Journal of Applied Computing Research, 1(1):33-47
January-June 2011

Conflicts treatment for ubiquitous collective and context-aware applications[1]

Thais R.M. Braga Silva
Universidade Federal de Viçosa, Campus Florestal. Rodovia LMG, 818 – Km 6.
35690-000, Florestal, MG, Brasil. thais.braga@ufv.br

Linnyer B. Ruiz
Universidade Estadual de Maringá. Av. Colombo, 5790, Jd. Universitário.
87020-900, Maringá, PR, Brasil. linnyer@gmail.com

Antonio A.F. Loureiro
Universidade Federal de Minas Gerais. Av. Antônio Carlos, 6627 - Prédio do ICEx, Sala 4010.
Pampulha, 31270-010, Belo Horizonte, MG, Brasil. loureiro@dcc.ufmg.br

Abstract: Context-aware computing is a research field that defines systems capable of adapting their behavior according to any relevant information about entities (e.g., people, places and objects) of interest. The ubiquitous computing is closely related to the use of contexts, since it aims to provide personalized, transparent and on-demand services. Ubiquitous systems are frequently shared among multiple users, which may lead to conflicts that occur during adaptation actions due to individual profiles divergences and/or environment resources incompatibility. In such situations it is interesting to detect and solve those conflicts, considering what is better for the group but also being fair enough with each individual demand, whenever possible. This work presents the important concepts on the collective ubiquitous context-aware applications field. Furthermore, it proposes a novel methodology for conflicts detection and resolution that considers the trade-off between quality of services and resources consumption. A case study based on a collective tourist guide was implemented as a proof-of-study to the proposed methodology.

Key words: context and awareness in collaborative systems, ubiquitous computing, adaptive collaborative environments.

Introduction

The great advance observed in the last few years in the way how it is possible to connect the physical world to computational systems, getting data from the environment and its entities, has made possible the expansion of a research field called context-aware computing. Contexts can be defined as data related to entities (e.g., objects, people or environments) of interest to a certain application (Dey, 2001). The context-aware systems aim to perform physical or computational adaptations considering such data and, consequently, according to the needs and characteristics of their users (Baldauf *et al.*, 2007).

Contexts might be applied to different computational scenarios (Abowd *et al.*, 1999), as the applications that belong to a specific area called ubiquitous computing. Ubiquitous systems can be defined as computational solutions that can provide their services to the end

[1] This article is an extended version of "A conflict resolution methodology for collective ubiquitous context-aware applications", originally published in the *13th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2009)*, 2009, p. 426-461. DOI: 10.1109/CSCWD.2009.4968096.

users all the time and everywhere (Greenfield, 2006; Weiser, 1993). The ubiquitous computational devices are generally embedded into everyday objects, such as furniture, electronics, clothes and personal belongings, among others. Such devices can communicate among each other and with other systems (the Internet, for example) through wireless communication, using adequate standards and protocols. In general, these elements have severe computational restrictions, such as limited energy source, low processing and storage capacities and low data transmission rates. Ubiquitous applications must provide services to their users in a transparent way, and with the best quality of service possible, according to the amount of available resources. Intelligent environments capable of adapting their elements according to the needs of users can be seen as one of the classic examples of ubiquitous applications (Edwards and Grinter, 2001).

Ubiquitous computing is a field that has great affinity with the use of contextual data. The use of contexts allows the development of applications that are more adapted to the different situations of many users (Greenfield, 2006). Once this kind of system has the computational and communication capabilities integrated to the environment and used in a transparent and on-demand way by users, the use of contexts as inputs helps on improving the personalization and adaptation of tasks. Without context-awareness, ubiquitous computing becomes static systems, based on preprogrammed rules to the automatic execution of standard services (Weiser, 1993).

Many issues are yet to be solved for the context-aware computing, especially considering ubiquitous environments, to be used as broadly as it was initially proposed (Schilit *et al.*, 1994). One of them is the study of collective contexts conflicts resolution, which was selected to be studied by the current work.

Collective contexts can be defined as contextual data shared among two or more users belonging to the same context-aware application, comprised of a group of people that wants to perform the same set of tasks in a collaborative way. Examples of this kind of applications are tourist groups, shared smart environments (e.g., houses, cars, offices) and public presentations or conferences. In general, users in those scenarios have common goals. On the other hand, individuals may diverge on the desired adaptations due to differences on their individual profiles. In this way, conflicts can be detected and their resolution must be performed in a way that considers the group as a whole but also being fair enough with each individual, whenever possible. In particular, collective contexts occur frequently in ubiquitous context-aware applications since they are designed to operate into everyday environments, which are normally shared by many users. In this case, besides collective and individual efficiency, the conflict resolution methodology must be flexible, robust and resources consumption efficient.

The main goal of this work is to present issues and problems related to collective ubiquitous context-aware applications and then introduce a new and efficient methodology to be used by such applications to detect and solve collective conflicts. In particular, this methodology is capable of dynamically select which is the best conciliation algorithm, considering the amount of resources available (e.g., energy, memory and communication network) and the desired quality of service level. This adaptive behavior is adequate to ubiquitous systems, which frequently present modifications on their characteristics and configurations through their life-time.

The basic motivation to the development of this work is the existence of a great number of collective context-aware applications, especially those related to the ubiquitous computing area. Such applications need a computational support in order to deal with the occurrence of situations that are specific for systems that are shared among multiple users. Besides, to the best of our knowledge, there is no work in the literature that deals with this theme in a consistent way, providing modeling and resolution actions that are broad, adequate and with minimal efficiency requirements.

The rest of this work is organized as follows: the second section presents the main concepts related to the collective context theme as well as some related work found in literature. The third section presents our proposal of a new methodology to deal with collective contexts conflicts. The fourth section presents a collective tourist guide case study as a proof-of-concept to demonstrate the operation of the proposed methodology. Finally, the last section presents the final comments and some future work perspectives.

