A flexible framework to prototype, develop and manage ubiquitous applications for intelligent environments

Matheus Erthal
Institute of Computing
Federal Fluminense University
Rio de Janeiro, Brazil
Email: merthal@ic.uff.br

David Barreto
Institute of Computing
Federal Fluminense University
Email: dbarreto@ic.uff.br

Douglas Mareli
Institute of Computing
Federal Fluminense University
Email: dmareli@ic.uff.br

Orlando Loques
Institute of Computing
Federal Fluminense University
Email: loques@ic.uff.br

***Abstract*—Due to recent advances in mobile computing and wireless communication technologies, we can see the emergence of a favorable scenario for building ubiquitous/pervasive applications. This work proposes a state-of-art framework for building those applications by solving some of the main issues of the field, namely the heterogeneity of devices, the amount and diversity of context informations and services, and the difficulty of developing and testing ubiquitous applications. The abstractions provided, and the management services conceived, allow developers to handle resources spread within the intelligent environment in a simple and homogeneous way. Furthermore, they enhance the capability of implementing context-aware applications, while still keep the flexibility on developing such applications. In order to demonstrate the applicability of the proposal, we implemented the concepts in a platform named *SmartAndroid*, which contemplates most of the framework's features and has enabled the construction of a set of example applications to validate the framework. Above the platform we implemented a management and prototyping interface that uses a tablet device to allow the testing of applications running on different configurations of the environment. This intelligent environment may be loaded with physical and/or virtual resources, what enables the emulation of devices as well as the simulation, without the need of purchasing all those devices. The interface also allows the composition of context rules either by developers or non trained users.**

I. Introduction

The field of Ubiquitous Computing (UbiComp), nowadays widely discussed, was first proposed by Mark Weiser in the 1990's [1]. Also referred by pervasive computing, the field aims at providing a different paradigm of human-machine interaction. As traditional applications provides services to users through their explicit interaction (using devices as mouse, keyword, monitor, for example), in ubiquitous applications (UbiApp) the interaction happens without the need of explicit interaction, i.e. the application tries to discover the users needs through the acquisition of context (using sensors) and the knowledge of their preferences, and providing services in the environment (using actuators). These environments enhanced with sensors and actuators to provide automated services to persons are also called Intelligent Environments (IE).

The construction and manipulation of UbiApps represent major challenges for developers, especially in terms of technical knowledge required and the availability of real devices during application development. Some of these challenges can be well highlighted: (i) there are difficulties in establishing a common protocol for communication between the components of the distributed system, because of the *heterogeneity of devices* involved, (ii) the interactivity of UbiApps is hampered depending on the amount and *variety of context information and services* available in the environment, (iii) *developing and testing applications* require high availability of resources, such as sensors (e.g. presence, lighting, temperature), actuators (e.g. keys, alarms, smart-tvs), including new embedded devices, or physical spaces, such as a house for applications of type smart home.

In this work we propose a framework for developing UbiApps in IE. The goal is to provide support for the programming, testing and execution of applications, thus allowing to deal consistently with systems great complexity. The framework stands out for addressing the challenges already identified in UbiComp [2]. The *heterogeneity of devices* is handled through the definition of a Distributed Component Model, that provides abstractions to encapsulate these devices, also called resources, enabling developers to interact with them seamlessly. Regarding the *variety of context information and services* issue, we propose a solution for context interpretation that allows developers to create and manage context rules at runtime, and users to set their preferences in the IE. Finally, regarding the issue on the *resource availability*, the framework includes an application interface for the prototyping and management of pervasive applications (IPGAP), focused on visualization and testing UbiApps, mixing real and virtual components.

The concepts of the framework were materialized on a platform called *SmartAndroid*[1], whose development has enabled appraisal in order to prove that the conceptual framework facilitates the process of building UbiApps. We implemented some use cases to validate different aspects of the framework proposed and to testify its capabilities, supported by the elaboration of competency questions, which is a mechanism mostly used to evaluate ontologies, but can also be used as an assessment to this work.

The remainder of this paper is organized as follows. First we present in Section III an overview of the main concepts

[1] www.tempo.uff.br/smartandroid

used as a basis for the framework's development. In Section IV, we present the framework's overall architecture and main features. We then, in Section VII, present a proof of concept demonstrating the feasibility of building UbiApps using the framework and we show examples of applications that explores the key features of the IE. Finally, in Section II we compare our approach with related work and in Section VIII we conclude this paper with the main remarks. [refactor this paragraph]

II. Related Work

The benefits to users and developers that UbiComp foresees reach beyond the horizon of many imaginative researchers. The possibilities that arise from it surpass the bounds established by keyboards and mice, our standard interfaces. Nonetheless, building such adaptable applications requires much effort from developers that see themselves immersed in a universe of device's specifications and communication technologies, apart from the problem itself that the application must solve. Therefore, many researchers have focused on diminishing those obstacles by providing abstractions, middlewares, services, tools, and other supportive techniques, not only for development but also for prototyping of UbiApps.

