The Journal of Systems and Software 88 (2014) 138–146

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A context awareness framework for cross-platform distributed applications

C. Ntanos[*], C. Botsikas[**], G. Rovis, P. Kakavas, D. Askounis

Electrical and Computer Engineering Department, National Technical University of Athens, 9, Iroon Polytechniou str., Athens 157 80, Greece

ARTICLE INFO

Article history:
Received 17 August 2013
Received in revised form 5 October 2013
Accepted 10 October 2013
Available online 6 November 2013

Keywords:
Context awareness
Webinos
Cross-platform

ABSTRACT

With the introduction of interconnected cross-platform middleware, a new area of opportunities for ubiquitous/pervasive computing has emerged. Context aware applications can be enhanced to practically and realistically incorporate multiple facets of human–machine interactions in everyday life that are not limited to a device-centered model for deducing context. In this paper, we propose that they can rather extend this model to a human-centered, device and platform independent model, based on a personal distributed application and data cloud ecosystem. For this to be achieved, webinos, a set of web runtime extensions that enable web applications and services to be used and shared consistently and securely over a broad spectrum of converged and connected devices, is used to provide this ecosystem. The webinos Context Awareness Framework described here is accessible to each webinos-enabled application. After strict policy enforcement, it can collect contextual information, either via an automatic mechanism that intercepts native calls made by webinos applications through the various webinos APIs, via an automatic polling mechanism to these APIs, or via custom, application-specific context schema extensions. It can then distribute the contextual information from its own personal cloud storage mechanism, in the form of simple, manageable and intuitive Context Objects, to and from all webinos-enabled devices owned by the same user, or even other, authorized users.

1. Introduction

Context is a broad concept. According to the Merriam-Webster dictionary, context is defined as “the interrelated conditions in which something exists or occurs”. In computer systems, this context involves every measurable metric surrounding a device, its use, or the user himself. Determining and utilizing this context is a key factor in improving human–machine interfaces. Various areas of computer science have been investigating this concept over the last forty years, in order to relate information processing and communications to aspects of the situations in which such processing occurs. We will use a definition that is specific to the context-aware computing field. “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves” (Dey, 2001).

This definition makes it easier for an application developer to enumerate the context for a given application scenario. If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is context. Note that context information is dependent on individual systems, as a type of information might be considered as context information in one system but not in another.

Context can be either a single piece, a combination, or a time series of data, for example, the current state of the compass on a mobile phone, the current state of the ambient light sensor of a laptop computer or the geographic location and the closest Wi-Fi networks. The context can then be used to make an automatic contextual reconfiguration (e.g. increase the brightness of the monitor, enable Wi-Fi connectivity) or enable a proximate selection (e.g. highlight POIs geographically located near the user or print a document on the closest printer).

A ubiquitous computing environment is characterized by a diverse range of hardware (sensors, user devices, computing infrastructure, etc) and an equally diverse set of applications, which anticipate the need of users and act on their behalf in a proactive manner (Ngo et al., 2004). The vision of ubiquitous computing is only recently becoming a reality in a scale that can be practically distributed to end users.

Being context-aware allows software not only to be able to deal with changes in the environment the software operates in, but also being able to improve the response to the use of the software. This means that context-awareness techniques

* Principal corresponding author.
** Corresponding author.
E-mail addresses: cntanos@epu.ntua.gr (C. Ntanos), cbot@epu.ntua.gr (C. Botsikas), askous@epu.ntua.gr (D. Askounis).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.10.018

aim at supporting both functional and non-functional software requirements.

Three important context-awareness behaviors are (Dey et al., 2001):

1 The representation of available information and services to an end user.
2 The automatic execution of a service.
3 The tagging and storing of context information for later retrieval.

A context-aware system must be able to gather context information available from user interface, pre-specified data or sensors and add it to a repository (*Context Acquisition and Sensing*). Furthermore, the system converts the gathered raw context information into a meaningful context, which can be used (*Context Filtering and Modeling*). Finally, the system uses context to react and make the appropriate context available to the user (*Context Reasoning, Storage and Retrieval*).

Some of the main challenges in the area of context-aware computing are:

- The development of a taxonomy and uniform representation of context types.
- The infrastructure to promote the design, implementation and evolution of context aware applications; and a discovery of compelling context-aware applications that assist our everyday interactions with ubiquitous computational services.

2. State-of-the-art

Several cross-device context-aware application middleware systems have already been developed. In their majority, especially the most recent ones, these are web service-based systems. However, there exist a wide variety of middleware systems, developed mainly in the early '00s that does not rely on web services and is not designed to function under web service-based environments (Truong and Dustdar, 2009).