Collective and context-aware applications

There are many context-aware applications that demand actions to be performed for a group of users. Those are called collective

context-aware applications and the occurrence of them in ubiquitous systems generates even more interesting scenarios: typical collaboration problems that occur in collective applications are aggregated to challenges that are specific to the ubiquitous area, such as resource limitation and dynamic system configuration.

Defining Related Concepts: Collective applications are those context-aware scenarios that must consider the interests and contextual data from a group of users to adapt their tasks. The tasks of an application are the services offered by it to the end users, such as the tourist attractions in the case of a collective tourist guide, or the parameters adaptation actions of the domestic elements for the smart environments.

A collective context can be defined as data collected from the environment and from each group element, used by a collective application to perform its adaptations. Two types of collective contexts can be observed:

- The first one is the environmental collective contexts, which represent the physical and/or computational environment and its elements. They also represent the relationship among the users and the environment, characterizing situations such as property, sharing, permanent or temporary association, among others. These contexts and their values are shared among all involved users. Weather conditions, season and the use of environment elements are examples of this type of context.
- The second one is the personal contexts that reflect the current state of an individual, represented by a set of values. They reflect the characteristics, preferences and personal situations of a user. These contexts must also show the relationship between the user and the other elements of the group. Hunger, asleep and relatedness degree are some possible types of personal contexts. The personal contexts also reflect the characteristics of the devices (hardware, software and data communication) used by the users to perform a collective application.

Collective contexts are not the processing results or the fusion of many individual contextual data. Rather, they are the set obtained with those data and used by a collective context-aware application. The set of contextual data containing the level of sleep presented by each one of the users participating in a collective application is an example of a personal collective context called "Sleep".

The combination of many collective contexts types should be used to perform the adaptation of a collective application. While analyzing collective context input values presented by each user, as well as the current resources and characteristics of the environment, a collective application can reach an inconsistency state. It may be unable to decide what to do regarding the adaptations to be performed, in order to answer individual and collective demands at the same time. In this case, a collective conflict is said to be occurred.

Once the occurrence of a collective conflict is identified, the application must find ways, which might be either simple or sophisticated, to solve the inconsistency or deadlock. The execution of a technique or algorithm that allows answering in a smart and convenient way the differences on the collective contexts used as inputs to the execution of adaptations is called collective conflicts resolution or conciliation.

Related Work: The proposals found in Masthoff (2004) and McCarthy and Anagnost (2000) are related to the main goal of this work: to perform collective conflicts resolution. However, while those studies keep their focus on a single application based on a close set of contexts and specific software architectures, this paper has a broader and dynamic view of the theme and offers a new solution, capable of considering a range of implementations, configurations and applications.

Roy *et al.* (2006) present a solution to the development of a smart house that considers the activities and localization of multiple inhabitants. Its main contribution is to provide an environment that adapts itself to every participant without giving preference to any of them. As opposed to the Roy et al. proposal, this work presents a generic collective conflicts treatment solution, which can be implemented for any application and adapted to their current conditions.

Shin and Woo (2005) also discuss the adaptation problem for collaborative ubiquitous environments. They propose the assignment of priorities to each user and always apply the same static solution to solve a given problem. The collective conflicts treatment methodology proposed in this work is based on the provisioning of multiple resolution algorithms types selected according to the application and their characteristics.

Brézillon and Araujo (2005) have a study that presents the contexts sharing issues for collaborative systems. The authors discuss how the use of contexts is important for the collaboration among actors of a shared system, especially to facilitate the communication, interaction and sharing of knowledge. The main focus of the work is related to contexts representation, as well as its implications and opportunities to improve the collaborative work support.

Ardissono *et al.* (2002) present a Web system to prepare a tasks list to a heterogeneous tourist group. The system splits the initial group into sub-groups based on their ages, interests, visual capabilities and physical disabilities. Considering each sub-group's information, the system chooses the best tasks to them, presenting separated lists or a single and unified version. The system described in Ardissono's work is not a context-aware application since its goal is to help users to schedule the trip and not to guide them during the visits considering current contexts. The collective tourist guide application presented as a case study in this work interacts with smart environments, collects contexts and indicates the tasks to be performed at each moment according to them.

Collective conflict resolution: A dynamic approach

While developing a collective conflict treatment solution, a context-aware application life-cycle model proposal was elaborated. The following overview of this cycle allows a greater comprehension of the operational dynamics for the context-aware systems, the identification of the main related challenges and, specially, the perception of the points in which specific actions to collective applications must be performed.

Vieira *et al.* (2009) describe a development process proposal to aid designers in the definition of context-aware systems. The focus of Vieira *et al.* study is on the description of a systematic methodology to the development of context-aware software. Issues related to the selected hardware, data communication, details on the smart environment used and, specially, the aspects connected to the collective applications were not directly treated by the authors. Therefore, a particular model to this work, that considers the issues described above, was designed.

The model proposed by this work is a diagram comprised of phases and their respective activities. Each phase configures a period on the life-cycle of a context-aware application and each activity represents an important action to be performed at that period in order to the system to operate in a correct and complete way. Although proposed to collective applications, the model can also be used to the development of individual applications since all modules developed specifically to collective applications were encapsulated into a single block, called *Conflict Engine*, which is performed during a single activity. The model also considers that the applications use a context-aware software architecture (Baldauf *et al.*, 2007) to obtain contextual data (environmental and personal), as well as other necessary services such as contextual data processing (correlation or fusion, for example), security, among others.

The *Conflict Engine* is a framework responsible for processing the collective tasks. The suggested implementation of its components configures the new methodology designed to detect and solve collective conflicts.

