clinic Without Capability Support*, and (2) *Register The Patient In A Dental Clinic With Capability Support*. In the dental clinic application, the software agents decided to use the latter task in order to positively contribute to the specified softgoals.

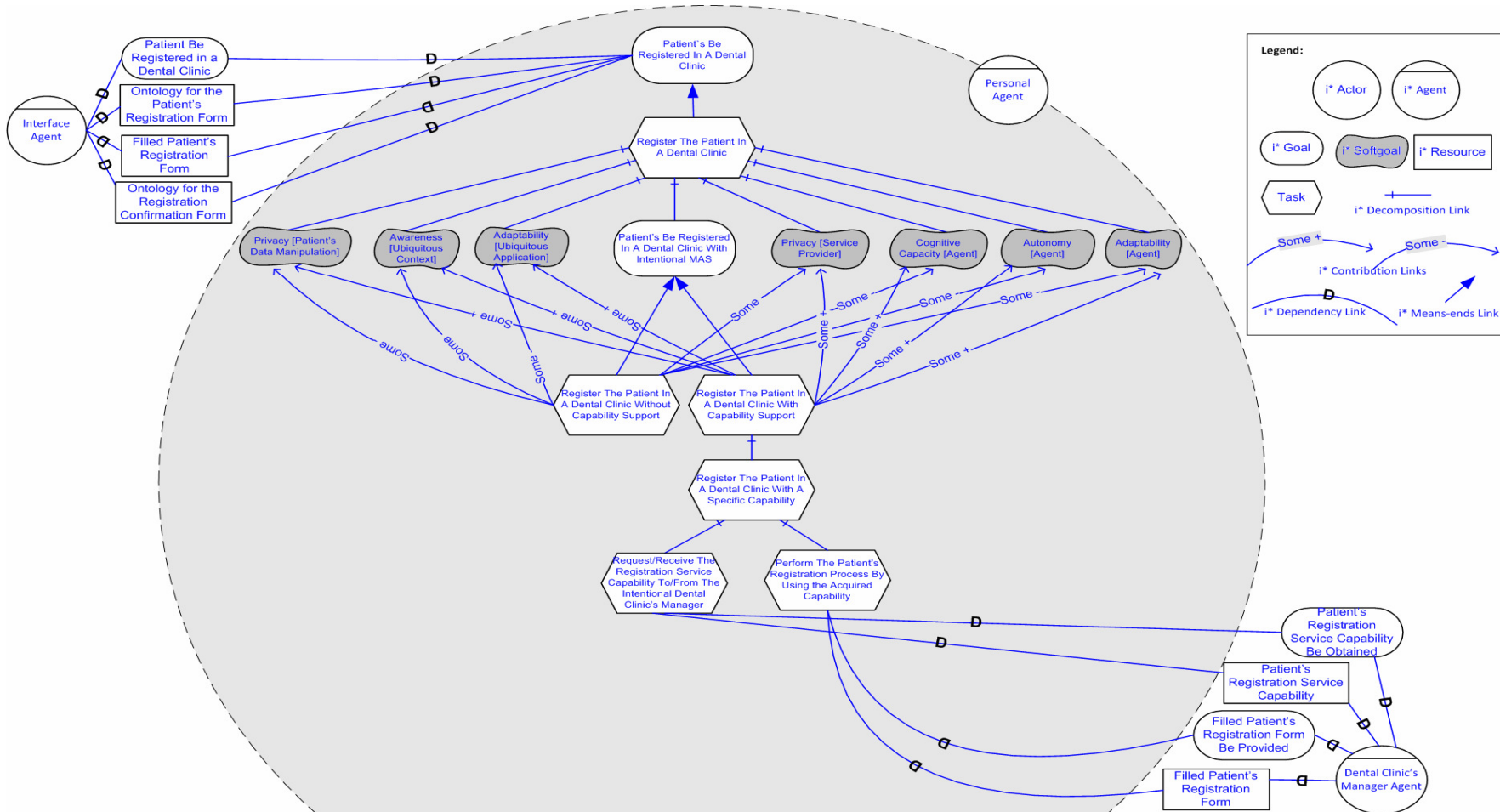


Figure 6.19 - i* SR model for the intentional *Personal Agent*

The mentioned task is decomposed in the task *Register The Patient In A Dental Clinic With A Specific Capability* and the sub-tasks: (i) *Request/Receive The Registration Service Capability To/From The Intentional Dental Clinic's Manager*; and (ii) *Perform The Patient's Registration Process By Using the Acquired Capability*. This alternative task demands a collaborative work between the Personal Agent and the Dental Clinic's Manager Agent.

The last activity in the Architectural Design discipline is the specification of the technological support that will be used in the development of the intentional-MAS-driven ubiquitous application from dental clinic cognitive domain. Based on the operationalizations provided by the *NFR Catalogue Building Block* as well as on the “tips” suggested by the conceptual models of different proposed building blocks, it is possible to specify a suitable technological set to deal with different concerns of the dental clinic ubiquitous application under development. The BDI model, intentional MAS and ontologies to facilitate the communication and inter-operability of the agents are examples of support sets proposed by our approach and specified in those building blocks.

In the Detailed Design discipline, the software agents will refine the obtained models by improving the agents' capabilities, the agents' plans and the technological support. Only to illustrate how to refine the technological support, the software engineers must consider the computational technologies used to specifically deal with different ubiquitous issues. In this field, the proposed building blocks suggests, for example, the use of the JADE-LEAP execution modes to support the integration of heterogeneous devices with the MAS platform; the use of the IFCAUC API to deal with the content adaptability issue by applying intentional agents; and the use of the Fuzzy Logic Library to deals with non-functional requirements at runtime by running pre-defined fuzzy conditional rules centered on fuzzy sets and variables.

6.1.3. Implementation

In the Implementation discipline, the software engineers configured the **I**ntegrated **D**evelopment **E**nvironment (IDE) (e.g. NetBeans or Eclipse), and then reuse frameworks, models, and other technological support – specified in the

Architectural & Detailed Design disciplines – to systematically implement the dental clinic ubiquitous application. Therefore, it is necessary to add libraries and to use APIs, which can help the software engineers in specific issues.

In order to promote, improve and facilitate the software reuse in the Implementation discipline, there are some building blocks and packages focused on lower abstraction levels. As follows, we describe the use of them in the *Dental Clinic Ubiquitous Application Engineering*.

6.1.3.1.

Reuse of the Intentional Agents' Reasoning Building Block in lower abstraction levels

In this Section we zoom in the sub-packages of the *Intentional Agents' Reasoning Building Block* by reusing them in the *Dental Clinic Ubiquitous Application Engineering*. Our focus is on (i) the *Autonomous Entity Support* from the *Agents' Cognitive-Ability-Based Support*, and (ii) the *Ubiquitous-Application-Based Capability* from the *Dependability- Accountability- & Security-Based Support*.

(i) Reuse of the Autonomous Entity Support

The *Autonomous Entity Support* – from the *Intentional Agents' Reasoning Building Block* – is reused by instantiation (Figure 6.20) to obtain the *Autonomous Entity Support* for the dental clinic ubiquitous application. The latter support is focused on the development of intentional agents with powerful cognitive capacity in order to perform the services offered by the dental clinic ubiquitous application. The *Dental Clinic Ubiquitous Application's Agent* hypothetically represents different intentional agents of the dental clinic under analysis. This agent must deal with, for example, the patient's registration and the scheduling of the dental treatment by taking into account the dental clinic ubiquitous application's issues (e.g. the adaptability need and the protection of the clinic's business rules from other clinics' access).

An intentional agent, following the JADDEX specification, is composed of an *Agent Definition File* (ADF) (i.e. XML file) and different *Plans* (i.e. Java files). The ADF contains goals, beliefs and plans by specifying the agent's purposes, the agent's knowledge and the names of the plans. The *Plans* represent sequence of actions to achieve specific goals. Figure 6.21 shows the ADF for the Mobile Agent of the dental clinic ubiquitous application.

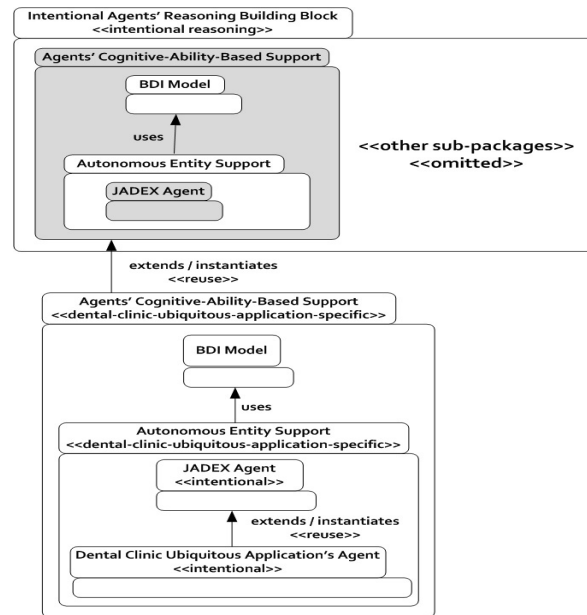


Figure 6.20 - Reuse of the *Autonomous Entity Support* in the *Dental Clinic Ubiquitous Application Engineering*

```

- <agent xmlns="http://jadex.sourceforge.net/jadex"
  xmlns:xsi="..." xsi:schemaLocation="..." name="Mobile" package="ifcauc.mobile">
- <imports>
  ...
</imports>
- <beliefs>
<belief name="serviceDescription" class="String" exported="true" />
<belief name="adaptationRequest" class="AdaptationRequest" exported="true" />
<belief name="adapters" class="ArrayList" exported="true" />
<belief name="selectedAdapter" class="AgentIdentifier" />
  ...
</beliefs>
- <goals>
- <achievegoal name="adapters_be_analyzed">
  <creationcondition>$beliefbase.selectedAdapter == null</creationcondition>
</achievegoal>
<achievegoal name="adaptation_be_requested" />
  ...
</goals>
- <plans>
- <plan name="analyze_adapters">
  <body type="mobile">new AnalyzeAdaptersPlan()</body>
- <trigger>
  <goal ref="adapters_be_analyzed" />
</trigger>
</plan>
- <plan name="move_agent">
  <body type="mobile">new MoveAgentPlan()</body>
- <trigger>
  <beliefchange ref="selectedAdapter" />
</trigger>
</plan>
- <plan name="request_adaptation">
  <body type="mobile">new RequestAdaptationPlan()</body>
- <trigger>
  <goal ref="adaptation_be_requested" />
</trigger>
</plan>
- <plan name="receive_adaptation_result">
  <body type="mobile">new ReceiveAdaptationResultPlan()</body>
- <trigger>
  <beliefchange ref="adaptation_result_be_received" />
</trigger>
</plan>
- <plan name="move_agent_back">
  <body type="mobile">new MoveAgentBackPlan()</body>
- <trigger>
  <goal ref="agent_be_moved_back" />
</trigger>
</plan>
  ...
</plans>
  ...
</agent>

```

Figure 6.21 - *Agent Definition File* for the *Mobile Agent* of the dental clinic ubiquitous application

Figure 6.22 shows a code fragment of a Mobile Agent’s plan, which is dispatched by a goal related to the patient’s registration process in order to perform some adaptation in a dedicated server. The *MoveAgentPlan.java* describes a sequence of actions – partially shown in the illustrated code fragment – to perform the migration from one server to another by selecting an appropriate Adapter Agent to deal with the adaptability issue. It is important to notice that both the *Mobile.agent.xml* and the *MoveAgentPlan.java* are obtained from the IFCAUC API, more specifically from its *ifcauc.mobile package*.

```

public class MoveAgentPlan extends MobilePlan {

    private AgentIdentifier adapter;
    private AgentIdentifier personal;
    ...

    /** Creates a new instance of MoveAgentPlan */
    public MoveAgentPlan() {
        this.adapter = (AgentIdentifier) this.getBeliefbase().getBelief("selectedAdapter").getFact();
        this.personal = (AgentIdentifier) this.getBeliefbase().getBelief("personal").getFact();
        ...
    }

    public void action(IEvent event) {
        System.out.println("\n\nMobile Agent running MoveAgentPlan.\n\n");
        ...
        if (!(event instanceof IGoalEvent) && ((IGoalEvent)event).getGoal().getType().equals("request_be_sent")) {
            System.out.println("\n\nAdapter Agent running MoveAgentPlan Step 1.\n\n");
            ...
            IMessageEvent requestLocation = null;
            requestLocation = this.createMessageEvent("request_adapter_location");
            requestLocation.getParameterSet(SFipa.RECEIVERS).addValue(this.adapter);
            this.sendMessage(requestLocation);
            ...
        }
    }
}

```

Figure 6.22 - *Mobile Agent's plan – MoveAgentPlan.java* – to perform the migration from one server to another in the dental clinic ubiquitous application

The BDI-model-based engine of intentional agents in the dental clinic ubiquitous application is illustrated in Figure 6.23. Heterogeneous devices (basically, Smartphones, notebooks and desktops) – represented as *Sensors* – are used by the clinic’s stakeholders to request contents or services in the dental clinic. The requests dispatch *Events* that invoke the *Reasoning and Learning Engine*. The *Reasoning and Learning Engine* is composed of a *knowledge Base* centered on the agents’ *Beliefs*; a *Fuzzy-Logic Library* support; a *Dynamic Database* to deals with the ubiquitous profiles; and the *Goals* specification centered on the users’ *Desires*. Moreover, the agents’ brain also contains an *Intentional Engine* – called PLAN LIBRARY – that is mainly composed of *Tasks* associated with the goals and centered on the users’ *Intentions*. Finally, *Actuators* are responsible for the users’ feedback by using their own devices’ resources.

