Journal of Web Engineering, Vol. 16, No.1&2 (2017) 103-125
© Rinton Press

CONSTRAINT-BASED CONTEXT MODELING AND MANAGEMENT FOR PERSONALIZED MOBILE SYSTEMS

JAWAD BERRI

College of Computer and Information Sciences
King Saud University, Riyadh, Saudi Arabia
jberri@ksu.edu.sa

Received May 30, 2016
Revised October 25, 2016

The capability of adapting to environmental changes and fulfilling specific needs of nomadic users empowers mobile devices with new value-added features. Users on the move are expecting real time and personalized services that are adjusted to their needs and that fit within their current time and space settings. Context-aware systems are distinguished by: i) their ability to profile users; ii) their awareness about device capabilities; and iii) their environmental knowledge. The availability of wireless networks supports context-aware systems through ubiquitous sensors and web services used to gather contextual information in order to offer users exceptional interactive experiences. In order to cope with information overload, collected data on the changing environmental context needs efficient management. In this research, we present a constraint-based context management system which handles efficiently complex situations in adopting a desired behaviour whenever a specific change occurs in the environment. This is accomplished through a set of knowledge-based rules which validate the consistency of the context by monitoring system constraints to trigger automatic context updates. We evaluate our dynamic context-consideration approach through real-life scenarios while comparing three consistency-validation strategies.

Key words: Context-aware system, consistency constraint, context model, reasoning engine, mobile system.
Communicated by: D. Schwabe & E. Mendes

1 Introduction

The development of mobile applications for nomadic users is witnessing an increasing interest following persistent demands from mobile users. Developers are racing to develop innovative applications to best serve users in their daily life [1]. While mobile users are keen to make use of these applications and services on the move, classical Web applications and services are moving to the mobile Web. This recent tendency is leading to mobile ecosystems for nomadic users that are autonomous and separate from the classical Web, supported by the increasing Internet mutation from wired to the wireless connectivity [2, 3].

Mobile applications are inherently ubiquitous. Services proposed to users could understand users' contextual needs while offering exceptional interaction experiences. Mobile users are expecting real time and personalized services that are adjusted to their needs and that fit within their current time and space settings. Context-aware systems are emerging as a viable capability to fulfil these needs for the following reasons: i) they know who the user is by accessing personal user's profile data stored in the mobile phone; ii) they are aware about the capabilities of the mobile device and are able to make use of mobile hardware resources or even to control some of them; and iii) they have knowledge about the environment since they are able to get ambient data from embedded sensors, and also to access a multitude set of web services. Two major components constitute the core of context-aware systems: Context Model and Reasoning Engine. The context model is a multifaceted data structure that represents the context within which mobile applications thrive. The reasoning engine is an intelligent module that is able to handle environmental changes and take adequate decisions with respect to the application and the user's activity.

Context-aware systems gather data on a regular basis and need to process it efficiently in order to cope with the dynamic environment of nomadic users [4]. For multi-user mobile applications which expect multi-users cooperation and data sharing, context accuracy and application performance represent two main challenges. For these applications, each user has a context instance running on his mobile device which needs to reflect his personalized interactions with the application, while at the same time accommodating the dynamic changes triggered by other users interacting with the same application. A context is accurate if it reflects the real world. A context can be incorrect due to the following factors and technical challenges: i) *Data accessibility*: application data is not available due to many reasons such as unavailable sensors, services or networks to update the context attribute-values; ii) *Synchronization*: the context is updated regularly but the changes in the real world are not reflected in real-time, resulting in an outdated context image that is recorded in the mobile application; iii) *Context state inconsistency*: the context state is inconsistent, where some attribute values are not within the expected domain, or are not coherent with other attribute values. The scope of this work is limited to context state inconsistency. In a multi-user mobile environment, application consistency and application performance are tightly coupled. Consistency of users' context instances is ensured if any change that occurs in a context instance is automatically relayed to all the users in real-time. However, this will generate huge communication overhead between mobile devices and the application resulting in poor application performance. A trade-off needs to be found in order to address both of the challenges mentioned above. Multi-user mobile applications developed for sensitive fields such as healthcare, finance, human rescue, etc., must be designed to guarantee consistency, as context data will be used for decision-making, while not compromising the application performance which may generate time delays resulting in an outdated information that may lead to harmful consequences.

In this research, we propose a constraint-based approach to handle efficiently multi-user context management. Constraints are defined to monitor the consistency of context instances and are implemented as rules which are triggered automatically within a strategy to update context attribute values. Activating systematically these rules for every change detected from mobile device sensors or the users' input can lead to highly complex algorithms resulting in poor performance of mobile applications. Monitoring context validity needs to be done within a strategy that integrates these rules and controls their activation to avoid unnecessary activations. These strategies should exploit domain information to organize the rules activation and prioritize the events which trigger context updates. The approach has been used to manage emergency surgeon operations in a hospital ward. For this application, three consistency validation strategies have been defined, implemented and evaluated using real-life scenarios. The results show that the strategy that exploits domain information leads to best trade-offs between response time and information consistency.

This paper is organized as follows: next section presents research works on context-aware systems and focuses on how these systems deal with the dynamic nature of contexts in a multi-user environment. Section 3 gives a formal description of the context model along with the attributes and consistency-constraints used in this work, and presents a comprehensive context modelling approach for mobile systems. Section 4 describes the context management and focuses on the function, the structure and the reasoning process of the reasoning engine. The following section presents the system design and an explanatory architecture of the implemented system. Section 6 describes the implementation of the system and presents a case study which illustrates how the system deals with events that occur dynamically during surgical operations in an actual hospital. Section 7 shows a system evaluation that has been done using a real-life scenarios to compare three consistency validation strategies defined for the hospital application. Finally, the last section concludes this work and proposes some future directions to extend this research.

2 Related Work

Context is a key element in many fields of science such as linguistics, history, archaeology, psychology and computer science [5, 6]. Context defines generally the conditions gathered from the surrounding environment of an entity or an object in order to understand it and use it correctly. The advances in information technology and the design of user-friendly mobile computing devices has led to the development of early location-based systems that take into account the current user location which can be acquired from GPS-embedded mobile devices [7]. An interesting development of context-aware systems has followed this first generation of location-based systems. Context-aware systems are defined as *using* "*context to provide relevant information and/or services to the user, where relevancy depends on the user's task*" [8]. This definition is general and encompasses a broad range of systems in the field of mobile computing. The development of context-aware systems has been enticed by recent technologies that access Web services anytime, anywhere. For this new generation of systems, the context represents all possible parameters that allow an application running on a mobile device to adapt autonomously to surrounding environments, while being used by a specific user. Context-aware systems are capable of providing high quality experiences to users in their daily tasks. A computing device which makes an efficient use of context has the ability to respond intelligently to the user's needs and offers a rich human-computer interaction experience. In the field of mobile computing, users are nomadic as they request different services by using their handheld devices while on the move. As context-aware systems are gaining momentum, several works have been dedicated to present surveys and comparisons about available context-aware systems [9, 12]. In this section, we discuss some of the prominent and recent works dealing with context consistency for mobile systems.