Life-cycle Activity Diagram: the approach selected to the development of the diagram is based on the distribution of activities to be performed during three distinct phases of a context-aware application life-cycle: (i) Pre-application, (ii) Application and (iii) Post-application. Figure 1 illustrates the diagram.

The Pre-application phase deals with the preparation of all aspects needed to the correct and complete operation of a ubiquitous context-aware application with collective conflicts resolution. In this phase the following activities must be performed:

- Configure smart environments: the environments to be used must be instrumented with any mechanisms to offer the applications diverse contextual data, physical and computational resources and communication facilities.
- Configure hardware: every user needs hardware devices to interact with the application, which can be embedded sensors, wearable computing or even traditional handhelds or cell phones. Once there are many available devices' models and configurations, it is necessary not only to detail minimal operational requirements, but also to choose a strategy to deal with the potential observed heterogeneity.

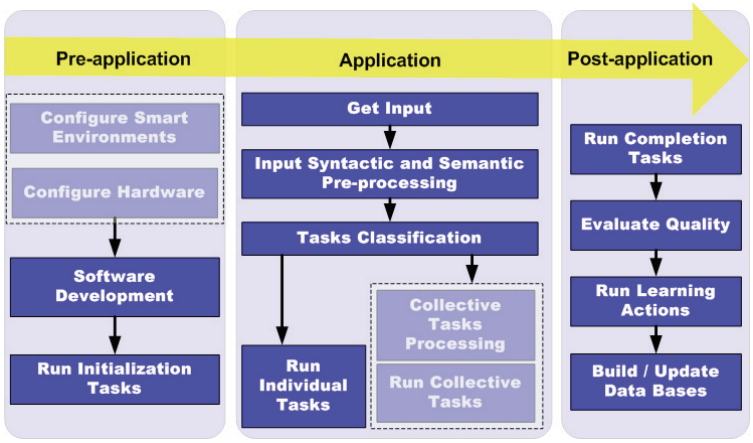

Figure 1. Ubiquitous and context-aware applications life-cycle diagram.

- Software development: includes the implementation of the application, all software architecture modules and the *Conflict Engine*, which is the software block responsible for the collective conflicts resolution. All software implementation details must be addressed during this activity.
- Run initialization tasks: activity that performs all the actions necessary to start the application, the software architecture modules and the *Conflict Engine* block. In some cases it might be necessary, for example, time or actions synchronization, ensure users views (e.g., environment, applications task, contexts), uniformity, among others.

The Application phase is the one in which users, carrying their devices correctly selected and configured, enter themselves into a myriad of smart environments previously instrumented, and perform, according to their needs, a series of tasks made available by the application. During the Application phase, users want to be served with tasks in the best way possible according to contextual data collected from them, the elements of the group and the surrounding environment. In this phase the following activities must be performed:

- Get input: the application acquires from the system architecture, the three basic necessary inputs, which are the personal contexts, environmental contexts and tasks list. It is the architecture responsibility to provide modules that are able to get that information.
- Input syntactic and semantics pre-processing: contextual data and application tasks must receive standard syntactic and semantic associations to improve their storage, exchange and final use. It is also necessary to associate contexts to tasks in order to collect and consider for possible adaptations only those data that are really important in the definition of current users and environment states. Other specific pre-processing tasks can also be executed during this activity.
- Tasks classification: tasks can be categorized into individual and collective ones. Individual tasks are ready to be offered to users since their execution do not have direct relationship with the contexts and tasks of other users. On the other hand, collective tasks can only be offered to users after being evaluated. Different techniques can be used in order to perform tasks classification, such as semantic analysis, associated contexts analysis and exchanging messages among users.
- Run collective tasks: the system will search for conflicts among the collective contexts presented by the participating users, considering the current state of physical and computational shared environments and the tasks set to be processed. If conflicts are not detected, the tasks set can be released and then executed by everyone. On the other hand, if the activity indicates a conflict, it must be solved at some action level before the tasks can be offered to the group. It is worth to highlight that the set of processed tasks may vary from only one task to all tasks made available by the application. This activity is performed only to the tasks that were previously classified as collective.

At some point in time, the context-aware application will be closed by the user or spontaneously once all the tasks have already been performed. At this moment begins the Post-application phase, which has some associated activities used to close the application, verify the quality of the performed actions and store data and information acquired during the application life-time. This phase, in particular, is application dependent, since each one may present specific needs.

It is worth to mention that the presented activities diagram is one possible approach to model the life-cycle of collective context-aware applications. The diagram was built based on the observation of the main activities performed to the development of context-aware applications on many studies found in literature. The model design was performed in a generic way that is enough to embrace the general needs of different applications classes. It can be extended or restricted according to the application's characteristics.

Conflict Engine - Detecting and Resolving Conflicts: A set of modules must be implemented together with the collective context-aware application in order to allow, whenever necessary, the execution of the collective tasks processing activity (activity that belongs to the Application phase of the collective context-aware application life-cycle). Such set can be developed as a software framework called *Conflict Engine*. Its components are described in the following, together with a discussion on the implementation possibilities for them. The organization of the framework modules, as illustrated by Figure 2 is a contribution of this work and was proposed to operate in conjunction with different context-aware software architecture and independently of the applications.

In order to solve collective conflicts it is necessary that the proposed conciliation solution modifies the initial application tasks set. In this case, it is important to choose on which tasks aspects changes will be performed in order to use the most appropriated resolution algorithm. Such aspects are called action levels and some examples of them are tasks parameters, schedule, grouping, composition, among others. It is worth to highlight that modifications over the tasks may lead to changes over the physical and/or computational environments. The Action Levels may be handled dynamically to the *Conflict Engine* through configuration files such as XML (eXtensible Markup Language) files or properties files. Such levels configure a key aspect of the proposed *Conflict Engine*, once they allow the use of the same strategy to solve collective conflicts for different applications, in which conflicts may occur under distinct circumstances.

