Journal of Universal Computer Science, vol. 16, no. 15 (2010), 1973-1991
submitted: 31/1/10, accepted: 28/7/10, appeared: 1/8/10 © *J.UCS*

Multi-Level Context Management and Inference Framework for Smart Telecommunication Services

Carlos Baladrón
(ETSIT, University of Valladolid, Spain
cbalzor@ribera.tel.uva.es)

Alejandro Cadenas
(TelefónicaI+D, Madrid, Spain
cadenas@tid.es)

Javier Aguiar
(ETSIT, University of Valladolid, Spain
javagu@tel.uva.es)

Belén Carro
(ETSIT, University of Valladolid, Spain
belcar@tel.uva.es)

Antonio Sánchez-Esguevillas
(TelefónicaI+D, Valladolid, Spain
ajse@tid.es)

Abstract: Telco operators and other players are searching for intelligent value-added services, i.e., communication applications that take advantage of the huge amount of user data available to the operators (location, contact lists, etc.) in order to adapt themselves to the preferences and context of each individual. However, the current networks of the operators lack the proper infrastructure to handle context data in a clean and unified way, so smart context-aware applications are extremely difficult to engineer, develop and deploy. Accordingly in this paper a global context processing architecture is presented. In addition, the monitoring of users in order to extract and process the context is a task potentially resource consuming. That is a significant problem in global telco deployments.This paper also presents a proposal for a multi-level context management framework for smart telecommunications services, whose objective is to optimise the available processing resources of the presented architecture to provide contextual monitoring to a high number of subscribers with limited resources.

Keywords: Context-Aware, Application, IMS, Convergent, Service Layer, Network Architecture, Ubiquitous Computing
Categories: C.2.1, I.2.4, I.2.9, I.2.13, I.2.12

1 Introduction

The current landscape of the telecommunications market could be considered at least convulsed. Constant innovations appearing in fields like mobility and multimedia

introduce new features and services to be offered, and the increasing convergence with IP technologies, the Internet and the World Wide Web (WWW) point to an integrated user experience in everything related to personal communications.

A direct consequence of this convergence is that the information about users at the disposal of services and devices is increasing day by day. A clear example is the recent launch of Google's operating system for smartphones, Android, which pursues the integration in a single device of data and functionalities from phone and contact lists, friends from social communities, positioning and location, images and photos and potentially everything that the user has delivered to the Internet. While the management and processing of this enormous amount of integrated and correlated data about an individual becomes a challenge, it also fosters an unlimited array of new applications. Such applications would be able to process and react smartly to the user data. Continuing with Google's Android (but also in other platforms), several applications are taking advantage of context data to produce intelligent results, like for instance modifying the desktop and/or the Graphical User Interface (GUI) of the device depending on the location or parameters of the ambient around the user.

It is therefore easy to imagine the kind of advanced services that could be implemented taking advantage of context data: if I enter a meeting room at the office, my smartphone can automatically configure itself to silent mode; if I am stuck at a traffic jam when driving to an appointment, messages could be automatically sent to the friends waiting for me informing them of the delay; my job's email inbox could set its own auto-response when I am on holidays, or even better, if I am ill and have to unexpectedly stay at home.

This kind of context-aware services, although at a much simpler scale, are already starting to permeate the current communication landscape from the professional to the residential markets, and a crystal clear example is the plethora of location aware services offered in the AppStore and the Android Market.

However, when implementing these services, developers usually find two fundamental big problems. First, a lot of information could be available, but in any case it is fragmented into different data repositories, which means that to access it, it becomes necessary to contact different providers. This includes Web services, telco operators, sensor networks, other devices, etc. And that means using very different protocols, Application Programming Interfaces (APIs), message exchanges and file and data formats.

And secondly, the information is not only fragmented, but raw, that is, is stored mostly in the same format the provider of the information retrieves it. Any additional processing has to be done directly by the application that consumes the information. This is known as a "vertical" solution, in which the processing of the information needs to be done in all the servers in which that information is used. This option is normally far from a good solution because the sharing of processed information among different services is very low, algorithms may be specialized and difficult to design, the processing power required too high for a mobile device and the specific user trying to access the raw information may not have permission to do so.

A clear example of these difficulties may be seen when designing an application to post if a professor is in the middle of a class, available for questions in the office, not in the University or unavailable. That kind of high-level information could be extracted combining location, calendar, proximity, etc. In a vertical implementation,

this reasoning has to be done directly by the application, which means a huge waste of resources both at development and execution stages that shall be performed again by a similar different application that does not share information with the first one.

Accordingly the optimum architecture is a "horizontal" one, in which the information processing is performed in a central element or module, and is reused or consumed by all applications. Several initiatives like [Tao, 05], [Van Kranenburg, 06], [Capra, 03] have appeared in order to offer integrated frameworks for the management of context data and represent good alternatives to control how the data is handled, but they usually describe in detail the software plane while not dealing in depth with architectural issues and deployment in real situations.

If there was a trusted entity providing easy access to processed context, the complexity and costs of developing and deploying context-aware applications would decrease significantly which could potentially lead them to a quick expansion. Some high-level functionalities of such element would be retrieving information from all kinds of available sensors and services, processing and reasoning over it to obtain higher level context data useful to applications, allowing easy control and management of Authentication, Authorization and Accounting (AAA) features so users are not reluctant to exposing personal information, and providing a centralized and standardized access point to it.