Yang et al. [10] propose a survey on dependable pervasive computing systems and a comparison based on defined factors that affect those systems. The problems addressed in this work are the plethora of contexts, their correctness and ability to detect faults. Another challenge addressed by the authors is the criticality of detecting and resolving context inconsistencies in real-time environments. The proposed techniques can be categorized in two groups: i) using predefined consistency constraints to check and detect context inconsistencies, and ii) defining strategies that can efficiently and effectively resolve these inconsistencies. The paper proposes a taxonomy scheme to classify works in the domain and compares between different context strategies to resolve inconsistencies.

Xu et al. [11] propose an approach named Partial Constraint Checking which aims at detecting context inconsistencies based on reusing previous checking-results to optimize constraint checking. The originality of the approach is to perform checking on changes that may affect only part of a constraint. The performance evaluation of the proposed approach is compared with the Entire Constraint Checking (ECC) technique where the whole constraint is checked. Experimentation done show that the proposed approach is adaptable to dynamic environments and is faster than ECC in tested scenarios.

Oh et al. [13] present context-aware software architecture for heterogeneous smart environments. The proposed architecture integrates contexts from multiple heterogeneous sensors and uses a reasoning module that is able to take decisions about collected context data. The architecture includes also a community manager module that manages communities. A community is a collection of entities (users, sensors and services) that share context information to achieve common goals. The community manager is able to predict common interests among users in a community and accordingly recommends appropriate services. The architecture time-performance and efficiency of context-delivery have been evaluated trough an experimentation. The results show that the decision time is slightly higher than the collection time while the architecture reduced unnecessary contexts, while integrating some of them to generate high-level contexts. The accuracy of received contexts shows that the architecture enables a near zero number of inappropriate context transmissions. Although context information can be shared by community members who have common goals, the consistency of contexts within the community is not addressed by the authors in this research.

Degeler and Lazovik [14] introduce diagrams named reduced context consistency diagrams (RCCD) that look like a semantic network allowing interpretation of context information that may hold conflict or ambiguity. They propose a mechanism for reasoning and inferring the most probable situations using RCCDs. The approach consists in defining interdependency rules which express constraints between data gathered by sensors. For instance, an interdependency rule states that if a user is sitting on a chair in front of a computer, then the sensor on the chair should detect pressure and the sensor on the computer screen to infer motions. An evaluation has been done with a real life scenario showing that RCCD has succeeded to reduce half of the erroneous readings representing conflictual sensory data.

Bu et al. [15] use a context ontology to model the context and propose to detect inconsistencies using an inconsistency resolution algorithm. The proposed approach relies on context-relative frequencies which are calculated and used to decide on which context to discard in case of context conflicts. The experimental evaluation shows that the proposed context-inconsistency resolution algorithm has succeeded to improve the application robustness by reducing more than half of incongruous behaviors.

Lu et al. [16] propose a data flow framework where context-inconsistency resolution is integrated within the application middleware which detects and resolves context inconsistencies. A set of testing criteria are also proposed as means to test applications. The empirical evaluation shows that the performance of the proposed approach to detect faults depends mainly on the used criteria.

In a previous work [17, 18], we presented a context-aware mobile learning system that adapts composition and delivery of educational material to the learner's context. As the user progresses in mobile learning sessions, he is exposed to scenarios that are continuously re-arranged, packaged and conveyed to fit his actual his progress in the learning process. In [4], we further showed how context can be used efficiently to retrieve information by a mobile search engine. Exploiting context information in a mobile search engine helps fine-tuning Web services selection enabling applications to deliver custom-made information to end users. In the present work, we introduce a formal approach for context modelling and representation. We propose also consistency constraints and strategies to be part of the context model, and show how these are implemented within the context management cycle. Then, we unveil an evaluation of the system with a real-life scenario.

While context management is the core processing of any context-aware system as it must keep the context updated; maintaining a consistent context is an essential part of context management [14, 15, 16]. Data gathering from mobile sensors is done continuously, as long as mobile applications are running, but data needs to be aggregated, interpreted and checked for consistency validation, otherwise mobile system applications will deliver unreliable information and services that are not appropriate for users' needs. Another important factor is context management efficiency, whereby obtained data from sensors and Web services update the current context but may generate latency with some serious impacts on system efficiency. Context management task needs to be designed carefully, taking into account domain information which may lead to substantial system's response time enhancements [10, 11, 13]. In the present work, we focus on context management for mobile systems as a cycle that must incorporate context consistency-validation checks. The latter is designed as a set of rules that check consistency of information processed from gathered data. System efficiency is also addressed through the introduction of three consistency checking strategies. The experimental study shows that the strategy which makes use of domain information leads to an improved response time.

3 Context Model

Context for mobile systems is a representation of the environment involving a user requesting a service from an application running on a mobile device.

3.1 Formal Model

In order to describe formally the context, we use first-order logic including classical connectives and quantifiers [19].

Definition 1: *Context*

A Context C is a triplet $C = <S, T, R>$ where

- S is the context schema denoted by $S(A_1, A_2, ..., A_m)$ that is made up of a context name S and a list of attributes $A_1, A_2, ..., A_m$;
- $T = \{t_1, t_2, …, t_p\}$ is a set of constraints on attributes A_j;
- R is a set of rules which associate actions affecting the values of context attributes A_j under certain constraints t_k.

This definition incorporates in the model context constraints that are unavoidable forcing context-aware systems to continuously satisfy their inherent constraints in order to remain coherent.

Definition 2: *Context State*

A Context State s of the context schema $S(A_1, A_2, ..., A_m)$, denoted by $s(S)$ is a set of n-context instances $s = \{u_1, u_2, ..., u_n\}$. Each context instance u_i is an ordered list of n values $u_i = <v_1, v_2, ..., v_m>$, where $u_i.A_j = v_j$. In this statement, each value v_j, $1 \leq j \leq m$, is an element of dom(A_j) – the domain of A_j, represents all the possible atomic values of attribute A_j. For instance, if A_j is the attribute *gender* then dom(*gender*) = {*male, female*}.

The context state s is a two dimensional vector of values $s(u_i, A_j)$ where the rows represent the context instances u_i of the application users, and the columns are the attributes A_j for a specific mobile application context.

Property 2.1: *Context State Domain*

The context state domain of $s(S)$ represents all the possible context instances. It is defined as a subset of the Cartesian Product of the domains that define S:

$$s(S) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_m)).$$

Property 2.2: *Invalid State*

A context state $s(S)$ is invalid if some constraints t_k are not satisfied:

Given a context state $s(S)$, $\exists\, k \;\; 1 \leq k \leq p, \neg t_k \Rightarrow s(S)$ is invalid.

Accordingly, a Valid State is a context state where all the constraints are satisfied.

Definition 3: Attribute Constraint

An Attribute Constraint t^a is a type of constraint requiring that for any context state s, the values v_j of attribute A_j must be within the domain of A_j

$$t^a(A_j): v_j \in \text{dom}(A_i)$$

Definition 4: Consistency Constraint

Given a context schema $S(A_1, ..., A_l, A_{l+1}, ..., A_m)$ and a context state $s(S)$.

