
# PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 07/10

# Dynamic Database for Intentional Development of Ubiquitous Systems

**Milene Serrano**
**Carlos José Pereira de Lucena**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**
**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900**
**RIO DE JANEIRO - BRASIL**

Monografias em Ciência da Computação, No. 07/10 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena May, 2010

# Dynamic Database for Intentional Development of Ubiquitous Systems*

Milene Serrano[1] and Carlos José Pereira de Lucena[1]

[1]Departamento de Informática – Potifícia Universidade Católica do Rio de Janeiro

milene@les.inf.puc-rio.br and lucena@inf.puc-rio.br

**Abstract.** In this paper, we present our Dynamic Database Building Block centered on our Intentional Systematic Software Development for Ubiquitous Systems (ISSD for UbSystems). This specific building block is composed of a Dynamic Database Architecture and an agent-oriented Layer Structure. Our main goal is to store, search, recovery, and protect the stakeholders' information considering different ubiquitous profiles, and special privacy policies through reusable standard solutions. Moreover, we use this building block to dynamically allow the insertion of new devices' features, network specification, users' preferences, and contract information. This dynamic mechanism is particularly important in ever-changing environments to improve the traditional database models, in which you must previously specify the entities and fields. Furthermore, we evaluated our support in an extensive dental case study, and also compared it with other related work.

**Keywords**: Agent-oriented Dynamic Database Architecture, Intentional Systematic Software Development, Ubiquitous Computing, Multi-Agent Systems, Ubiquitous Issues, Reusable Support.

**Resumo**. Esse artigo apresenta um bloco de construção de banco de dados dinâmico centrado no desenvolvimento sistemático intentional para sistemas ubíquos. Esse bloco de construção é composto de uma arquitetura específica e uma estrutura em camadas orientada a agentes. Nosso principal objetivo é armazenar, buscar, recuperar e proteger as informações dos usuários considerando diferentes perfis ubíquos e políticas de privacidade através de soluções baseadas na reutilização. Além disso, nós usamos esse bloco de construção para dinamicamente permitir a inserção de novas características dos dispositivos, especificações de rede, preferências dos usuários, e informações de contrato. Esse mecanismo dinâmico é particularmente importante em ambientes marcados por constantes mudanças para melhorar os modelos de banco de dados tradicionais, nos quais precisamos previamente especificar as entidades e os campos. Adicionalmente, nós avaliamos o suporte proposto em um estudo de caso no domínio odontológico, e também comparamos o mesmo com outros trabalhos relacionados.

**Palavras-chave**: Arquitetura Dinâmica Orientada a Agents, Desenvolvimento de Software Systemático e Intentional, Computação Ubíqua, Sistemas Multi-Agents, Supporte baseado na Reutilização.

---

* This work has been sponsored by CAPES and CNPq, Brazil.

**In charge of publications**:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

# Table of Contents

# 1 Introduction

The Mark Weiser's vision (Weiser 1991) is technically becoming viable nowadays, by augmenting the number of sensors, devices, actuators, and adequate technological support. We can use, for example, the combination of Multi-Agent Systems (Shoham and Leyton-Brown 2008), Goal-Orientation (Mylopoulos 2008), Distributed Intentionality modeling (Yu 1997), and BDI model (Bratman 1999) to respectively support: **(i)** automation, controlling, and personalization using reasoning and learning techniques, reducing the human intervention need; **(ii)** the Goal-Oriented Requirements Engineering focused on the stakeholders' goals, perceived in ubiquitous contexts; **(iii)** the stakeholders' intentionality modeling based on their goals, softgoals, beliefs, resources, and tasks; and **(iv)** the stakeholders' interests implementation centered on their beliefs, desires, and intentions.

In group, these emergent technologies can contribute to achieve some Ubiquitous Computing principles, outlined by Mark Weiser (Weiser 1991): **(i)** *Omnipresence* – the computer purpose is focused on helping the users in their daily activities by offering services/contents anywhere and anytime; **(ii)** *Complexity Invisibility* – the computers must offer support - as personal servants - to the users without disturbing them or even distracting them; **(iii)** *Intentionality* – the computers should extend the users' unconscious based on her/his intuitions, personal needs and privacy policies; **(iv)** *Calm Technology* (Weiser and Brown 1995) – the computer must create calm to reduce the "frenzy" of information and to allow the user selecting what kind of information is at the center of her/his attention and what information is peripheral; and **(v)** *User's Satisfaction* and *Context-Awareness* – the ubiquitous systems must be designed/implemented to be aware with the user's preferences and the intelligent spaces policies. Moreover, the wireless communication combined with the large bandwidth and different mobile devices improve the information dissemination. Although it represents a desired evolution and the possibility to achieve the content/services omnipresence, it also requires special privacy-issue-based mechanisms to deal with organization and users' data; and a specific and suitable support to deal with the technological intrinsic evolution of ever-changing contexts. In this sense, the Ubiquitous Computing poses some modern challenges, such as:

- *Devices are heterogeneous in terms of the technologies they use and their physical features.*
    - **Ubiquitous Scenario Example:** the devices can be mobile, small, and just-call-phone, or they can be a powerful smartphone. Ubiquitous systems must deal with this heterogeneity – e.g. avoiding the limited device overload using content adaptability support (Serrano et al. 2008).
- *Devices are in constant evolution.*
    - **Ubiquitous Scenario Example:** yesterday the devices have CD reader/writer; today they have DVD reader/writer; and tomorrow all of them will have blue-ray reader/writer. Ubiquitous systems must store these new devices' features, by preferentially reducing the cost, spent time, and efforts to perform changes on the database.
- *Services and contents depend on the organization's policies that offer them.*
    - **Ubiquitous Scenario Example:** some services/contents are paid, some are free, and some are shareware. These decisions (e.g. be paid, be free, or be

shareware) can frequently change (e.g. every year, month, day, minute, or second.) Ubiquitous systems must be flexible to deal with these new commercial needs, by providing mechanisms to allow these organizations deciding – at runtime – what kind of information is at the center of their attention and/or is peripheral.

- *Devices, services, and contents constantly enter and leave different environments.*
    - **Ubiquitous Scenario Example:** devices perform their actions independently/autonomously by storing, sharing, searching, and recovering information. Ubiquitous systems must provide resources to facilitate the communication between the environments and their embedded devices using, for example, specific and pre-defined protocols/ontology, and autonomous entities.
- *Different organizations are involved in these communication processes.*
    - **Ubiquitous Scenario Example:** organizations have different ways to store, access, and share information. Some of them allow sharing data; but others prefer to ride their data. Ubiquitous systems must deal with different privacy policies, needs, and interests in the persistence layer.
- *End-users have different preferences and needs.*
    - **Ubiquitous Scenario Example:** sometimes the users want to offer personal information – e.g. to be registered in an online store –, and sometimes they do not want to share this information in order to protect themselves. Ubiquitous systems must deal with different users' interests according to their preferences and needs in the domain and persistence layers.
- *Environments[1] (e.g. home, and work space) are distributed and changeable.*
    - **Ubiquitous Scenario Example:** someone decides to use the printer in her/his office using her/his cellphone; few minutes later, she/he accesses her/his e-mails using her/his home desktop, and, she/he uses a smartphone to access the internet to pay an electric bill when she/he is waiting her/his little son at school. Ubiquitous systems must deal with – using dynamic ubiquitous profiles – different contents, services, environments, and devices, which are distributed and in constant evolution.

We are only mentioning some challenges of ubiquitous contexts. However, we have other important ones (Weiser 1993), organized in: hardware components issues, network issues, and privacy issues. Our focus is on offering a reusable technological support to particularly attend to: **(i)** the stakeholders' privacy issue; and **(ii)** the intrinsic and constant evolution of the information, services, contents, devices, and technologies in ubiquitous contexts. In our approach we are suggesting the use of a Dynamic Database to aim the data storage and management, and to avoid conflicts among the stakeholders' requirements – their preferences, needs, security, and privacy. Our Dynamic Database combined with our intentional agent-oriented Layer Structure offer a suitable building block to dynamically **i**nclude (e.g. new entity/class, and new

1 We call these environments as smart-spaces in our approach.
A smart-space is basically composed of a physical space, different devices (mobile or fixed), sensors, and actuators.
We believe that the Multi-Agent Systems paradigm can improve a lot the communication between two or more different smart-spaces. We investigated the pertinence of this paradigm using various experimental case studies.

field/attribute); **u**pdate; and **e**xclude information (e.g. old record, and unusual data). Our structure is divided in layers, which are organized to store/access/share/recovery/search personal information based on the *Software Engineering domain*, *Ubiquitous Computing transversal domain*, *different cognitive domains* (e.g. e-health domain, and ecommerce domain), *different cognitive sub-domains* (e.g. dental domain; and medical domain), and the *ubiquitous application* (e.g. ABCD Dental Clinic; JJ Cancer Hospital, and WYZ Media Store). We also propose ways to respect the end-user profile, and consequently her/his preferences, and personal data through the application of cognitive agents and specific protocols/ontology/capabilities.