Telco operators are in a perfect position to take the role of this kind of context provider. They already own a lot of information about the user, including location, addresses, contact lists, accounting data, etc.; their infrastructures are converged (thanks to Next Generation Networks - NGN trends [Cadenas, 04] and IP Multimedia Subsystem - IMS [Schmidt, 07], [González, 08]) with packet-based IP networks jointly with the Internet, as well as all kinds of sensor networks, so they have access to virtually every bit of data published about the user. Additionally, telco operator's servers have enough processing power to run intelligent algorithms over the data and extract higher level information, integrating also facts about other users. And they are probably the closest to a trusted entity that a big company in the IT (Information Technologies) world can be.

This paper presents a proposal for a context management infrastructure to be integrated within a telco operator network. Thanks to it, applications would be able to access to every bit of context data about a user just knowing an identifier key of the user, such as a SIP URI (Session Initiation Protocol – Uniform Resource Identifier). It provides functionalities for smart context data retrieval (managing subscriptions to sensor networks, synchronous and asynchronous data updates), information integration and cleaning (ensuring coherence and validity), security, accounting, authorization, authentication, and access control lists (so the users are able to select which entities are able to access their data) and intelligent inference to extract higher level data.

This paper is organized as follows. After this introduction, Section 2 presents the proposed context management infrastructure and details how context management is performed. Section 3 deals with the details of the Context Intelligence, inference and reasoning process, while Section 4 explains the multi-level algorithm implemented to optimize the resource usage for reasoning and inference in a wide area deployment. Finally, Section 5 exposes the conclusions of this work.

2 Framework for Smart Telecommunication Services

2.1 Global Architecture and Context Enabler

The global picture of the proposed framework is depicted in Figure 1. It represents a generic operator, NGN-style infrastructure evolving around an IMS core, with links to the WWW, different access networks and sensor networks. These elements act as context sources and context consumers.

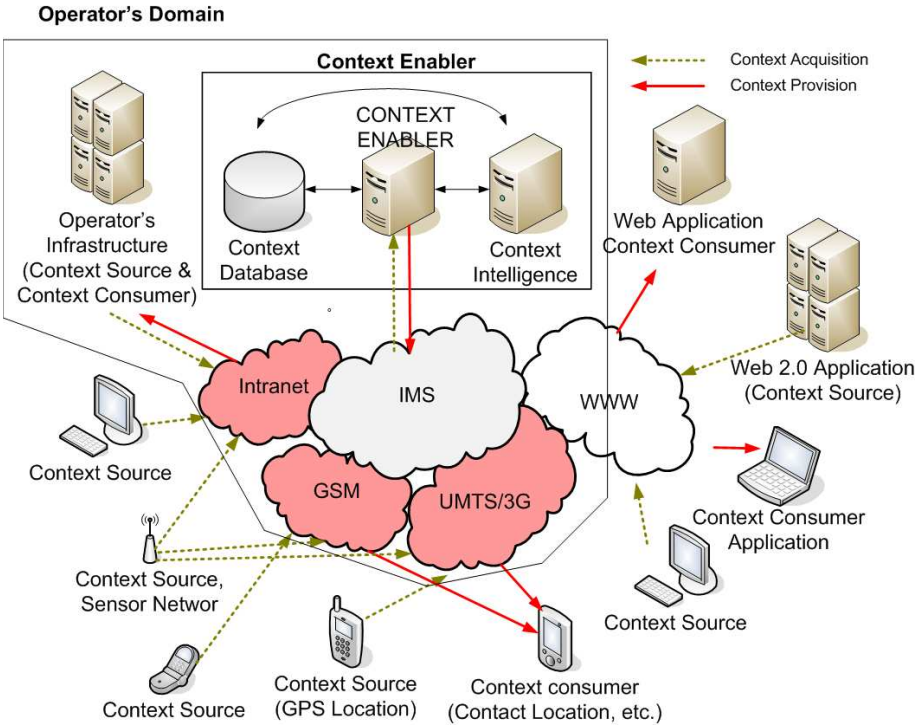

Figure 1: Global Architecture

The proposed approach for context information management and processing is a centralised one. Such architecture is simpler on the orchestration side, as the different entities exchaing context information will have a much simpler logic. This is especially clear in the case of the services that use aggregated context information. In a centralised approach these services do not need to acquire or request specific types of information, as all that logic will be implemented in a single complexity point. The service development, which is the main problem for time-to-market and service integration, is much simpler.

On the other hand, a distributed context information approach would rely on service transaction orchestration, in such a way that all information required by the service should be acquired and processed by the service itself. This second option would distribute the complexity across all context-aware services.

This information processing centralised approach means a significant advantage in global deployments, in which the user's context information acquired may be diverse, and not available on a constant basis. The required flexibility with respect to the nature, quality, reliability and diversity of user's context information is provided by the centralised element that is described in following sections. Taking such complexity to each service integfrated in the system would jeopardize the service development and deployment.

The Context Enabler, the heart of the context-aware system proposed in this paper, is plugged into this infrastructure through the IMS core, which provides basic functionalities on several fields like numbering, addressing, user provision and management, security/AAA and transport, allowing for instance SIP based communications or performing some authorization operations.

Inside the Context Enabler there are three submodules:

- The Context Database is a pasive element that stores the context data, and is accessed by the other modules of the Context Enabler.
- The Context Intelligence submodule packs a semantic reasoner and several artificial intelligence algorithms, used to process the raw data received from the sensors in order to extract additional implicit data (high-level context), clean up and repair inconsistencies among several context sources or infer and/or predict missing values from malfunctioning sensors using other values and history information.
- The Context Manager acts as a single point interface towards the rest of the world and directs all context operations, including reception of context data, subscription management and government of the Context Intelligence.

For reception of context data, the Context Manager is the point where sensors and context sources entering the system (a new sensor network, a new smartphone acting as a context feed, a new Web service, etc.) are registered and validated. The Context Manager is able to poll and configure these context sources, asking for a specific measure or value, or setting a new refresh rate. These requests are based on current requirements of the client applications or the Context Intelligence.