2.1. Non-web service-based context aware systems

1. RCSM (Yau and Karim, 2004) is a middleware supporting context sensitive applications based on an object model, where context-sensitive applications are themselves modeled as objects. RCSM supports situation awareness by providing a special language for specifying situation awareness requirements. Based on these requirements, application-specific object containers for runtime situation analysis will be generated. RCSM runtime system obtains context data from different sources and provides the data to object containers, which conduct the situation analysis.
2. The JCAF (Java Context Awareness Framework) (Bardram, 2004) supports both the infrastructure and the programming framework for developing context-aware applications in Java. Contextual information is handled by separate services to which clients can publish and from which they can retrieve context. The communication is based on Java RMI (Remote Method Invocation).
3. The PACE middleware (Henricksen and Robinson, 2006) provides context and preference management, together with a programming toolkit and tools for assisting context-aware applications to store, access and utilize contextual information managed by the middleware. PACE supports context-aware applications, in order to make decisions based on user preferences.
4. The GAIA project is a CORBA-based middleware, supporting active space applications (Roman et al., 2002). GAIA middleware provides a context service to support applications to retrieve as well as to publish contextual information.
5. CAMUS is an infrastructure for context-aware network-based intelligent robots (Ngo et al., 2004; Kim et al., 2005). It supports various types of context information, such as user, place and environment, and context reasoning. However, this system is not based on Web services and it works in a closed environment.
6. SOCAM is a middleware for building context-aware services (Gu et al., 2005). It supports context modeling and reasoning based on OWL. However, its implementation is based on RMI.
7. Citron is a context information acquisition framework for a personal device (Yamabe et al., 2005). It captures context information about a user and his surrounding environment; and the information is used to adapt the behavior of applications running on the personal device.

2.2. Web service-based context-aware systems

1. CoWSAMI is a middleware supporting context-awareness in pervasive environments (Athanasopoulos et al., 2008). CoWSAMI provides a context manager to manage context sources. Contextual information is represented as relations and defined by using context collectors, while context information can be queried.
2. The ESCAPE framework (Truong and Dustdar, 2009) is a web service-based context management system for teamwork and disaster management. ESCAPE services are designed for a front-end of mobile devices and the back-end of high end systems. The front-end part includes components supporting context sensing and sharing that are based on web services and are executed in an ad hoc network of mobile devices. The back-end includes a web service for storing and sharing context information among different front-ends.
3. Han and colleagues present the Anyserver platform, which supports context-awareness in mobile web services (Han et al., 2005). The Anyserver platform uses various types of contextual information, such as device information, networks and application type. However, it does not support direct context sharing and is not fully based on web services.
4. The inContext project (Truong and Dustdar, 2009) provides various techniques for supporting context-awareness in emerging team collaboration. It is designed for web service-based collaborative working environments. inContext provides techniques for modeling, storing, reasoning and exchanging contextual information between web services.
5. CARISMA is a mobile computing middleware, which exploits the principle of reflection to enhance the construction of adaptive and context-aware mobile applications (Capra et al., 2003). The middleware provides software engineers with primitives to describe how context changes should be handled using policies.
6. The AmbieSense looks into the future of the ambient intelligence landscape. Miniature and wireless context tags are mounted in everyday surroundings and situations. Project vision: "Relevant information to the right situation and user" (Kofod-Petersen and Mikalsen, 2005).
7. Hydrogen Context-Framework is a three-layered architecture and a software framework trimmed to the special needs of mobile devices, which is extensible to consider all kind of context information (Hofer et al., 2003).

Data control and privacy are common missing features. Most existing middleware do not help users keep control of their personal data across multiple devices, despite the rise in web applications storing data in the cloud.

3. The webinos platform

With an understanding of what context is and the different ways in which it can be used, application builders can more easily determine what behaviors or features they want their applications to support, and what context is required to achieve these behaviors. However, something is still missing. Application builders may need help moving from the design to an actual implementation. This help can come from a combination of architectural services or features that designers can use to build their applications from. This requirement for a more sophisticated web application middleware motivated the creation of webinos. webinos is an EU funded project aiming to deliver a platform for web applications across mobile, PC, home media and in-car devices. The webinos project has over twenty partners from across Europe spanning academic institutions, industry research firms, software firms, handset manufacturers and automotive manufacturers. The webinos vision is to build a multi-platform, applications platform based on web technology that is fit for purpose, across a wide range of connected devices, taking computing to its next logical step, that of ubiquity. In order to do so, knowing the state of the device and the user at any given time, and making decisions based on that context is crucial. Context awareness and true cross-platform and cross-device applications cannot be achieved without the developer having access to an environment able to synchronize content and application state between devices, adapt to changes in user context and provide standard JavaScript APIs to let web applications access device features. Therefore, webinos provides a cross-platform level of abstraction for procedural calls, but at the same time, incorporate an additional data abstraction layer for use in third party context-aware and context-sharing applications that are webinos-enabled (Vergori et al., 2013; Voulgaris, 2011).