The conflicts detection module performs a three-dimensional analysis whose axis are: involved users profiles, environment profile and application tasks. It can be programmed to use all data available through these dimensions to the analysis, as well as only a subset of them. The choice on how the analysis will be performed depends on the chosen action levels, which indicates how tasks can be processed during the search for conflicts occurrence. In case conflicts can occur due to differences of values on the users' profiles and/or the environment, an Action Level called "Parameters", for example, will be handled to the detection module. Previously knowing the semantics

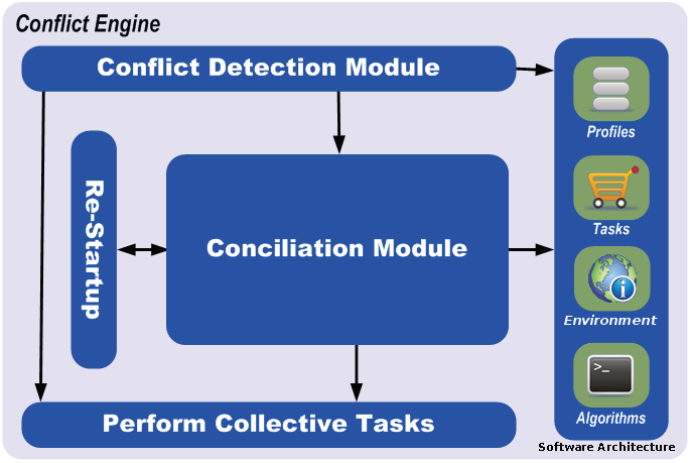

Figure 2. Conflict engine modules.

associated to the Action Level, the module will perform a comparative analysis among the contextual data from the profiles, searching for the occurrence or not of collective conflicts.

The Conciliation module receives as inputs all conflicting tasks, profiles of involved users, environment contextual data, conciliation action level areas, resolution algorithms and techniques available and any other necessary information stored in the software architecture database. This module must perform algorithms that solve the conflicts detected to a certain Action Level. In case the target applications are ubiquitous, the module must perform conciliation techniques considering thresholds for resource consumption and quality of services. As the result, adapted tasks are produced answering the collective interests, but also trying to consider as much as possible the individual demands and current contextual possibilities. Once more, in case the Action Level called "Parameters" was used, the conciliation module will act over the contextual data from the related profiles, modifying them according to what is possible, in a way to solve the associated collective conflict.

In some cases, as part of the conflicts verification process or during the execution of adjusts needed to adapt tasks that involve collective contexts, it might be necessary the execution of the Re-startup module, which contains actions that were already performed previously by the start-up Pre-application phase activity.

A collective conflicts resolution methodology: The conciliation module of the *Conflict Engine* can be implemented in several ways, using diverse algorithms and techniques capable of offering a solution to the conflicts detected. Some studies in literature have already approached this subject, offering specific solutions to certain situations, applications and contexts (see second section). However, since they represent particular approaches, they would not be easily applicable to scenarios that are different from the ones to which they were initially proposed.

This work, besides the collective context-aware applications life-cycle diagram proposal, as well as the software framework to collective tasks processing, also defines a novel implementation methodology to the conflicts conciliation module, dynamic enough to be applied to different applications according to their current situation. Such methodology was elaborated to collective, ubiquitous and context-aware applications and it is illustrated by Figure 3. It aims to guide the execution of the actions that are necessary to, in case of conflicts occurrence, automatically classify and select resolution algorithms that can perform the conflicts treatment inside some previously defined parameters to resources consumption, according to their availability, and quality of service levels, reflecting the users demands by the final satisfaction level towards the used resolution.

Energy, memory, processing and data communication network are examples of resources that should be used in a controlled way during the conflicts resolution for collective, ubiquitous and context-aware applications. The

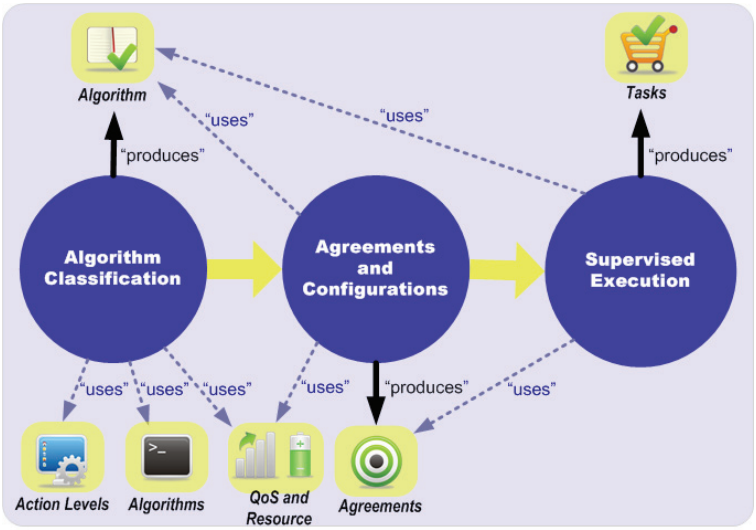

Figure 3. Novel conflicts resolution methodology.

quality of service for such applications should be measured by the efficiency obtained with the collective conflicts situation treatment, i.e., the individual and collective satisfaction levels presented by users, after the execution of the conciliation strategy.

Inside the conciliation module, implemented according to the proposed methodology, the execution of the following actions is defined: classification and selection of the algorithm to be performed, setting it according to resources consumption and quality levels agreements and the supervised execution. The proposed actions, as well as the elements used and produced by them are described in the following and illustrated by Figure 3.