Client applications will also request the Context Manager specific data, or will subscribe to some parameters of the context of certain user. The Context Manager will in this case notify the client application periodically, or when changes occur in those paramenters. The Context Manager applies the configurable authorization rules for all requests. Each user is able to set up parameter and/or requester specific permission through a Web interface.

The Context Manager also controls the Context Intelligence, requesting an inference process to obtain some parameters and specifying which users should be monitored semantically and/or with artificial intelligence, for example. This is done in order to save the huge resources required to perform Artificial Intelligence (AI) or semantic based analysis. The algorithm used for this will be further detailed in Section 4.

2.2 Context Management

Context acquisition is made by different elements or context sources, like sensing devices located in sensor networks with external connectivity to report the results. However, as shown in previous Sections, the elements capable of capturing useful information about the situation of a user (context) can also be telco services, Web applications like social networks, etc.

In parallel, there are service entities whose service functionality or quality can be enhanced by taking advantage of context information about the target user; those are certain applications, telco services, third party applications like Web application servers, or even applications running in mobile phones, etc. Each one of those services will be interested in a very specific type of context related to the actual functionality of the application itself, not in any other type of user context that the system may obtain (a location-triggered messaging service requires location information, other type of user's context is not really useful).

Accordingly, the generic procedure in a global context monitoring system is as follows:

- Context sources located in sensor networks or in applications or services acquire user's context information (probably low-level information, usually at a physical level like temperature, presence, location, etc., although other types of user context are also possible like user mood, user activity, etc.).
- Such low-level information is progressed to the Context Enabler, where all information is processed and aggregated. By having such centralised processing element, the quality of the context information obtained is much higher due to inter-domain processing: it is much easier to detect wrong signals if the global amount of information is higher.
- Any application that may require of certain type of context information can subscribe to the Context Enabler. In that subscription process the application will specify the type of context that it is interested in and wants to receive notifications about. In addition there will be a negotiation between the application and the Context Enabler to come up with a set of notification configuration parameters, like the periodicity or the contextual trigger that will generate the context notifications sent to the subscribing application.
- The Context Enabler will generate context notifications for a given user to the corresponding application that subscribed to those notifications. That will happen as per the notification parameters during the subscription lifetime or until any of the peers may decide to cancel the subscription.

This global sequence is depicted in Figure 2.

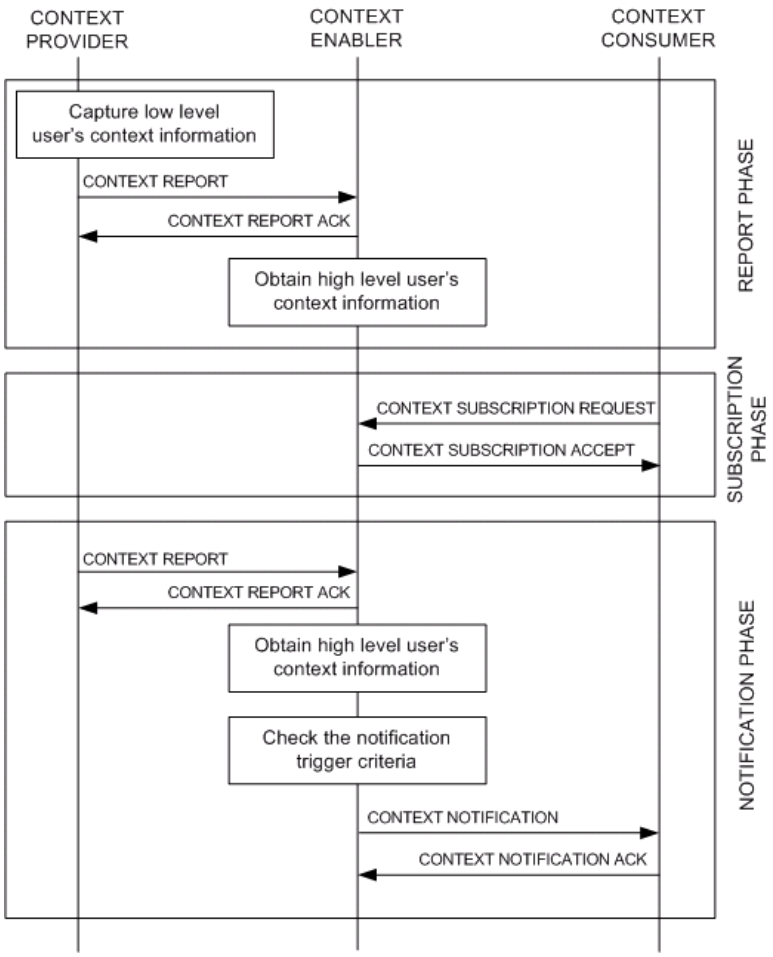

Figure 2: Sequence followed by the Contextual entities

Such general procedure has the following key advantages:

- Higher quality of the context information.
- Higher variety of types of context information. Different and traditionally isolated sensor networks can progress the captured context information to the Context Enabler through a convergent NGN control layer that guarantees the global accessibility.
- Dynamic allocation of subscriptions from external context consumer applications.
- External context consumer applications are made agnostic of the specific procedures followed or devices involved to capture the context of the user, becoming a robust end to end system.

2.2.1 Context Source Interface

The elements located in the sensor networks in charge of capturing the user context information will implement this interface with the Context Enabler.
The main requirement for this interface is to provide the adequate flexibility to carry the context information captured by the sensing device or application to the Context Enabler. Accordingly, the transport protocol shall be interoperable and supported over the different transport transit networks between the context source element and the Context Enabler.