Most approaches focus on one aspect of UbiComp, such as security [], location [], context representation [], SmartHome specific [3], among others. [+++]

As one of the most famous, the Gaia middleware is designed to facilitate the construction of applications for IE. It consists of a set of core services and a framework for building distributed context aware applications. The Gaia embraces many goals that include the acquisition of context, the maintenance of hardware and software descriptions, mechanisms to find resources and to compose context rules, among others.

The Context Toolkit provides a set of abstractions for the composition of reusable components that focus on context acquisition and interpretation. [+++]

[Gator Tech]

The solution presented in this paper strives to accommodate central issues in the development of UbiApps, as identified by the challenges (i), (ii) and (iii). The framework proposed facilitates the construction of those applications but don't restrict its development, being flexible enough to accommodate other concerns that are system specific.

III. Overview

It is easy to notice that the context information is a first concern topic in ubiquitous systems. But what is "context"? Many authors have proposed definitions to this concept, though most are little accurate [4]. Synthesizing previous definitions, Dey and Abowd have proposed that context is "any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves" [5]. In a IE "Places" are the rooms, the floors of a building, or spaces in general that enable the localization of other entities, "People" are individuals that populate the environment and interact with it,

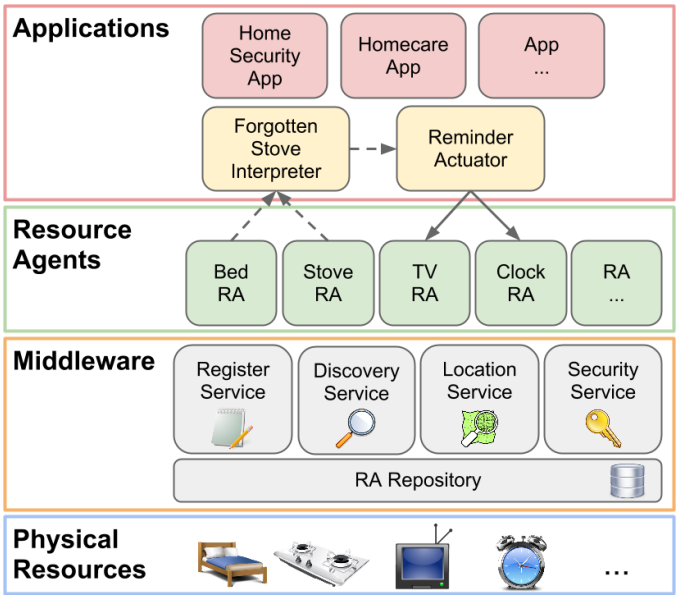

Fig. 1. Framework Layer Architecture

and "things" are virtual representations of physical objects or software components.

Context aware (or context sensible) applications is a designation to those applications that not only are capable of knowing the context of the environment, but also react to it either by means of changing the environment or in a software level. []

[distributed systems]

IV. Framework Architecture

The framework provides facilities and programming standards typically required in context-aware applications focused on IE. Among the features is the ability to abstract details that involve communication, context acquisition and resource location, by the applications. For the sake of abstracting resource details we have used components called *Resource Agents* (RA). The RA is the basic modularization unit of the framework being used in the modeling, implementation and management of applications (Subsection V-A).

The decentralization of components of the framework (the RAs) follow the principles of the Service-oriented Architecture (SoA), where the more decentralization the more flexibility on connectivity, since the dependence of centralized services is lowered.

The general architecture of the framework includes a Distributed Components Model, which defines the RA structure and the Resource Management Support Services (RMSS). The Figure 1 presents a layer architecture that highlights main aspects of the framework. The layers are described in the following items.

1) Physical Resources Layer: The deepest layer is the one where is found the resources present in the environment. In a IE these resources are the smarter versions of daily used objects, such as beds, stoves, TVs, clocks, and so on. Since all these resources have their own mechanisms of communication

and operation, then a higher level entity is required to deal with them.

2) Middleware Layer: This layer comprehends the main services that sustain the interoperability of the RAs and provides basic features to the development of UbiApps. The management services and the RA Repository, that compose the middleware, will be further discussed in Subsection V-B.

3) Resource Agents Layer: This layer is composed by all the RAs that represent physical resources already coupled with the IE. The RA will be presented in Subsection V-A.

4) Applications Layer: The higher layer includes all UbiApps that are enjoying the middleware features and also all the software level RAs, which are represented in the figure by the "Forgotten Stove Interpreter" and the "Reminder Actuator".