Algorithm classification and selection: The classification and selection of algorithms can be implemented in a simple or sophisticated way. In order to perform this action, two important parameters are considered: the amount of resources available and the desired quality of services. The conciliation algorithm that is capable of acting over the configured action level, using the amount of resources available and providing a quality of service that is nearest to the desired one should be selected. Pre-programmed policies and machine learning techniques are some examples of interesting implementation options to the algorithms classification action.

Considering the possibilities for the algorithms to be used as a way to find a solution to the occurred conflicts due to applications sharing by a group of users, different options that have already been described in literature can be used. Such options diverge by the sophistication and complexity levels and are present in different areas, such as distributed systems, game theory, bio-inspired computing and optimization.

Agreements and configuration: Once selected, the conciliation algorithm must pass through a parameterization phase in order to adequate itself as much as possible to the application and its current characteristics, considering the desired conciliation results and the possible resources consumption. A set of service level agreements must be determined to the selected algorithm, considering which are the maximum and minimum thresholds to the resources consumption and provided quality of service levels.

Supervised execution: Finally, the execution of the conciliation strategy will be performed. However, this execution will be done supervised, and with the goal of not allowing surpass or violation of the limits stated to the service level agreements in any way. Actuation strategies or polices for the violation case can be implemented, such as allowing the selection of a new algorithm or to distribute grades to the algorithms according to their adequateness to the service level agreements, using them on the new iterations to the classification and selection.

It is worth to highlight that the proposed methodology is based on the selection of an algorithm among different available options. In this way, it is expected that an algorithms repository is provided, and that each option includes the meta-data that reflects information such as related Action Levels, average energy consumption, complexity, average services quality, among others.

The architecture model to be used, that is, the definition of the network places where the methodology actions must be performed will depend on the application characteristics. It can, therefore, be client-server, peer-to-peer or any other possible variation over these two basic models.

The methodology proposed in this work has an operation profile capable of adapting itself to different situation as an advantage over the other collective conflicts resolution options. It knows and respects the amount of available resources and tries to find the solution that will bring the best quality of service possible. This characteristic is very advantageous for ubiquitous systems, considering the resources restrictions and dynamic nature for the configuration of such scenarios.

Collective tourist guide: A case study

A collective tourist guide was implemented as a case study for the methodology proposed in this work. The selected scenario for this case study is a one-day tourist ride in one of the historic cities from the state of Minas Gerais, Brazil. In such city, tourists can perform tasks related to the region history, ecology, religiosity, gastronomy and shopping. In this scenario the execution of tasks can be highly affected by the collected environment contexts, as well as the profile of each individual user. Therefore, this collaborative application may be affected by the occurrence of conflicts, which must be identified and solved by the conciliation solution implemented in conjunction with the application.

A group of users share a tourist guide computational tool, and, although each of them has his/her own device, all users will only perform tasks together, as a group. Before executing any application task, each participant must perform a task indication, pointing out which would be his/her preferred task at that moment, according to personal contexts. The *Conflict Engine* will receive all indicated tasks and execute the verification and conflict resolution modules. The *Conflict Engine* was implemented in a centralized way. One of the users devices from the group previously selected should perform the server role.

Given the goal of testing the operation of the *Conflict Engine* and the proposed collective conflict resolution methodology, three different conflict conciliation algorithms were implemented: majority (M), random (R) and priorities (P). Table 1 presents the main characteristics of each algorithm, where *n* and *m* in the complexity row (row 1) refer to the number of users and the number of tasks, respectively. The table also presents for each algorithm, the amount of transmission messages needed to perform the collective conflicts treatment (row 2) characterized into levels (high, average and low). Once started, the *Conflict Engine* enters a training and warm-up phase to learn the average satisfaction percentage of each used algorithm (row 3) and to allow differentiations on the profiles of each application user. Each algorithm works as follows:

- Majority (M): The most indicated task is selected;
- Random (R): The task indicated by a randomly obtained user is selected;
- Priorities (P): The task that was most indicated by the users with high priority is selected.

To observe the methodology behavior, as well as to compare it with other possible collective conflicts resolution approaches, three scenarios were designed and simulated:

- *Conflict Engine (Eng)*: implements the new methodology, which dynamically selects the conciliation algorithm;
- *Resources (Res)*: selects always the algorithm with smallest energy consumption;
- *Satisfaction (Sat)*: selects always the algorithm with highest users' satisfaction percentage.

A collective context-aware application simulator was developed using the Java language and used to implement the collective tourist guide with collective conflicts resolution support. Each proposed scenario was configured to contain 50 users, mobile devices with 10 or 30 Joules batteries, 10 different tourist tasks and the 3 conflict resolution algorithms mentioned above. The scenarios were executed using the simulator 33 times each and the average results obtained are presented in the following.

The decision tree algorithm called J48 (Witten and Frank, 2005) was used to classify and select conciliation algorithms for collective conflicts. In order to obtain a collective conflicts treatment algorithm, it is necessary to provide to the J48 algorithm the residual energy, the users' profiles similarity level and the communication channels quality to the identification of a classification rule and, consequently, the indication of the most indicated option considering the circumstance. The decision tree algorithm used was developed in a way to privilege the selection of the Majority algorithm whenever the residual energy is high, the users' profiles similar and/or the communications channel quality low, the Priorities whenever the users' profiles are divergent, the residual energy with an average level and/or the communication channels in good state, and finally, the Random otherwise.

The conciliation algorithms provided to the case study do not allow the execution of parameterization after they are selected by the methodology. The service level agreements defined to the execution of the selected algorithm define that it has to provide the highest quality

Table 1. Conflict engine resolution algorithms.

	Majority	**Random**	**Priorities**
Complexity	*O(nm)*	*O(1)*	*O(n)+O(m)*
# messages	Low	Medium	High
Satisfaction	High (52.5%)	Low (40.0%)	Medium (45.0%)

of service possible, with resources consumption always under devices' current capacity. Agreements violation policies were not implemented in this study case.