The rest of this paper is organized in sections: in Section 2, a briefly description of our *Dynamic Database Architecture*; in Section 3, the *Dynamic Database Architecture* applied to a dental scenario; in Section 4, the *Dynamic Database Architecture* applied to different ubiquitous profiles; in Section 5, our dental case study description; in Section 6, our Dynamic Database Building Block applied to our dental case study; in Section 7, some related work; and in Section 8, the final considerations, including a comparative evaluation and further work.

## 2 Dynamic Database Architecture Overview

As presented in the Introduction, it is particularly important to have a dynamic mechanism to store and recover contents in ever-changing contexts. In these contexts, it is difficult (or even impossible) to previously define all the attributes that will be necessary to describe/catalog a specific device or a specific service. Both, devices and services, are frequently evolving, following the technological innovation. Moreover, it is difficult to deal with the privacy policies centered on knowledge that the ubiquitous systems have, desire to protect, and avoid sharing with the concurrent systems.

Centered on these concerns (quick technological innovation and different privacy policies), we will first analyze how we normally deal with them in a fixed and traditional database. As the traditional database structure – its entities' model – must be previously defined, it is difficult to incorporate new and specific entities, properties, and their types without modifying and reorganizing its structure in the design level. These updates request/spend time, efforts, and money. Commonly, the software engineers must suspend the offered service in order to adjust its database model. The resources consumed in this process are not very important in unchangeable contexts. However, we are talking about ubiquitous and pervasive contexts, in which these constant updates are really difficult to be performed as they must be performed daily and monthly. Imaging how difficult and expensive would be if for every new device, or technological feature of a specific device, or even adjusts in an online-offered service, a set of updates was necessary in the system database model.

In order to provide an adequate support in this sense, we are proposing a flexible database, designed to be reused. Its model is combined with a flexible layer structure – supported by smart agents – that allows data sharing considering different cognitive domains, and protecting the system data/knowledge/business rules when it was requested. Firstly, we will discuss about our Ubiquitous Dynamic Database proposal, by presenting its general architecture, and its entities' model. Our database model is centered on the *TypeObject Pattern* (**Figure 1***)* and the *TypeSquare Architecture*. Both the pattern and the architecture were proposed by Yoder (Yoder 2001).

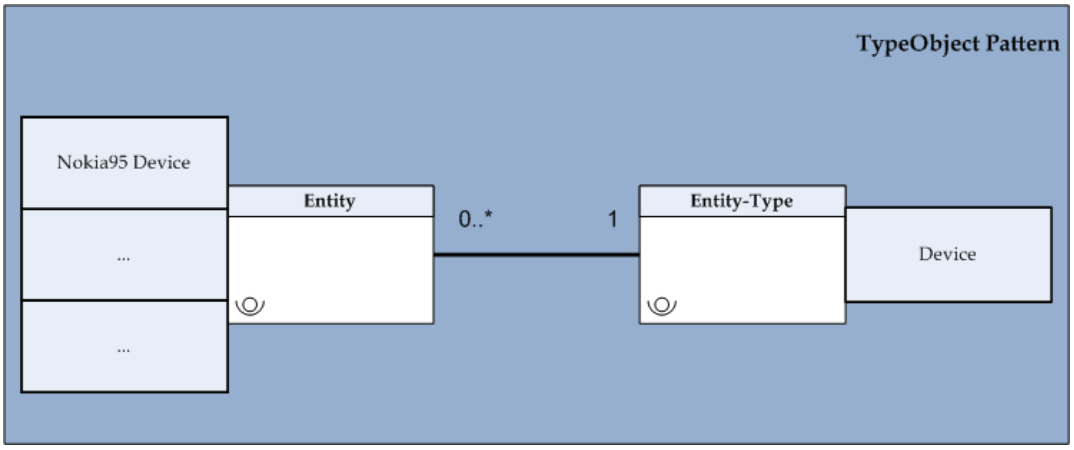


**Figure 1: TypeObject Pattern – Adapted from [Yoder 2001]**

The *Entity-Type* represents the classes and the *Entity* represents the classes' instances. As a simple example, we have the *Entity-Type* "Device" and the *Entity* "Nokia95 Device". "Nokia95 Device" is an instance of "Device." Moreover, the *Entity* is associated with a specific *Entity-Type*, and the *Entity-Type* can be associated with zero or various *Entities*. Thus, the cardinality in the first way is one to one (**1..1**), and in the opposite way is one to zero or more (**1 to 0..***). We can have different devices – e.g. "BlackBerryBold9700 Device," "NokiaN86 Device," "MotorolaWX390 Device," and "SonyEricssonXperia-X10 Device." Each of them (*Entity*) is an instance of "Device" (*Entity-Type*). Yoder proposed to apply the *TypeObject Pattern* to the entity level, and to the property level as presented in **Figure 2**. For this structure, Yoder called *TypeSquare Architecture*.

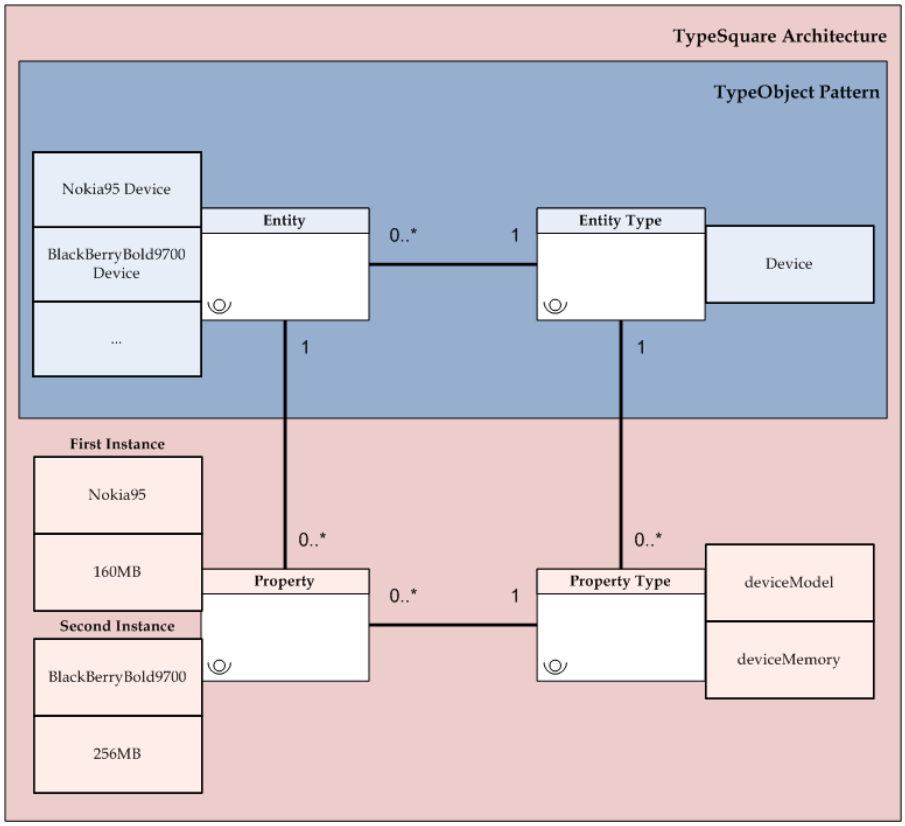


**Figure 2: TypeSquare Architecture – Adapted from [Yoder 2001]**

The "Device" *Entity-Type* can have different *Properties-Types*, such as: "deviceModel," "deviceMemory," "deviceScreenSize," "deviceBattery," "deviceMemoryCard," and "deviceOperatingSystem." Centered on this idea, the architecture stores the properties' values of a specific *Entity* (e.g. "Nokia95 Device") as *Properties*. Thus, for the "Nokia95 Device" *Entity*, the "deviceModel" is "Nokia95," and the "deviceMemory" is "160MB". The architecture also specifies other important associations between:

- *Property* and *Property-Type*: one *Property* must be associated with only one *Property-Type* – e.g. the "Nokia95" *Property* is only associated with the "deviceModel" *Property-Type.*

- *Property-Type* and *Property*: one *Property-Type* can be associated with zero or more *Properties* – e.g. the "deviceMemoryCard" *Property-Type* is associated with the "Nokia95," and "BlackBerry Bold 9700" *Properties.*

We designed and implemented our Dynamic Database by extending the *Type-Square Architecture* as presented in Figure **3**. The *Entity-Property* represents a new class/table in our Dynamic Database model that is created based on the association between *Entity* and *Property*, which cardinality is ***..***. It means that an *Entity* can be associated with zero or more (**0..***) *Property*; and a *Property* can be associated with zero or more (**0..***) *Entity*.