In order to be fully compatible with the NGN Convergent control layer proposed, the transport protocol implemented is SIP, and the contextual message is embedded in the body of the SIP method in the format of an eXtensible Markup Language (XML) structured set of tags.

The information structure required in the reporting message from the context source to the Context Enabler is conceptually simple. The captured magnitude shall be carried properly, including also the identity of the sensing device. In Figure 3a an example of XML structure for the context report message is presented.

2.2.2 Context Consumer Interface

This interface is more critical given the complexity involved in the dialogues. This interface will be implemented between the context consumer elements (the applications subscribing to the Context Enabler to receive context notifications for a given user or set of users) and the Context Enabler.

Accordingly, apart from the requirements applicable for the previous interface, the functionality and flexibility required in this case is much higher. It is necessary to implement a subscription mechanism, as well as a notification one in return. The notifications can happen periodically, on demand or event triggered, so the specific condition for the notification to happen shall be specified in the subscription message. Also the type of context information to be included in the context report from the Context Enabler to the Context Consumer application shall also be included. These requirements will add complexity to the interface.

The API is defined after a rigorous analysis of all the possible use cases that can be found during execution, including successful and not successful registrations and notifications of user context information.

A snapshot of the context report generated by the Context Enabler back to the Context Consumer Application is shown in Figure 3b.

```
<?xml version="1.0" encoding="UTF-8"?>
<entityname>Mobile_application</entityname>
<ContextFunction>Context_Provider</ContextFunction>

<ContextMessage>
    <MessageType>Information_Report</MessageType>
    <ReportType>OnDemand</ReportType>
    <Timestamp>#Sat Apr 25 19:07:00 CEST 2009</Timestamp>
    <ContextCaptured>
        <ContextName>Position</ContextName>
        <ContextUnit>LatLong</ContextUnit>
        <ContextValue>
            <Lat>N41,39.717</Lat><Long>W4,42.571</Long>
        </ContextValue>
    </ContextCaptured>
</ContextMessage>
```

a)

```
<?xml version="1.0" encoding="UTF-8"?>
<entityname>Context_Enabler</entityname>
<ContextFunction>Context_Enabler</ContextFunction>

  <ContextMessage>
    <MessageType>Context_Notification</MessageType>
    <NotificationData>
        <ReportType>OnDemand</ReportType>
    </NotificationData>
    <ContextCaptured>
        <ContextName>Position</ContextName>
        <ContextUnit>LatLong</ContextUnit>
        <ContextValue>
            <Lat>N41,39.717</Lat><Long>W4,42.571</Long>
        </ContextValue>
    </ContextCaptured>
    <SubscriptionData>
        <SubsID>112765</SubsID>
        <SubsStatus>Unsubscribed</SubsStatus>
    </SubscriptionData>
  </ContextMessage>
```

b)

Figure 3: a). XML structure for the context report message, b). Report generated by the Context Enabler to the Context Consumer Application

3 Context Intelligence, Inference and Reasoning

The aim of the Context Intelligence submodule, depicted in Figure 4, is to process the raw data received from the context sources (known as low-level or sensed context) using semantic logic and/or artificial intelligence algorithms, in order to extract better or improved information. The purpose of this information is threefold:

- Clean, update, correct and resolve inconsistencies in the sensed context: it is easy to imagine that some context sources could eventually malfunction due to hardware or software failures and throw faulty lectures. If they were isolated, it would probably be impossible to determine that the context source is no longer trustable and therefore the whole context system could be compromised. However, in combination with lectures from other context sources, it is possible to infer that something is wrong and even identify the faulty value and/or sensor. It is possible for instance to imagine that the Global Positioning System (GPS) unit in certain user's smartphone could lose the GPS signal and continue throwing the last location stored. However, if the Context Intelligence scans that user's context, it will realize that the Global System for Mobile Communications (GSM) based location throws a different value and that he is connected via Bluetooth to a couple of fixed devices hundreds of meters away from the position sent by the GPS. Such infrastructure Bluetooth location devices (beacons) may broadcast the location over Bluetooth, so that the mobile handset is able to handle that information and progress that to the Context Enabler via standard data connection. In that case, the Context Intelligence will ask the Context Manager to tag the GPS location of that user as untrustable, and maybe even stop receiving data from it for a period of time. The reliable location information would then be the one acquired from the nearby location beacons over Bluetooth protocol and transmitted to Context Enabler by the mobile handset.

  It is also easy to imagine that due to a mismatch between sensor report frequency and application usage, certain context values could result outdated for some purposes. An easy example could be the case in which a user's location is being monitored with a period of 5 minutes, but if that user enters his car and starts moving, it is likely that the location data could be outdated when retrieved by an application. The Context Intelligence is able to understand this situation and ask the Context Manager to refresh location more frequently.
- Infer and predict missing sensed values: the Context Intelligence is able to use the available information also to actually infer additional data which is not even present due to limitations in the sensing network. It is possible to imagine for instance a user whose location is not determined through GPS or GSM means, but is connected using Bluetooth to another mobile phone for which there is actually location information. The Context Intelligence is then able to infer the location of the first user.

  This kind of inference is more or less straightforward, meaning that, semantically, the location of the user is a concept close to the location of near devices (connected via Bluetooth). However, the Context Intelligence is also capable of further inferences involving seemingly unrelated concepts, thanks to AI algorithms and history analysis.
- Inference of high-level context: apart from the magnitudes directly measured by sensors and context sources, there are other important context parameters which cannot be directly provided by any means, such as the status. For instance, if a user's location is quite the same than his car's location, his

speed is more than 30 km/h and he is not near any of their contacts, the Context Intelligence will infer that he is probably alone in the car and driving, so the user status will be set to "driving".