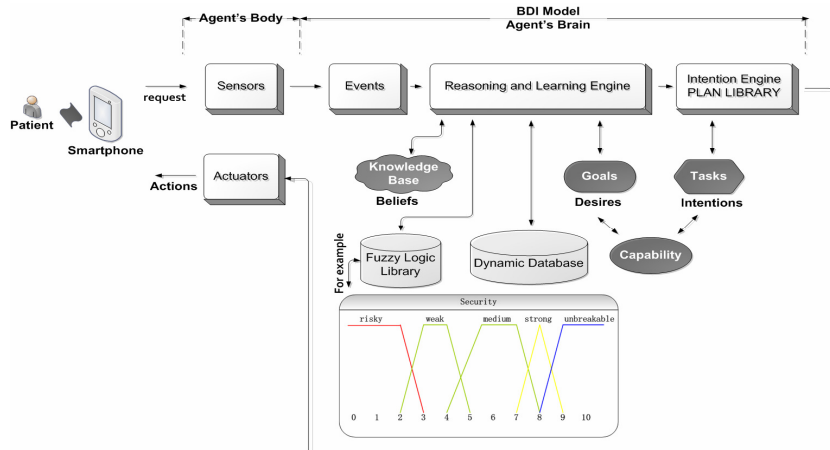


Figure 6.23 - BDI-model-based engine for dental clinic's intentional software agents

Figure 6.24 shows another short fragment of the BDI-based code, specific for the dental clinic application that registers the patient, by reusing the *Cognitive-Ability-Based Support* focusing on the BDI-model-based engine.

```

...
<goals>
  //Goal centered on the patient's registration
  <achievegoal name="patient_be_registered_in_the_dental_clinic" exported="true"/>
  ...
  //An example of goal centered on the privacy investigation "on the fly"
  <achievegoal name="clinic_rules_and_procedures_be_accepted_by_patient"/>
  ...
  //Reuse of the JADEX request protocol to deal with the patient's request by
  //standardizing the communication between the agents of the platform
  <achievegoalref name="request_protocol_be_instantiated">
    <concrete ref="procap_rp_initiate"/>
  </achievegoalref>

  <performgoalref name="rp_receiver_interaction">
    <concrete ref="procap_rp_receiver_interaction"/>
  </performgoalref>

  <querygoalref name="rp_request_be_decided">
    <concrete ref="procap_rp_decide_request"/>
  </querygoalref>

  <achievegoalref name="rp_request_be_executed">
    <concrete ref="procap_rp_execute_request"/>
  </achievegoalref>
  ...
</goals>

<plans>
  //JADEX plan instantiation to be dispatched by the patient's registration goal
  <plan name="register_patient_in_the_dental_clinic">
    <body class="RegisterPatientInDentalClinic"/>
  <trigger>
    <goal ref="patient_be_registered_in_the_dental_clinic"/>
  </trigger>
</plan>
  ...
</plans>
...

```

Figure 6.24 - Code fragment for the patient's registration in the dental clinic ubiquitous application

(ii) Reuse of the Ubiquitous Application-Based Capability

The *Ubiquitous Application-Based Capability*, more specifically the *Privacy-Aware Capability* model – from the *Intentional Agents' Reasoning Building Block* – is reused by instantiation (Figure 6.25) in the incremental and systematic

development of the dental clinic ubiquitous application. Among other contributions, it intends to protect the patients' personal information (e.g. *I desired to protect my history of dental diseases, which were previously treated by different clinics, from unapproved third party accesses*) and the dental clinic's commercial strategies by taking into account its specific privacy policies and rules (e.g. *It is possible to disclose my prices for other dental clinics. However, it is important to protect the access of other commercial strategies, such as: if we facilitate payment for our patients and how much we pay our dentists*).

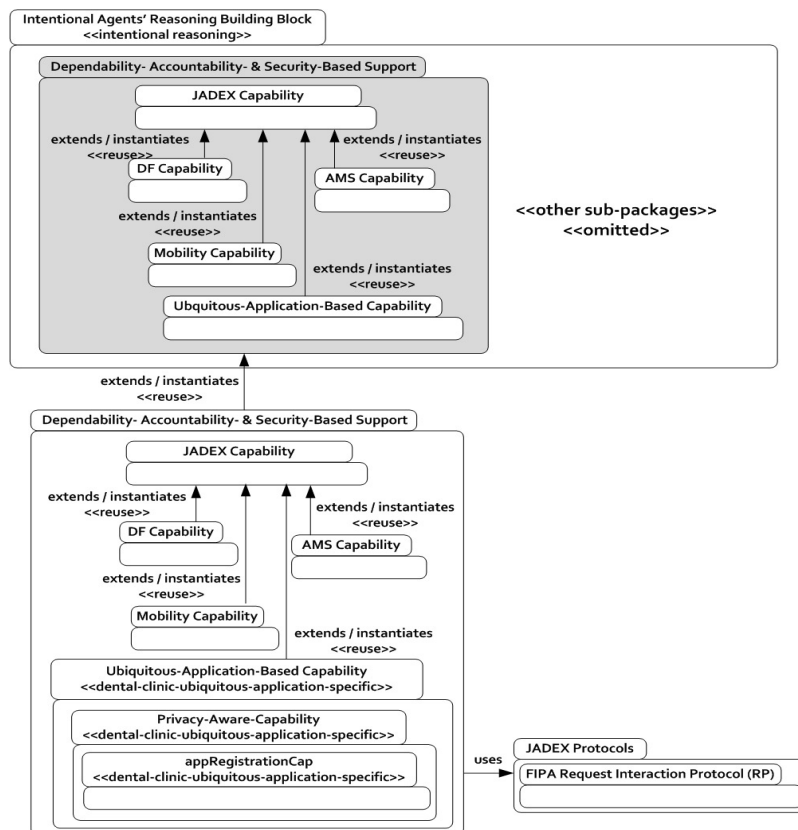


Figure 6.25 - Reuse of the *Ubiquitous Application-Based Capability* in the *Dental Clinic Ubiquitous Application Engineering*

In the *Dental Clinic Ubiquitous Application Engineering* our focus is on *Privacy-Aware Capability* when it is dealing with privacy issues in the dental clinic. As well as the stakeholders of the dental clinic desire to protect their personal information in the dental clinic application, a specific dental clinic desires to protect their information (e.g. dental clinic's commercial policies and business rules) from their concurrent dental clinics in the application level. Therefore, and only to illustrate its applicability, the *Privacy-Aware Capability* model can be reused in the *Dental Clinic Ubiquitous Application Engineering* by

focusing on intentional agents specialized in the dental clinic's services. They control the access of the *Personal Agent* and others (coming from different dental clinics applications) by providing or not – at runtime – the service capability to support those services.

Each service capability confers to the agent that receives it the ability to communicate with the intentional agents responsible for the service in the dental clinic application. More than know how to communicate with dental clinic's service agents, the agent with the service capability is capable to use the service and also to access the shared information. Therefore, if an agent desires to use the service, to interact with the service's agents and to access the dental clinic's information, it must contact the agent responsible for the service (normally registered at the Yellow Pages of the MAS Platform). Thus, it requests the service's capability. Finally, it receives or not the capability from the responsible agent. Depending on the dental clinic's desires and intentions, the *Privacy-Aware Capability* model can be instantiated based on different access policies, such as: (i) only agents registered at the Yellow Pages can receive the service's capability; and (ii) only patients (represented by Personal Agents into the MAS Platform) with high level of confidentiality/dependability/integrity can receive the service's capability. As follows, we illustrate code fragments based on the *Privacy-Aware Capability* model in the dental clinic ubiquitous case study, in which the Personal Agent requests a specific capability – e.g. to use the patient's registration service – and the Dental Clinic's Manager Agent sends it by previously taking into account that the Patient under analysis – represented by the Personal Agent – is an old client of the dental clinic registered at the Yellow Pages. This analysis process is not demonstrated in the code fragments. The described mechanism improves the dependability/accountability/security of the dental clinic ubiquitous application. The Personal Agent's request is performed in the *KnowTheContextPlan.java* (Figure 6.26) and the Dental Clinic's Manager Agent sends the capability by using the *ExecuteRPRequestPlan.java* (Figure 6.27). The process also demands the use of the JADEX protocols (e.g. *FIPA Request Interaction Protocol (RP)*) to send/receive messages. Now, the Personal Agent has the capacity to register the patient into the dental clinic by following the service's business rules and policies – specified into the received *appRegistrationCap* – as well as to share information between the patient and the Dental Clinic's Manager Agent.

```

public class KnowTheContext extends Plan {
    public void body() {
        System.out.println("Personal agent running KnowTheContext plan.");
        this.findTheResponsiblesForTheDentalApps();
        ...
        this.requestTheApplicationRegistrationCapability();
    }
    private void findTheResponsiblesForTheDentalApps() {
        System.out.println("Personal agent running findTheResponsiblesForTheDentalApp task.");
        String domain = (String) this.getBeliefbase().getBelief("domain").getFact();
        IGoal df_search = createGoal("df_search");
        ...
        AgentDescription ad = SFipa.createAgentDescription(null, sd);
        df_search.getParameter("description").setValue(ad);
        dispatchSubgoalAndWait(df_search);
        if(result.length == 0) {
            System.out.println("Manager Agent not found in DF.");
            fail();
        }
        for(int i = 0; i < result.length; i++) {
            String appManager = result[i].getName().getName() + ",";
            appManager = appManager + result[i].getServices()[0].getOwnership();
            this.getBeliefbase().getBeliefSet("apps_managers").addFact(appManager);
        }
    }
    private void requestTheApplicationRegistrationCapability() {
        System.out.println("Personal agent running requestTheApplicationRegistrationCapability task.");
        IGoal requestProtocolBeInitiated = this.createGoal("request_protocol_be_initiated");
        AgentIdentifier managerAgentID = (AgentIdentifier) this.getBeliefbase().getBelief("manager_agent_identifier").getFact();

        requestProtocolBeInitiated.getParameter("receiver").setValue(managerAgentID);
        requestProtocolBeInitiated.getParameter("action").setValue("interaction_request");
        this.dispatchSubgoalAndWait(requestProtocolBeInitiated);
        String appRegistrationCap = (String) requestProtocolBeInitiated.getParameter("result").getValue();
        // Register capability at runtime.
        IMCapability capabilityModelElement = (IMCapability) this.getScope().getModelElement();
        IMCapabilityReference capabilityReference = capabilityModelElement.createCapabilityReference("appregistration", appRegistrationCap);

        this.getScope().registerSubcapability(capabilityReference);
        ...
        // Register goals at runtime
        IMGoalbase goalbaseModelElement = (IMGoalbase) this.getGoalbase().getModelElement();
        IMAchieveGoalReference goalref = goalbaseModelElement.createAchieveGoalReference("patient_be_registered", IMGoal.EXPORTED_FALSE, "appregistration.patient_be_registered");
        this.getGoalbase().registerGoalReference(goalref);
    }
}

```

Figure 6.26 - *KnowTheContextPlan.java* based on the *Privacy-Aware Capability* model

```

public class ExecuteRPRRequestPlan extends Plan {
    public ExecuteRPRRequestPlan() {
    }
    public void body() {
        System.out.println("Manager agent running ExecuteRPRRequestPlan.");
        Object action = this.getParameter("action").getValue();
        AgentIdentifier personalAgentID = (AgentIdentifier) this.getParameter("personal").getValue();
        if (action instanceof String) {
            String request = (String) action;
            //interaction_request
            if (request.equalsIgnoreCase("interaction_request")){
                System.out.println("Manager agent executing interaction_request.");
                this.getParameter("result").setValue("abcd.appregistration.AppRegistration");
            }
            else {
                //registration_request
                if (request.equalsIgnoreCase("registration_request")){
                    System.out.println("Manager agent executing registration_request.");
                    IGoal create = createGoal("ams_create_agent");
                    create.getParameter("type").setValue("abcd.registrationAgent.RegistrationAgent");
                    Map argsMap = SCollection.createHashMap();
                    argsMap.put("personal_agent_id", personalAgentID);
                    create.getParameter("arguments").setValue(argsMap);
                    dispatchSubgoalAndWait(create);
                    AgentIdentifier createdRegistrationAgentID = (AgentIdentifier) create.getParameter("agentidentifier").getValue();
                    this.getParameter("result").setValue(createdRegistrationAgentID);
                }
                else {
                    this.getParameter("result").setValue("fail.");
                }
            }
        }
        else {
            this.getParameter("result").setValue("fail.");
        }
    }
}

```

Figure 6.27 - *ExecuteRPRRequestPlan.java* based on the *Privacy-Aware Capability* model

6.1.3.2 Reuse of the Fuzzy-Logic-Based Support in lower abstraction levels

Based on the dental clinic's non-functional requirements, the software engineers instantiated the *Fuzzy-Logic-Based Support*, more specifically the *Fuzzy-Logic-Library* sub-packages (Figure 6.28) by considering different NFRs as quality criteria, represented as fuzzy variables.

