3.
Related Work: Traditional & Emergent Approaches

"As more computational devices are put into place, designing the user experience as an ensemble of interactions will become paramount… Consequently, it's important to begin planning and designing how real / virtual devices will cooperatively interoperate".

Daniel M. Russell and Mark Weiser, "The Future of Integrated Design of Ubiquitous Computing in Combined Real & Virtual Worlds," ACM - CHI, 1998.

In this Chapter, we provide an overview of traditional and emergent development approaches, methodologies and processes. This related work was formulated in a way to provide an adequate view of what Software Engineering offers for the development of applications in different cognitive domains. We focus on RUP, TROPOS, GAIA, Agile Methods, Mobile-D, and our reuse-oriented approach. Finally, Section 3.7 summarizes the Chapter by presenting some closing remarks.

3.1.
RUP

RUP (Kroll and Kruchten 2003) is a framework for object-oriented processes, which defines a large number of conceptual elements (e.g. roles, artifacts and work flows). It is possible to instantiate this framework according to specific needs of the software development processes, such as the RUP instance to SOA (RUP SOA 2008). RUP is not agent-oriented as the methodologies presented in Sections 3.2 to 3.3. The main phases of the RUP are described as follows: (i) *Conception*: this phase comprehends the planning and the analysis of requirements, in which the users' profiles and the essential use cases are defined; (ii) *Elaboration*: this phase considers the modeling of the project using diagrams (e.g. UML sequence, collaboration, state, and class diagrams) and the interface design; (iii) *Construction*: this phase comprehends the implementation of the modeled application, by also considering the configuration of the development environment and the persistence details; and (iv) *Transition*: this phase organizes

and performs the deployment of the developed application, which includes the software packages availability, the distribution and the installation of the application. We are particularly interested in the RUP because of its meta-process feature, which allows other processes to extend its development model to deal with specific purposes.

3.2. TROPOS

TROPOS (Giunchiglia et al. 2003) is an agent-oriented software development methodology. Basically, this methodology has two key principles: (a) the agent concept associated with goal, task and various other knowledge levels, which are used throughout the entire software development process; and (b) the notion that the main phase of the development process is assigned to analysis and specification of the application's requirements. This methodology comprehends five disciplines: (i) *Early Requirements*: this discipline consists of the identification of the domain's context under analysis as well as its modeling by using goal-oriented approaches; (ii) *Late Requirements*: this discipline introduces the application as another actor. Therefore, other application's stakeholders are elicited. The application as another actor is probably associated with other stakeholders. Thus, it is important to consider the dependencies among them. These dependencies represent the obligations of the application towards its environment and actors; (iii) *Architectural Design*: this discipline considers some new actors (e.g. other correlated applications) and the subgoals and subtasks of the goals associated with them and the application under development; (iv) *Detailed Design*: this discipline details the application's actors by focusing on the design of their capabilities using a detailed modeling to capture the actors' rationale. It is also relevant to establish, for example, the protocols and other technological support that will be used to orient the application's implementation; and (v) *Implementation*: this discipline is dependent on the details specified in the design disciplines. The TROPOS suggests a mapping between the detailed design and the implementation template. The code is added to the template using an agent-oriented programming language supported by a programming platform, such as JACK (Busetta et al. 1999).

Additionally, this methodology is centered on interesting concepts and models, such as: actor, goal, dependency, task, resource, belief, Strategic Dependency model and Strategic Rationale model (Yu 1997). All these concepts and models are used in our approach from the modeling to the implementation of intentional-MAS-driven ubiquitous applications. Moreover, our approach – in the *Ubiquitous Application Engineering* – is inspired by the TROPOS disciplines and its model-driven nature.

3.3. GAIA

The GAIA Project (Román et al. 2002), developed by the Illinois University at Urbana-Champaign, is a distributed middleware operating system infrastructure that manages the resources contained in an intelligent environment and provides support for Pervasive Computing. The Cerberus core service of GAIA integrates identification, authentication and context awareness. GAIA allows applications to be partitioned between different computers and to be moved from computer to computer to satisfy the user in a controlled environment. This group developed a methodology, called GAIA (Moraïtis et al. 2003). The GAIA methodology group is particularly interested in the definition of a complete and general methodology, which is specifically tailored to the analysis and design of Multi-Agent Systems. According to GAIA, Multi-Agent Systems are composed of autonomous interactive agents that live in an organized society in which each agent performs some specific roles. The Multi-Agent System is determined based on a role model, which identifies the roles that each agent will perform in the MAS platform. Moreover, it establishes the protocols that will be used in the interaction between various roles. Our interest in GAIA is centered on its experience (e.g. popularity, several case studies, abstractions, protocols) in dealing with Multi-Agent Systems challenges. Moreover, GAIA works with the intelligent environment concept. In our approach we represent this intelligent environment as a smart-space, which is composed of different stakeholders, devices, servers and contents. These smart-spaces are distributed and connected using a MAS platform, supported by novel technologies.