A Consistency Constraint t^c requires that the values v_j of a subset of attributes A_j must be within a subset D_j of the domain dom(A_j).

$$t^c(A_j){:}\ 1 \leq j \leq l,\ v_j \in D_j,\ D_j \subset \text{dom}(A_j),\ s(S) \subseteq (D_1 \times D_2 \times ... \times D_l \times \text{dom}(A_{l+1}) \times ... \times \text{dom}(A_m))$$

Consistency Constraints are more restrictive than Attribute Constraints, they are suitable whenever a fine-grain context-state check needs to be made on only a subset of domain attribute values to allow specific updates of the context state. For example, the identification of the user's current task can be made through checking the place where he is located currently. Therefore, if we are in an educational environment with rooms dedicated for teaching, others for meeting and so on. If a teacher is located in a classroom dedicated for teaching, then he is most likely involved in a 'teaching' task. In this case, a consistency constraint will express the fact that 'teaching' task can happen only in the set of rooms dedicated for teaching. Accordingly, an association rule will check the validity of the consistency constraint and will trigger an action in case of constraint violation. Also, these constraints are paired with an activation strategy, as shown later in the evaluation as part of Section 7, they allow context-aware systems to save time and gain in efficiency. Indeed, consistency constraints are able to monitor the context validity on an event basis by mean of rules without requiring frequent updates from the system sensors which is a real burden on mobile systems' efficiency.

3.2 Context Schema Description

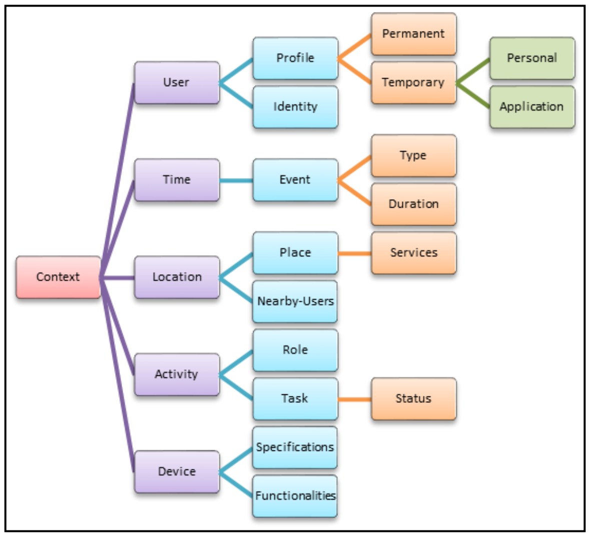

Figure 1 Context Schema Representation.

Lots of studies have provided descriptions of context from different perspectives, [20, 21, 22] and consider context to be the user's environment, while [23, 24] consider it to be the application's environment. In this work, we adopt the description provided by Dey et al. in [22] where "*context is typically the location, identity and state of people, groups and computational and physical objects*". The context schema shown in Figure 1 includes attributes that are necessary to represent the complexity of real life situations for mobile applications which are categorized in five main categories namely: user, time, location, activity and device. Each of these categories is further detailed at different levels resulting in a finer grain representation of context. In this context schema, we consider the user's device as part of the context as well to allow an application checking the functionalities and capabilities of the mobile device. Indeed, some services, applications, and communication facilities may or may not be supported by devices' specification and operating system.

3.2.1 User

User is a system representation of the user. This context attribute includes the user's profile and identity. It has been found that this distinction between the user's profile and his identity promotes the context model to be general, which makes it suitable to deal with various applications [4].

Profile: The user profile embraces general information about the user. User related information can be of two types: i) *permanent information that* does not change in time such as personal information including: name, gender, date and place of birth, mother tongue, and other personal traits like biometric data; ii) *temporary information* on the other hand is subject to change in time. This element consists of personal and application-related information. *Personal information* includes but is not limited to the following data: preferred language, occupation, interests, preferred currency (for currency conversion), and hometown. *Application related information* refers to personal information that is related to the nature of the application such as the user's preferences (interface options, types of notifications, …), skills including for instance the user's expertise with regard to the application (beginner, intermediate, expert), history which stores a record of past interactions (such as past requests or accessed websites), and progress representing the advancement of the user (for applications consisting of episodes such as learning sessions in m-learning).

Identity: The user identity is to be distinguished from his profile. It defines specific rights for a user to access data or to use some services. A user may have different identities but his profile is unique. He may be involved as a member in different groups with a different identity for each. For instance, a lecturer at the university has different identities: he is a teacher for a group of students; he is also a member in the department council, and he is a researcher, member of a research group.

While the user's profile holds basic information about a user, the identity is the complement for specific applications and services. The user's identity is interesting as it holds more personalized information resulting in an adapted and fine-tuned interaction of the proposed services to the user needs.

3.2.2 Time

Information related to time is relevant to time-stamped events, which allows an efficient context management and delivery of content with reference to the current time and date. For instance, a mobile application should be sensitive to restaurant opening time while receiving a user request about nearby restaurants. Based on the request time and the restaurant opening time, the application should propose restaurants that are opened for sufficient time to allow the user getting there, order and enjoy his meal. Time is used to define Event – a context element that characterizes current and upcoming events. Scheduled events of interest for the user trigger alerts on the user's mobile. Event represents all activities limited in time which might interest the user. For instance, a tourist might be interested in events such as exhibitions, expositions, trade fairs, shows, and so on. Event information consists of elements to identify the event such as: event name, event type, event location, and start date/time and duration. Event related information is obtained from specific web services maintaining a schedule of different types of events. Event has two major data sub-elements: Type for the event type (business, education, leisure, and art) and Duration which is calculated according to the time and schedule of the event. Information related to events in a given city worldwide can be retrieved and updated automatically from specific services available on the web [25].

3.2.3 Location

The user's location correlates the user's interests with services that are available nearby the current user's location. The location is an important attribute for location-based systems to offer personalized mobile services. For instance, a mobile tourist advisor system should notify the user about interesting sightseeing nearby his location [25]. In mobile applications, the user's current location is retrieved from the built-in GPS (Global Positioning System) receiver or through a connection to an external GPS receiver. The system uses location to process the user's current place (home, workplace, park, and school). *Place* is a data element that allows the context manager to adapt services to the current user’s place. For instance, information requests from an employee working for a multisite corporation should return in priority information related to the current employee site. *Place* defines also the available services nearby the user. All services that are accessible from the user's current location are stored into a specific data element called *Services*.

Another data element related to the location is the context attribute *Nearby-Users* representing users located nearby the current user with which he can have direct interactions. This attribute is very useful for user groups sharing common interests or collaborating to achieve a common task.

3.2.4 Activity

Activity represents the current activity the user is involved in. The current user's activity is input by the user or derived from the handheld task manager. This context attribute allows the system architecture to be aware of the current user’s task and accordingly enhance semantically the interaction between the user and the system in one hand and the user and the user community in the other hand. The activity allows also the system to monitor and manage user tasks towards an efficient use of the mobile resources. *Role* is a data element that might be very helpful in some activites involving many users. Although users are involved in the same activity, they might have different roles. For instance, a surgical operation is an activity involving many users with different roles such as surgeon, assistant, patient, nurse, technologist, and anesthesiologist. An activity might be complex requiring fine grain details and hence needs to be represented as a set of tasks. Accordingly, *Task* stores the current task description the user is involved in. For instance, a student might be solving an exercise, reading the problem or listening to the tutor instructions. These tasks are part of the learning activity.

While a user has been assigned a task, he might have different types of involvements such as reserving resources for the task, performing the task, having a break or finished but not yet released. The data element *Status* represents the involvement of the user within a task to allow an efficient management of tasks assignment and to define the user availability for different services such as alerts or pop up calls.