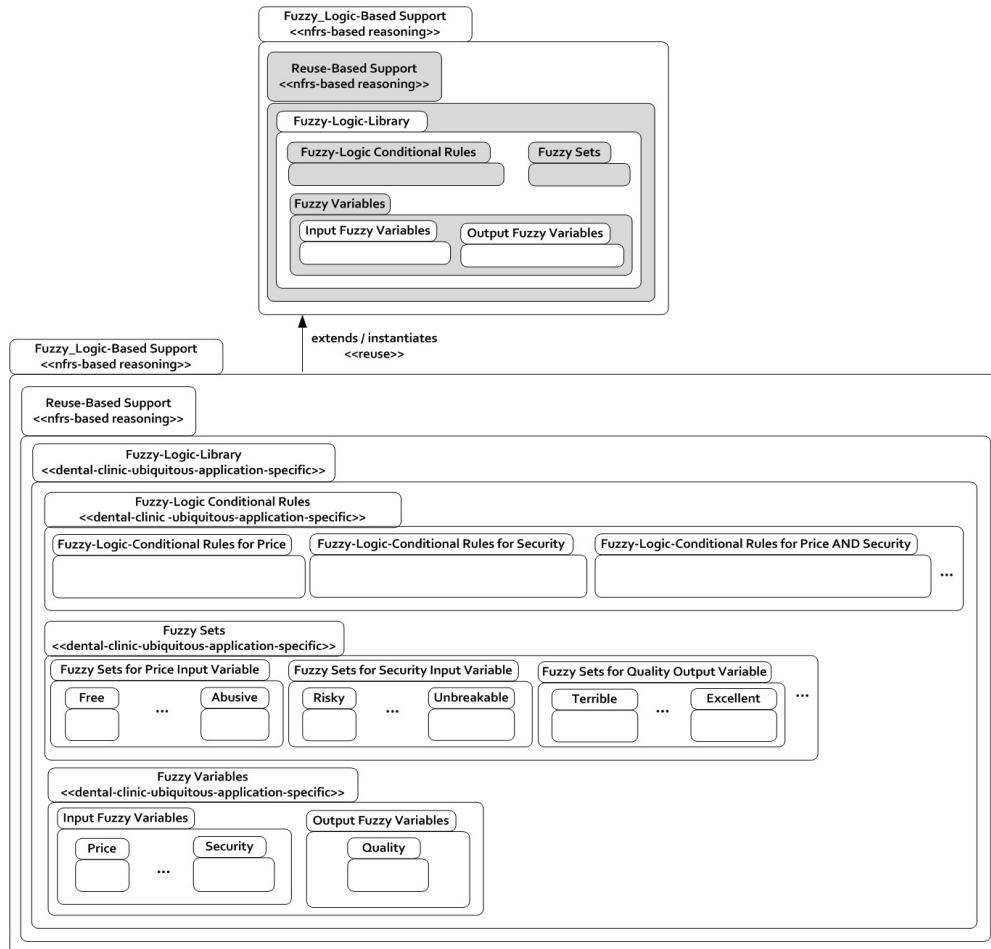


Figure 6.28 - Reuse of the *Fuzzy Logic-Based Support* in the *Dental Clinic Ubiquitous Application Engineering*

In our dental clinic ubiquitous application, the variables used in the fuzzy conditional rules are organized into: (i) fuzzy input variables, such as *Price*, *Security* and others, with their fuzzy sets; and (ii) fuzzy output variables, such as *Quality* and others, with their fuzzy sets. Figure 6.29 shows the fuzzy sets for the *Price* input variable – free, cheap, medium, high, expensive and abusive; the *Security* input variable – risky, weak, medium, strong and unbreakable; and the *Quality* output variable – terrible, low, bad, medium, good, high and excellent.

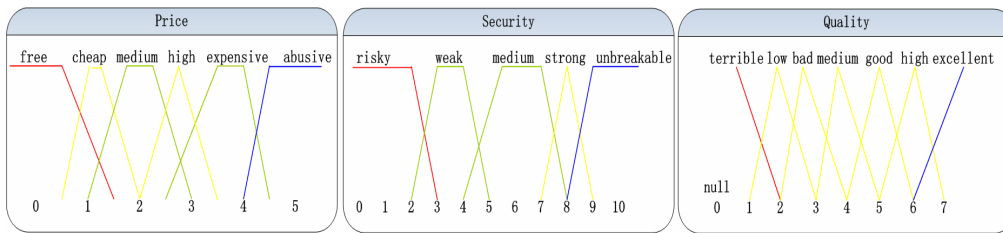


Figure 6.29 - The fuzzy logic sets for fuzzy variables

Table 6.3 shows the fuzzy-logic base for the dental clinic ubiquitous case study centered on those variables. In this ubiquitous application, the Personal Agent analyzes the price, the security and other specified quality criteria to choose the best service provider to attend the specified goals.

Table 6.3 - The fuzzy logic base for the dental clinic case study

Fuzzy Variables	Code
Price	<pre>// create price continuous variable ContinuousFuzzyRuleVariable price = new ContinuousFuzzyRuleVariable(sfrb, "price", 0.00, 5.00); price.addSetShoulder("free", 0.1, 0.50, 1.50, FuzzyDefs.LEFT); price.addSetTrapezoid("cheap", 0.1, 0.50, 1.00, 1.25, 2.00); price.addSetTrapezoid("medium", 0.1, 1.00, 1.75, 2.25, 3.00); price.addSetTriangle("high", 0.1, 2.00, 2.75, 3.50); price.addSetTrapezoid("expensive", 0.1, 2.50, 3.50, 4.00, 4.50); price.addSetShoulder("abusive", 0.1, 4.00, 4.50, FuzzyDefs.RIGHT);</pre>
Security	<pre>// create security continuous variable ContinuousFuzzyRuleVariable security = new ContinuousFuzzyRuleVariable(sfrb, "security", 0.0, 10.0); security.addSetShoulder("risky", 0.1, 2.0, 3.0, FuzzyDefs.LEFT); security.addSetTrapezoid("weak", 0.1, 2.0, 3.0, 4.0, 5.0); security.addSetTrapezoid("medium", 0.1, 4.0, 5.5, 7.0, 8.0); security.addSetTriangle("strong", 0.1, 7.0, 8.0, 9.0); security.addSetShoulder("unbreakable", 0.1, 8.0, 9.0, FuzzyDefs.RIGHT);</pre>
Quality	<pre>// create quality continuous variable ContinuousFuzzyRuleVariable quality = new ContinuousFuzzyRuleVariable(sfrb, "quality", 1.0, 7.0); quality.addSetShoulder("terrible", 0.1, 1.0, 2.0, FuzzyDefs.LEFT); quality.addSetTriangle("low", 0.1, 1.0, 2.0, 3.0); quality.addSetTriangle("bad", 0.1, 2.0, 3.0, 4.0); quality.addSetTriangle("medium", 0.1, 3.0, 4.0, 5.0); quality.addSetTriangle("good", 0.1, 4.0, 5.0, 6.0); quality.addSetTriangle("high", 0.1, 5.0, 6.0, 7.0); quality.addSetShoulder("excellent", 0.1, 6.0, 7.0, FuzzyDefs.RIGHT);</pre>
Others	...

Using fuzzy conditional rules, the Personal Agent establishes – at runtime – the quality output values and decides which provider is the best to satisfy the user (e.g. patient, dentist and attendant). These values are also used to update the Personal Agent's belief base by storing this information for a short period of time. Therefore, if an instantaneous request from the same user and device by asking to

perform the same goal is sent to the Personal Agent, this agent can quickly decide by using its proper knowledge – stored into its belief base – to answer the user. It avoids the Personal Agent’s overload and reduces the spent time by considering that the service’s price and security do not vary in a very short period of time.

Considering the specified fuzzy variables and sets the fuzzy-logic library was instantiated to attend the dental clinic application’s issues by specifying the fuzzy conditional rules. Therefore, the previous mentioned Personal Agent’s decision depends on running these fuzzy conditional rules at runtime. Table 6.4 shows, for example, some fuzzy conditional rules based on the fuzzy variables *Price* and *Security*.

Table 6.4 - The fuzzy conditional rules for the price and the security variables

Fuzzy Variables	Conditional Rules
Price	<pre> // if the price is abusive then the quality is terrible FuzzyRule price1Rule = new FuzzyRule(sfrb, "price1Rule", new FuzzyClause[]{ sfrb.createClause(price, compareIs, "", "abusive") }, sfrb.createClause(quality, assignIs, "", "terrible")); // if the price is expensive then the quality is low FuzzyRule price2Rule = new FuzzyRule(sfrb, "price2Rule", new FuzzyClause[]{ sfrb.createClause(price, compareIs, "", "expensive") }, sfrb.createClause(quality, assignIs, "", "low")); // if the price is high then the quality is bad FuzzyRule price3Rule = new FuzzyRule(sfrb, "price3Rule", new FuzzyClause[]{ sfrb.createClause(price, compareIs, "", "high") }, sfrb.createClause(quality, assignIs, "", "bad")); //Others ... </pre>
Security	<pre> // if the security is risky then the quality is terrible FuzzyRule security1Rule = new FuzzyRule(sfrb, "security1Rule", new FuzzyClause[]{ sfrb.createClause(security, compareIs, "", "risky") }, sfrb.createClause(quality, assignIs, "", "terrible")); // if the security is weak then the quality is bad FuzzyRule security2Rule = new FuzzyRule(sfrb, "security2Rule", new FuzzyClause[]{ sfrb.createClause(security, compareIs, "", "weak") }, sfrb.createClause(quality, assignIs, "", "bad")); //Others ... </pre>

For the *Price* variable: “*if the price is abusive then the quality is terrible.*”

For the *Security* variable: “*if the security is risky then the quality is terrible.*”

Applying the same reasoning, we dealt with the fuzzy conditional rules for the variables *Price* AND *Security* (the combination of those variables) by considering their respectively fuzzy sets: for price (abusive and others) and for security (risky, weak, medium and others). The output results are specified by the fuzzy variable *Quality* based on its fuzzy sets. To illustrate, Table 6.5 presents the fuzzy

conditional rules for the variables *Price* AND *Security* by focusing on the **abusive** fuzzy set. For example: “*if the price is abusive and the security is risky then the quality is terrible.*”

Table 6.5 - The fuzzy conditional rules for the variables price AND security (abusive fuzzy set)

Fuzzy Variables	Conditional Rules for the Abusive Fuzzy Set
Price and Security	<pre> // if the price is abusive and the security is risky then the quality is terrible FuzzyRule priceSecurity01Rule = new FuzzyRule(sfrb, "priceSecurity01Rule", new FuzzyClause[]{ sfrb.createClause(price, compareIs, "", "abusive"), sfrb.createClause(security, compareIs, "", "risky")}, sfrb.createClause(quality, assignIs, "", "terrible")); // if the price is abusive and the security is weak then the quality is terrible FuzzyRule priceSecurity02Rule = new FuzzyRule(sfrb, "priceSecurity02Rule", new FuzzyClause[]{ sfrb.createClause(price, compareIs, "", "abusive"), sfrb.createClause(security, compareIs, "", "weak")}, sfrb.createClause(quality, assignIs, "", "terrible")); // if the price is abusive and the security is medium then the quality is bad FuzzyRule priceSecurity03Rule = new FuzzyRule(sfrb, "priceSecurity03Rule", new FuzzyClause[]{ sfrb.createClause(price, compareIs, "", "abusive"), sfrb.createClause(security, compareIs, "", "medium")}, sfrb.createClause(quality, assignIs, "", "bad")); //Others ... </pre>

The Fuzzy-Logic Library is an API that can be instantiated as previously presented. The specified fuzzy conditional rules are used by the intentional agents of the dental clinic application to improve their cognitive capacity (their reasoning engine) – at runtime – by deciding about services that better satisfy the users according to some quality criteria. In this context, it is relevant to consider that the fuzzy conditional rules are previously specified and implemented. However, the decision is made – *on the fly* – by taking into account the services offered at the moment of the user’s request; the user’s preferences based on the profile information investigation and/or the user’s navigation investigation; and other ever-changing features. The *Intentional Agents’ Reasoning Building Block* of our approach already contemplates a reuse-based support that – among other contributions – integrates the Fuzzy-Logic-Based support with the agents’ reasoning engine. Therefore, in the *Ubiquitous Application Engineering*, it is only delegated to the software engineers the fuzzy variables, sets and conditional rules specification by following the API conceptual model and the application’s requirements under analysis.