Quality of service is measured by the percentage average satisfaction obtained by users regarding the final selection of tasks to be performed. A user is considered satisfied if, in case of a conflict, the task selected to be performed is the same one s/he has initially indicated. Therefore, by analyzing the satisfaction of a user on each task indication round, it is possible to calculate the percentage of times that s/he was considered satisfied.

A simplified model to the users' devices energy consumption was implemented on the simulator. The computation used to the consumption associated to each algorithm was performed based on their computational complexity. An empirical value proportional to low energy consumption was attributed to the Random algorithm, whose complexity is constant. The consumption for the other algorithms was computed based on these values and proportionally to their respective complexity. Whenever one of the conciliation algorithms was performed (on all simulated scenarios) its correspondent energy amount is consumed from the battery source.

Main achieved results: Table 2 shows how many users have indicated each available task for all application's rounds, considering one specific simulation from the 33 performed. One round indicates one simulation instance in which users must indicate and select one of the application's tasks to be performed. The analysis of this information is performed by the conflicts verification module as a way to indicate whether or not an indication of divergence has occurred. Whenever there are users indicating different tasks in a given round, a conflict is identified. Otherwise, as the table's last row shows, all users converge to the same task and, in this case, a conflict does not occur.

Since indications depend basically on users' profiles, each user is free to indicate any task. Once performed, a task becomes unavailable for indication in subsequent rounds. According to the previously described policies, the methodology selects one of the available algorithms to perform the conflict resolution in case it effectively occurs.

Considering specifically the Conflict Engine scenario, once detected the occurrence of a conflict, it is initiated a work to find an algorithm that is able to solve the generated impasse. Figures 4, 5 and 6 present the operation of the methodology during the execution of one simulation (Conflict Engine scenario). They show the parameters values used by the scenario to the selection of the algorithm to be used at each round during a specific simulation. The graph presented by Figure 4 shows the similarity among users profiles at each round, which is classified between very heterogeneous and very homogeneous. Such analysis is performed based on the tasks indications performed by users during the round. The more divergent the indications are the more heterogeneous is considered the group's profile. The graph presented by Figure 5 shows the quality of the communication channels during each round. A uniform probability distribution was used to generate this communication network quality profile.

Table 2. Number of indications per task at each round.

	Tasks										**Conflict**	**Performed Task**
	0	**1**	**2**	**3**	**4**	**3**	**6**	**7**	**8**	**9**		
Round 0	04	06	07	04	03	04	03	07	04	04	yes	7
Round 1	05	07	08	07	07	03	06	00	06	01	yes	2
Round 2	04	11	00	05	06	06	06	00	03	07	yes	1
Round 3	04	00	00	07	02	10	10	00	09	08	yes	3
Round 4	12	00	00	00	11	07	06	00	04	10	yes	5
Round 5	09	00	00	00	08	00	09	00	09	15	yes	9
Round 6	15	00	00	00	07	00	13	00	15	00	yes	6
Round 7	15	00	00	00	24	00	00	00	11	00	yes	8
Round 8	22	00	00	00	28	00	00	00	00	00	yes	4
Round 9	50	00	00	00	00	00	00	00	00	00	no	0

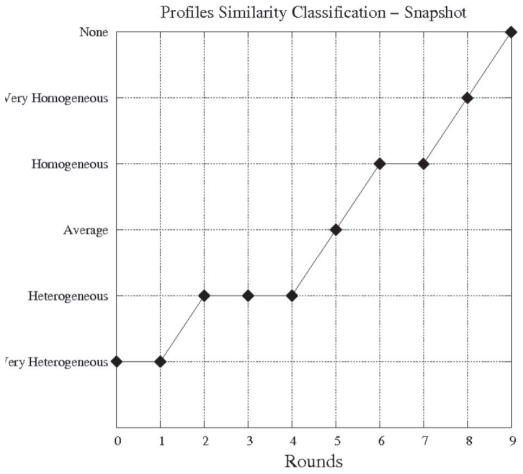

Figure 4. Users profiles similarity.

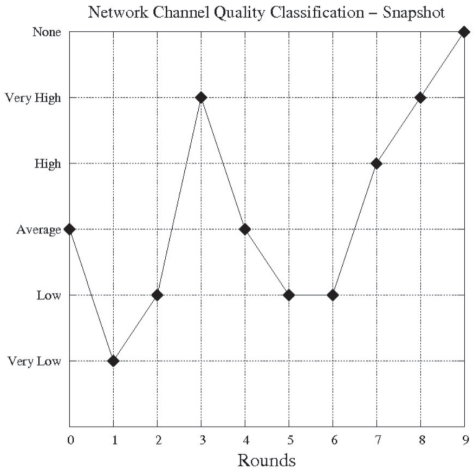

Figure 5. Network channels quality.

The graph presented by Figure 6 presents the percentage of residual energy of the end users' devices. This information is important to the execution of the Conflict Engine scenario once it allows the selection of algorithms that present a reasonable average satisfaction, without overconsumption of the scarce resources available. Considering all the information contained in the presented table and graphs, the conciliation algorithms are dynamically selected. The graph presented by Figure 7 shows all the choices made during each performed round. At each moment, the trade-off between resources and satisfaction is analyzed and a choice that contemplates in the best way possible both sides is made.

Consider for example the first indication round presented by Table 2. A conflict has occurred since users did not select the same task. In this case the methodology had to select a conflict resolution algorithm among the available ones. According to the previous explained rules, the majority algorithm was performed.