3.2.5 Device

This context element is necessary since users use mobile devices that have different specifications and functionalities. Information displayed on the user’s mobile need to suit the device technical constraints. It includes a set of attributes about the device type and its capabilities which are classified in two classes. *Specifications*: represents the hardware specifications of the mobile device. It includes the mobile model, supported browsers, multimedia support, screen size, navigation tools, bandwidth limitations, multi-language capabilities, cookies support, caching support, memory size, and processing power capabilities. *Functionalities*: is a data element which stores all the functionalities the mobile device is equipped with such as additional applications that may be important to perform user tasks.

Context model expansion should be included as a core requirement in the development of context-aware systems to allow flexible mobile applications design and implementation. The context model presented in this section is extensible as application requirements may differ necessitating additional data elements during the development life cycle. Complying with this constraint allows low coupling among the context model and the reasoning engine and frees the developer from the frequent revisions of the application code. It will also allow the user to add any data element he feels is necessary to include while using the application.

4 Context Management

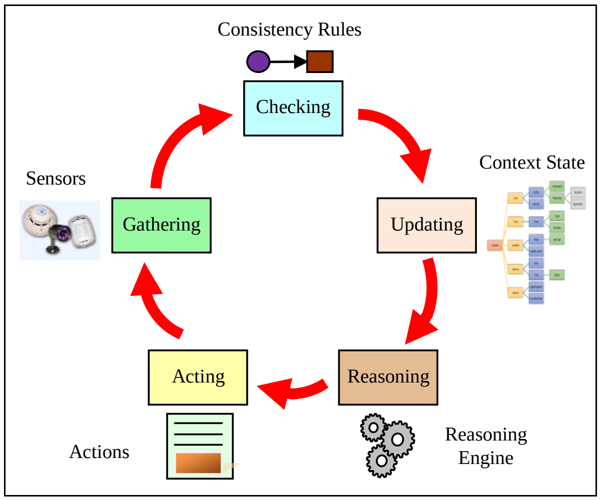

Figure 2 Context management cycle.

Context management is a cornerstone constituent for a mobile application. This module’s role is to reflect the changes occurring in the real world on the services provided by the mobile application. Context management is a process cycle including five steps as shown in Figure 2: Gathering, Checking, Updating, Reasoning and Acting. This cycle is triggered by the user whenever a mobile application is initiated. Context management needs to be done in real-time to allow a seamless interaction between the user and the mobile application. One of the main factors that can affect the performance of the system is the systematic application of consistency rules. Although these rules need to check the coherence of context instances at each context change, the propagation of updates in multi-user applications may trigger an exponential activation of rules which may affect negatively the performance of the system during the Checking, Reasoning and Acting steps. For this reason, the activation of these rules must be done within a strategy that takes into account domain information to avoid unnecessary activations.

4.1 Gathering

Gathering is the first activated step in the cycle which feeds the application with the necessary data from the real world. This step consists of two tasks: gathering static context data and gathering dynamic context data. Static context data are gathered only once at the launch of the mobile application; it does not change during a session. This includes data related to the user and the device.

Dynamic context data needs to be periodically gathered and processed during a session as it will affect the user's activity and his interaction with the device. Dynamic context data are context elements that are related to time, location and activity. Gathering data for context management is a task that relies on embedded sensors in the mobile device or present in the surrounding environment that feed the context manager with the necessary data. Also, some data is gathered by invoking web services to provide some context related data. The frequency of gathering dynamic data depends mainly on the nature of data and the needs of the application. It is worth mentioning that a high frequency update clogs the network and drops down the application performance but provides a real time interaction experience which is a requirement for some sensitive applications such as patient monitoring in the medical field.

4.2 Checking

Checking is a necessary step in the context management cycle to ensure that data gathered is consistent with existing data in the context state. This verification is done for dynamic data gathered periodically. Consistency rules are periodically invoked to check any change in the context state to ensure that data gathered does not affect the user's activity. For instance, if the user is involved in an event scheduled between 8:00 AM and 5:00 PM, then after 5:00 PM consistency rules spot the change in time and accordingly update the attribute *Event*. Another example is about updating the user's current place. While on the move, an application collects continuously GPS coordinates and processes the user's current location. Any change showing that the user has moved away from a place while performing a task may signify a possible change in the user's activity.

4.3 Updating

During this step, all the context elements are processed and the context state is updated. More specifically, the attribute *Event* is updated with the events at the current location and time. This is done by accessing a specific web service that provides information about organized events nearby the user's location. Also the services available currently are retrieved and the element *Services* is updated. Users nearby the current location are gathered and the context element *Nearby Users* is also updated.

4.4 Reasoning

Reasoning is about using current context data to propose a customized experience to users by adapting information and services to their needs. Based on the context state, which has been updated in the previous step, Reasoning Engine infers high-level context information that is used by an application to provide a level interaction that meets the users' preferences and needs.

User's initial requests are contextualized to match the current context. This is done by the reasoning engine which aggregates the user's request with the current context information. When a user's request is contextualized, it is bound to the current context, hence information retrieved and services proposed satisfy in priority the actual user's needs in terms of place, event and task. For instance, if the user requests information about "turbine reports" while performing an inspection task on a particular company's plant, then reports on turbines of the current plant nearby the current location are retrieved in priority.

Accurate reasoning presupposes that ontologies are available. Indeed, an ontology specifying the user's task and sub-tasks permits a better understanding of the user's current involvement and allows the application to adjust to fine-grained tasks at hand. For instance, a university teacher can be lecturing in a classroom, advising students in his office or meeting with peers in the meeting room. Each of these sub-tasks requires particular behaviors of the application. Moreover, an ontology of tasks allows a better management of events handling during the current task such as phone calls that may need to be put on hold or to notify the caller that the user is busy (such as *will call you later*) and other possible notifications. In addition, the use of domain ontology might be necessary for some applications requiring a representation of the workplace environment.

The reasoning engine allows also inferring high level information from the current and past user's interaction. Making use of the GPS data, the reasoning engine can infer whether the user is walking or moving with a vehicle. The user's current task can also be known from its current location. A teacher located in a classroom is most likely involved in a *teaching* task, while he is *meeting* in the meeting room. Also, user's acquaintances can be known from historical interaction traits. Acquaintances represent the users with whom the user has frequent interactions.

4.5 Acting

Actions decided by the reasoning engine are executed during this step. For some actions, user's approval is necessary to have a seamless interaction with the user. Indeed, the user should keep full control of major decisions that may affect directly his duties. An example of such case is a change in the current activity's location or event; the user should be notified and should express his consent. Indeed, in case a change in the location requiring a change in the current task is processed by the reasoning engine, then a confirmation is needed from the user.

The graphical user interface is updated during this step to display information requested by the user and any information inferred by the reasoning engine. Also recommendations for the user to deal with the task at hand and to manage his interactions with upcoming events are executed. Examples of upcoming events are related to requests from other users requiring a reply from the user. Actions' post-conditions feed the next cycle's step *Gathering* to update the context state.

5 System Design and Architecture

Context management as shown in the previous section constitute the heart of the system architecture (Fig. 3) which includes two other components: a knowledge module, and the external world.