Figure 6.30 shows a code fragment – an XML based on the proposed reasoning engine – with some beliefs, goals and intentions specified to find a

dental clinic service by considering an analysis of all dental clinic services centered on the fuzzy rules base. After the analysis, it is possible to determine the best dental clinic service. Moreover, Figures 6.31, 6.32 and 6.33 presents code fragments based on the fuzzy rule base's creation, service analysis and best service determination.

```

...
<beliefs>
<beliefset name="dental_clinic_services" class="AgentIdentifier"/>
<beliefset name="service_reports" class="ServiceReport"/>
<beliefset name="reasoning_results" class="ServiceQuality"/>
<belief name="fuzzy_rule_base" class="FuzzyRuleBase"/>
...
</beliefs>

<goals>
...
</achievegoal>
<achievegoal name="setup_reasoning_engine">
<creationcondition>$beliefbase.fuzzy_rule_base == null</creationcondition>
<targetcondition>$beliefbase.fuzzy_rule_base != null</targetcondition>
</achievegoal>
...
<achievegoal name="find_dental_clinic_service">
<creationcondition>$beliefbase.dental_clinic_services.length == 0</creationcondition>
<maintaingoal name="analyze_all_reports">
<maintaincondition>$beliefbase.service_reports.length == 0</maintaincondition>
...
</goals>

<plans>
<plan name="df_search_dental_clinic_service">
<body>new SearchDentalClinicServicePlan()</body>
<trigger>
<goal ref="find_dental_clinic_service"/>
</trigger>
</plan>
...
<plan name="analyze_service_report">
<body>new AnalyzeServiceReportPlan()</body>
<trigger>
<goal ref="analyze_all_reports"/>
</trigger>
</plan>
<plan name="create_fuzzy_rule_base">
<body>new CreateFuzzyRuleBasePlan()</body>
<trigger>
<goal ref="setup_reasoning_engine"/>
</trigger>
</plan>
...
<plan name="send_best_reports">
<body>new SendBestReportsPlan()</body>
<trigger>
<goal ref="limit_time_be_considered"/>
</trigger>
</plan>
...
</plans>

<configurations>
<configuration name="default">
<goals>
<initialgoal name="start1" ref="setup_reasoning_engine"/>
...
<initialgoal name="start4" ref="find_dental_clinic_service"/>
<initialgoal name="start5" ref="analyze_all_reports"/>
...
</goals>
</configuration>
</configurations>

```

Figure 6.30 - XML code fragment with beliefs, goals and intentions specified to find a dental clinic service centered on the fuzzy rules base


```

public class CreateFuzzyRuleBasePlan extends Plan {
    /** Creates a new instance of CreateFuzzyRuleBasePlan */
    public CreateFuzzyRuleBasePlan() {
    }
    public void body() {
        FuzzyRuleBase sfrb = new FuzzyRuleBase("Service Fuzzy Rule Base");
        // set correlation method one of Product, Minimize
        sfrb.setCorrelationMethod(FuzzyDefs.PRODUCT);
        // set inferencing method, one of FuzzyAdd, MinMax, ProductOr
        sfrb.setInferenceMethod(FuzzyDefs.FUZZYADD);
        // set defuzzification method, one of Centroid, MaxHeight
        sfrb.setDefuzzifyMethod(FuzzyDefs.CENTROID);
        ...
        // create price continuous variable
        ContinuousFuzzyRuleVariable price = new ContinuousFuzzyRuleVariable(sfrb, "price", 0.00, 5.00);
        price.addSetShoulder("free", 0.1, 0.50, 1.50, FuzzyDefs.LEFT);
        price.addSetTrapezoid("cheap", 0.1, 0.50, 1.00, 1.25, 2.00);
        price.addSetTrapezoid("medium", 0.1, 1.00, 1.75, 2.25, 3.00);
        price.addSetTriangle("high", 0.1, 2.00, 2.75, 3.50);
        price.addSetTrapezoid("expensive", 0.1, 2.50, 3.50, 4.00, 4.50);
        price.addSetShoulder("abusive", 0.1, 4.00, 4.50, FuzzyDefs.RIGHT);
        // create security continuous variable
        ContinuousFuzzyRuleVariable security = new ContinuousFuzzyRuleVariable(sfrb, "security", 0.0, 10.0);
        security.addSetShoulder("risky", 0.1, 2.0, 3.0, FuzzyDefs.LEFT);
        security.addSetTrapezoid("weak", 0.1, 2.0, 3.0, 4.0, 5.0);
        security.addSetTrapezoid("medium", 0.1, 4.0, 5.5, 7.0, 8.0);
        security.addSetTriangle("strong", 0.1, 7.0, 8.0, 9.0);
        security.addSetShoulder("unbreakable", 0.1, 8.0, 9.0, FuzzyDefs.RIGHT);
        ...
        // create quality continuous variable
        ContinuousFuzzyRuleVariable quality = new ContinuousFuzzyRuleVariable(sfrb, "quality", 1.0, 7.0);
        quality.addSetShoulder("terrible", 0.1, 1.0, 2.0, FuzzyDefs.LEFT);
        quality.addSetTriangle("low", 0.1, 1.0, 2.0, 3.0);
        quality.addSetTriangle("bad", 0.1, 2.0, 3.0, 4.0);
        quality.addSetTriangle("medium", 0.1, 3.0, 4.0, 5.0);
        quality.addSetTriangle("good", 0.1, 4.0, 5.0, 6.0);
        quality.addSetTriangle("high", 0.1, 5.0, 6.0, 7.0);
        quality.addSetShoulder("excellent", 0.1, 6.0, 7.0, FuzzyDefs.RIGHT);
        ...
        // Rules based on Unique Variable //
        // make conditional rule
        FuzzyRule price1 = new FuzzyRule(sfrb, "price1Rule",
            new FuzzyClause[] { sfrb.createClause(price, compareIs, "", "abusive"),
                               sfrb.createClause(quality, assignIs, "", "terrible") });
        ...
        // make conditional rule
        FuzzyRule security1Rule = new FuzzyRule(sfrb, "security1Rule",
            new FuzzyClause[] { sfrb.createClause(security, compareIs, "", "risky"),
                               sfrb.createClause(quality, assignIs, "", "terrible") });
        ...
        // Rules based on Rules Price AND Security //
        // make conditional rule
        FuzzyRule priceSecurity01Rule = new FuzzyRule(sfrb, "priceSecurity01Rule",
            new FuzzyClause[] { sfrb.createClause(price, compareIs, "", "abusive"),
                               sfrb.createClause(security, compareIs, "", "risky"),
                               sfrb.createClause(quality, assignIs, "", "terrible") });
        ...
        this.getBeliefbase().getBelief("fuzzy_rule_base").setFact(sfrb);
    }
}

```

Figure 6.31 - Code fragment based on the fuzzy rule base's creation

```

...
public class AnalyzeServiceReportPlan extends Plan {
    /** Creates a new instance of AnalyzeServiceReportPlan */
    public AnalyzeServiceReportPlan() {
    }
    public void body() {
        ServiceReport[] reportsToAnalyze = (ServiceReport[]) this.getBeliefbase().getBeliefSet("service_reports").getFacts();
        for(int i = 0; i < reportsToAnalyze.length; i++) {
            FuzzyRuleBase sfrb = (FuzzyRuleBase) this.getBeliefbase().getBelief("fuzzy_rule_base").getFact();
            ContinuousFuzzyRuleVariable price = (ContinuousFuzzyRuleVariable) sfrb.getVariable("price");
            ContinuousFuzzyRuleVariable security = (ContinuousFuzzyRuleVariable) sfrb.getVariable("security");
            ...
            ContinuousFuzzyRuleVariable quality = (ContinuousFuzzyRuleVariable) sfrb.getVariable("quality");
            double priceVal = reportsToAnalyze[i].getEvaluation().getPrice();
            double securityVal = reportsToAnalyze[i].getEvaluation().getSecurity();
            ...
            System.out.println("\n\nValues: "+priceVal+" "+securityVal+"\n\n");
            price.setNumericValue(priceVal);
            security.setNumericValue(securityVal);
            ...
            quality.setNumericValue(7);
            System.out.println("\n\nSymbolic: "+price.getSymbolicValue()+"\n\n");
            System.out.println("Symbolic: "+security.getSymbolicValue()+"\n\n");
            ...
            sfrb.forwardChain();
            System.out.println("\n\nService Analysis: "+quality.getNumericValue()+"\n\n");
            ServiceQuality reasoningResult = new ServiceQuality(quality.getNumericValue(), reportsToAnalyze[i]);
            this.getBeliefbase().getBeliefSet("reasoning_results").addFact(reasoningResult);
            this.getBeliefbase().getBeliefSet("service_reports").removeFact(reportsToAnalyze[i]);
            ...
        }
    }
}

```

Figure 6.32 - Code fragment based on the service analysis

```

...
public class SendBestReportsPlan extends Plan {
    /** Creates a new instance of SendBestReportsPlan */
    public SendBestReportsPlan() {
    }
    public void body() {
        System.out.println("\n\nRunning SendBestReportsPlan.\n\n");
        Long limit_time = (Long) this.getBeliefbase().getBelief("limit_time").getFact();
        if (limit_time == 0) {
            this.getBeliefbase().getBelief("timer").setFact(0);
        }
        else {
            ServiceQuality[] qualities = (ServiceQuality[]) this.getBeliefbase().getBeliefSet("reasoning_results").getFacts();
            IMessageEvent newMSG = this.createMessageEvent("inform_analysis_result");
            newMSG.getParameterSet(SFipa.RECEIVERS).addValue((AgentIdentifier) this.getBeliefbase().getBelief("personal").getFact());
            newMSG.getParameter(SFipa.REPLY_WITH).setValue("analysis"+System.currentTimeMillis());
            IMessageEvent reply = this.sendMessageAndWait(newMSG);
            ...
            reply = this.sendMessageAndWait(reply);
            if (qualities.length == 0) {
                reply = reply.createReply("service_not_found");
                reply.getParameter(SFipa.REPLY_WITH).setValue("analysis"+System.currentTimeMillis());
                reply = this.sendMessageAndWait(reply);
            }
            else {
                ServiceQuality bestServiceQuality = qualities[0];
                ServiceQuality secondServiceQuality = null;
                ServiceQuality thirdServiceQuality = null;
                for (int i = 1; i < qualities.length; i++) {
                    if (qualities[i].getQuality() > bestServiceQuality.getQuality()) {
                        thirdServiceQuality = secondServiceQuality;
                        secondServiceQuality = bestServiceQuality;
                        bestServiceQuality = qualities[i];
                    }
                    ...
                }
                reply = reply.createReply("send_service", qualities[0]);
                reply.getParameter(SFipa.REPLY_WITH).setValue("analysis"+System.currentTimeMillis());
                reply = this.sendMessageAndWait(reply);
                System.out.println("\n\nProtocol ended.\n\n");
            }
        }
    }
}

```

Figure 6.33 - Code fragment based on the best service determination

6.1.3.3 Reuse of the Dynamic Interface Construction Building Block in lower abstraction levels

In the *Dental Clinic Ubiquitous Applications Engineering* the software engineers reused the MIDP GUI Ontology, obtained from the *Dynamic Interface Construction Building Block*, to dynamically construct interface elements at runtime for limited devices (i.e. MIDP devices). Therefore, they are also centered on the *FIPA Standards Ontological Support* and the *Graphical User Interface (GUI)*. Both support packages helped us to develop the *GUI Generic Ontology* to improve the agents' communication and inter-operability as well as to standardize the interface construction "on the fly".