V. Framework Features and Application

This section approaches the main features provided by the framework, which includes the definition of the RA (Subsection V-A), the management services (Subsection V-B) and the context interpretation (Subsection V-C). In the next section we will present the current status of the implementation.

A. Resource Agents

A resource may be defined as any hardware or software entity that exposes services (of data acquisition or actuation) which can be used by other entities and applications present in IE. Thus, sensors, actuators and smart devices (e.g. stove, fridge, air-conditioning), as well as software modules that provide a service to the environment, fall within this definition.

The RAs are entities that represent resources. They encapsulate the specificities of resources and expose their information to the environment through standard interfaces, so that others can access them uniformly reducing the complexity of integration.

The Dey and Abowd's definition to context proposes three entities as the most basic of a ubiquitous system, i.e. person, place and object (or thing). Although the difference of concepts, we have conceived all these three entities as RAs on the framework, but observed their particular features. Thus they inherit the convenience of being a RA.

[wrapper]

1) RA Interfaces: The RA architecture includes two interfaces for input and output. The former is called Context Variable (CV) and is responsible for exposing the RA's context information. The latter is called Operation (OP) and is responsible for exposing features (or actuation services) belonging to RAs, enabling applications to actively interact in the environment.

Consider the simple example of a smart stove where the instantiation of its RA exposes services to get its current context (e.g., oven is turned on, oven temperature in $^{\circ}\mathrm{C}$) and services that may change its context (e.g., turn oven off, set temperature as $150\,^{\circ}\mathrm{C}$). Hence, the RA concept meets the first challenge (i), which is the *devices heterogeneity*, since decoupling sensor and actuator from the sensor platforms favors the flexibility.

2) Communication: In order to meet requirements typical of distributed environments, two communications mechanisms have been included in the framework: the Remote Procedure Call (RPC), that implements synchronous communication; and a publish-subscribe mechanism (pub-sub) [6], that implements asynchronous communication. The RPC is used to access directly the RA, querying for its context by accessing its CVs, or calling its OPs. On the other hand, through the pub-sub a RA can subscribe to a CV of another one (consumer) and further receive a notification, when the first RA publishes its CV state change (provider).

The RA has a domain name, the RA Name System (RANS), that allows its unique identification in the IE. This identification is similar to the DNS, where the name remains the same even if the network address (IP) changes, the only change is the updating of the addressing table.

3) RA Structure: In analogy to object-oriented programming we define classes of RAs to represent the type of the resource with specific interfaces. Besides other informations present in the AR's structure (such as name, location, subscribed RAs, among others) it has its type hierarchy. The type hierarchy is composed by a sequence of classes that classify the entity and has been inspired by the ontology described by Ranganathan [7]. In the Subsection V-B we will present how this feature can be used to find a RA and to define its functionalities.

B. Resource Management Support Service

A set of services is responsible for providing basic functionalities to manage the RAs. The four main services are the RA Register Service (RRS), the RA Discovery Service (RDS), the RA Location service (RLS) and the Security Management Service (SecMS). All these services access the RA Repository (RR) that maintains informations about the RAs (addresses and descriptions) and the map data structure (map of the environment). The Figure 1 represents, in the Middleware layer, the RR and above it the management services.

The management services are also designed as RAs, hence they take advantage of the communication mechanisms already provided by it and communicate using the same primitives.

1) RA Register Service: The RRS allows the registration and deregistration of RAs instances references and description. This service also has CVs that allows the developer to know when RAs enter and leave the IE

The naming and addressing scheme can be generalized to work over the internet, and comply with the Internet of Things concept.

2) RA Discovery Service: The RDS allows the search for registered RAs based on specific attributes, which are the RA name, RANS and type. The search by RANS will only return one RA, but the search by name or type will return all RAs that meet this criteria.

After the discovery of a RA, the stakeholder can instantiate a stub (i.e. a proxy) of this RA, when both the stub and the current implementation of the RA implement the same interface. Through the stub the CVs and OPs can be accessed, what internally will be converted to network calls to the RA object (RPC functioning).

During the application initialization the only information required get started is the RDS address. And even this information can be obtained, if within a LAN, by sending a broadcast message with the request; or using the DNS, if within a WAN. With the RDS address on hands any RA can be found, including the other management services (i.e., the RRS, RLS and RSS).

3) RA Location Service: The RLS explores the map of the environment to run the required queries. This service performs searches related to the location of RAs, such as the physical location of RAs, the RAs located on some place, the RAs of some type located on some place, the RAs close to another and order by proximity, among others. Thus, if a developer needs audiovisual devices close to some person, for example, she can run this search through the RLS and will receive a list, where an utility function can choose the more relevant result.

The RLS exposes CVs that allows not only the query for if a RA is located at some place, but also to subscribe to that place for the sake of being notified of the entrance or exit of a specified RA type. Therefore, even if the RA is still not registered, an application can subscribe to a place and wait for its appearance.