3.4. Agile Methods

The Agile Methods are methods based on iterative and incremental development, in which the requirements evolve through cooperation between self-organizing and cross-functional teams.

Ideas centered on the incremental software development have been traced back to 1957 (Larman and Basili 2003). However, after the Agile Manifesto (Beck et al. 2001) and according to (Larman 2004), lightweight methods (e.g. Scrum (Schwaber and Beedle 2002), Crystal Clear (Cockburn 2004), Extreme Programming (Beck 1999), Adaptive Software Development (Highsmith 2000), Feature Driven Development (Palmer and Felsing 2001), and Dynamic Systems Development Method (Stapleton 1999)) have been considered as agile methods. The main idea of these methods is to develop the software based on small iterations and without involving long-term investigation. Therefore, these iterations are performed in short time-boxes, from one to four weeks.

Moreover, the Agile Manifesto comprehends some principles (Beck et al. 2001), such as: (i) rapid software delivery is used to satisfy the customers; (ii) development progress measurement is based on the teams' efforts/working; (iii) constant interactions between the customers and the developers – face-to-face – are encouraged; (iv) even late changes in requirements are welcome; and (v) simplicity and regular adaptations to follow ever-changing situations are desired.

In order to achieve these principles, each iteration in agile methods demands a team working through all the software development cycle. This process includes activities from the requirements elicitation to the software implantation. In addition, and to contribute to the software domain's investigation, a representative customer is appointed by the stakeholders to respond according to their intentions. The purpose is to reduce undesirable outcome as well as to allow constant adaptations (Highsmith 2000) throughout the project's life-cycle.

Another particular characteristic of agile methods is the usage of small teams to develop the software (Beck et al. 2001). This practice facilitates the teams' collaboration and communication. In case of larger dedication, it is recommended to use multiple small teams. If it is possible, it is also desired to establish a face-to-face communication among teams' members. However, the

usage of e-mail, video-conferences, Skype and other resources are suggested to make communication possible and to improve it. Finally, specific strategies centered on pair programming, design patterns, domain-specific development, and other support sets also reinforce the success of the agile-method-based project.

There are some project features that do not contribute to the success of agile methods application, such as: (i) projects with large-scale development dedication; (ii) projects with spread developers – i.e. dispersed teams that do not share common facilities; and (iii) projects in which failures are really critical.

We have applied some principles of agile methods to our approach as the ubiquitous scenarios are in constant evolution and the ubiquitous applications must follow these changes over time. In this case, the use of sequential approaches is not recommended. We found it desirable the use of an incremental support to quickly adapt the application under construction according to the actual context. Moreover, ubiquitous applications demand special attention from their users. Therefore, it is also relevant the use of a method centered on the user satisfaction, such as the agile methods.

3.5. Mobile-D

Mobility-centered software development is an increasingly important development approach for software companies. However, the mobile application development is burdened with several challenges. Due to the mobility issue, we can mention an interesting approach proposed in (Abrahamsson et al. 2004). This approach suggests that agile methodologies can offer a good solution for mobile application development. It is aimed at evaluating the suitability of agile methods for mobile application development projects by bringing a set of improvements to an established agile method, called Mobile-D. This recent approach is object-oriented. Mobile-D offers support for dealing with technical constraints of mobile application development by including some guidelines to improve and facilitate the development process. Moreover, this methodology offers a comprehensive specification for each phase and stage and for the associated tasks by considering some principles that fit mobile development projects. Furthermore, Mobile-D has been extended to include a new phase, called Evolve. This phase is concerned

with improving the product through constant end-user feedback. This suitable development approach impressed us as it is specific for mobile application development. Thus, it concerns with distribution, mobility, and some ever-changing situations. However, it does not deals with some specific concerns of ubiquitous applications development, such as: (i) intelligent smart-spaces, (ii) ubiquitous profiles investigation at runtime, (iii) the balancing between invisibility and transparency as well as personalization and privacy issues, and (iv) several non-functional requirements that must be investigated "on the fly" by also considering the ubiquitous context under analysis.