However, the dental clinic ubiquitous application demands specific interface elements. In order to implement an ontology specific for the dental clinic under analysis, the software engineers must define the dental-clinic-based asserted model for this ontology. To illustrate, we consider the asserted model for the ontology of dental clinic forms in Figure 6.34. Moreover, the XML code (Protege 2011) of the dental clinic forms ontology is shown in Figure 6.35.

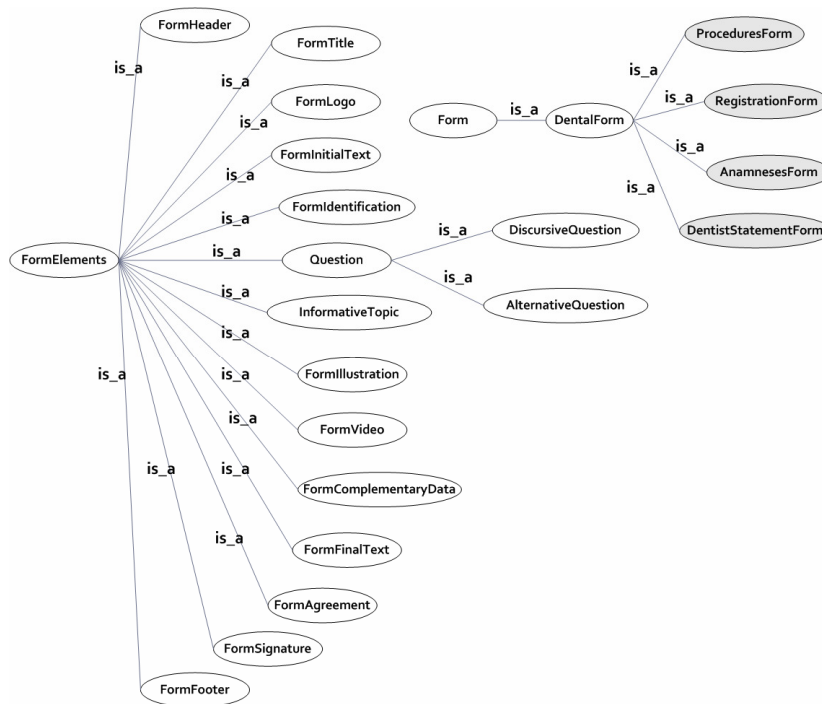


Figure 6.34 - The asserted model for the ontology of the dental clinic forms

```

<?xml version="1.0"?>
<!DOCTYPE Ontology [
...
<!ENTITY dentalFormsOntology
...
]>
<Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#"
...
  <SubClassOf>
    <Class URI="&dentalFormsOntology;AnamnesesDentalForm"/>
    <Class URI="&dentalFormsOntology;DentalForm"/>
  </SubClassOf>
  <SubClassOf>
    <Class URI="&dentalFormsOntology;AnamnesesDentalForm"/>
    <ObjectSomeValuesFrom>
      <ObjectProperty URI="&dentalFormsOntology;canContain"/>
      <Class URI="&dentalFormsOntology;AlternativeQuestion"/>
    </ObjectSomeValuesFrom>
  </SubClassOf>
</Ontology>
<!-- Generated by the OWL API (version 2.2.1.1138)
http://owlapi.sourceforge.net -->

```

Figure 6.35 - The XML code of the dental clinic forms ontology

According to our empirical studies, the development of this ontology helped the software engineers to improve their knowledge in relation to the dental clinic ubiquitous application by also contributing to refine the requirements and design assumptions modeled in previous disciplines. As a contribution of our research, this ontology is now available for reuse and it can be instantiated or extended in other projects from the dental clinic cognitive domain to deal with different dental clinic forms.

Based on the described ontology, the dental clinic forms can be: procedures form, registration form, anamneses form or dentists' statements. Moreover, these forms are composed of different form elements (e.g. form title

and question). For example, the anamneses form must contain question(s) (e.g. alternative question and/or discursive question), agreement, identification, signature and title. Furthermore, it can be specialized by using the elements: supplemental data, final text, footer, header, illustration, initial text and logo. Figure 6.36 shows an example of the anamneses form and its interface elements.

Figure 6.36 - Instances of the developed dental forms ontology for the *Anamneses Form*

The presented elements are instances of the *Dental Clinic Forms Ontology*. Therefore, different dental forms can be obtained by instantiating this dental-clinic-based ontology.

Another important step for dealing with the dynamic interface construction based on the *Dental Clinic Forms Ontology* is the determination of the heuristics by permitting the correlation of this new ontology and the *GUI Generic Ontology*. Table 6.6 illustrates these relationships. For example, in the dynamic interface construction of the dental form shown in Figure 6.36 – at runtime – the *AnamnesesDentalForm* of the *Dental Clinic Forms Ontology* is represented as a *Form* in the *GUI Generic Ontology* and the *FormTitle* as a *StringItem*.

Table 6.6 - Dental Forms Elements & GUI Elements

Dental Form Element	GUI Element
AnamnesesDentalForm	Form
FormLogo	ImageItem
FormTitle	StringItem
FormIdentification	StringItem
FormInitialText	StringItem
FormAlternativeQuestion	ChoiceGroup
...	...
FormSignature	DigitalSignature
FormFooter	StringItem

Again, the dental clinic forms ontological support can be reused in different dental clinic applications. However, for applications from other cognitive domains (e.g. e-commerce and educational), it is necessary to construct an ontology that describes the interface elements of the desired application and to establish the heuristics that associate this ontology with the *GUI Generic Ontology*.

Based on the *Dental Clinic Forms Ontology* asserted model, the software engineers will implement it by defining an Ontological Java class and a vocabulary. Therefore, it is necessary to add the elements schemas that describe the structure of the concepts, agent actions, and predicates of the exchanged messages. Figure 6.37 shows the resulting Java class for the *Dental Clinic Forms Ontology* as well as Figure 6.38 presents its resulting vocabulary.

```

public DentalFormsOntology() {
    super(ONTOLOGY_NAME, BasicOntology.getInstance(), introspector);
    try {
        // classes
        add(new AgentActionSchema(SEND_DENTAL_FORM, new SendDentalForm().getClass());
        add(new ConceptSchema(FORM_ELEMENT), new FormElement().getClass());
        add(new ConceptSchema(QUESTION), new Question().getClass());
        add(new ConceptSchema(INFORMATIVE_TOPIC), new InformativeTopic().getClass());
        add(new ConceptSchema(FORM_TITLE), new FormTitle().getClass());
        add(new ConceptSchema(DENTAL_FORM), new DentalForm().getClass());
        ...
        // schemas
        AgentActionSchema aasSendDentalForm = (AgentActionSchema) this.getSchema(SEND_DENTAL_FORM);
        aasSendDentalForm.add(SEND_DENTAL_FORM_FORM_TYPE, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

        ConceptSchema csFormElement = (ConceptSchema) this.getSchema(FORM_ELEMENT);
        csFormElement.add(FORM_ELEMENT_TEXT, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

        ConceptSchema csQuestion = (ConceptSchema) this.getSchema(QUESTION);
        csQuestion.addSuperSchema(csFormElement);
        csQuestion.add(QUESTION_TYPE, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
        ...

        ConceptSchema csFormTitle = (ConceptSchema) this.getSchema(FORM_TITLE);
        csFormTitle.addSuperSchema(csFormElement);
        csFormTitle.add(FORM_TITLE_POSITION, (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
        ...

    } catch (OntologyException oe) {
        oe.printStackTrace();
    }
}
}

```

Figure 6.37 - Dental Clinic Forms Ontological Java class

```

public class DentalFormsOntology extends Ontology {
    /** The vocabulary of the ontology. */
    public static final String ONTOLOGY_NAME = "dental-forms";

    public static final String SEND_DENTAL_FORM = "SendDentalForm";
    public static final String SEND_DENTAL_FORM_FORM_TYPE = "formType";

    public static final String FORM_ELEMENT = "FormElement";
    public static final String FORM_ELEMENT_TEXT = "text";

    public static final String QUESTION = "Question";
    public static final String QUESTION_TEXT = "text";
    public static final String QUESTION_TYPE = "type";

    public static final String INFORMATIVE_TOPIC = "InformativeTopic";
    public static final String INFORMATIVE_TOPIC_FONT_STYLE = "fontStyle";

    public static final String FORM_TITLE = "FormTitle";
    public static final String FORM_TITLE_TEXT = "text";
    public static final String FORM_TITLE_POSITION = "position";

    public static final String DENTAL_FORM = "DentalForm";
    public static final String DENTAL_FORM_FORM_ELEMENTS = "formElements";
    ...
}

```

Figure 6.38 - Dental Clinic Forms Ontology vocabulary for *Interface Elements*

There are three steps to conclude the ontology: (i) define the content language; (ii) register the content language and the ontology using a software agent; and (iii) create or manipulate the content expressions as Java Objects.

In the first step, the software engineers decided to use the SL language in the dental clinic case study. Moreover, Figure 6.39 shows the registration of the content language as a JAVA code fragment by using a behavioral agent called *Interface Agent*. This agent only works with the MIDP GUI Ontology.

```

protected void setup() {
    myGui = new MIDPGuiOntology(this);
    this.interfaceAID = this.getAID();
    this.getContentManager().registerLanguage(this.codec);
    this.getContentManager().registerOntology(this.ontology);
    ...
}

```

Figure 6.39 - Registering SL content language and *MIDP GUI Ontology* using the behavioral *Interface Agent* in the dental clinic ubiquitous application

Figure 6.40 illustrates the registration using an intentional agent – called *Personal Agent* – by following the JADEX specifications and the BDI notation. It is represented as an XML code fragment of the *Personal Agent* (*property tag*).

```

...
<properties>
  ...
  <property name="contentcodec.fipas1-dentalforms">
    new JadeContentCodec(new SLCodec(), new DentalFormsOntology())
  </property>
  ...
</properties>
...

```

Figure 6.40 - Registering SL content language and *Dental Forms Ontology* using the intentional *Personal Agent* in the dental clinic ubiquitous application

Furthermore, Figure 6.41 focuses on the creation and manipulation of content expressions as Java Objects by presenting the code fragment about this step using the *Interface Agent*.

```

public void retrieveMIDPForm() {
    addBehaviour(new OneShotBehaviour() {
        public void action() {
            //Creation
            System.out.println("Creating");
            SendMIDPForm sendForm = new SendMIDPForm("registration_form");
            Action action = new Action(personalAID, sendForm);

            ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
            request.addReceiver(personalAID);
            request.setLanguage(codec.getName());
            request.setOntology(ontology.getName());
            request.setProtocol("fipa-request");
            request.setConversationId("Data Request " + myAgent.getName());
            request.setReplyWith(String.valueOf(System.currentTimeMillis())); // Unique value

            //Manipulation
            try {
                System.out.println("Encapsulating");
                myAgent.getContentManager().fillContent(request, action);
            } catch (CodecException ex) {
                System.out.println("Error with codec.");
                ex.printStackTrace();
            } catch (OntologyException ex) {
                System.out.println("Error with ontology.");
                ex.printStackTrace();
            }

            System.out.println("Sending");
            myAgent.send(request);
            ...
        }
    });
}

```

Figure 6.41 - Creating/manipulating the content expressions of the *MIDP GUI Ontology* as Java objects using the *Interface Agent* in the dental clinic ubiquitous application

Finally, in order to create and manipulate the content expressions using intentional agents, it is necessary to implement the *DecideRPRequestPlan* and the *ExecuteRPRequestPlan* as plans of the *Personal Agent*. Figures 6.42 and 6.43 respectively present code fragments of these plans.

```

...
public DecideRPRequestPlan() {
}
public void body() {

    System.out.println("Personal Agent running DecideRPRequestPlan.");
    ...
    if (action instanceof Action) {
        System.out.println("Personal Agent received fipa-sl Action.");

        Action fipaslAction = (Action) action;
        Concept concept = fipaslAction.getAction();

        if (concept instanceof SendDentalForm) {
            this.getParameter("accept").setValue(true);
        } else {
            this.getParameter("accept").setValue(false);
        }
    } else {
        this.getParameter("accept").setValue(false);
    }
}
}
}

```

Figure 6.42 - Manipulating the content expressions of the *Dental Forms Ontology* using the *Personal Agent (DecideRPRequestPlan)* in the dental clinic ubiquitous application

```

...
public ExecuteRPRequestPlan() {
}
public void body() {
    System.out.println("Personal Agent running ExecuteRPRequestPlan.");
    Object action = this.getParameter("action").getValue();

    if (action instanceof Action) {
        System.out.println("Personal Agent received fipa-sl Action.");
        Action fipaslAction = (Action) action;
        Concept concept = fipaslAction.getAction();

        if (concept instanceof SendDentalForm) {
            System.out.println("Personal Agent executing SendDentalForm.");

            SendDentalForm sendDentalForm = (SendDentalForm) concept;

            if (sendDentalForm.getFormType().equalsIgnoreCase("anamneses")) {
                DentalForm AnamnesesForm = new DentalForm();
                FormTitle formTitle = new FormTitle("Anamneses Form", "centered");
                ...
                Question q1 = new Question("Do you have a good health", "yesno");
                ...
                ArrayList all = new ArrayList();
                all.add(formTitle);
                ...
                all.add(q1);
                ...
                anamnesesForm.setFormElements(all);
                ...
            }
        }
    }
}
}

```

Figure 6.43 - Manipulating the content expressions of the *Dental Forms Ontology* using the *Personal Agent (ExecuteRPRequestPlan)* in the dental clinic ubiquitous application

Figure 6.44 summarizes the described process by demonstrating the association of the *Clinic Dental Forms Ontology* and the *GUI Generic Ontology* by using an *Adaptation Service*.