Nowadays there are several technologies to map and/or track the location of entities, that may include from people to small things. These technologies vary mainly based on precision, ubiquity (capable of being "invisible") and price. As examples we have the GPS, the smart floor, cameras (associated to image processing softwares), Wi-Fi triangulation, RFID tags, and so on. In this approach we do not comply with any specific technology, and we assume that the RA can discover its own location or the RLS can.

4) Security Management Service: The security concern is always a central topic on UbiComp systems, moreover because of diverse aspects of security required. Some aspects are already inherited depending on the technology adopted, e.t., the Wi-Fi already ensures against the registration of outside devices and the exchange of informations.

[credentials]

[domains]

C. Context Interpretation

[motivation]

The context interpretation serves to aggregate context information from different sources in accordance with some specific logic and considering the passage of time. This logic is defined by the developer and the interpretation is implemented with the framework's support.

The entity that performs the context interpretation may be encapsulated by a RA, thus allowing the subscription to other RAs, what is called Context Interpreter (CI). This concept allows dealing with context in a higher level and also provides a separation on concerns, since a context interpreter decouples the steps for context acquisition, evaluation, timing and notification, that are repeatedly performed by the UbiApps. Therefore, using the CIs also promotes the reuse, either by the creator application or by other ones.

The framework uses a generic approach for creating context rules, which are composed by two parts: the interpretation and the actuation. The interpretation part, as already stated before, concerns on binding the CI itself with CVs from different RAs, evaluating the expression that interrelates those CVs, and notifying whoever is interested. The actuation part is a RA that is subscribed to the CI, and develop some job, that may change the context of the IE or change some state in a software level.

An example of context rules has been presented on Figure 1. At the Applications Layer a set of two components implements the interpretation-actuation pattern: the "Forgotten Stove Interpreter" and the "Reminder Actuator" repectivelly, where the goal of the rule is to remind the elderly that he has forgotten the oven on while he went to sleep. The former component, which is a CI, is subscribed to two RAs, namely the "Bed RA" and the "Stove RA" (the dashed arrow means notify subscriptors). The latter is subscribed to the CI and calls the RAs, namely the "TV RA" and the "Clock RA", directly using RPC (represented by the straight arrow). Internally to the CI a rule expression relates the RAs as "bed is occupied and stove is turned on for 30 min", which means that the CI only will be valid if this condition remains true during "30 min". When the CI changes its state from invalid to valid (what includes the timing), the subscriptors are notified, the "Reminder Actuator" in case. This actuator, in its turn, calls the OPs "fire alarm" from "Clock RA" and "set message" from "TV RA", with the parameter "You forgot the stove on!". Then it is expected that an accident is prevented.

An actuator can be simple as a list of actions to be invoked sequentially, the OPs of RAs in case such as the previous example, or it can be more complex, e.g. a program that send a SMS to some person in case of emergency, or that consults the whether from an website and shows on some visual interface at morning, among others.

[talk more about actuators]

[can have many actuators]

[timer can be CV]

[how to aggregate others standards such as rule engines]

Through the CI the framework address the challenge (ii), which is related to the amount and diversity of context informations and services, since the context interpretation allows the autonomous behaviour of RAs.

VI. SmartAndroid

[android and other techs]

[we have implemented most of the framework features, thought much more remain to be done]

A. Distributed Components Model

[implementation of RA and services]

B. Context Rules Parsing

[implementation of context rules]

C. Context Rules Running

[context rules conflicts is an ongoing study that aims at discover a conflict in runtime,]

D. Interface for Management and Prototyping

[description]

[main figure]

[rules composer]

VII. Application Examples

[smartlic]

[media follow me]

[prenda]

[review of capabilities]

VIII. Conclusion

[framework main features]

[smartandroid]

[future work] [more complex resource discovery] [aggregate other security concerns] [machine learning and data mining] [internet of things - expand to internet]

Acknowledgment

The authors would like to thank CNPq and FAPERJ for partial funding of this work.

References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 3, pp. 94–104, 1991. [Online]. Available: http://wiki.daimi.au.dk/pca/_files/weiser-orig.pdf

[2] R. de Araujo, "Computação ubíqua: Princípios, tecnologias e desafios," in *XXI Simpósio Brasileiro de Redes de Computadores*, vol. 8, 2003, pp. 11–13.

[3] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and V. Bahl, "An operating system for the home," *Proc. NSDI 2012*, 2012.

[4] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[5] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-computer interaction*, vol. 16, no. 2, pp. 97–166, 2001.

[6] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[7] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. Campbell, and M. Mickunas, "Olympus: A high-level programming model for pervasive computing environments," in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*. IEEE, 2005, pp. 7–16.