Observe the fifth indication round. Since users have indicated different tasks, once more a conflict was detected. The Random algorithm, which is the one with the smallest

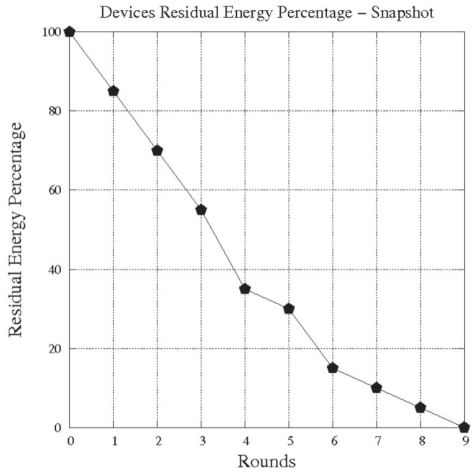

Figure 6. Residual energy.

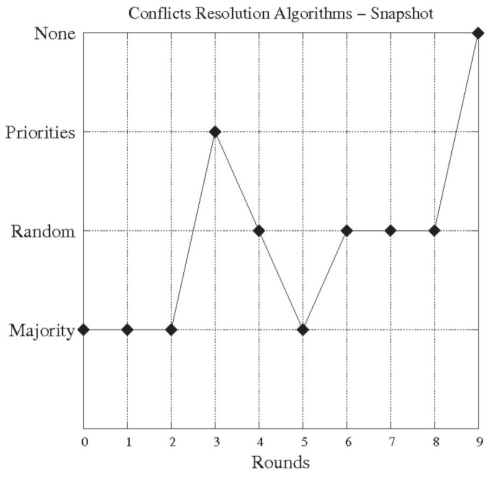

Figure 7. Selected algorithms.

associated complexity and therefore consumes less energy, was selected.

Tables 3 and 4 present a comparison among the three scenarios simulated in this work. They present the average users' satisfaction, average energy consumption and last performed round for each of them. The maximum value for the simulated mobile devices' battery capacity was varied between 10 (Table 3) and 30 (Table 4) Joules. Such values were empirically selected according to the operation of the simulation environment used in this work, in a way to allow the occurrence of two distinct situations: total and partial exhaustion of the used batteries. The goal of such variation is to observe the behavior of the simulated scenarios in front of resources scarceness, as well as with abundance of them.

The results presented show the adaptive behavior of the scenario that implements the proposed methodology. When the devices' batteries presented only 10 Joules, the Conflict Engine scenario behavior is closer to Resources' one. Even though trying to preserve resources from the moment that the batteries reach their lowest levels, the methodology keeps observing the users' satisfaction. Therefore, this scenario is able to reach, in average,

Table 3. Energy Consumption and users satisfaction averages (10 Joules).

	Engine	Resources	Satisfaction
Satisfaction	33.0	*28.0*	*35.0*
Energy	98.0	45.0	100.0
Round	8	9	6

Table 4. Energy consumption and users satisfaction averages (30 Joules).

	Engine	Resources	Satisfaction
Satisfaction	35.0	*30.0*	*36.0*
Energy	45.0	15.0	45.0
Round	9	9	9

the penultimate execution round, with a satisfaction level that is higher than the one presented by the Resource's scenario. It is worth to highlight that the Satisfaction scenario, although providing the best satisfaction average for the users, could only perform, in average, until the sixth round. When the devices' batteries have 30 Joules of energy, the Engine scenario tends to behave as that one that aims at maximizing the users' satisfaction. The energy consumption of both scenarios is identical, and the average satisfaction acquired very similar. However, the Resources scenario in this case, although it has consumed less energy amount, presented the lowest satisfaction rate to the users.

Finally, Figures 8 and 9 present, respectively, the energy consumption and users' average percentage satisfaction parameters for each round and during one simulation execution. The graph presented by Figure 8 shows that the energy consumption of the Engine Scenario is very close to the one presented by the Satisfaction scenario. However, the latter could only keep their devices alive until the sixth round, while the users' devices of the former still has around 20% of their batteries and can, therefore, be operational until the penultimate round. Although presenting smaller associated energy consumption, the graph presented by Figure 9 shows that the users' satisfaction of the resources scenario is always the worst one, once the algorithm with the lowest resources consumption presents also the lowest average satisfaction level. The Engine scenario oscillates between the other two scenarios' curves, providing a trade-off between satisfaction and consumption, according to the choices made for the algorithms to be used.

The joint analysis of both graphs show that the methodology implemented by the scenario Engine has accomplished the desired goals, once it has became closer to the best results of the other two simulated scenarios. It was possible to keep the operation of the scenario until the penultimate round, with an average users' satisfaction level very close to the maximum possible value, considering the algorithms that were available to use.

The acquired results, besides showing the correct operation of the Conflict Engine scenario, demonstrated that the goals for this computer supported collaborative work methodology have been reached. According to Table 3, although the Resources scenario presents the smallest energy consumption average and the Satisfaction scenario the highest average for users' satisfaction, the Conflict Engine scenario presents a satisfaction average very close to the Satisfaction one with smaller related average energy consumption. In other words, users felt satisfied with the tasks selected in conflict cases and still presented interesting average energy consumption. Although providing the best user satisfaction percentage, the users devices of the Satisfaction scenario only operates until the sixth round due to lack of energy.

Conclusion

This work has presented a discussion on a specific problem found in context-aware ubiquitous systems shared by a group of users: the occurrence of conflicts on the adaptation of services due to the incompatibility among the users' profiles and/or the shared environment. Concepts related to the theme were

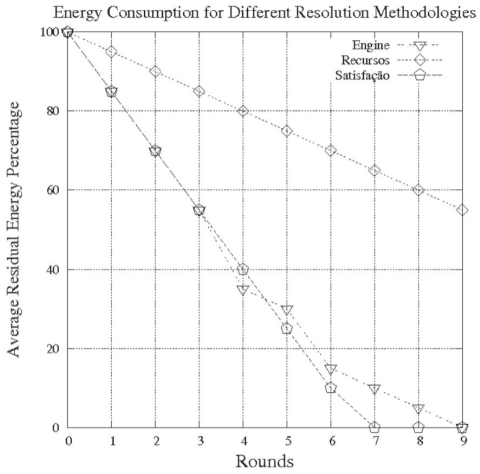

Figure 8. Energy consumption per scenario (one snapshot).