One of our main purposes is to deal with contexts such as:

**First Context:** *A specific device (Entity) is an instance of Device (Entity-Type). Thus, this specific device (e.g. Nokia95 Device) contains the Device's properties. Moreover, this same specific device (Entity) is an instance of Device with Camera (other Entity-Type). Thus, this specific device (e.g. Nokia95 Device) also contains the Device with Camera's properties. However, the Device with Camera is a Device! Thus, a Device with Camera inherits the Device's properties.*

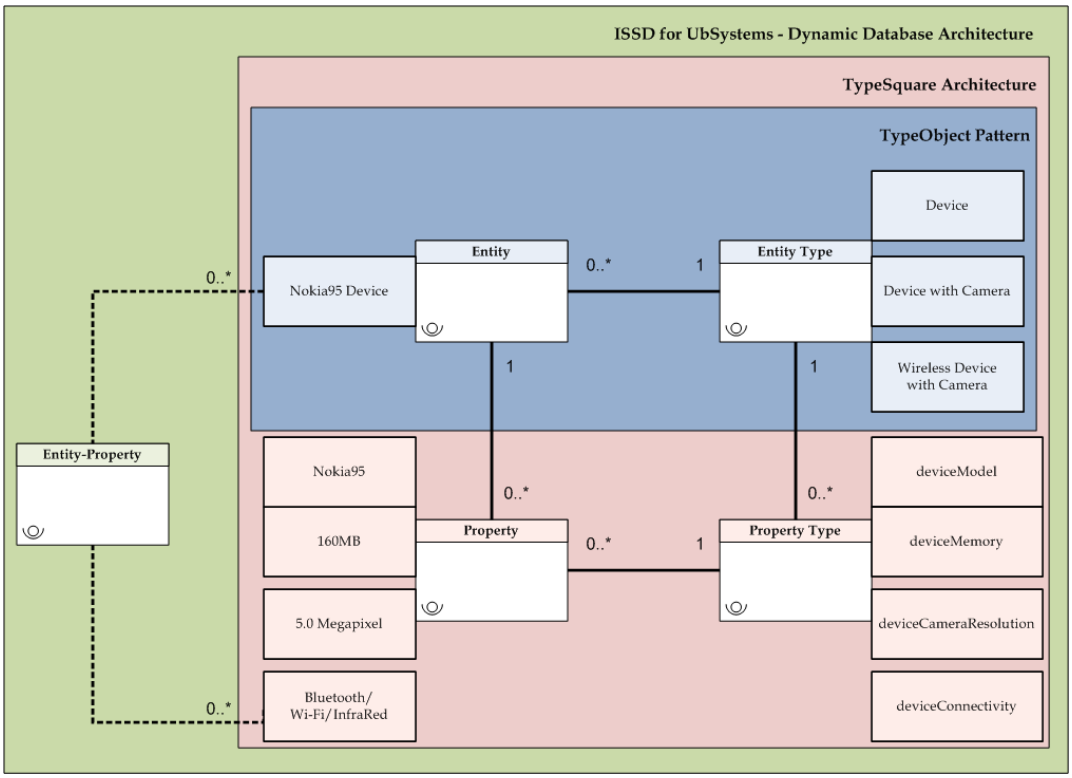


**Figure 3: Dynamic Database Architecture (First Example)**

We have the inheritance concept in the context presented before, and the *Type-Square Architecture* does not directly deal with this kind of context. Our Dynamic Database allows dynamically defining one or more levels of inheritance: a Specific Device (an *Entity*) is an instance of Device with Camera (an *Entity-Type*), which is a Device (an *Entity-Type* _ First-Level-Of-Inheritance.) Moreover, a Specific Device (an *Entity*) is an instance of Wireless Device with Camera (an *Entity-Type*), which is a Device with Camera (an *Entity-Type* _ Second-Level-Of-Inheritance.) Only to illustrate, consider the example previously presented in **Figure 3**, and the explanation as follows:

• **Instantiation:** As the "Nokia95 Device" is an instance of Wireless Device with Camera, it contains the "deviceConnectivity" *Property*, which value is "Bluetooth/Wi-Fi/InfraRed."

• **First-Level-Of-Inheritance:** As a Wireless Device with Camera is a Device with Camera, it also inherits the Device with Camera *Property-Types* (e.g. "deviceCameraResolution".) We represent this relationship as a special *Property-Type* called "SUPER," which type is Device with Camera.

Thus, the "Nokia95 Device," as an instance of Wireless Device with Camera, will contain two *Properties-Types* ("super" and "deviceConnectivity",) which values are respectively an object of Device with Camera (in which the "deviceCameraResolution" *Property-Type* is associated with the value "5.0 Megapixel"); and "Bluetooth/Wi-Fi/InfraRed."

• **Second-Level-Of-Inheritance:** As a Device with Camera is a Device, it also inherits the Device *Property-Types* (e.g. "deviceModel" and "deviceMemory".) We represent this relationship as a special *Property-Type* called "SUPER," which type is Device. Thus, the object "SUPER" of the "Nokia95 Device" will contain two *Properties-Types* ("super" and "deviceCameraResolution",) which values are respectively an object of Device (in which the "deviceModel" and "deviceMemory" *Properties-Types* are associated with the values "Nokia95" and "160MB"); and "5.0 Megapixel."

**Second Context:** *A specific device (Entity) is an instance of Device (Entity-Type). Thus, this specific device (e.g. Nokia95 Device) contains the Device's properties (deviceModel, deviceMemory, and deviceBattery). In this context, a Device (Entity-Type) has battery as Property-Type. BUT Battery is an Entity-Type, which has batteryType and batteryCapacity as Properties-Types. It means that Device (Entity-Type) is associated with Battery (another Entity-Type).*

We have a classical association in the context presented before, and again the *Type-Square Architecture* does not directly deal with this kind of context. Our Dynamic Database proposes a various-to-various association between the Device *Entity-Type* and the Battery *Entity-Type*, represented by the cardinality **0..*** to **0..*** and the new class/table *Entity-Property* ( see **Figure 4**.)

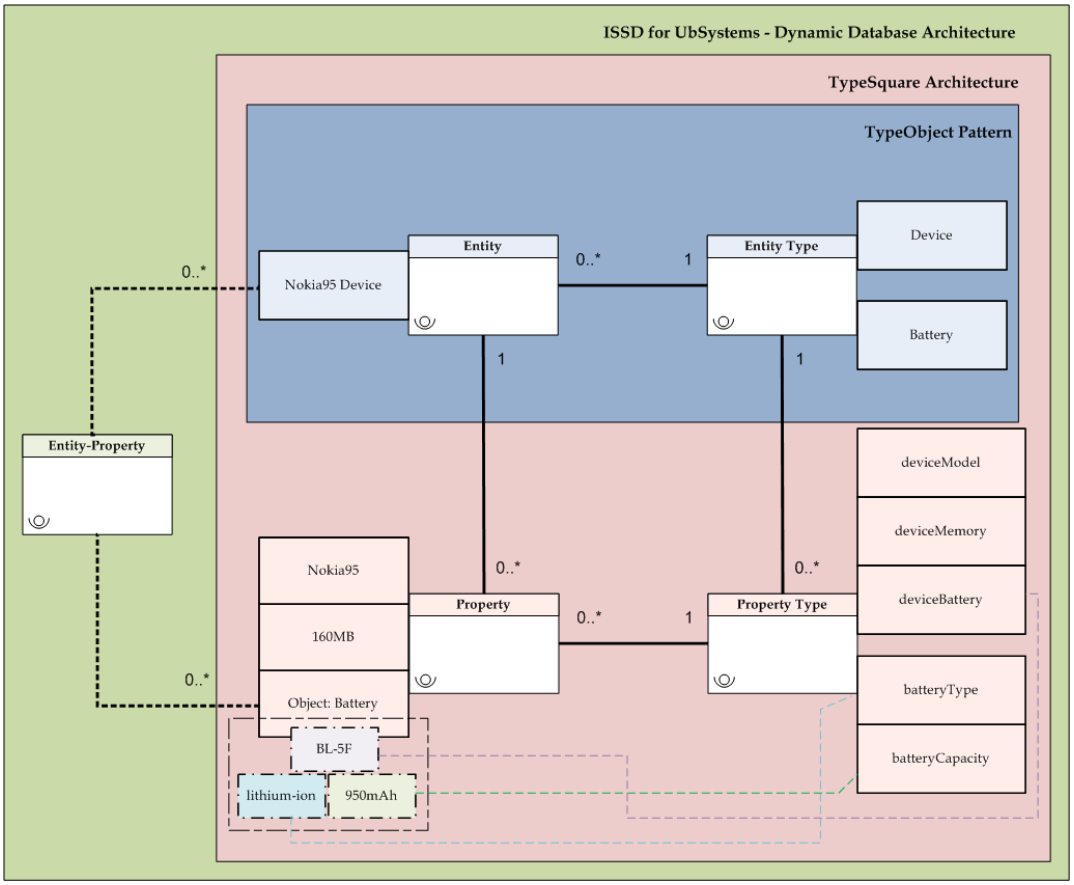


**Figure 4: Dynamic Database Architecture (Second Example)**

For example:

• **Instantiation:** As the "Nokia95 Device" is an instance of Device, it contains the "deviceModel," "deviceMemory," and "deviceBattery" as its *Properties*, which values are respectively "Nokia95," "160MB," and an object "BL-5F."

• **Association:** the object "BL-5F" is a Battery. It is represented as an association between Battery and Device. Thus, the "Nokia95 Device" also contains "batteryType" and "batteryCapacity" as its *Properties*, which values are respectively "lithium-ion," and "950mAh".

In order to illustrate/emphasize the applicability of our Ubiquitous Dynamic Database architectural model, we consider a ubiquitous scenario based on an extensive dental case study developed in our Software Engineering Laboratories at PUC-Rio and UofT.