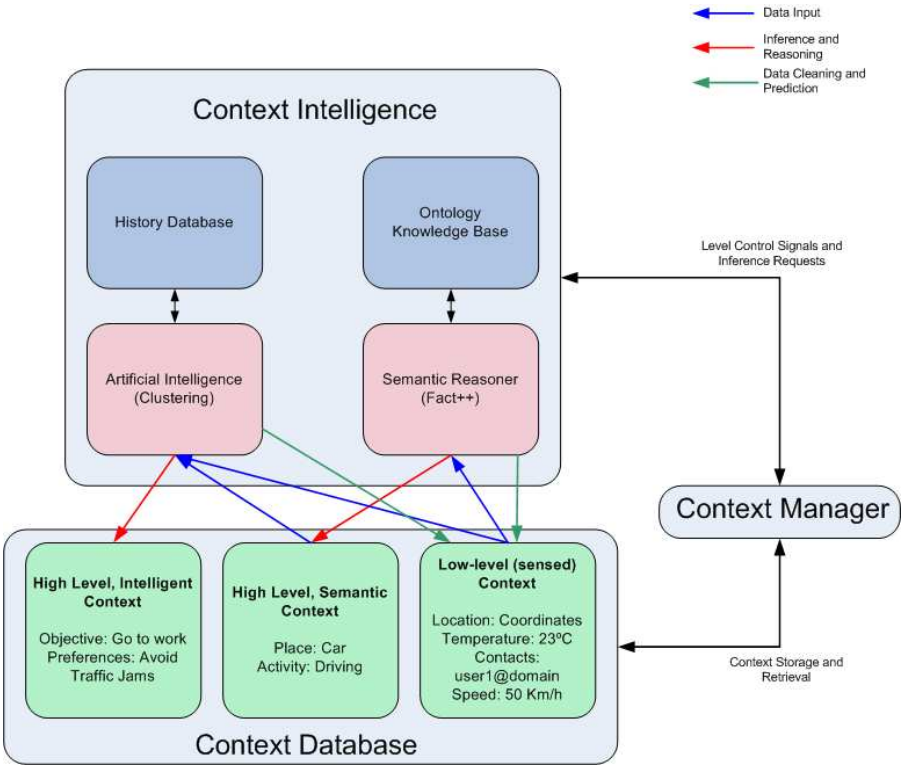

Figure 4: Scheme of the Context Intelligence submodule operation

The tools employed by the Context Intelligence to perform these three tasks are a semantic reasoner and several AI algorithms, which operate over a semantic representation ([Strimpakou, 06], [Wang, 04]) of the context of each user. This means that the information about context is in a machine-understandable format, thanks to a knowledge base representing the concepts handled known as ontology. This ontology specifies for instance that a user can only have a location, that if two users are "near" they will have the same location, and that if there is a Bluetooth connection between two users, they are "near".

The semantic reasoner takes the sensed context and applies the user ontology to identify implicit information. Following the previous example, the semantic reasoner will infer the location of a user because he is connected using Bluetooth to another located user. This inference might seem straightforward; however, when the ontology grows, this kind of implicit facts start to be difficult to establish at first sight, and there is where the power of the semantic reasoner appears. Our prototype implementation uses OWL (Ontology Web Language) representations of the context

and the Fact++ reasoner [Tsarkov, 06], while in theory any other reasoner could be easily plugged instead.

The AI component of the Context Intelligence processes context data together with historical data. The mechanisms applied are modified clustering algorithms adapted to the context awareness domain [Baladron, 08], and are aimed at identifying recurrent context situations of the user which may be classified as typical statuses.

In order to perform this computation, historic context data is maintained for each monitored user. The context of a user is considered a vector space where each meaningful component (latitude, longitude, temperature, presence status, etc.) is considered one dimension, so as to one snapshot of a user's context is represented as one point in this vector space. When enough points have been retrieved by monitoring the user, a clustering algorithm can be applied over them to identify clusters of similar context snapshots. These clusters are then considered recurrent context situations, typical statuses.

A simple and straightforward example of this kind of clustering can be imagined considering only GPS location information (retrieved for instance from a GPS-equipped *smartphone*), resulting in a 2D context vector space (where latitude and longitude are the two dimensions). If data about a user location is monitored for long enough, it is reasonable to imagine that there will be sample points scattered along the places the user has visited, but most of the points will be accumulated around meaningful places for his or her daily life, such as home, workplace, and preferred leisure places. Therefore, when a clustering algorithm is applied over this dataset, these preferred places will be identified as clusters, and for the purpose of the system, will be labeled as recurrent context situations and therefore typical statuses.

This kind of processing can be extended in the proposed system by introducing additional features (dimensions in the context vector space) supported by sensors on the devices carried by the users, such as environmental values (temperature, light, speed, etc.), biometric (blood pressure and body temperature when wearing intelligent clothes), network (WiFi or 3G connectivity, available bandwidth, etc.), activities (running an application or speaking on the phone), social (presence status, proximity to different friends and contacts), etc. Enriching the context with these features allows for a more detailed typical context identification. For instance, considering speed may result in identifying a typical status when the user is driving (speed will be high), or monitoring proximity to the user's friends may help to identify the status "hanging out with friends". The parameters to be monitored therefore may be chosen depending on the specific application.

A variety of clustering algorithms may be applied, resulting however in different performance figures depending on the situation and scenario. Additionally, incremental clustering [Charikar, 97] is also as a very desirable option since the system may incorporate new context snapshots received into the already identified typical statuses without performing a complete reclustering every time a new context measure arrives.

In any case, when the typical statuses are identified, the current context of the user may be assigned to one status with a given probability, so it is assumed that the context values of the original status and the current one could be similar.