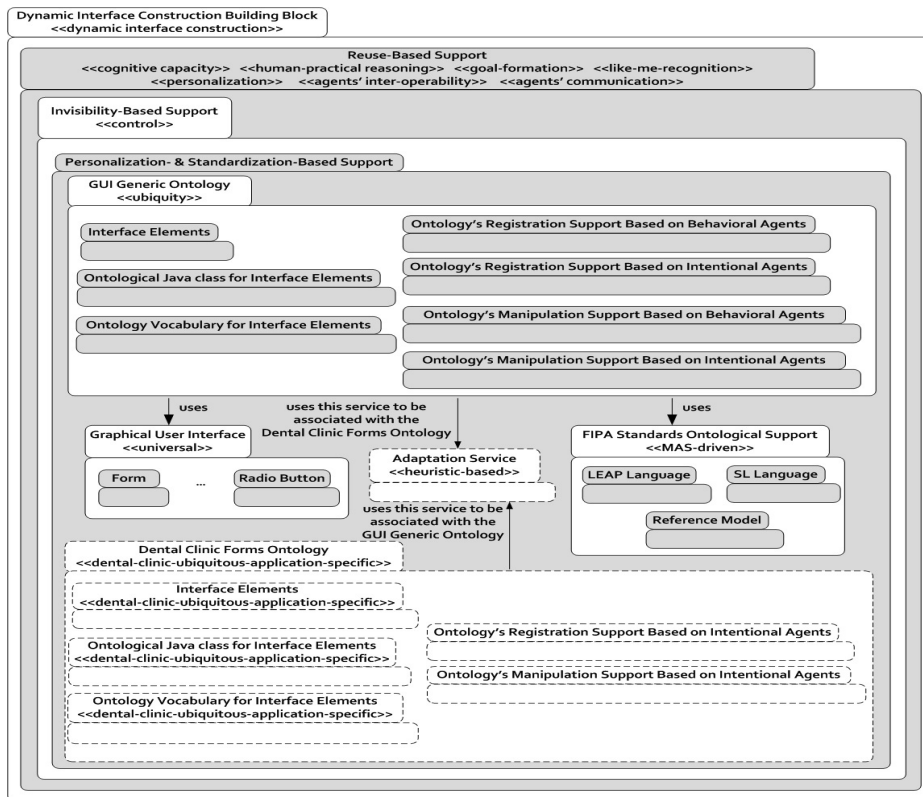


Figure 6.44 - *Adaptation Service* by associating the *Dental Clinic Forms Ontology* and the *GUI Generic Ontology* in the *Dental Clinic Ubiquitous Application Engineering*

6.1.3.4 Reuse of the Ubiquity Issues Building Block in lower abstraction levels

Other important packages that are instantiated to systematically develop the dental clinic ubiquitous application are based on the *Ubiquity Issues Building Blocks*. One of these specific ubiquitous issues is the content adaptation.

The content adaptation is a concern in ubiquitous scenarios as well as in the dental clinic ubiquitous application under analysis. Our main goal by providing reuse-based support for content adaptability issue is to facilitate the adaptation process modeling, design and implementation. We already demonstrated the reuse of the framework's models in the disciplines of higher abstraction levels. Now, we focus our attention on the reuse in the Implementation discipline. Motivated by this goal, the IFCAUC is provided as an API – called *ifcauc.jar*. The software engineers must add this file into the class library of the dental clinic ubiquitous application. Moreover, the framework offers different hot-spots based on plans generalization. Therefore, the software engineers must

implement these hot-spots. The IFCAUC already provides content adaptations based on, for example, image, text, video and audio contents. However, it is necessary to extend or instantiate some capabilities as well as the desired adaptation service(s) by attending the dental clinic ubiquitous application's specific goals. Here, it is relevant to remember that the ontological support, which was developed to adapt the dental clinic forms to heterogeneous devices by allowing the dynamic construction of dental-clinic-forms-based interfaces, can be viewed as a specific adaptation service. As the adaptation service is a hot-spot in the IFCAUC, it is also prepared to deal with this specific adaptation service.

There are different agents involved in the adaptation process. Figure 6.45 illustrates the *AbstractDecideServiceRequestPlan.java* and Figure 6.46 shows the *AbstractExecuteServiceRequestPlan.java* for the ContentAdapter Agent from the *ifcauc.contentadapter* package, which are instantiated to implement the Adapter Agent of the dental clinic ubiquitous application. In the *AbstractDecideServiceRequestPlan.java* code fragment, a new instance of the *AbstractDecideServiceRequestPlan* is created. The Initiator Agent represents the agent that requests the adaptation of a specific content. Thus, it represents the agent that initializes the adaptation process. In our dental clinic ubiquitous application, this agent is represented by the Personal Agent, whose content-adaptation-based goals are delegated to the Mobile Agent in order to achieve them by migrating to a dedicated server. Furthermore, the adaptation service is decided based on the request's analysis. In the *AbstractExecuteServiceRequestPlan.java* code fragment, a new instance of the *AbstractExecuteServiceRequestPlan* is created. Finally, the adaptation service request is executed.

```

public abstract class AbstractDecideServiceRequestPlan extends Plan {
    private AgentIdentifier initiator;
    private Object action;
    //Creates a new instance of AbstractDecideServiceRequestPlan
    public AbstractDecideServiceRequestPlan() {
        this.initiator = (AgentIdentifier) this.getParameter("initiator").getValue();
        this.action = this.getParameter("action").getValue();
        ...
    }
    public void body() {
        System.out.println("\n\nContentAdapter Capability running AbstractDecideServiceRequestPlan.\n\n");
        if (action instanceof String && ((String) action).equalsIgnoreCase("Location Request")) {
            this.getParameter("accept").setValue(true);
            ...
        }
        else {
            if (action instanceof AdaptationRequest) {
                Boolean accept = decideServiceRequest((AdaptationRequest) this.action);
                this.getParameter("accept").setValue(accept.booleanValue());
                ...
            }
            else {
                this.getParameter("accept").setValue(false);
            }
        }
    }
}
public abstract Boolean decideServiceRequest(AdaptationRequest action);
}

```

Figure 6.45 - Code fragment of the *AbstractDecideServiceRequestPlan.java* of the *IFCAUC*

```

public abstract class AbstractExecuteServiceRequestPlan extends Plan {
    private AgentIdentifier initiator;
    private Object action;
    //Creates a new instance of AbstractExecuteServiceRequestPlan
    public AbstractExecuteServiceRequestPlan() {
        this.initiator = (AgentIdentifier) this.getParameter("initiator").getValue();
        this.action = this.getParameter("action").getValue();
        ...
    }
    public void body() {
        System.out.println("\n\nContentAdapter Capability running AbstractExecuteServiceRequestPlan.\n\n");
        if (action instanceof String && ((String) action).equalsIgnoreCase("Location Request") ) {
            Agent myAgent = (Agent) this.getScope().getPlatformAgent();
            Location adapterLocation = myAgent.here();
            this.getParameter("result").setValue(adapterLocation);
            ...
        }
        else {
            if (action instanceof AdaptationRequest) {
                ...
                Object result = executeServiceRequest((AdaptationRequest) this.action);
                this.getParameter("result").setValue(result);
                ...
            }
        }
    }
    public abstract Object executeServiceRequest(AdaptationRequest action);
}

```

Figure 6.46 - Code fragment of the *AbstractExecuteServiceRequestPlan.java* of the *IFCAUC*

Figure 6.47 summarizes the content adaptation process by considering the ontological support developed for the dental clinic forms as well as other mechanisms that improve the intentional agents' cognitive capacity such as the fuzzy logic library support. All the agents' interactions are based on the ontology-based support that both standardizes and facilitates the inter-operability and the communication of these agents. Moreover, we emphasize the use of the Mobile Agent and the Adapter Agent in the adaptation process.

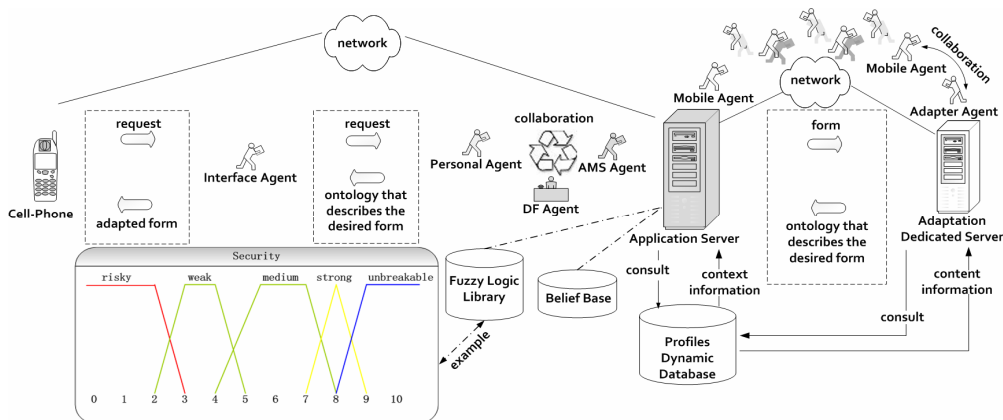


Figure 6.47 - Content adaptation process in the dental clinic ubiquitous application

Figure 6.48 illustrates the reuse of the *Ubiquity Issues Building Block*, more precisely the *Ubiquity-Based Frameworks* package (e.g. *IFCAUC*), in the *Dental Clinic Ubiquitous Application Engineering*. The dental-clinic-ubiquitous-application-specific Adaptation Service and the use of the *Fuzzy-Logic Library*, *Adaptation Techniques* (e.g. resizing and transcoding) and *Dental Clinic Forms Ontology* are also demonstrated.

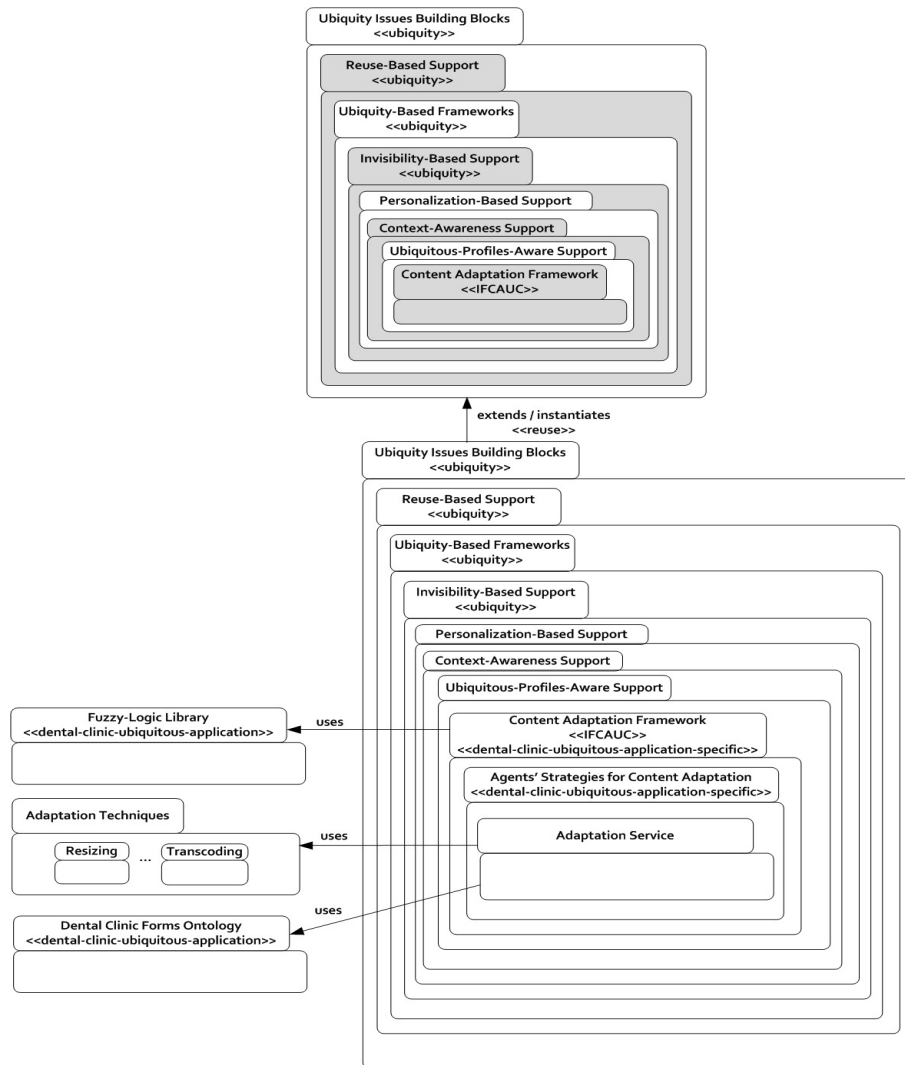


Figure 6.48 - Reuse of the *Ubiquity Issues Building Block* in the *Dental Clinic Ubiquitous Application Engineering*

6.1.3.5 Reuse of the Dynamic Database Building Block in lower abstraction levels

In the persistence layer, software engineers can instantiate the *Dynamic Data Model Support* package from the *Dynamic Database Building Block* (Figure 6.49). In this field, the ubiquitous profiles of the dental clinic ubiquitous application are obtained by instantiating this *Dynamic Data Model* to store: (i) information of patients, dentists, attendants and others in the *User Profile*; (ii) features of the devices in the *Device Profile*; (iii) network specification in the *Network Profile*; (iii) contract information between the clients and the services of the dental clinic under analysis in the *Contract Profile*; and others.