## 3 Scenario based on an Extensive Dental Case Study

We describe the dental domain as an e-health domain with specific policies, contexts, and needs. In this kind of context, it is necessary to deal with different dental forms at runtime. These forms have different structures, number of questions, and other information. Moreover, they must be dynamically adapted according to the devices' profiles (e.g. deviceScreenSize, and deviceMemory.) Thus, it is necessary a flexible database to deal with different forms' properties, allowing insertions, updates, and exclusions at runtime supported by autonomous entities. In order to illustrate the *Type-*

*Square Architecture* use, we defined a dynamic database to store the dental forms types (see **Figure 5**): clinic's rules and procedures dental form; clinic's dental registration form; clinic's dental registration payment information form; and clinic's dental registration payment confirmation form.

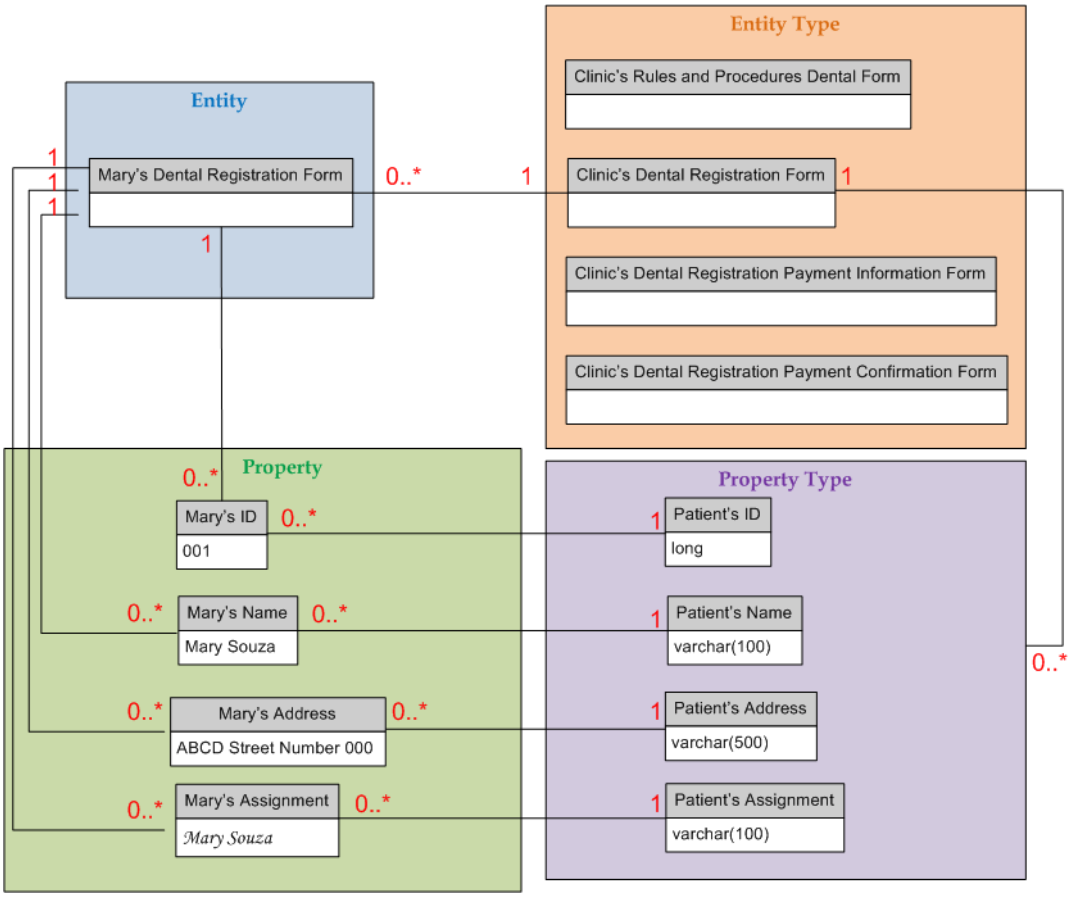


**Figure 5: TypeSquare Architecture applied in a dental domain context**

The "Mary's Dental Registration Form" (an *Entity*) is an instance of "Clinic's Dental Registration Form" (an *Entity-Type*), which contains different *Properties-Types*: "Patient's ID," "Patient's Name," "Patient's Address," and "Patient's Assignment." Each of them assumes the respective values (as *Properties*): "001," "Mary Souza," "ABCD Street Number 000," "*Mary Souza*." Thus, we used the *Entity*, *Entity-Type*, *Property-Type*, and *Property* to store the Mary's dental registration data as specified on the *Type-Square Architecture.*

Now, we can consider our *Dynamic Database* use in this same context – Dental Domain. Suppose that the "Clinic's Orthodontic Registration Form" is a specialization of the "Clinic' Dental Registration Form" considering the orthodontic dental branch. Thus, the "Mary's Orthodontic Registration Form" is an instance of "Clinic's Orthodontic Registration Form," which is a "Clinic's Dental Registration Form." **Figure 6** shows how we represented this situation using our *Dynamic Database Architecture.*

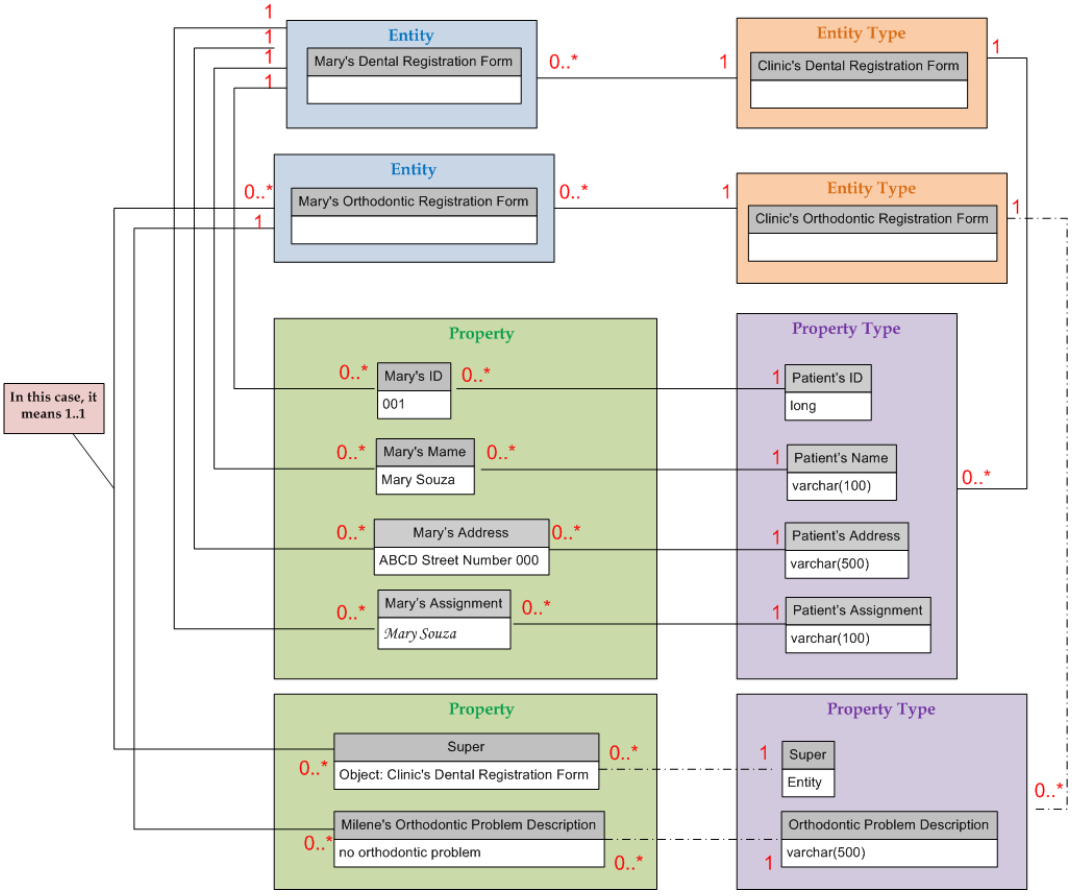


**Figure 6: Dynamic Database Architecture applied in a dental domain context**

We represented this relationship as *Various-Various Association* between the "Mary's Orthodontic Registration Form" *Entity* and the "Super" *Property*. "Super" is a *Property*, which type is "Clinic' Dental Registration Form." Thus, the "Mary's Orthodontic Registration Form" *Entity* has two *Properties*: "Super" and "Mary's Orthodontic Problem Description." Each of them respectively assumes the values: an object (type: "Clinic' Dental Registration Form",) and "no orthodontic problem". The object "Clinic' Dental Registration Form" has the *Properties-Types*: "Patient's ID," "Patient's Name," "Patient's Address," and "Patient's Assignment." Each of them assumes the respective values (as *Properties*): "001," "Mary Souza," "ABCD Street Number 000," "*Mary Souza*." In this case, we used the *Entity*, *Entity-Type*, *Property-Type*, *Property*, and the *Various-Various Association* to store the Mary's orthodontic registration data as specified on our *Dynamic Database Architecture.* All the activities involved into the described process are dynamically performed by cognitive agents using our reusable support proposed in this paper.