This information can then serve the different purposes described at the beginning of Section 3, context cleaning, inference and prediction. For instance, when the

current user context is identified as being inside one of the typical statuses for all but one of the dimension/features due to a malfunction in the sensor providing it, the context value for that feature can be autonomously inferred and filled by the AI module using the mean value of that feature inside that typical context. For example, in the case that a user has entered the typical status corresponding to being "at work", but proximity to his boss cannot be retrieved (maybe the GPS module in the boss' smartphone is not working), the parameter "proximity to boss" can be automatically fulfilled with the mean proximity value observed in the history data when the user is "at work" (and tagged with a confidence value so applications retrieving it may know that "proximity to boss" has not been monitored directly but inferred based on previous experience).

In any case, it is worth noting that due to the storage space required for historical data and the processing power to analyze it, the application of these AI algorithms is very resource intensive.

A note must be made on the information processing performed by the Context Enabler. It is not realistic to assume that the AI-based or semantic-based processing will always provide added value with respectto rule-based processing. However such techniques have proved to be extremely useful in specific domains for which the required context information to perform reliable AI processing is available, or domains in which the user is properly modelled by a domain-ontology. A control of the situations in which such processing techniques can and may be applied is also provided by the algorithm described in the following section.

4 Multi-level Context

The processing that takes place in order to obtain, from the low-level context information, the high-level context information that is useful to the external context consumer applications can be based on different information processing mechanisms. These processing algorithms are running in the Context Enabler. They will be consuming processing resources (typically, CPU and memory) available at the server. The Context Enabler will work on a context monitoring session basis, establishing a monitoring session per each single managed user.

Just like any other global service executed at a server with limited resources, a congestion policy that assigns properly the processing resources is necessary at the Context Enabler. If all context monitoring sessions are accepted and none of the previously existing ones can be cancelled, that would clearly lead to bottleneck situations in which high priority session requests cannot be accepted by a congested system while lower priority ones are running just because these ones arrived a bit earlier.

The information processing mechanisms can be diverse, but the ones mainly included in the research presented in this paper are the following:

- Database computing. The response can be obtained as a direct boolean calculation from the available database fields from the low-level context information.

- Database computing, intensive in CPU requirements. Similar to the previous one, but involving a high-level of low-level context information entries to produce the response.
- Semantic reasoning. A semantic reasoning process is used to produce the output from the available information. As has been shown previously, this mechanism consumes significant memory and processing resources as it requires a reasoner and a set of instances of ontologies for the user.
- Artificial Intelligence. This implies the application of several AI algorithms to the context and history data. Due to the sheer amount of data to be analyzed and monitored and the complexity of the algorithms, this mechanism is extremely resource intensive. However, in many situations, the performance of the semantic mechanisms and the AI procedures yield similar performances. However, in order to apply semantic mechanisms it is required the use of a ontology structure sufficiently complex and complete. That may be possible in some specific domains only. When the user is not clearly in one of the domains for which the semantic processing is possible, AI-based mechanisms are the option to handle the information processing.

Each one of these is suitable for different situations of the user whose context is being processed. For example, if the user is in a well determined situation and the context information requested from external application can be mostly obtained directly from the lower level context information, a database computing is enough to provide a valid response.

On the other hand, if the requested context information is very complex to obtain because it involves information from different domains, several users, or the situation of the user is not properly determined with the available low-level context information, a semantic processing is required in order to provide a valid response. Semantic reasoning will be enough when the kind of inference required is based on conceptual similarity, but when the habits of the users are important for the application, an AI processing could be necessary.

The point is that not all of the context processing mechanisms are valid for all the situations, and, moreover, the consumption of processing and memory resources is very different among them. Based on the different context information processing levels, a global multi-level context system is obtained. By processing level it is meant a specific processing algorithm or mechanism running at the Context Enabler that can be used for a given context monitoring session.

The following algorithm is proposed at the Context Enabler in order to optimise the number of context monitoring sessions (basically the users whose context can be processed and progressed to context consumer applications), and minimising the rejected monitoring session requests, by managing the sessions that are processed at each processing level.

The algorithm is based on a set of Markovian states, presented in Figure 5, each one of them associated to a given monitoring level. Based on this, each Markov state will have a given CPU and memory weight, given by the specific algorithm associated to the state.

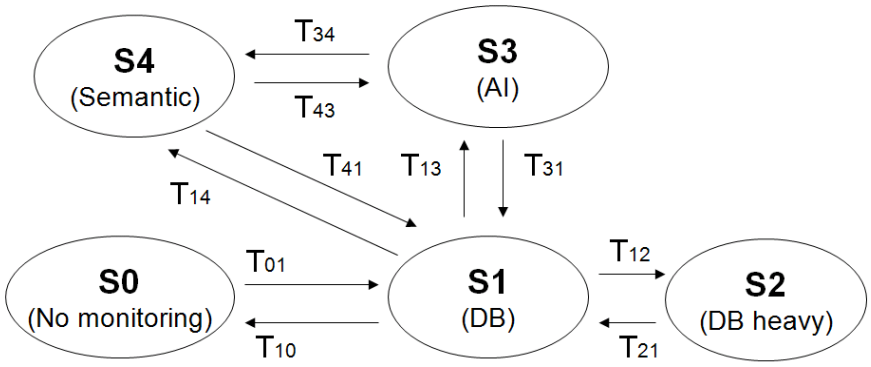

Figure 5: Markovian states associated to different context processing levels, with transitions probabilities Tij for transition to state i to j

The algorithm is relatively simple:

1. Initially, all users are in state *S0* (no monitoring). That means that there is no CPU or memory consumption due to processing activity.
2. For all users:
    a. If a context consumer subscribes to context notifications for user *A*, promote user *A* to state *S1*.
    b. If the number of subscriptions of context consumers providing contextual reports for user *A* increases over a configurable parameter *C*, promote the user *A* according to Figure 5.
    c. In addition, if the number of context sources available for user *A* reaches or exceeds a configurable number *P*, wait a time *T* (on the order of one minute) and if the number of context sources has not decreased below the limit, promote the user according to Figure 5. Such delay will help to avoid impacts on available CPU and memory due to transient effects that are over quickly, due to for instance, user mobility, etc.
    d. If the number of context consumers for user *A* decreases below the configurable parameter *C* and the number of context sources decrease below the parameter *P*, decrease the monitoring level for user *A* based on Figure 5.
3. Periodically, promote users in state *S0* to *S1* to perform security context monitoring even though there may be no context consumer applications subscribed to the given user.