3.6. Our Reuse-Oriented Approach vis-à-vis Related Work

As previously presented, the mentioned related work does not deal with specific concerns of ubiquitous applications development (e.g. dynamic interface construction, ubiquitous profiles manipulation at runtime, and quality criteria analysis “on the fly”). Moreover, it is difficult to find related work that offers engineering guidelines to promote reusable artifacts and facilitate the development of intentional-MAS-driven ubiquitous applications from requirements’ specification to evaluation process. Therefore, we propose a reuse-oriented approach for the systematic development of intentional ubiquitous applications by combining some interesting support sets provided by the related work with different technologies. These technologies were investigated and determined by our research group since 2007 in order to compose adequate technological support sets, specific to deal with commonly found ubiquitous issues. Figure 3.1 summarizes our approach’s main orientations based on emergent and traditional approaches.

3.7. Closing Remarks

This Chapter presented approaches, methodologies and processes focused on different orientations: (i) **RUP** is an object-oriented process, with can be instantiated to specific needs of the application; (ii) **TROPOS** is a goal-oriented methodology that emphasizes the importance of the requirements to the

application development. This methodology provides resources to deal with the intentions of the users from the requirements to code. Moreover, it is possible to evaluate the application under construction from the requirements modeling, which tries to previously "guarantee" the success of the developed application; (iii) **GAIA** in an agent-oriented methodology, whose success in pervasive scenarios is really great. However, this methodology does not apply intentional agents. As previously explained, the intentionality can contribute to the agents' cognitive capacity; (iv) **Agile Methods** that support iterative and incremental development with some particular principles (e.g. rapid software development and constant interactions to satisfy the users); and (v) **Mobile-D** that supports the mobile applications development by using an agile methodology centered on the object-oriented paradigm and mobility issues. This thesis proposes a *Reuse-Oriented Approach for Incremental and Systematic Development of Intentional Ubiquitous Applications*. The approach combines the ideas of Software Reuse, Goal-Orientation and Intentional Multi-Agent-Systems. Moreover, the *Ubiquitous Application Engineering* was defined by using the TROPOS disciplines and some agile methods principles, and by reusing building blocks.

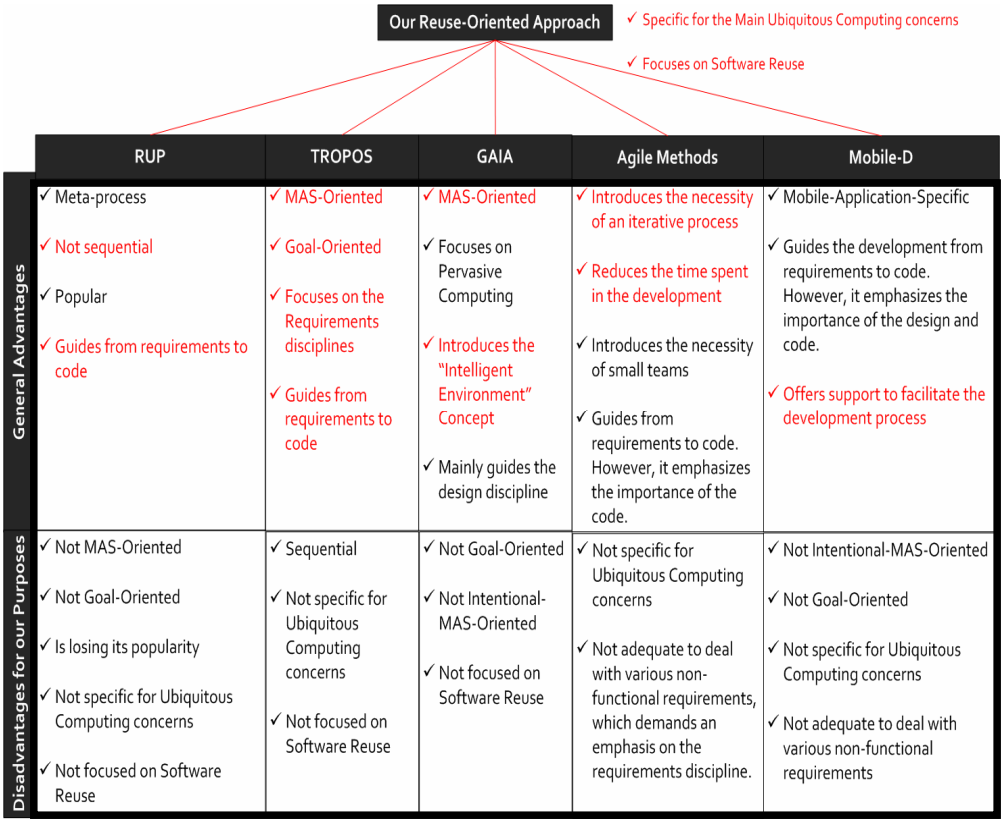

Figure 3.1 - Our approach vis-à-vis traditional and emergent approaches