We had different ways to deal with inherence and to extend the *TypeSquare Architecture*. We preferred to use a *Various-Various Association* as it represents the most generic relationship, and we can avoid drastically changing the *TypeSquare Architecture*, which contemplates other interesting resources that attend to ubiquitous systems development. The *Various-Various Association* can represent an inherence (normally, **1..1**), and all kinds of associations (e.g. **0..1**, **1..1**, **0..***, **1..***, **2..***, and ***..***.) In the inherence relationship, we also incorporated a *Property* "Super" to guarantee the access to the *Properties* (e.g. "Patient's ID," "Patient's Name," "Patient's Address," and "Patient's Assign-

ment") specified on the super class (e.g. "Clinic' Dental Registration Form".) This special *Property* "Super" is not necessary for all kinds of associations, only for inherence.

## 4 Dynamic Ubiquitous Profiles

In **Figure 7** we present an overview about the dynamic database proposed in our Intentional Systematic Software Development for Ubiquitous Systems (ISSD for UbSystems) based on the main ubiquitous profiles, such as: *User Profile* – to store the user's data; *Device Profile* – to store the device's features; *Network Profile* – to store the network's specifications; *Contract Profile* – to store the user-service business rules; *Content Profile* – to store the content's information; *Service Profile* – to store the service's information; and *Smart-Space Profile* – to store the environment's issues.

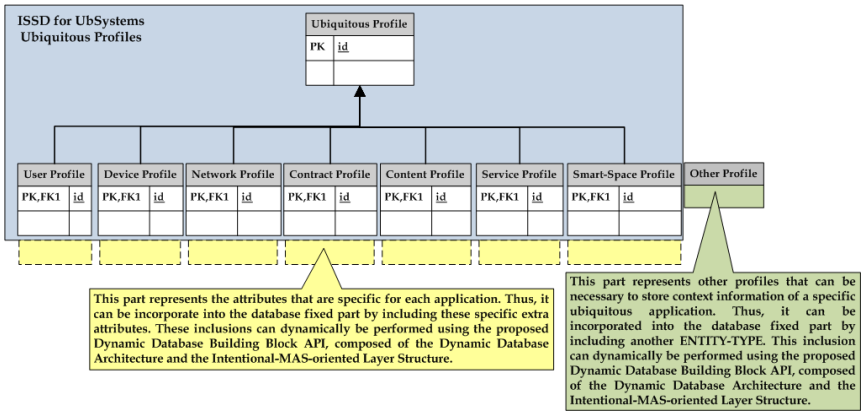


**Figure 7: Dynamic Ubiquitous Profiles**

We used our *Dynamic Database Architecture* to define the *Entities-Types*, *Properties-Types*, *Entities*, and *Properties* for the ubiquitous profiles, as presented in **Figure 8**. Only to exemplify, consider that the User Profile *Entity-Type* is composed of the *Properties-Types*: "ID," "userName," "userAddress," "userCellphone," "userE-mail," and "user-Preferences."

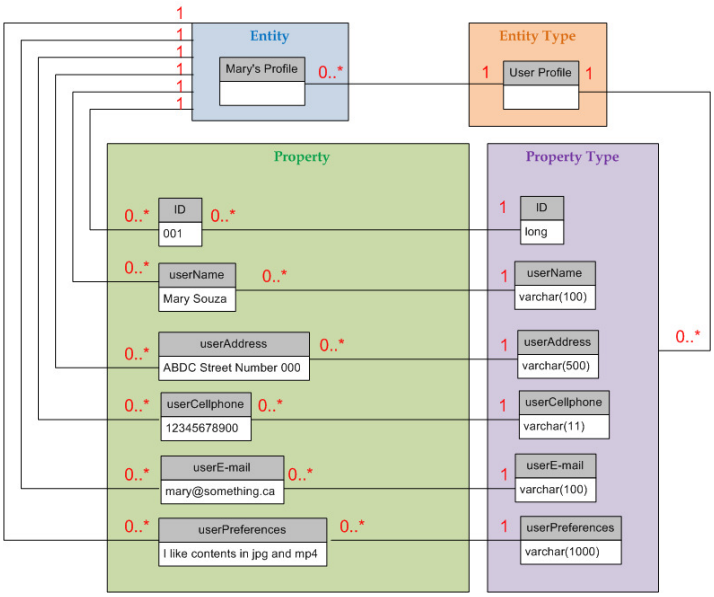


**Figure 8: User profile and the Dynamic Database Architecture**

Suppose that we have the Mary's Profile *Entity* as an instance of the User Profile *Entity-Type*. Thus, the *Properties-Types* ("ID," "userName," "userAddress," "userCell-phone," "userE-mail," and "userPreferences") respectively assume the values "001," "Mary Serrano," "ABCD Street Number 000," "12345678900," mary@something.ca, "I like contents in jpg and mp4," as *Properties*.

Moreover, it is possible to dynamically create new *Entities-Types*, *Properties-Types*, *Entities*, and *Properties* using the agent-driven *Dynamic Database Architecture* API. We developed an API to be reused by ubiquitous systems in different cognitive domains. This API is centered on our *Dynamic Database Architecture*, and potential interested developers can reuse this API to develop specific dynamic databases that better attend to their ubiquitous system's needs. The developers of a ubiquitous system in e-commerce domain (**Figure 9**) can define, for example, a database extending our Device Profile *Entity-Type* to address their specific devices' *Properties-Types*: "ID," "deviceModel," "isSmartphone," and "hasUSB." Thus, if the device has a new feature (e.g. webcam), it is possible to dynamically create a new "webcamResolution" *Property-Type* (associated with the Device Profile *Entity*) in the database model. The use of our API as a *Building Block* based on the reuse principle is simple and intuitive. It can be added to the application project as a ***jar*** file. Our *Dynamic Database Architecture* is really interesting in contexts that are in constant evolution – e.g. ubiquitous and pervasive contexts – in which the devices are aggregating more and more features, following the technological innovation; the organizations' policies are frequently changing; and the users want to modify and to personalize their preferences for each daily activity anywhere and anytime.

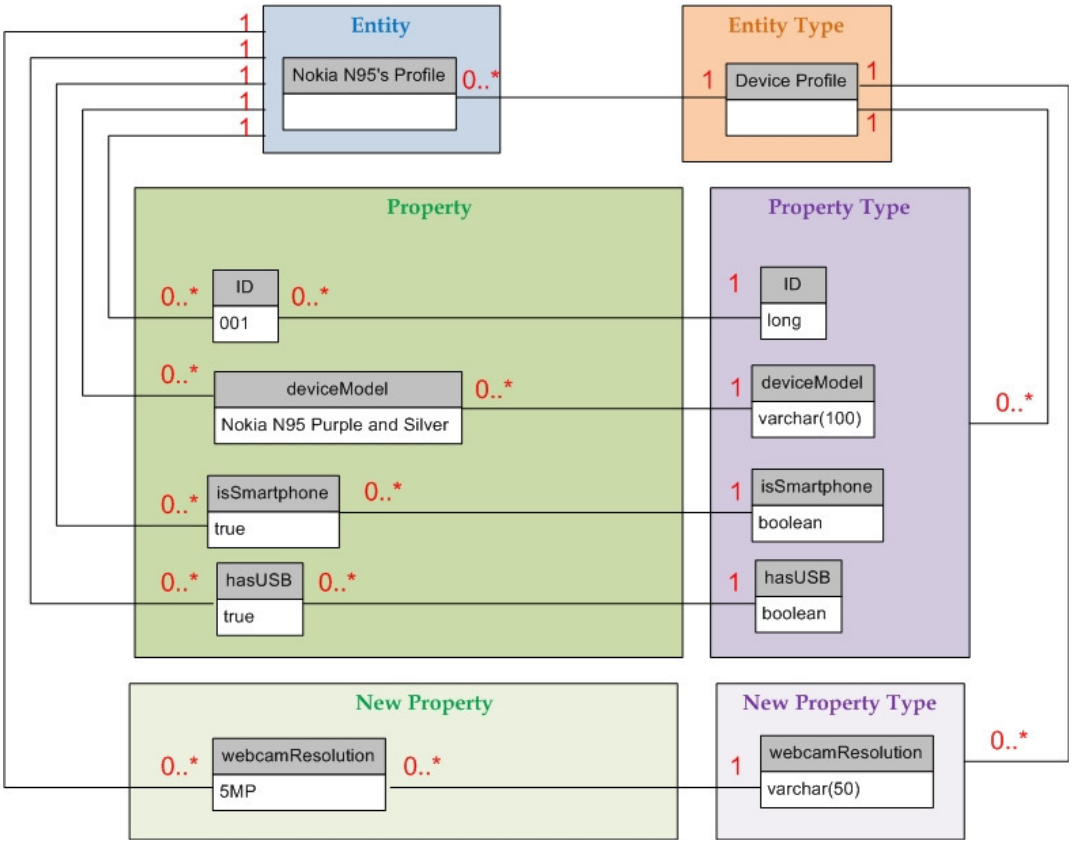


**Figure 9: Dynamic Database Architecture API in an e-commerce context**

We also propose the use of Intentional MAS centered on the BDI Model to facilitate the data creation, update, and exclusion at runtime, when it was necessary, and according to the context analysis. We tested our Dynamic Database Architecture API in the development of a complex and extensive ubiquitous system based on a dental clinic, briefly presented in Section 5.