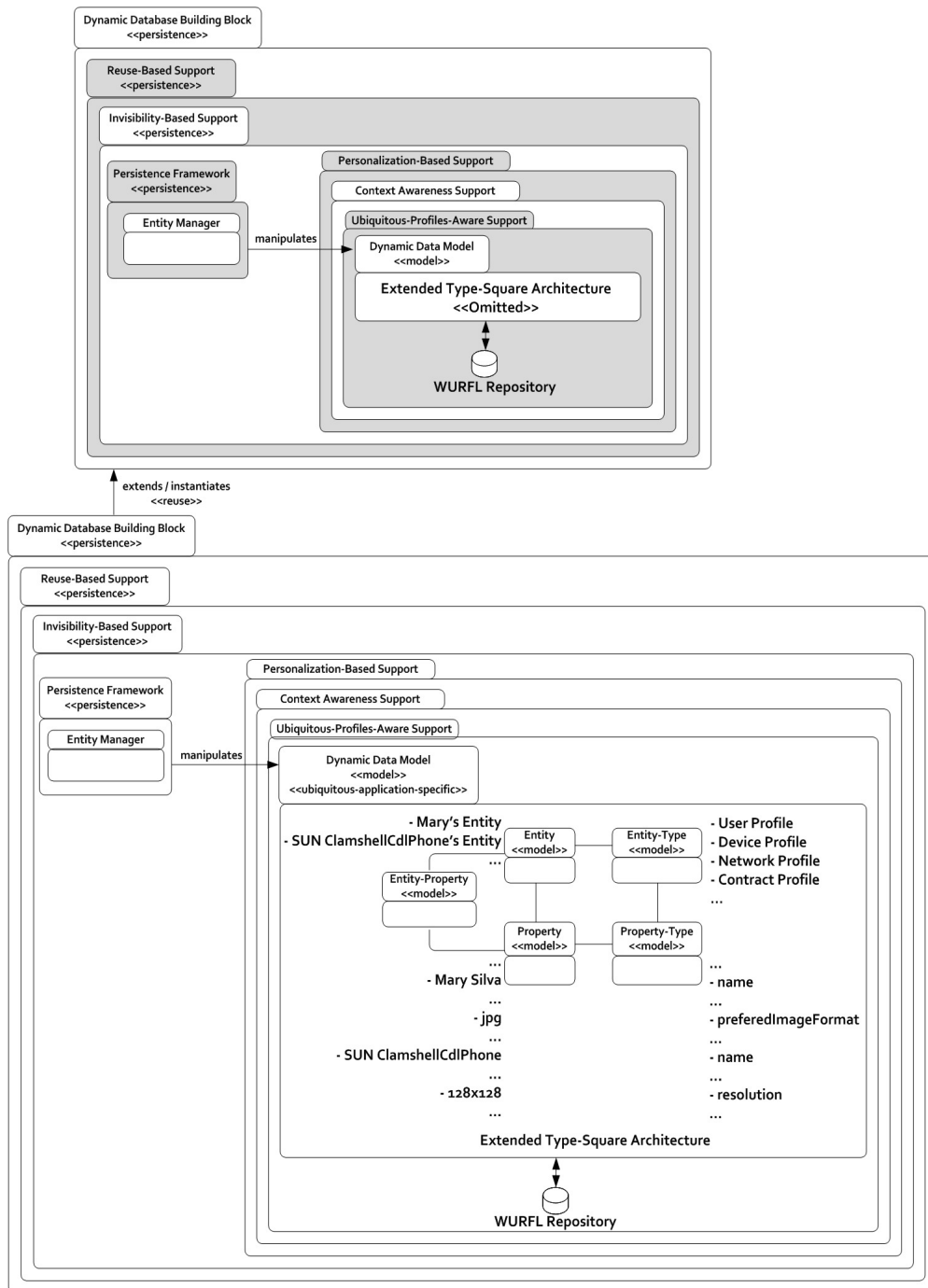


Figure 6.49 - Reuse of the *Dynamic Database Building Block* in the *Dental Clinic Ubiquitous Application Engineering*

Figure 6.50 presents an instance of the profiles – at the moment of the access – obtained from the dynamic database. This instance is used to adapt a specific content (e.g. an x-ray) according to the dentist’s device features (e.g. 128x128 pixels in terms of resolution) and the dentist’s preferences (e.g. jpg for the image format).

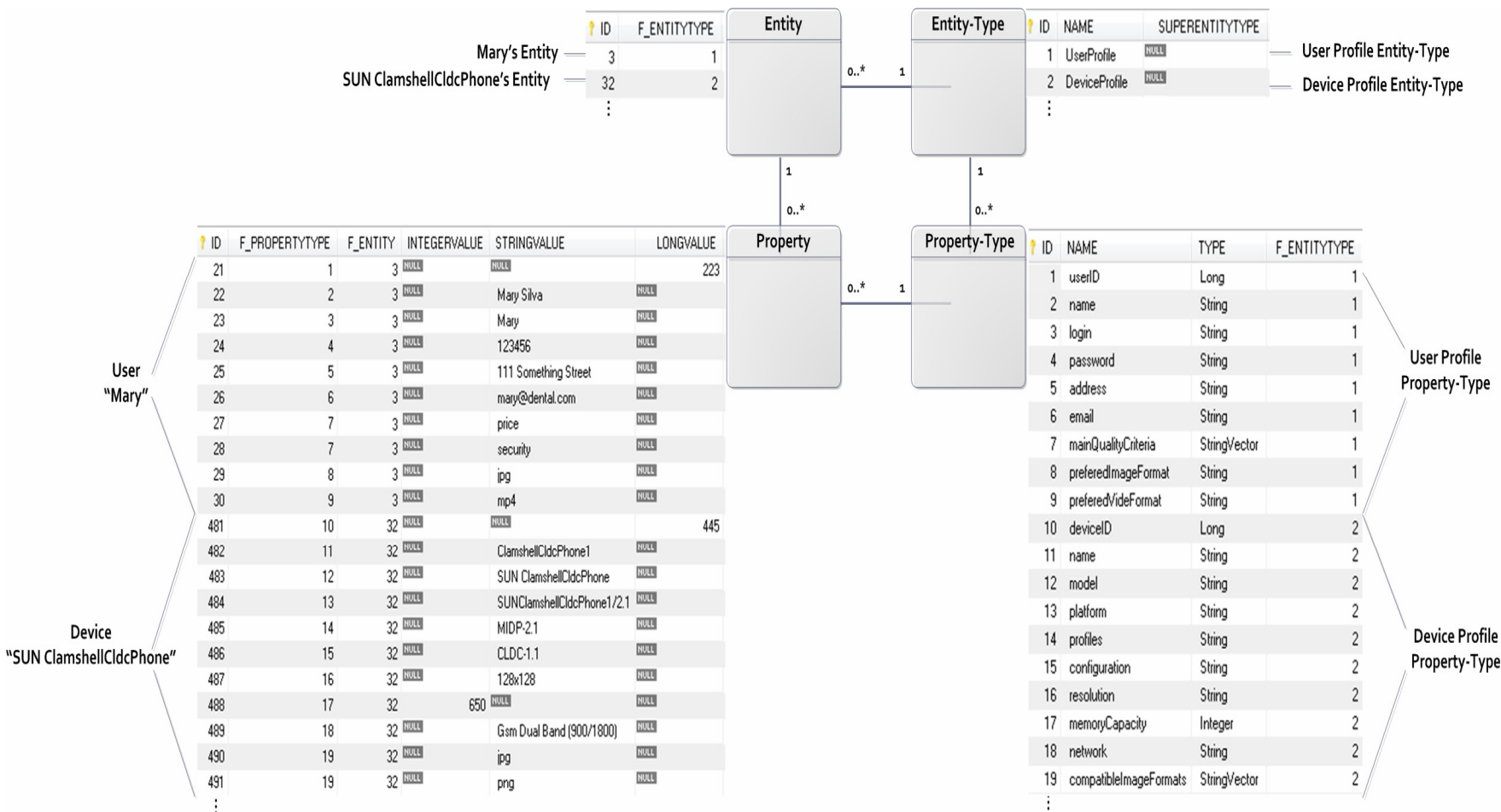


Figure 6.50 - Instances of the *User Profile* and *Device Profile* in the dental clinic ubiquitous case study

It is important to notice that the dentist under analysis – represented by *Mary Silva* – is not previously associated with a specific device and its features are constantly updated based on the *WURFL Repository*. Therefore, the dentist can change the device, perform another request and the intentional MAS of the dental clinic ubiquitous application is able to identify the device at runtime, without disturbing the dentist or even distracting her/him, as idealized by Mark Weiser's vision – the complexity invisibility need centered on Calm Technology. The device's identification – as previously mentioned (Chapter 4 - Section 4.6) – is performed by using the microedition properties. An intentional agent identifies the platform details, which means the proper Java platform type and the device's model. Based on the model, it is possible to retrieve its profile from the dynamic database and determine its limitations in terms of memory capacity, processing capacity, resolution and other features at runtime.

In most of the ubiquitous application development, the application under analysis will probably have some particular problems that require special attention. In these specific fields, it can be difficult to find ready and extraordinary solutions by demanding the developers' extra efforts and dedication time. Therefore, the last activity of the *Implementation* discipline is dedicated to the implementation of specific issues – not illustrated in this thesis for dental clinic case study.

The last discipline of our life-cycle is the Test discipline. Therefore, in order to conclude the presentation of the proposed iterative process, in the next Section we describe some tests performed to evaluate the intermediary versions of the dental clinic ubiquitous application.

6.1.4. Test

The evaluation of the dental clinic ubiquitous application was performed after each iteration of the life-cycle. Figure 6.51 illustrates a dental clinic's scenario used to test the agents' working from the registration requested by a patient to the response with the adapted registration form.