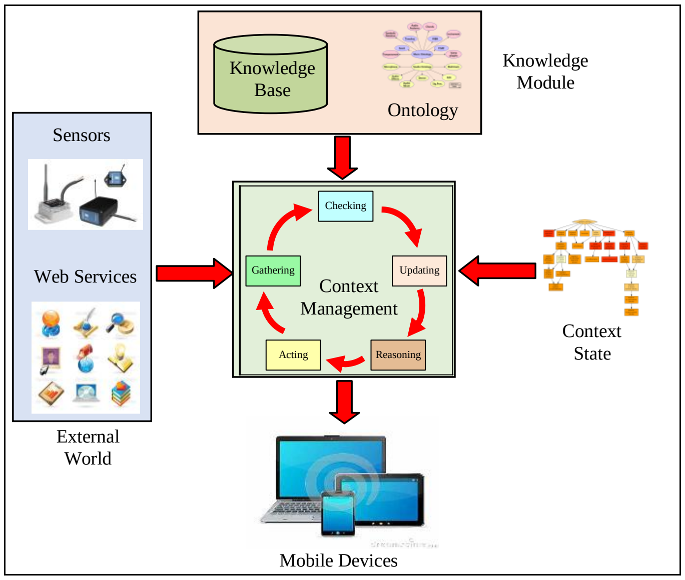

Figure 3 System Architecture.

5.1 The Knowledge Module

The Knowledge Module aims to organize logically the system rules and actions allowing the reasoning engine to make the suitable inferences for a given context. This module includes the knowledge base and the ontology.

5.1.1 The knowledge base

The knowledge base contains three types of rules that allow taking decisions regarding a given context [26]: Consistency rules, Updating rules and Action rules.

Consistency Rules. Consistency rules implement consistency constraints; they verify continuously the coherence of the context at each cycle when data is gathered. These rules make sure that a context update never leaves the context state inconsistent. For instance, a surgeon operates inside an operating room. Accordingly, when he is involved in a surgery task, his location should be within the available operating rooms. If this is not the case, then there is an inconsistency, in which case the system should take the suitable action.

Updating Rules. Updating rules consists in matching a context state and update it with new data. These rules are applied after consistency checking has been done by consistency rules. For instance, a lecturer involved in weekly lectures should be located inside a lecture room at lecture time. If it is the case, then his task is updated to “lecture”.

Action Rules. Action rules decide on the suitable action to take on behalf of the user. For example, when the surgeon is involved into a critical task for which he should not be disturbed, all incoming calls should be handled by the system either by putting them on hold or by sending a notification to the caller based on his identity.

5.1.2 The Domain Ontology

Domain ontologies are generally necessary to reason accurately and infer fine grained facts. In order to allow the system to handle accurately the current task in which the user is involved, an ontology specifying the user activity, relations between different activities and the tasks of each one is necessary. The Activity Ontology is frequently used in domains where the specification of complex activities permits a better handling of the current task. For instance, a bank teller involved in the activity of paying a check to a customer is involved in the following tasks: customer authentication, check verification, money preparation, and customer signature matching. Moreover, some applications require using a Job Ontology specifying jobs and how they are organized to handle the roles of users within their organization. It will enhance also the management of different events that may popup during the task execution such as phone calls or other popup assignments. Inferred information is then displayed on the user's mobile device taking into account the device specifications

5.2 External World

The External World is the place where the user activity occurs. The Real World includes the user's place (home, school, workplace, hospital, shopping mall, …), the mobile device and the environment surrounding them. It is characterized by a set of data elements that define the values of properties under which the interaction occurs. It includes the mobile device's location which is supposed to be the user's location and the surrounding of the user that affects the interaction including available services and nearby users that may interact with the current user.

6 System Implementation

The system has been implemented as a knowledge-based system offering many advantages to represent context and to perform efficient reasoning. In this section, we present the implementation of the system and a related real-life healthcare emergency case study.

6.1 JESS Environment

The implementation of the reasoning engine has been done with JESS the Java Expert System Shell [27]. Jess is a knowledge-based system which mainly uses facts, rules and templates to represent knowledge. Jess has a syntax that is similar to LISP [28] which is symbolic and declarative allowing a compact expression of knowledge and programs. Using the declarative programming paradigm in developing our system is the natural way to express knowledge and to deal with context management. This paradigm allows focusing on expressing the context and the management of events in the form of templates, actions and pattern matching; the way knowledge is processed is handled by Jess's inference engine. The latter is implemented in Java [29] benefiting from all the qualities of the Java language and more specifically platform independence feature which reliefs the developer from the tricky integration problems as the reasoning engine is meant to be embedded into diverse mobile platforms. Besides, Jess can embed pure Java code within rules and functions which opens wide possibilities in programming web applications.

6.2 Case Study

One of the sensitive fields where context aware applications might be very helpful concerns managing medical staff daily activities. Managing the work of doctors and other medical personnel is vital to provide patients with prompt care and quality medical service. The following is a real life case that is used as a scenario to illustrate the reasoning process described in Section 5.

Dr. Kamel is a surgeon specialist at Oran University Hospital (OUH)[1], a public hospital in the city of Oran, the second largest city located in the west of Algeria. Strong of his 26-years practical experience, Kamel has many duties. He is a full-time chief surgeon of the Emergency Unit at OUH which serves a wide population of more than three million. Emergency cases of Oran region are first treated at local hospitals but if the case is judged critical, it is transferred to OUH. Kamel's main duties at OUH are to perform surgery operations for emergency cases that pop up generally with no prior notification to the hospital. Once the surgery completes a treatment, patients are followed-up in the hospital under his supervision. Kamel leads a team composed of surgeon doctors, residents in surgery and nurses who are assigned specific medical tasks at the department such as preparing the surgery bloc and patient medical-care assistance during shifts.

6.3 Context Model Implementation

The context model has been implemented as a template including all the context attributes specified in Figure 1. In order to fit with the template construct requirements provided by Jess, the tree representation of the context is flattened by mean of a depth first search traversal of the context tree. The context template is defined by *deftemplate* construct (Fig. 4) that works like a class definition; it includes slots representing context elements and specified by the *slot* keyword. An instance of the context class is specified by the *assert* command that creates a fact which can be seen as an object-like instance of the template. In our case the newly introduced instance correspond to Dr. Kamel.

```
(deftemplate context
;; User profile
  (slot id) (slot fname) (slot phone) (slot position)
  (slot identity)
;; Event
  (slot event) (slot type) (slot duration)
;; User location
  (slot place) (multislot nearby_users)
  (multislot services)
;; User activity
  (slot role)
  (slot task) (slot status)
;; Device
  (slot model)
  (…))
```

```
(assert (context
  (id cc130764) (fname kamel) (phone 055757689)
  (position chief-surgeon) (identity nil)
  (event nil) (type nil) (duration nil)
```

[1] We would like to thank Dr. Kamel Anouar, his medical team and the Emergency Unit at Oran University Hospital for their support and cooperation.

```
  (place nil) (nearby_users nil) (services nil)
  (role chief-surgeon) (task nil) (status nil)
  (model samsung-s3)
  (…)))
```

Figure 4 A fragment of the context model implementation.

Jess *deftemplate* construct is flexible allowing context model expansion with no impact on facts and rules. For facts, slot values can be omitted while creating instances of the context. For rules, pattern matching is done only for slots specified in the right hand side of rules (conditions) and is bypassed for any slot of the context template that is not mentioned in the rule.