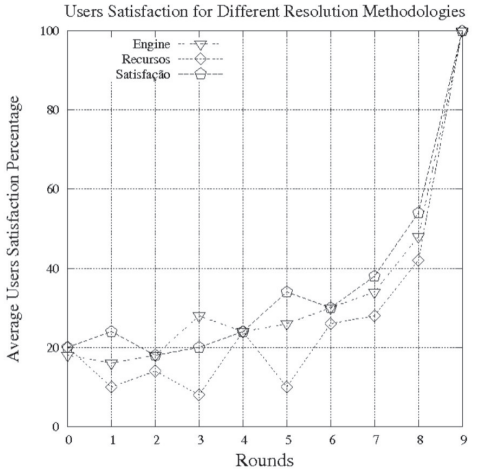

Figure 9. Users satisfaction per scenario (one snapshot).

formalized, and in order to solve this problem, this work presented an activity diagram to the ubiquitous context-aware applications life-cycle, a framework to the development of modules responsible for processing collective tasks, called *Conflict Engine*, and a methodology to perform techniques to the collective conflicts resolution.

A case study based on the implementation of a collective tourist guide was presented as a way to illustrate the collective applications concept and, therefore, to validate all the above propositions.

As future research directions to the presented work, we intent to implement other case studies as a way to evaluate the occurrence of a higher number of conflicts for different Action Levels and to implement and evaluate new algorithms to collective conflicts treatment.

References

ABOWD, G.D.; DEY, A.K.; BROWN, P.J.; DAVIES, N.; SMITH, M.; STEGGLES, P. 1999. Towards a better understanding of context and context-awareness. *In:* HUC'99 NTERNATIONAL SYMPOSIUM ON HANDHELD AND UBIQUITOUS COMPUTING, 1, London, 1999. *Proceedings...* Berlin, Springer-Verlag, p. 304-307. http://dx.doi.org/10.1007/3-540-48157-5_29

ARDISSONO, L.; GOY, A.; PETRONE, G.; SEGNAN, M.; TORASSO, P. 2002. Tailoring the recommendation of tourist information to heterogeneous user groups. *In:* S. REICH; M.M. TZAGARAKIS; P.M.E. de BRA, *Hypermedia: Openness, Structural Awareness, and Adaptivity.* Berlin, Springer-Verlag, **2266**:228-231.

http://dx.doi.org/10.1007/3-540-45844-1_26

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. 2007. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, **2**(4):263-277.

http://dx.doi.org/10.1504/IJAHUC.2007.014070

BRÉZILLON, P.; ARAUJO, R.M. 2005. Reinforcing shared context to improve collaboration. *Revue d'Intelligence Artificielle*, **19**(3):537-556.

http://dx.doi.org/10.3166/ria.19.537-556

DEY, A.K. 2001. Understanding and using context. *Personal Ubiquitous Computing*, **5**(1):4-7.

http://dx.doi.org/10.1007/s007790170019

EDWARDS, W.K.; GRINTER, R.E. 2001. At home with ubiquitous computing: Seven challenges. *In*: G.D. ABOWD; B. BRUMITT; S. SHAFER, *Ubicomp 2001: Ubiquitous Computing*. Berlin, Springer-Verlag, p. 256-272.

http://dx.doi.org/10.1007/3-540-45427-6_22

GREENFIELD, A. 2006. *Everyware: The dawning age of ubiquitous computing*. Berkeley, New Riders Press, 267 p.

MASTHOFF, J. 2004. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction*, **14**(1):37-85.

http://dx.doi.org/10.1023/B:USER.0000010138.79319.fd

MCCARTHY, J.E.; ANAGNOST, T.D. 2000. Musicfx: an arbiter of group preferences for computer supported collaborative workouts. *In:* ACM CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, New York, 2000. *Proceedings...* ACM, New York, p. 363-372.

http://dx.doi.org/10.1145/289444.289511

ROY, N.; ROY, A.; DAS, S.K. 2006. Context-aware resource management in multi-inhabitant smart homes: A nash h-learning based approach. In: ANNUAL IEEE INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS, 4, Washington, 2006. *Proceedings...* Washington, p. 148-158.

http://dx.doi.org/10.1109/PERCOM.2006.18

SCHILIT, B.; ADAMS, N.; WANT, R. 1994. Context-aware computing applications. *In:* IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, 1, Santa Cruz, 1994. *Proceedings...* Santa Cruz, p. 85-90.

http://dx.doi.org/10.1109/WMCSA.1994.16

SHIN, C.; WOO, W. 2005. Conflict resolution method utilizing context history for context-aware applications. *In:* INTERNATIONAL WORKSHOP ON EXPLOITING CONTEXT HISTORIES IN SMART ENVIRONMENTS (ECHISE'05), 1, Munich, 2005. *Proceedings...* Munich, p. 105-110.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. 2009. A process for the design of context-sensitive systems. *In:* INTERNATIONAL CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK IN DESIGN, 13, Santiago, 2009. *Proceedings...* Santiago, p. 143-148.

http://dx.doi.org/10.1109/CSCWD.2009.4968049

WEISER, M. 1993. Some computer science issues in ubiquitous computing. *Communications of the ACM*, **36**(7):75-84.

http://dx.doi.org/10.1145/159544.159617

WITTEN, I.H.; FRANK, E. 2005. *Data mining: practical machine learning tools and techniques.* Amsterdam/Boston, Morgan Kaufman, 525 p.

Submitted on September 30, 2010.

Accepted on October 10, 2010.