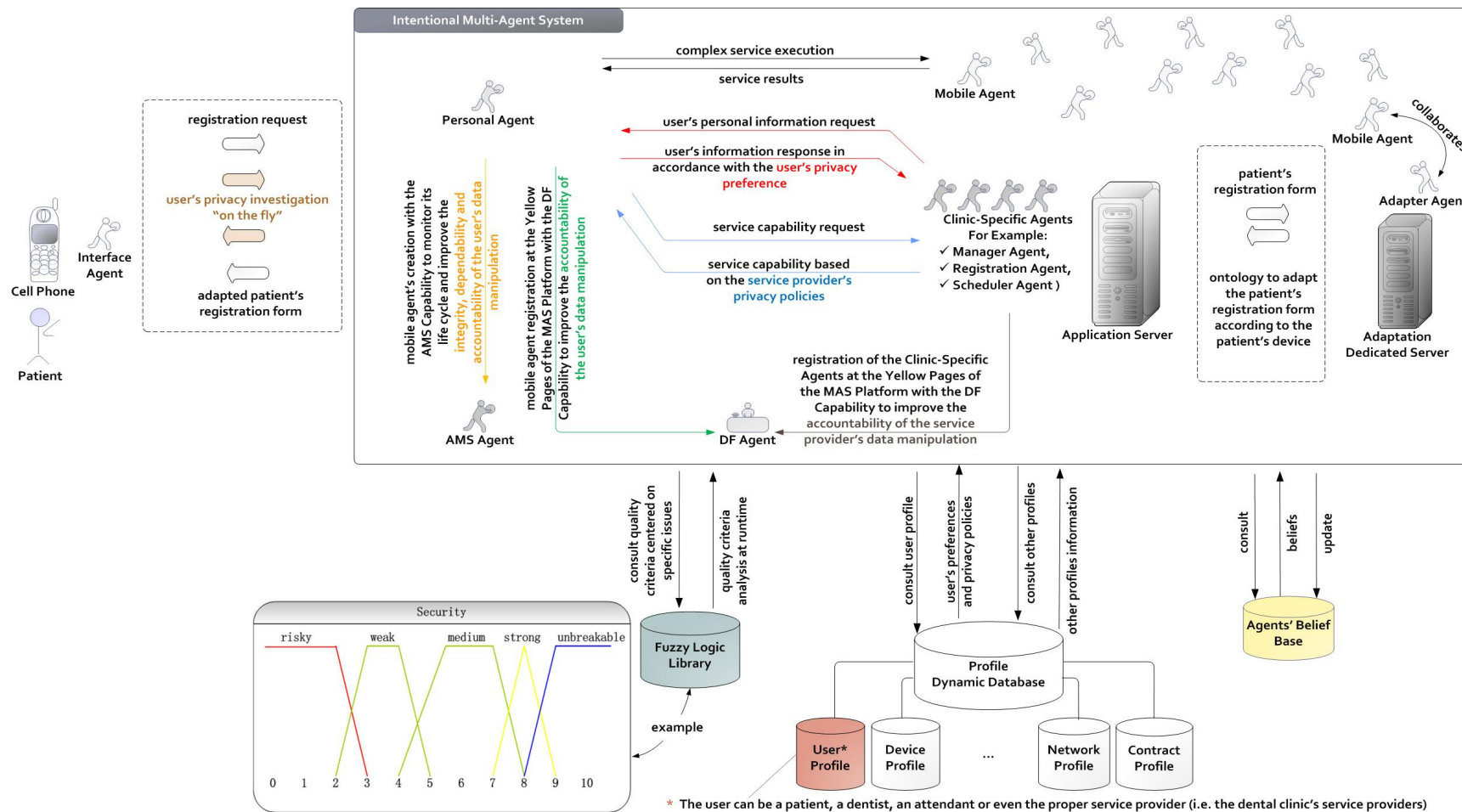


Figure 6.51 - Agents' way-of-working during the patient's registration process

The execution process centered on privacy issues starts with the patient by accessing the network using her/his personal device. The patient requests its registration. Therefore, the device's integration with the MAS platform is performed and supported by the *Integration Building Block*. In the dental clinic ubiquitous application, the integration is performed by a *JADE-LEAP Agent*, called Interface Agent. This agent is based on behavior and runs inside the user's device – i.e. in the patient's device according to the illustrated example. When this agent receives the patient's registration request, it identifies the device's model by applying the microedition properties. The device is integrated by using the split execution mode or the standalone execution mode, depending if the device is a MIDP device or a PJava device. At the moment that the device is integrated with the MAS platform, it is possible to use the platform's services (e.g. agents' creation service performed by the AMS Agent with the AMS Capability and the agent's registration/deregistration service performed by the DF Agent with the DF Capability).

As the Interface Agent does not have sufficient cognitive capacity to achieve the patient's goal, it delegates this goal to an intentional agent – an instance of the *JADEX Agent* – called Personal Agent. This latter agent represents the patient in the dental clinic ubiquitous application. There is one Personal Agent for each user (e.g. each patient, each dentist and each attendant). This agent is specialized in the user's preferences and privacy policies. Therefore, any requisition or manipulation in the MAS platform that involves the user's data must be performed and/or authorized by this intentional personal agent. Moreover, this agent only decides about those accesses by: (i) consulting the *User Profile*, in which it is possible to previously store the patient/dentist/attendant's privacy preferences and personal data as mandatory, private or public; or (ii) dynamically acquiring the necessary information by using the proper patient/dentist/attendant's navigation process or by asking her/him at runtime.

In order to achieve the patient's goal – her/his registration into the dental clinic – the Personal Agent interacts with other platform's agents (e.g. Mobile Agent to perform complex content adaptations in a dedicated server). In this collaborative process, the Personal Agent can acquire capabilities to improve its interactive activities. For example, this agent must acquire the dental clinic's specific capabilities to know how to use some offered services (e.g. the patient's

registration service), how to communicate with the dental clinic's Multi-Agent System and to improve its knowledge centered on the dental clinic's privacy policies. Moreover, the patients may wish to limit and control the amount of information released to the service providers (e.g. the service providers of the dental clinic), and the service providers would also need to protect their commercial strategies (e.g. prices and ethical rules). Furthermore, both parties, patient and dental clinic, desire to ensure the accuracy and the authenticity of the information provided by them. The capability usage represents a mechanism to allow the patients and the dental clinic to control their release as well as authenticating and protecting the integrity/confidentiality of the information provided by them. In this scenario, if the Personal Agent requests and receives from the dental clinic's agents the capability to use a specific service, it means that this agent improves its abilities by now being aware of the dental clinic's privacy policies and capable of acting/reacting by respecting these rules.

Finally, the Personal Agent with the registration service's capability collaboratively interacts with the intentional MAS of the dental clinic's application to register the patient by considering the ubiquitous profiles information – stored by following the *Dynamic Database Building Block*. Figure 6.52 shows some device screens, which were visualized by the patient during her/his registration process. The illustrated MIDP device is a SUN ClamshellCldcPhone model with 128x128 pixels of screen resolution. It is relevant to say that various content adaptations were performed in order to deal with the device's limitations, such as its memory and processing capacity. All necessary adaptations were performed by intentional agents – provided by the API of the IFCAUC from the *Ubiquity Issues Building Blocks* – without disturbing the patient. In addition, each agent in the described process is registered – with a unique identifier – at the platform from the moment of its creation in order to facilitate the interaction among the agents, the identification of an agent and the accountability in case of abusive actions or illegal access. Moreover, the agents of the platform can be dynamically traced by helping the user to know what is going on anywhere and at any time. Furthermore, these “privacy-based resources” also represent interesting mechanisms to deal with the device disconnection because of low battery, lack of signal, and other communication problems. The disconnection is an intrinsic concern in ubiquitous contexts.



Figure 6.52 - Adapted *Registration Form* for a MIDP device

In order to illustrate the difference between an application supported by “privacy-based resources” and an application without them, we also show – Figure 6.53 – some screens that were used to investigate the patient/dentist/attendant’s preferences “on the fly” to better satisfy her/him. For example, the screen that investigates if the patient desires to share her/his dental information with the dental clinic. In this case, since the patient agreed, this information (e.g. her/his history of dental problems previously treated by other dental clinics) was stored by the actual dental clinic for further consultation. Moreover, because the patient desired to know who was the person responsible for dealing with this personal information, the application – based on the AMS and the DF Services – provided the identifier of the responsible agent (e.g. RegistrationAgent@187.24.139.123:1099/JADE) as well as the name of the responsible individual of the dental clinic that this agent represents (in this case, the attendant Diane). According to our evaluation process, this kind of resource provides accountability and improves the patient’s satisfaction. If we did not have

this kind of resource, we would probably decide: (i) to not share the patient's information, by disabling, for example, that the dental clinic adequately deals with the current dental problem of the patient; and (ii) to share this information without the patient's permission, which could lead to the patient's dissatisfaction. Moreover, without knowing who is responsible for handling the patient's information, she/he could not react in case of improper usage of her/his personal information.



Figure 6.53 - Privacy-centered investigation from a MIDP device

Finally, Figure 6.54 shows an Anamneses Form adapted for the same SUN ClamshellCldcPhone device as well as Figure 6.55 illustrates an x-ray adapted for two different devices centered on their ubiquitous profiles and the fuzzy-logic conditional rules' specifications for the dental clinic ubiquitous application by choosing the content provider that offers the desired x-ray with secure download.

The result of the systematic and incremental development process – after various iterations – is the *Intentional-MAS-Driven Dental Clinic Ubiquitous*

Application, whose evaluation detailed process – discussed in the next Chapter – contributed to report on the application’s strengths and weaknesses. Furthermore, the evaluation FEEDBACK was important to improve the application as well as the proposed incremental and systematic development.



Figure 6.54 - Adapted Anamneses Dental Form for a MIDP device

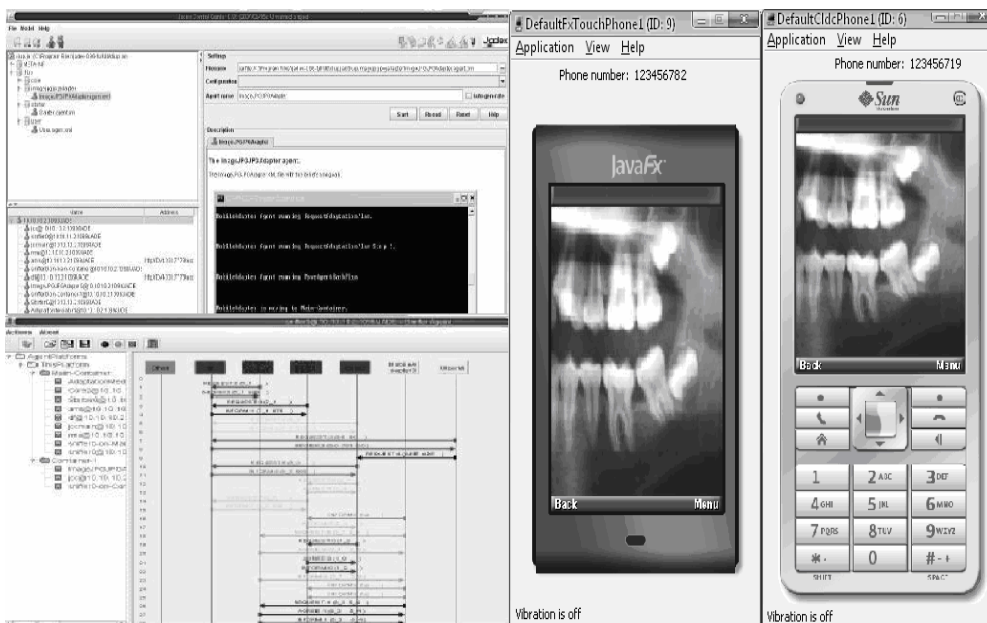


Figure 6.55 - Adapted x-rays for heterogeneous devices and the JADEX Control Center support

6.2. Closing Remarks

In this Chapter we zoom in the *Ubiquitous Application Engineering* by considering a dental clinic ubiquitous application. We applied the proposed incremental and systematic development to the dental clinic case study based on the reusable building blocks. From the *Early & Late Requirements* disciplines to the *Architectural & Detailed Design* disciplines we demonstrate the model-driven development by using, for example, the *Intentional Modeling Building Block* and the *NFR Catalogue Building Block*. Moreover, we also present how to incorporate pre-defined modeling (e.g. the IFCAUC modeling for intentional-MAS-driven content adaptation) to the modeling of the ubiquitous application under analysis. The resulting model already contemplates the mechanisms to deal with content adaptation in ever-changing contexts by improving it with intentional MAS. Based on the models – obtained by performing various iterations to refine the early models – we show how to implement (*Implementation* discipline) the modeled dental clinic ubiquitous application by configuring the development environment and reusing different building blocks in lower abstraction levels, such as: (i) the *Autonomous Entity Support* from the *Intentional Agents' Reasoning Building Block* to implement intentional agents based on the BDI model; (ii) the *Ubiquitous Application-Based Capability* from the same *Intentional Agents' Reasoning Building Block* to deal with, for example, the privacy issues; (iii) the Fuzzy-Logic-Based Support to deal with non-functional requirements at runtime in order to improve the user satisfaction centered on specific quality criteria; (iv) the *Dynamic Interface Construction Building Block* to adapt interface elements that must be visualized from heterogeneous devices. In this field, we use an ontological support to improve and standardize the agents' communication and inter-operability; (v) the IFCAUC from the *Ubiquity Issues Building Block* to perform the content adaptation centered on adaptation services, adaptation techniques and ontologies; and (vi) the *Dynamic Database Building Block* to support the persistence layer in ever-changing environments by using a dynamic data model to consult/retrieve/update/delete the ubiquitous profiles' information at runtime. Finally, we present some tests performed to evaluate the dental application's intermediary versions, developed during the iterative process.