There is a variety of processor assignment algorithms [Brunstrom, 95], but they usually lack of a subscriber's sense. In the present system, the information associated to a subscriber will need to be processed with a different mechanism depending on intrinsic aspects of the information itself (the situation of the user will provoke using a specific type of information processing). Accordingly the subscriber's processing session may migrate fropm one processing mechanism to another. This particularity is

usually not foundin the existing processor assignment algorithms, that seek to optimise the processing load of the existing processors with respect to each arriving process. The algorithm presented in this paper has the key advantage of including the possibility of reassigning a specific information processing session (a monitored subscriber) if his situation or the global workload is modified.

The algorithm presented has undergone a series of simulations in order to be validated. During these simulations a distribution of users with different mobility patterns has been considered. This mobility pattern is the average one observed in a Urban residential environment in a regular working day as per the available mobility logs from commercial cellular access networks:

- 10% of the users have high mobility during most of the day. These would typically be workers in mobility.
- 70% of the users have normal mobility pattern during the day. That is mobility during the usual time windows to commute to the office in the morning and go back home.
- 20% of the users have low mobility pattern. These users spend most of the day at home (elderly, students, etc.) with sporadic short mobility during the day.

The main configuration parameters for the simulation process are depicted in Table 1.

Parameter	**Value**
Number of users in the simulation	100
Maximum number of context reports generated by context sources per hour for high mobility users	0.75
Maximum number of context reports generated by context sources per hour for medium mobility users during high mobility periods	0.75
Maximum number of context reports generated by context sources per hour for high mobility users during medium mobility periods	0.4
Maximum number of context reports generated by context sources per hour for high mobility users during low mobility periods	0.25
Maximum number of context reports generated by context sources per hour for low mobility users	0.25
Number of context reports from context sources to produce a state change	6
Time (seconds) after which the received reports are discarded to produce a state change for a user	30
Time (seconds) after which, if there is no received context information from providers, the state change is decreased	40
Time (seconds) after which a user in state *S0* is awakened to check the context	100

Table 1: Configuration parameters employed for the multi-level simulation

The simulations are performed as per these main configuration parameters and the results of number of users and CPU consumed for states *S0* (no monitoring), *S1*

(database processing) and *S4* (semantic processing) are presented in the following Figure 6. The congestion control is activated for a 60% of maximum CPU occupancy for each processing state. The simulation time is 1 whole day, is executed on MATLAB version 7.4.0.287 (2007a) over a WindowsXP, DualCore processor, with 4 GB RAM.

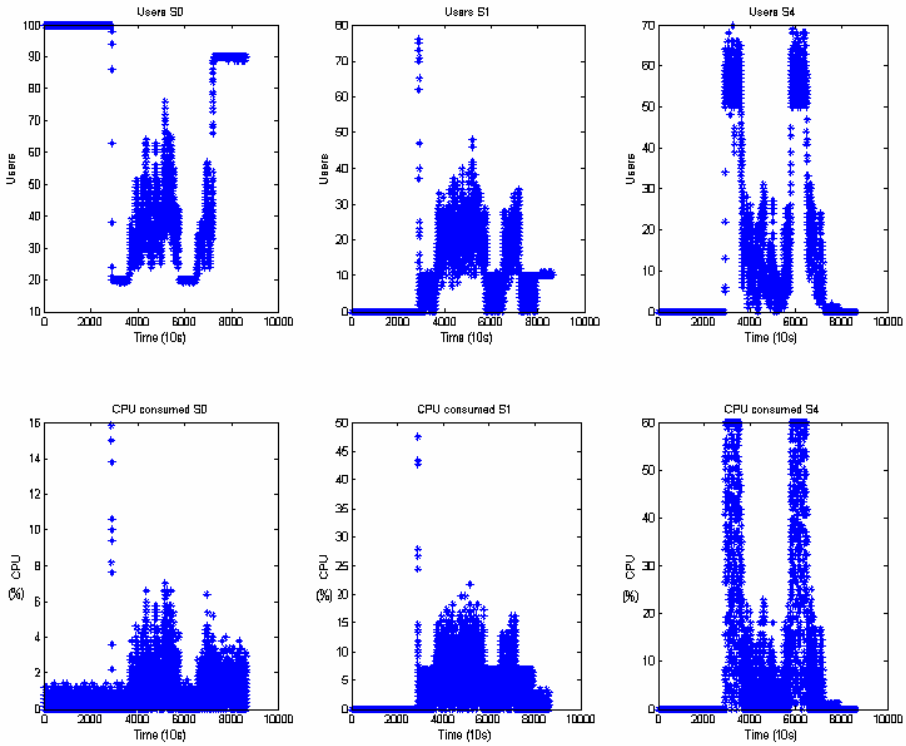

Figure 6: Simulation results for S0, S1 and S4 (number of users and CPU consumed)

The conclusion that can be obtained is that the algorithm is able to handle the contextual monitoring of all the users with minimum congestion (that can be observed only at some points of the CPU consumed by the *S4* contextual monitoring level, in which the 60% of CPU occupancy is reached). All the users are processed with a limited CPU resource (memory consumption is not depicted as no congestion is reached for memory).