6.4 Handling an Emergency

Critical cases coming to the emergency need to be handled urgently. The criticality of each case is evaluated by the doctor on duty who rates it by assigning a number between 1 and 10 representing the severity of the case. Cases above 6 are very critical and necessitate the intervention of a specialist surgeon. Accordingly, a medical team and an available operating room are assigned to the case. Also a notification is sent to each member of the medical team on his mobile phone which includes the severity rate, operating room number and a short description of the case.

```
(deftemplate newcase       (assert (newcase
  (slot patientid)           (patientid 5689612)
  (slot name)                (name "Wassim Ali")
  (slot severity)            (severity 8)
  (slot team)                (team "Surgery - Dr. Kamel")
  (slot room)                (room OP2)
  (slot description)         (description "Abdominal Trauma")
  (slot date)                (date 18/6/2013)
  (slot time))               (time 10:32)))
```

Figure 5 New case template and instance.

Today, the emergency ward received a patient suffering from an abdominal trauma. The template along with the representation of the current case are shown in Figure 5. The *newcase* template represents a class implemented by the *deftemplate* construct. Any new case accepted by the emergency ward will be represented as an instance of the template *newcase*.

While in his office, Kamel receives a notification that a case of severity 8 has been assigned to his team in operating room *OP2*. Kamel immediately moves to the assigned operating room. When he enters inside the operating room, the application detects a change by checking the current location from the GPS data and accordingly the slot *place* in the context instance is assigned a value that corresponds to the current location; in our case it is *OP2*. The rule then updates the context's task slot value to *operate*.

Context management is sensitive to the current location. When the doctor is informed about an assigned task, the slot *status* is changed to *reserved* which means that no other task can be assigned to him. As the application is sensitive to location changes; it detects that the current location has changed to operating room *OP2*. Accordingly, the user's *task* slot is updated to *operate*. It is worth mentioning that this change applies only when he is inside the operating room and not at notification time. This avoids any inconsistency of the surgeon's current task during the time he is on his way to the operating room. During the surgery all notifications and calls are put on hold into a queue preventing any disruption. They will be shown to the user after the *Task* slot value is updated.

```
(defrule surgeon-operates
 ?surgeon <- (context (status reserved) (place ?p))
 (test (member$ ?p ?*operating-room*))
  =>
 (modify ?surgeon (status operate)))
```

Figure 6 Changing the context status to "operate".

The rule that manages the change of the doctor's task is represented in Figure 6. The rule is defined using the *defrule* command provided by the Jess language. Rule *surgeon-operates* matches any context instance corresponding to a medical staff member having the status value *reserved*, i.e. whoever has been reserved for a surgery and is located currently in an operating room (*?*operating-room** is a list of surgery operating rooms). If a match is found, then the status of the user will be updated to *operate*.

At the end of the surgery, Kamel moves outside the operating room back to his office. A new place is detected triggering the following consistency rule in the Checking step of the reasoning engine (Fig. 7). This rule requests a confirmation from the user that he is no more involved in *operate* task. In case the confirmation is positive, then *status* is assigned empty value *nil*, meaning that the user is no more involved in any task and can receive calls and notifications on his mobile. The rule in Figure 7 allows detecting such change and updates the status to nil, in case the user confirms. The user confirmation is necessary in this case to ensure that the user did not move away temporarily from the operating room and will come back to complete the surgery.

```
(defrule clear-status
 ?surgeon <- (context (status operate) (place ?p))
 (test (not (member$ ?p ?*operating-room*)))
 (eq (get-confirmation) yes)
 =>
 (modify ?surgeon (status nil)))
```

Figure 7 Changing the context status to "empty".

The two rules *surgeon-operates* and *clear-status* allow a seamless and efficient management of the surgeon's context when involved in a specific task. Both rules check consistency constraints and take action accordingly. Other system rules check the context state associated with different medical tasks that need to be performed at specific locations and/or specific time.

6.5 Context-aware Implementation Principles

The implementation of the system relied on two major principles that we believe should govern the development of any context-aware system: *application independence* and *user ultimate control*. Application independence means that the application should be self-dependent relying as much as possible on data gathered by sensors, in order to infer decisions and actions. The aim of this principle is to relieve the user from the unnecessary user input and should force the development of intelligent and autonomous applications that must understand the user needs and provide prolific interaction. The second principle: user ultimate control, means that at any time the user should have the final word. He should be able to intervene to control and change regarding any decision taken or action proposed by the application. This principle favors the prevalence of the user decisions over the system inferences to guaranty system adaptivity and a seamless interaction. In general, the two abovementioned principles conflict and a good designed system should find a reasonable compromise to satisfy both of them.

The implemented system complies with the application independence principle to a great extent as it fully exploits context data especially the current place and current event to infer user current status and task. The user is requested to intervene in case the course of interaction can be affected badly by a system decision which will have an impact on the user's activity. For instance, the system requests from the user to confirm his availability after a critical task while a consistency rule checks that his current place is not conform to the current task (see example in Figure 7). As for the second principle: user ultimate control, the execution of the system is nested inside an interruption loop that listens continuously to any interruption triggered by the user. When such an interruption occurs it is handled by an exception handler that associates to every user request a suitable action.

7 System evaluation

The system has been tested to evaluate its performance and how fast it reacts to context changes. The main step which may constitute a real burden on the system is the activation of consistency rules. These rules check the coherence of context instances whenever there is a context change requiring henceforth a context update. Three consistency checking strategies have been defined and tested; the experiment and tests are presented in this section.

Duration	P Staff	S Staff	P Events	S Events	PS	ES	US
39	3	6	2	10	1.107	0.602	0.114
45	2	5	2	12	0.999	0.605	0.094
52	2	5	3	14	1.142	0.723	0.126
68	5	6	3	12	2.341	1.171	0.14
84	4	8	5	15	3.144	1.452	0.345
97	4	7	6	14	3.328	0.897	0.259
98	6	8	5	18	4.286	0.994	0.221
102	6	10	6	22	5.082	1.809	0.372
122	6	8	8	16	5.314	1.399	0.326
162	7	11	9	19	9.085	1.838	0.488
187	7	10	8	18	8.376	1.371	0.417
189	6	12	7	24	8.969	1.725	0.398
193	6	12	10	27	9.16	2.051	0.547
195	8	13	12	32	10.782	2.56	0.738
199	7	12	12	30	10.866	1.965	0.551
206	8	10	16	25	12.435	2.263	0.838
209	8	11	14	28	14.279	2.454	0.789
210	6	13	12	25	12.433	2.17	0.685
222	7	14	13	31	14.544	3.375	1.082
230	7	14	15	34	15.06	2.552	0.772

Table 1 Medical operations' data.