## 5 Dental Case Study Description

Our case study – developed on our Software Engineering Laboratories at PUC-Rio and UofT – is based on an academic dental clinic, which belongs to a dental association in the São Paulo State, Brazil. It members perform social activities, taking care of the community; and contributes to the dentists' academic life by specializing them in different dental branches. Our dental system involved different smart-spaces, several content servers, heterogeneous handle devices, various quality criteria (e.g. privacy, mobility, flexibility, satisfaction, context-awareness, adaptability need, e-health and academic issues) and different stakeholders' preferences, and daily activities. In the dental clinic environment, the main stakeholders were: *Patient* – user of the available dental clinic services to take care of her/his dental problem; *Dentist* – active position at the dental clinic that performs several tasks – e.g. triage process and patient's treatment; *Professor* – active position at the dental clinic that performs academic tasks – e.g. dentist's supervision and dentist's evaluation; *Attendant* – active position at the dental clinic that performs several tasks – e.g. patient's registration and registration payment; and *President*, *1o Vice-President*, *2o Vice-President*, *Secretary*, and *Bursar* as administrative positions that manage/control the dental clinic way-of-working.

## 6 Dynamic Database Building Block

As previously mentioned, our Dynamic Database Building Block is composed of the *Dynamic Database Structure* (see Sections 2, 3, and 4) and the *Layer Structure*. Our intention in this section is to present some details about the *Layer Structure*. The *Layer Structure* is basically composed of six layers: User Layer, Interface Layer, Domain Layer, Application Layer, Service Layer, and Persistence Layer. A brief description about each layer using the dental case study is presented below:

• **The User Layer:** This layer represents the users of the ubiquitous system. In the modeling level, different users are modeling as different stakeholders. In the i* models, for example, the most common used abstraction to represent the stakeholder is the actor abstraction. However, we can also use the position and the role i* abstractions to improve the modeling. In our dental case study, we had patients, dentists, professors, attendants, and other administrative positions in this layer.

• **The Interface Layer:** This layer represents the interface between the users and the ubiquitous system. The responsible for the communication between the user and the system is the *Interface Agent*. This agent runs inside the user's device. Thus, it is a simple and "light" agent, which structure is based on behavior instead of intentionality. We suggest a "light" agent as we must deal with simple devices, in which the processing and memory capacities are limited. In this case, these devices cannot support complex agents, in which the rationale requests some special devices' resources (e.g. speed, processing, and memory). In our dental case study, we also used the execution modes (e.g. split and standalone modes) of the JADE-LEAP Platform (Caire 2003) to facilitate the use/integration of limited devices in this layer. The standalone mode

allows integrating the platform and the Personal Java devices, which are capable of running the platform container as these devices are powerful and have adequate resources (e.g. processing and memory capacities) that support the container and its intentional and complex agents. The split mode allows integrating the platform and the MIDP devices, which are limited. This mode allows the limited device sharing resources with another computer that is more powerful. When the limited device connects with the powerful machine that is running "Container-1", through a wireless network, it requests that a "heavy" part of the container, called "Back-End," be maintained in that powerful machine. The other part, called "Front-End," is lighter than the first, and runs in the limited mobile device. For the user (e.g. patient and dentist,) this sharing process is invisible and performed by the proper *Interface Agent* using the JADE-LEAP Platform resources. Now, the limited device can run the container, interact with other agents, and use the services provided by the platform and the system, including the dynamic database support previously described.

• **The Domain Layer:** This layer represents the domain layer, subdivided in *Main Domain Layer*, *Transversal Domains Layer*, and *Cognitive Domains Layer*. The *Main Domain Layer* represents the Software Engineering as the master domain, in which different transversal domains are associated. The *Transversal Domains Layer* represents all transversal domains that impact in the ubiquitous systems in development – e.g. our dental ubiquitous system, such as *Pervasive Computing Domain*, *Mobile Computing Domain*, *Ubiquitous Computing Domain*, and *Multi-Agent Systems Domain*. The *Cognitive Domains Layer* represents different cognitive domains that have specific policies, contexts, content, and services. Some of them are: *Dental Domain*, *Medical Domain*, and *E-Commerce Domain*. It is also possible to define other cognitive domains categories in order to improve the layer structure, such as: *E-Health Domain*, in which we have all cognitive domains in health area (e.g. *dental*, *medical*, and *medical biology*.) We suggest the use of one *Domain Agent* for each user – e.g. for each dental clinic's patient. These agents dynamically react when it is necessary, when the users request something. Thus, they are like *Personal Agents* in the domain level, prepared to help the user in the interaction between her/him *Interface Agent* and the application layer. These agents are intentional, and have the knowledge centered on their cognitive domain. They are, for example, the responsible for the controlling of the users' personal information access in the domain level. If any other agent wants to access a specific user's personal information, this other agent must request the access to the *Domain Agent*. This last agent asks the user (e.g. patient) about this request. The user can: decide at runtime (automatically updating the database), or previously specify the preferences in her/his profile (e.g. as mandatory or not; and as public or private). This mechanism tries to guarantee that the final decision about the access depends on the user's goals, beliefs, and intentions, by respecting the user's privacy policies, and avoiding third persons' illegal invasion. This issue is a serious concern in Ubiquitous Computing that we are trying to deal with and to contemplate in our reuse-based approach.

• **The Application Layer:** This layer represents the application layer. In this layer we have the applications' specific privacy policies, and business rules. As well as the users desire to decide what are their personal access policies, the organizations – associated with the applications – sometimes desire to share information, and sometimes not. The information can be shared with the *Domain Layer* and/or other applications in Application Layer. In order to control the access, we use the concept of Capability[2]. Every agent with access/permission in relation to a specific application (e.g. "Dental Clinic A") receives the Capability that allows the interaction between this agent and the *Mul-*

2 The quality and ability of being used and improved.

*ti-Agent System* responsible for the application's information, services, and contents. Thus, the *Domain Agent* that represents the patient can only interact with the agents of

the *Application Layer* if it (the *Domain Agent*) has the application specific Capability. The process is easy, and its complexity is invisible for the final user. The patient just selects the domain she/he is interested (e.g. Dental Domain), and the *Domain Agent* finds all applications in this domain (e.g. Dental Clinic A to Dental Clinic N), according to specific issues (e.g. user's location.) The user can select herself/himself which of them she/he wants or can delegate the selection for the *Interface Agent* and the *Domain Agent*. This last agent requests the Capability to know how to interact with the application's *Multi-Agent System*. The application's *Multi-Agent System* decides if the *Domain Agent* will have the access. If they agree (according to the Dental Clinic A's internal policies and specification,) the *Domain Agent* receives the Capability and the communication starts. If they do not agree, the *Domain Agent* can automatically decide to select another application (e.g. Dental Clinic N), or even it can ask the patient to select another one, and the process will continue. Both, the application's *Multi-Agent System* and the *Domain Agent*, are intentional, and prepared to reason, learn, and react based on, respectively, the dental clinic's system and the patient's beliefs, desires, and intentions. These agents try achieving the system's and the patient's goals considering their ubiquitous profiles, preferences, privacy policies, and business rules. Moreover, the Capability support allows extending the agents knowledge and, consequently, obtaining a higher degree of reusability. We use different application's services (e.g. agents' communication protocols, services access, agents' creation, agents' registration and deregistration) as different Capabilities. Thus, for each Capability that the agent has, it knows how to perform special activities. For example, the agents' communication protocols capability, which belongs to the JADEX Plan-Lib, provides ready-to-use implementations of some common interaction protocols. In other words, the agent with this Capability has the ability of communicating with other platform agents. Another example, as previously mentioned, is the Capability offered by our ISSD for UbSystems [Serrano et al. 2008a] that provides support for the *Domain Agent* to use and to interact with the application's *Multi-Agent System*. The *Domain Agent* knows what services are offered by the application; who are the responsible agents for each service; how to interact with these agents, and how to use the services.

• **The Service Layer:** This layer contains the services offered by different ubiquitous applications (e.g. patient's registration, dental treatment payment), and others that are available when the application extends a specific framework – e.g. JADEX Framework that offers specific "capabilities": the Directory Facilitator (DF Capability) or Yellow Pages Service, the Agent Management System (AMS Capability) or White Pages Service, and FIPA Protocols (FIPA Request Interaction Protocol - RP Protocol Capability.) This layer is also coordinated by the applications' *Intentional Multi-Agents Systems*. Thus, the *Domain Agent* can only access the services if it has the permission. In other words, if it has the Capability, which must be requested for and are provided by the application's *Multi-Agents System*.

• **The Persistence Layer:** This layer is subdivided in *Domain Persistence Layer*, and *Application Persistence Layer*. The *Domain Persistence Layer* represents the domain information, which is stored in dynamic ubiquitous profiles (e.g. user's profile, and device's profile,) and can be only modified (e.g. database enquiries, database insertion, database exclusion, and database update) by the *Domain Agent*, according to its cognitive domain (e.g. dental) and the patient's privacy policies. The application's *Multi-Agents System* can receive the domain persistence layer information if it requests and has the agreement of the *Domain Agent*. Again, this mechanism tries to guarantee that the final

decision about the information access depends on the patient's desires and her/his privacy policies. The *Application Persistence Layer* represents the application information, which is stored in a dynamic database, and *a priori* can be only modified (e.g.

database enquiries, database insertion, database exclusion, and database update) by, for example, the Dental Clinic A's *Multi-Agents System*, according to the clinic's privacy policies and its business rules. This mechanism tries to avoid that important knowledge be shared without the Dental Clinic A's permission/agreement. Most of the time, the ubiquitous system is associated with an organization (e.g. Dental Clinic A) that has important knowledge – centered on their clients (e.g. patients); offered services (e.g. novel dental treatments), and business strategies (e.g. qualified dentists and adequate prices). Sometimes, a specific organization (e.g. Dental Clinic A) does not want to share this knowledge with other competitors (e.g. Dental Clinic N). This decision depends on the organization's interests and its "marketing strategy." Figure 10 illustrates an overview of our *Layer Structure*, applied to our dental case study.