In terms of scalability of the algorithm, to fully understand the conclusions, it needs to be considered that usually the different levels of processing are executed in separate processing servers. Being imporssible for each individual server to handle all the subscribers, thanks to the algorithm presented in this work, the total processing capacity to be deployed is much smaller in each case, and its use will be much more optimised- The processing capacity will be leveraged across all subscribers and the

number of subscribers that do not access to the processing service is minimised at a global level.

The multi-level treatment of contextual processing is, based on these results, a sensible option to consider for wide area deployments like the ones performed by operators.

5 Conclusions

This paper presents a viable alternative for a context management framework for telco operators, which is ready to be deployed in a real market situation. This architecture presents several advantages against other options (like fully distributed or ad-hoc context frameworks where there is no centralized Context Manager) detailed along this section.

First, the architecture proposed is designed to be owned by a trustable context management entity. All context operations are centralized in this entity, which means that the endpoint serving the context information is well-known. It is not necessary to search for a provider of the required data.

The Context Enabler presented offers all the support operations required in order to make context management transparent for the service developer. The Context Enabler takes care of a lot of operations that no longer have to be performed by the client applications, like configuring sensors, checking the integrity and coherence of the data, AAA, etc. The application developer just has to request or subscribe to the context parameter needed, and the infrastructure takes care of finding the value, retrieval, validation, inference if necessary, authorization and permission checks, etc.

The architecture proposed exhibits sheer processing power and memory for intelligent context inference. Most of the client devices carrying out context-aware operations are limited mobile terminals. They do not have enough power to track and process detailed, fast-changing context data with resource and data intensive algorithms. Even more, most application developers do not have enough knowledge on semantics or AI in order to implement this kind of algorithms themselves. The Context Enabler proposed in this paper takes care of these operations intelligently, and puts the results at the disposal of the clients.

Most remarkably, unlike other context management frameworks that provide just protocols, formats and message exchange patterns, this Context Enabler has been architecturally designed for a real deployment in a telco environment.

Finally, the proposed architecture presents configurable permission and authorization rules. It is possible for the users to select which context information would be available for the rest of the people. For instance, detailed positioning information could be made available for relatives and friends, while the rest of the users are only able to access the name of the city in which the user is located.

In summary, the proposed solution represents an alternative full of potential that could foster the advance of context-aware applications.

References

[Baladron, 08] Baladron, C.; Aguiar, J.; Carro, B.; Sanchez-Esguevillas, A., "Integrating User-Generated Content and Pervasive Communications," Pervasive Computing, IEEE, vol.7, no.4, pp.58-61, Oct.-Dec. 2008.

[Brunstrom, 95] Brunstrom, A.; Simha, R.; , "Dynamic processor assignment in a task system with time-varying load," *Southeastcon '95. 'Visualize the Future'., Proceedings., IEEE* , vol., no., pp.300-306, 26-29 Mar 1995

[Cadenas, 04] Cadenas, A.; Hermida, A.; Arias, A.; Serna, J., "Distributed PBX gateways to enable the hosted enterprise services architecture in a NGN scenario," Innovations in NGN: Future Network and Services, 2008. K-INGN, 2008. First ITU-T Kaleidoscope Academic Conference, pp.203-210, 12-13 May 2008.

[Capra, 03] Capra, L.; Emmerich, W.; Mascolo, C., "CARISMA: context-aware reflective middleware system for mobile applications," Software Engineering, IEEE Transactions on, vol.29, no.10, pp. 929-945, Oct. 2003.

[Charikar, 97] Charikar, M.; Chekur, C.; Feder, T.; Motwani, R., "Incremental clustering and dynamic information retrieval", Proceeding of the 29th Annual ACM Symposium on Theory of Computing, pp. 626–635, 1997.

[González, 08] González, J.M., Cadenas A., Solá O, "Adaptation Middleware to enable Presence and call control for corporate fixed lines: evolution to convergent network over IMS". NGNM 2008, 5th International Workshop on Next Generation Networking Middleware. September 2008.

[Schmidt, 07] Schmidt, M.; Wilde, A.; Schulke, A.; Costa, H., "IMS interoperability and conformance aspects [IP Multimedia Systems (IMS) Infrastructure and Services]," Communications Magazine, IEEE, vol.45, no.3, pp.138-142, March 2007.

[Strimpakou, 06] Strimpakou, M.A.; Roussaki, I.G.; Anagnostou, M.E., "A context ontology for pervasive service provision," Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on, vol.2, pp. 5, 18-20 April 2006.

[Tao, 05] Tao Gu, Hung Keng Pung, Da Qing Zhang, "A service-oriented middleware for building context-aware services", Journal of Network and Computer Applications, Volume 28, Issue 1, January 2005, Pages 1-18.

[Tsarkov, 06] Tsarkov D., Horrocks I., "FaCT++ Description Logic Reasoner: System Description," chapter in Lecture Notes in Computer Science: Automated Reasoning, ISBN 978-3-540-37187-8, Springer Berlin/Heidelberg, 2006.

[Van Kranenburg, 06] Van Kranenburg, H.; Bargh, M. S.; Iacob, S.; Peddemors, A., "A context management framework for supporting context-aware distributed applications," Communications Magazine, IEEE, vol.44, no.8, pp.67-74, Aug. 2006.

[Wang, 04] Wang, X.H.; Zhang, D.Q.; Gu, T.; Pung, H.K., "Ontology based context modeling and reasoning using OWL," Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on, pp. 18-22, 14-17 March 2004.