7.1 Consistency checking strategies

Based on how rules are activated, three consistency checking strategies have been defined and tested. The rationale of defining the three consistency checking strategies is to come up with a strategy that optimizes consistency rules activation, and at the same time asserts the best system response time. The first strategy is the Periodical Strategy (PS); it is the classical consistency checking strategy which consists in checking context changes periodically. For this strategy the time period is very important as all consistency rules are checked for every user at fixed time intervals. The second strategy is the Event-based Strategy (ES) that consists in updating contexts whenever a location or time event are detected. Accordingly, any change in the location or time of a medical staff triggers the activation of consistency checking rules. This strategy relies on the number of events recorded during medical operations. The third strategy uses domain information that is specific to the case study. It is the User-based Strategy (US) whereby events are categorized into two types: events related to principal staff and those related to secondary staff. This strategy came from the observation that not all events have equal impact on users' contexts. Indeed, events related to principal staff involved in medical operations (including surgeons, gynecologists and pediatricians), have an impact on the course of a surgery operation and consequently on all the team members while events related to secondary staff (including interns and nurses) do not have an apparent impact on the surgery operation. For instance, a surgeon operation has pre-operation and post-operation periods where nurses, anesthetist and interns are involved to prepare and arrange the operating room. The surgery operation itself is conducted under the principal surgeon presence, i.e. it starts whenever he is located inside the operating room. Accordingly, location events have been categorized on the basis of principal staff and secondary staff. Whenever an event related to a principal staff occurs, then all consistency rules are checked for all staff involved in a medical operation.

7.2 Experiment

In order to test the system, a set of 20 real-life emergency scenarios have been recorded from Oran hospital. These emergency scenarios correspond to surgery operations conducted at surgery ward. The objective of the tests was to evaluate the system behavior in real-life. Table 1 shows the values used in this experiment. Staff involved in each experiment have been categorized into principal and secondary (*P staff* and *S staff*) and for each category we have recorded the event that occurred during the operation. We recorded also the *Duration* that is the time dedicated to each operation. It is noted that context instantiation occurs only once when a staff is assigned the new medical case. The time starts to be recorded at activation of rule *surgeon-operates* (Fig. 6) resulting in the update of the user's task slot to *operate*. Starting at this time all context updates are recorded resulting in the times shown in Table 1.

7.3 Performance results

The main contextual factors that will play a role in managing the surgeon *status* in the case study are i) *place* which determines the user's current place (operating room, office, laboratory…); ii) *status* (reserved, free, operate, …) to manage the tasks that are assigned to users. For instance, if the surgeon status is *reserved* then no task can be assigned to him; and iii) *role* that categorizes staff involved in the surgery operation as *P Staff* and *S Staff*.

The scenarios are ordered by the duration (in minutes) of the medical operation reported in the first column of Table 1. *P Staff* and *S Staff* columns represent the number of principal staff and secondary staff involved in the operations. *P Events* and *S Events* report the number of events recorded respectively for principal and secondary staff. PS, ES and US are the response times related to the three strategies defined in this experiment (in seconds). Figure 8 shows the graphs related to each strategy.

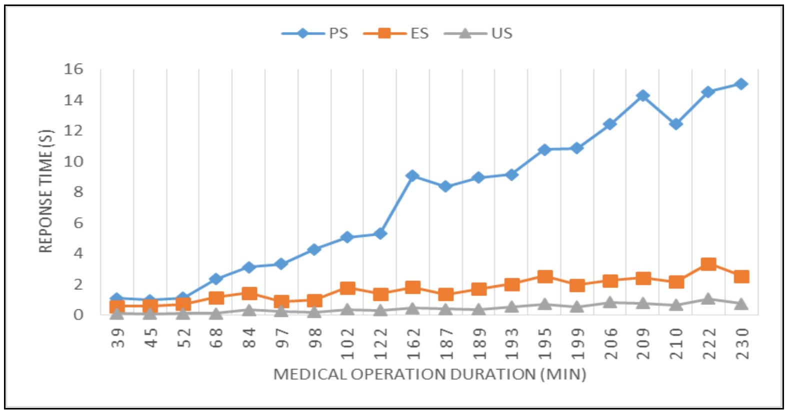

Figure 8 System response time using the three consistency checking strategies.

The tests have been conducted on a personal computer Core i7, 2.3 GHz on Windows 8.1. PS has been tested with a time period of 6 minutes. The test results show that the system response time using US strategy outperforms ES and PS strategies as it required less time in all the cases tested (Fig. 8). US that is tightly related to the events of principal staff that occur during medical operations reduces the complexity and the processing time related to context instantiation and consistency checking. This strategy always guarantees to have less (or equal) processing time then PS. Although the number of events that occur during medical operations is correlated to the duration as staff need to move in and out of the operation room, it has been noticed that this number increases somehow linearly with regard to the operation duration and does not affect much the response time specifically for US which considers only events related to principal staff.

7.4 Context management consistency

Context management consistency is also an important indicator to evaluate in order to ensure that users' context instances are consistent. In order to measure the system consistency, a system evaluation has been done to check the consistency of the context instances during the medical operations using the three consistency checking strategies.

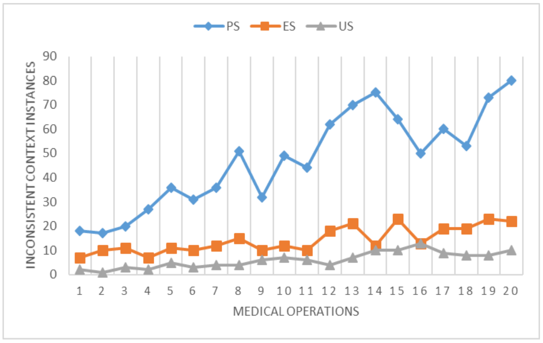

Figure 9a Number of inconsistent context instances

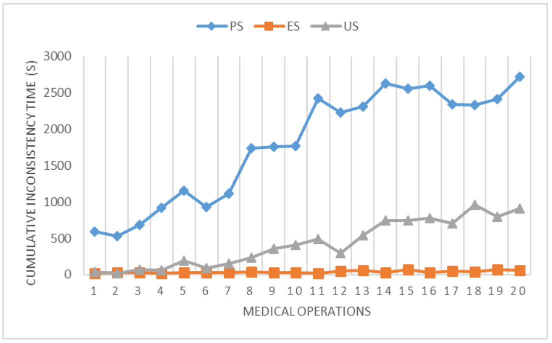

Figure 9b Cumulative inconsistency time.

For this evaluation, we used the log files recorded for each experiment and we used two measures to evaluate the system consistency. The first measure is the number of inconsistent context instances. This measure counts the number of context instances that were inconsistent during a medical operation (Fig 9a). The second measure is the cumulative inconsistency time which reports the total time during which context instances were inconsistent (Fig. 9b).

Results show that US and ES are better than PS in both experiments. US showed the lowest number of inconsistent context instances during operations (Fig. 9a). However, ES performed better than US in cumulative inconsistency time (Fig. 9b) as ES checks the consistency of context instances for each staff event unlike US which is activated only on events related to principal staff and hence has the tendency to preserve inconsistency context instances of secondary staff for a longer time.

7.5 Discussion

The experiments allowed us to evaluate thoroughly the system and to have a good appreciation of the performance and consistency of the three strategies w.r.t their ability to manage efficiently scenarios at the hospital surgical ward. In terms of response time, US is clearly better than ES for the twenty scenarios tested while PS is the most time consuming strategy. This is mainly due to the fact that PS exhaustively runs all consistency checking rules for all users periodically. Besides, this strategy is the lesser consistent. PS remains interesting to consider for medical operations necessitating frequent principal staff movements. In terms of system consistency and the ability of strategies to guarantee consistency of context instances, ES and US had the best performance. US recorded the less number of inconsistent instances while ES was better in cumulative inconsistency time. This is explained by the fact that US is less sensitive to changes than ES which checks the context instances for every event; and this may occur many times for the same staff during an operation. US is the most stable strategy and generally guarantees the best trade-off between response time and context consistency for the medical operation experiments done. Although it showed less cumulative inconsistency time due to the accumulation of secondary staff inconsistent context instances which do not trigger automatically consistency checking, this had no noticeable impact on the course of medical operations which are mostly monitored by principal staff.