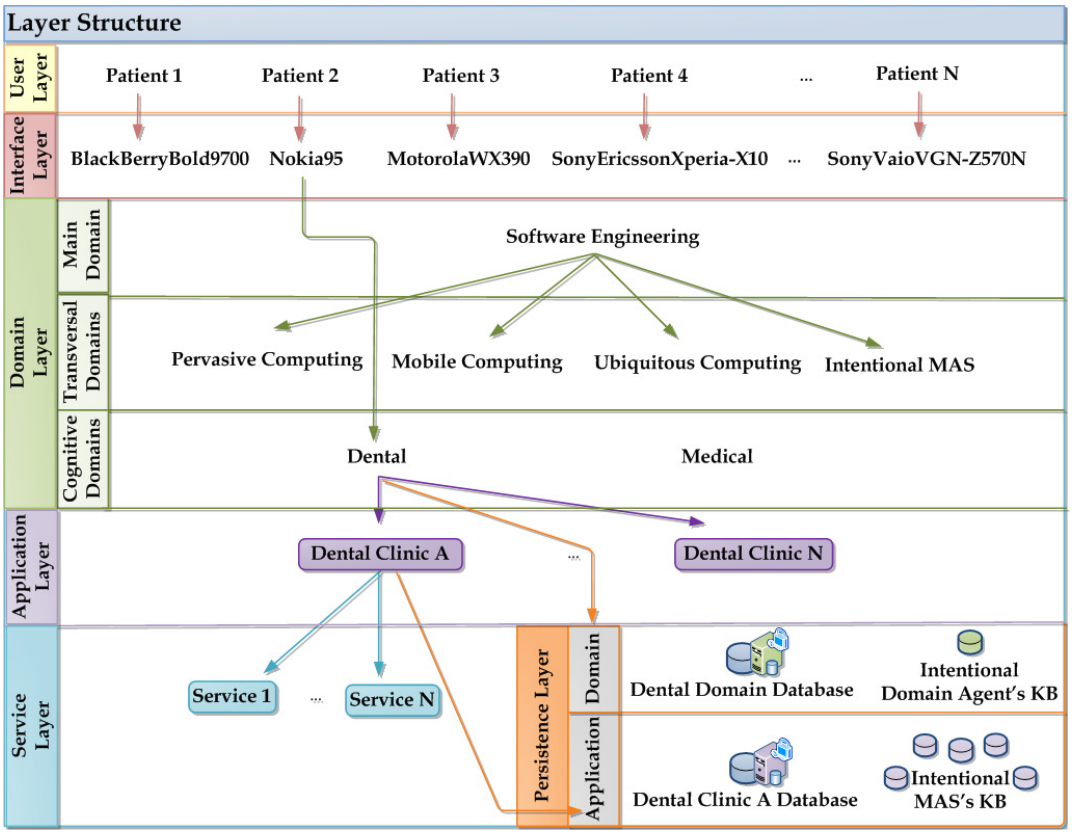


**Figure 10: Detailed Layer Structure**

## 7 Related Work

Here we will present some related work, which contains interesting ideas for dealing with stakeholders' privacy, personalization, and data management concerns.

## 7.1 Multi-Policy Access control considering Privacy in Ubiquitous Environment

In (Kyu-il Kim et al 2006) the authors Kyu-il Kim et al propose a mechanism to controls the accesses to users' private information in ubiquitous environments. This mechanism is developed by extending the Context Roles from the current RBAC/MAC. In other words, it automatically controls the accesses using specific policies centered on the Role-Based Access Control (RBAC) and the Media Access Control (MAC) address. These policies are based on different control permissions. The analysis of these permissions is made in runtime in order to know if the access is allowed or not. The main interesting point here is that the authors argue that the traditional techniques for data accessing are based on static security policies, which are pre-defined and not suitable to support the constant changes that commonly occur in ubiquitous and pervasive environments. They emphasize that in this kind of approach the dynamic location, for example, is normally not considered. Thus, they suggest a context-aware access control solution prepared to provide a flexible and suitable users' information control access in ubiquitous contexts. They also use subjects to allow specifying the authorizations' structure. This structure is not only based on the user identity, but also on the user characteristics. Thus, each user is associated with one or more credential.

*We agree with the authors in relation to the necessity of a dynamic mechanism to deal with the users' information access. However, we particularly suggest another way to provide control instead of using the RBAC/MAC architecture. We use an intentional personal agent. This personal agent can interact with other collaborative and intelligent agents centered on the beliefs, desires, and intentions of the users. Upgrades in relation to the users' beliefs, desires, and intentions can be performed in runtime anywhere and anytime, increasing the users' satisfaction. If the user desires, some information, privacy policies, privileges, and security control can previously be specified as default values (mandatory or not). However, anywhere and anytime that the user desires modifying them, they can do with the personal agent's help. We can also control the location, being location-aware, by using this personal agent inside the user's access device and some technological supports (e.g. JADEX resources* (Braubach et al. 2004) – *Yellow Pages, and Containers.) This device can be limited or not, and mobile or not as this platform can be combined with other lighter platform (e.g. JADE LEAP resources* (Caire 2003) – *Split and Standalone Execution Modes) to deal with the devices' heterogeneity and distribute smartspaces issues.*

## 7.2 A Privacy Agent in Context-Aware Ubiquitous Computing Environments

In (Zhang and Todd 2006) the authors Zhang and Todd present a personal context-aware protection centered on a privacy agent, which helps the users by notifying them about important information disclosure. In order to develop this special agent, they use an ontological platform for Privacy Preferences Project (P3P) (Cranor et al 2005). They also argue that the existing approaches focus on conventional support, which are centered on pre-defined and simple privacy policies specification that is extremely inappropriate to deal with dynamic requirements in ubiquitous contexts. They present that most of the time, these approaches suggest that the users/clients specify their privacy policies by *"filling forms with pre-defined layouts and options."* This inflexible support is particularly not convenient in ubiquitous scenarios, in which the users/clients information sharing depends on the time, the preferences, the location, the current activities, and may change overtime. The information disclosure must be controlled including getting notice, feedback, and explicit consent.

*We agree with the authors in relation to the necessity of a dynamic mechanism to deal with the users' information access. Moreover, we also agree on using agents to control the access of the users' information. In the authors' proposal, this agent is called Privacy Agent, and in our approach this agent is called Personal Agent. The first observation is that the authors do not specify how they constructed their agent, based on behavior abstraction or based on intentionality. In our case, we suggest the use of an intentional agent centered on BDI Model (Bratman 1999). The abstractions of the BDI Model are closer to the stakeholders' goals representation. The behavior abstraction is adequate to model pre-defined situation. However, in ubiquitous contexts the situations change, and the stakeholders' beliefs, desires, intentions, preferences, goals, and interests are in constant evolution. This ever-changing scenario demands different agent's strategies. Thus, the agent based on the intentionality is capable of reasoning, learning, solving problems, and taking decisions by considering the user's goals and preferences. The second observation and difference between us and the authors' approach is that we suggest the use of a dynamic database (see Sections 2, 3, 4, and 6) to improve, for example, the knowledge storing; the privacy policies dynamic specification; and the users' data security and integrity. Furthermore, we also suggest a specific and detailed layer structure (see Section 6) to support (among other things): the domain and the application privacy policies; the devices' heterogeneity, and the ubiquitous context evolution.*

### 7.3 The Agent Layer Concept and Ubiquitous Concept Databases

In (Mitsubuchi 2003) the author proposes an infrastructure for developing ubiquitous information society centered on a specific agent layer, called "Agent Layer Concept," and a database, called "Ubiquitous Concept Database." The Agent Layer Concept is modeled based on the information transfer mechanism of the human nervous system. This concept also categorizes this information transfer, which is commonly presented in our society, into five different layers: **(i)** Environment Layer represents the real world, including the user; **(ii)** Device Layer represents the devices that allow the communication between the real world and the electronic brain; **(iii)** Personal Layer represents the server that performs processing when a response is necessary/required; **(iv)** Agent Layer represents the center clustering that creates and controls the agents; and **(v)** Database Layer represents the database in which the data is stored, modified, and excluded. The Ubiquitous Concept Database allows the agents absorbing differences in different layers, and enabling users to send/receive information without necessarily considering the other party existence. In this case, the decentralized control is putting in practice over the agents and data. The author also describes the use of specific attributes (e.g. private, group, and public) in data in order to facilitate the management of them. The agents perform their actions in accordance with each specified attributes.