8 Conclusions and Future Work

Mobile applications can benefit a great deal from context management to increase application usability and to offer exceptional application interaction experience for nomadic users. This paper showed how context consistency constraints can be integrated and exploited in the mobile scene where context needs to be managed to encompass efficiently environment changes. Context management is presented as a cycle starting by gathering data, checking system data consistency, updating the context state, reasoning and inferring decisions and actions, and finally acting. Context management relies on two major components: i) the context model that represents all the attributes which influence the interaction between the user and the application in a wireless environment; and ii) the reasoning engine which proposes a customized experience to users by adapting information and services to their needs. The reasoning engine has been implemented as a knowledge-based system where the context model is represented as a template and context instances are derived from the template as facts. The system performance and consistency have been tested using a real-life case study where three consistency checking strategies are used to update users' contexts. The results show that the User-based Strategy that takes into account events related to principal staff is the most prominent as it improves the trade-off between response time and information consistency.

Future work will focus on developing a strategy for context change. The strategy aims to prioritize actions which will result on avoiding conflicts while many rules are candidate to be fired. This strategy should not be ad-hoc depending on the application but must be integrated in the system by mean of knowledge representations of the domain, actions, and users. Indeed, we would like to distinguish in our representations between general knowledge that is valid for any application and could be coded as a core in the system, and application- and domain dependent-knowledge that should be described and coded for each application.

Acknowledgements

This work was supported by the Research Center of the College of Computer and Information Sciences, King Saud University. The author is grateful for this support.

References

1. Charland A., Leroux B., Mobile Application Development: Web vs. Native Communications of the ACM, 54 (5), May 2011, 49-53.
2. Schilit B., Mobile Computing: Looking to the Future, Computer, 44 (5), May 2011, 28-29.
3. Anderson, J. Q., Rainie, L., The future of the Internet III. Pew Internet and American Life Project. 2008, Retrieved March 26, 2011, from http://www.pewinternet.org/ Reports/2008/ The-Futureofthe-Internet-III.aspx.
4. Berri, J., Benlamri, R., Context-Aware Mobile Search Engine, Next Generation Search Engines: Advanced Models for Information Retrieval, IGI Global, 2012, 371-385.
5. Desclés, J. –P., Langages applicatifs, langues naturelles et cognition, Hermès, Paris, 1990.
6. Givón, T., Context As Other Minds: The Pragmatics of Sociality, Cognition and Communication, John Benjamins, Amsterdam, 2005.
7. Lukowicz, P., Pentland, A., Ferscha A., From Context Awareness to Socially Aware Computing", IEEE Pervasive Computing, 11 (1), March 2012, 32-41.
8. Dey, A. K., Understanding and Using Context, Personal and Ubiquitous Computing J., 5 (1), 2001, 5-7.

9. Bellavista, P., Corradi, A., Fanelli, M., Foschinia, L., Survey of Context Data Distribution for Mobile Ubiquitous Systems, ACM Computing Surveys (CSUR) Surveys, 44(4), 2012.
10. Yang W H, Liu Y P, Xu C, et al. A survey on dependability improvement techniques for pervasive computing systems. Sci China Inf Sci, 2015, 58: 052102(14), doi: 10.1007/s11432-015-5300-3
11. Xu, Chang, Cheung, S. C., Chan, W. K. and Ye, Chunyang, Partial constraint checking for context consistency in pervasive computing. ACM Trans. Softw. Eng. Methodol. 19(3), 2010, 1–61.
12. Hong, J., Suh, E.-H., Kim, J., Kim S., Context-aware system for proactive personalized service based on context history, Expert Systems with Applications, 36, 2009, 7448–7457.
13. Oh, Y., Han, J., Woo, W., A Context Management Architecture for Large-Scale Smart Environments, IEEE Communications Magazine, 48 (3), March 2010, 118-126.
14. Viktoriya Degeler, Alexander Lazovik, Reduced Context Consistency Diagrams for Resolving Inconsistent Data, ICST Transactions on Ubiquitous Environments, Volume 12, Issue 10-12, October-December 2012.
15. Bu, Yingyi, Chen, Shaxun, Li, Jun, Tao, Xianping and Lu, Jian, Context Consistency Management Using Ontology Based Model, Lecture Notes in Computer Science 4254, 2006, 741–755.
16. Lu, H., Chan, W., and Tse, T., Testing pervasive software in the presence of context inconsistency resolution services. In Proc. Of International Conference on Software engineering (ICSE'08), May 10–18, Leipzig, Germany, 2008, pp. 61–70.
17. Berri, J., Benlamri, R., Atif, Y., Ontology-Based Framework for Context-aware Mobile Learning, In International Conference on Wireless Communication and Mobile Computing, Vancouver, Canada, 3-6 July, 2006, 1307-1310.
18. Basaeed, E. I., Berri, J., Zemerly, J., Benlamri, R., Learner-Centric Context-Aware Mobile Learning, IEEE Multidisciplinary Engineering Education Magazine, 2 (2), June 2007, 30-33.
19. Russel, S. J., Norvig, P., Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice Hall, N. J. USA, 2010.
20. Brown P. J., The Stick-e Document: a framework for creating context-aware applications. Proceedings of the Electronic Publishing '96, 259-272. Laxenburg, Austria: IFIP, 1996.
21. Hull R., Neaves, P. & Bedford-Roberts, J. (1997). Towards situated computing. Proceedings of the 1st International Symposium on Wearable Computers (ISWC'97), 146-153. Los Alamitos, CA: IEEE.
22. Anind K. Dey, Gregory D. Abowd and Daniel Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Journal of Human-Computer Interaction, Springer, 16 (2), 2001, pp. 97-166.
23. Ward A., Jones, A. & Hopper, A., A new location technique for the active office. IEEE Personal Communications, 4(5), 1997, 42-47.
24. Rodden T., Cheverst, K., Davies, K. & Dix, A., Exploiting context in HCI design for mobile systems. Proceedings of the Workshop on Human Computer Interaction with Mobile Devices, Glasgow, Scotland, 1998.
25. Echtibi, A., Zemerly, M. J., Berri, J., A Service-Based Mobile Tourist Advisor, International Journal of Computer Information Systems and Industrial Management Applications, Vol. 1, 2009, 177-187.
26. Jawad Berri, Context Reasoning for Mobile Systems, International Conference on Computer Information Systems (ICCIS'2014), Sousse, Tunisia, January 17-19, 2014.
27. Hill, E. F., Jess in Action: Java Rule-Based Systems. Manning Publications Co., Greenwich, CT, USA, 2003.
28. Winston, P., Horn's, B., LISP 3rd edition, Addison Wesley, 1989.
29. Schildt, H., Java A Beginner's Guide, 5th Edition, McGraw Hill/Osborne, 2011.