*We also use a special infrastructure centered on multi-agent systems, database, and layer structure. Moreover, our infrastructure also respects the user's privacy and preferences. Our differences are in the way we developed our agent, the way we divided our layers, and the use of a dynamic model to construct our database. Our agents are intentional, they reason and learn centered on the users' beliefs, desires, and intentions. We also improved the reasoning of our agents by considering different softgoals, which are presented as fuzzy variables. We propose different layers from the real world to the persistence level as presented in Section 6. Furthermore, this layer structure allows controlling different information according to the users' profiles, the domain privacy policies, and the ubiquitous applications particular needs. The privacy policies can be previously specified (as mandatory or not mandatory), and/or can be specified at runtime, anywhere and anytime, in accordance with the term "UBIQUITOUS," which means (in Latin): "existing everywhere." Defining the attributes (e.g. private, group, and public) without considering the runtime situation, concerns, and needs can be insufficient or even in-*

*adequate to deal with dynamic requirements presented in context-aware paradigms (e.g. Pervasive Computing and Ubiquitous Computing). Our database structure is dynamic, which allows the storage, updates, and exclusion, providing a flexible and reusable mechanism to manager data as presented in Sections 2, 3, 4, and 6.*

## 8 Final Considerations

We presented some ideas about our Dynamic Database Building Block, as well as the *Dynamic Database Structure* and the *Layer Structure* that compose it. We are particularly focused on the concerns: **(i)** the stakeholders' privacy; and **(ii)** the constant evolution of ubiquitous contexts.

Our *Dynamic Database Architecture* is based on the *TypeObject Pattern* and the *Type-Square Architecture*, both proposed by Yoder in (Yoder 2001). Our *Layer Structure* is organized into six layers, going from the real world level (*User Layer*) to the persistence level (*Persistence Layer*.) In order to facilitate the reuse of our Dynamic Database Building Block support, we developed an API, which can be added for the application project as a ***jar*** file. We tested our API – at PUC-Rio and UofT Software Engineering Laboratories – in a complex dental case study. Among other tests, we evaluate the users' satisfaction, agent's performance, database's response time, and developers' spent efforts and time. The results make us to believe that our structure is flexible and suitable to deal with the concerns previously presented. Figure 11 shows the results for the use's satisfaction in terms of the system's usability and dependability in two developed ubiquitous projects: Media Shop, without the Dynamic Database Building Block support; and Smart Dental Project, with the support. For the first project, the usability was evaluated by the users from good to excellent (Figure 11 – Part 1a), and the dependability from regular to excellent (Figure 11 – Part 1b). For the second project, both the usability (Figure 11 – Part 2a) and the dependability (Figure 11 – Part 2b) were evaluated from very good to excellent. In order to facilitate the comparison, we only considered the worst 16 evaluations for each issue and for each project. Moreover, the usability is associated with the system's invisibility issue and its ability to react according to the contexts and users interests' evolution. Both supported by the cognitive software agents; and the dependability is associated with privacy and reliability issues.

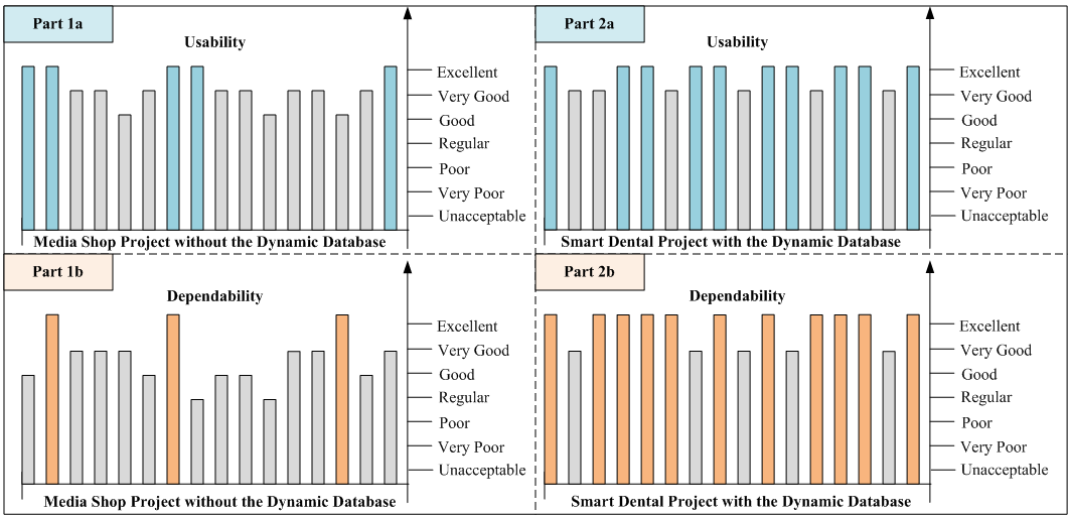


**Figure 11: Usability and Dependability Evaluation**

As further work we will combine this building block with our currently investigated ontological support (Serrano and Lucena 2010) in order to improve the communication among the agents in different smart-spaces. The ontological support is composed of different ontologies to standardize the knowledge representation, manipulation, and management in different levels: interface elements – to dynamically construct interfaces based on the ubiquitous profiles information; domain elements – to facilitate the communication among domain agents, interface agents, and application agents centered on users' privacy policies; and application elements – to manage the application data sharing and access, and to improve the communication among the applications' multi-agent-systems centered on applications' privacy policies and business strategies.

Moreover, we intend to incorporate a dynamic reorganization algorithm [Sockut and Iyer 2009], which can improve the database reorganization based on the agents' intuition and centered on different metrics (e.g. user's intentions, different priorities, most accessed knowledge, and data deterioration rate).

## References


Bratman, M. E. "Intention, Plans, and Practical Reason." Distributed by the University of Chicago Press, ISBN: 1575861925, 208 pages, March 1999.

Braubach, L., Pokahr, A. and Lamersdorf, W. "Jadex: A Short Overview." In Net. ObjectDays'04: AgentExpo, pp. 195-207, September 2004.

Caire, G. LEAP User Guide. TILAB, last update: December 2003.

Cranor, L.; Dobbs, B.; Egelman, S.; Hogben, G.; and Schunter, M. "The Platform for Privacy Preferences 1.1 (P3P1.1) Specification." July 2005. (last update: November 2006).

Kyu-il Kim, Hyun-Sik Hwang, Hyuk-Jin Ko, Hae-Kyung Lee, and Ung-mo Kim. "Multi-Policy Access control considering Privacy in Ubiquitous Environment." International Conference on Hybrid Information Technology (ICHIT'06), 0-7695-2674-8/06, pp. 216-222, November 2006.

Mitsubuchi, Keiji. "The Agent Layer Concept and Ubiquitous Concept Databases." January 2003. Available at www.nwco.com/jp/menu04/ucdb_e.html (Last Access: February 2010)

Mylopoulos, J. "Goal-Oriented Requirements Engineering." XI Conferencia Iberoamericana de Software Engineering (CIbSE'08), pp. 13-17, Pernambuco, Brasil, February 2008.

Serrano, Milene; Serrano, Maurício; Lucena, C. J. P. "Framework for Content Adaptation in Ubiquitous Computing Centered on Agents Intentionality and Collaborative MAS." Fourth Workshop on Software Engineering for Agent-oriented Systems (SEAS), 12 pages, 2008.

Serrano, Milene; Serrano, Maurício; Lucena, C. J. P. "Ubiquitous Software Development Driven by Agents' Intentionality." 11th International Conference on Enterprise Information Systems (ICEIS'09), vol. SAIC, pp. 25-34, Milan, Italy, May 2009.

Serrano, Milene; Lucena, C. J. P. "Applying FIPA Standards Ontological Support Intentional-MAS-Oriented Ubiquitous System," 8 pages (to appear in the Proceedings of

12th International Conference on Enterprise Information Systems (ICEIS'10), Funchal, Madeira, Portugal, June 2010.

Shoham, Y.; Leyton-Brown, K. "Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations." Cambridge University Press, ISBN 9780521899437, 496 pages, 2008.

Sockut, G. H.; Iyer, B. R. "Online Reorganization of Databases." ACM Computing Surveys, http://doi.acm.org/10.1145/1541880.1541881, Vol. 41, No. 3, Article 14, July 2009.

Weiser, M. "The Computer for the 21st Century." ISSN 0036-8733, Vol. 265, Number 3, p. 94, Scientific American, New York, NY, September 1991.

Weiser, M. "Some computer science issues in ubiquitous computing." Communications of the ACM, Computer augmented environments, Vol. 36, Issue 7, pp. 75-84, ISSN: 0001-0782, 1993.

Weiser, M.; Brown, J. S. "Designing Calm Technology." Xerox PARC, 1995.

Yoder, J. W.; Balaguer, F.; Johnson, R. "Architecture and Design of Adaptative Object Models." Conference on Object Oriented Programming Systems, Vol. 36(12), pp. 50-60, December 2001.

Yu, E. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering." In Proceedings of the 3 rd International Symposium on RequirementsEngineering (RE'97), pp. 226-235, Washington D.C., January 1997.

Zhang, N.; Todd, C. "A Privacy Agent in Context-Aware Ubiquitous Computing Environments." IFIP International Federation for Information Processing, H. Leitold and E. Markatos (Eds.): CMS 2006, LNCS 4237, pp. 196 – 205, 2006.