webinos introduces the concept of Personal Zone as an overlay network. It provides a basis for managing the user's devices, together with the services running on them. This also includes personal services the user uses in the Cloud. The Personal Zone supports:

1. Single sign-on, where the user is authenticated to a device and applications, and the device is authenticated to the Personal Zone.
2. Shared model of the context. This covers users, device capabilities and properties, and the environment. It enables applications to dynamically adapt to changes, and to increase usability by exploiting the context.
3. Synchronization across the devices in the zone. This includes support for distributed authentication, as well as personal preferences, and replication of service-specific data, e.g. social contacts, and appointments.
4. Discovery and access to services. This includes local discovery, e.g. of services exposed by the user's devices, whether connected through Wi-Fi, Bluetooth, or USB, as well as remote discovery for services exposed in the Cloud.

Applications operating within a webinos Personal Zone can share the state of the same application on any other device belonging to the same user, by seamlessly sharing context and application data across the Personal Zone. Moreover, the use of Remote Process Calls (RPCs) from the Webinos Run Time (WRT) to the exposed APIs of other devices within the Personal Zone will allow the developers to expand the functionality of their applications. The context descriptions can cover a broad range, including but not limited to, device capabilities, network access, user identity and preferences, location, behaviorally induced properties and finally the more complex issue of the users' social network context and social media engagement. Given that security and privacy are primary goals, a common distributed policy architecture and authentication model is implemented.

4. Context Acquisition in webinos

In webinos, the fundamental construct to hold a piece of contextual data is the *Context Object*. Specifically, "a Context Object is a unit for carrying data that uniquely defines a piece of contextual information" (Vergori et al., 2013). The webinos Context Framework provides access to contextualized data, in order to enable the design and operation of context-driven webinos applications and services. The framework is responsible for collecting and storing context data, through the identification of specific context related events that happen within webinos enabled devices. It enhances applications with a layer to access such data, either by querying against the storage or by being notified in real time for context changes, when specific events happen. A context-aware application either makes use of a single piece, a combination, or a time series of context related data. Such data are, for example, the current state of the compass on a mobile phone, the current state of the ambient light sensor of a laptop computer, or the geographic location and the closest Wi-Fi networks. These data serve in order to make an automatic contextual reconfiguration (e.g. increase the brightness of the monitor, enable Wi-Fi connectivity) or enable a proximate selection (e.g. highlight Points Of Interest geographically located near the user or print a document on the closest printer). In order to take advantage of such functionality, an application developer has to have access to means of acquiring contextual data, storing them, filtering them, combining them and performing commands based on the resulting information. For ubiquitous, distributed and context-aware computing applications, the aim is to provide appropriate middleware that can perform Remote Process Calls (RPCs), while at the same time introducing an abstraction layer that will facilitate the development process, by hiding the heterogeneity of the networking environment, supporting advanced coordination models among distributed entities and making as transparent as possible the distribution of computation. The webinos project aims to provide a cross-platform level of abstraction for procedural calls, but at the same time, incorporate an additional data abstraction layer for use in third party context-aware and context-sharing applications that are webinos-enabled.

The scope by which the Context Manager and Context API encompasses all the necessary activities that pertain to gathering, storing and making available context within webinos. Specifically, the following objectives are being addressed:

- Identification of Context Data relevant to the webinos operation.
- Detection of Context Data within webinos, through the webinos entities or related Context Sources.
- Acquisition of Context Data.
- Representation of Context Data within webinos.
- Storage of Context Data within webinos.
- Distribution of Context Data among webinos entities.

Context Data collection can be performed in three ways, namely RPC Interception, a Context Service and Application Context Objects.

4.1. RPC Interception

There is an automatic mechanism that, with the permission of the user via the Policy Manager, can intercept Remote Process Calls (RPCs) made by webinos-enabled applications to the various webinos APIs. The Context Manager transforms each message to a Context Object, by selecting fields that are considered to be

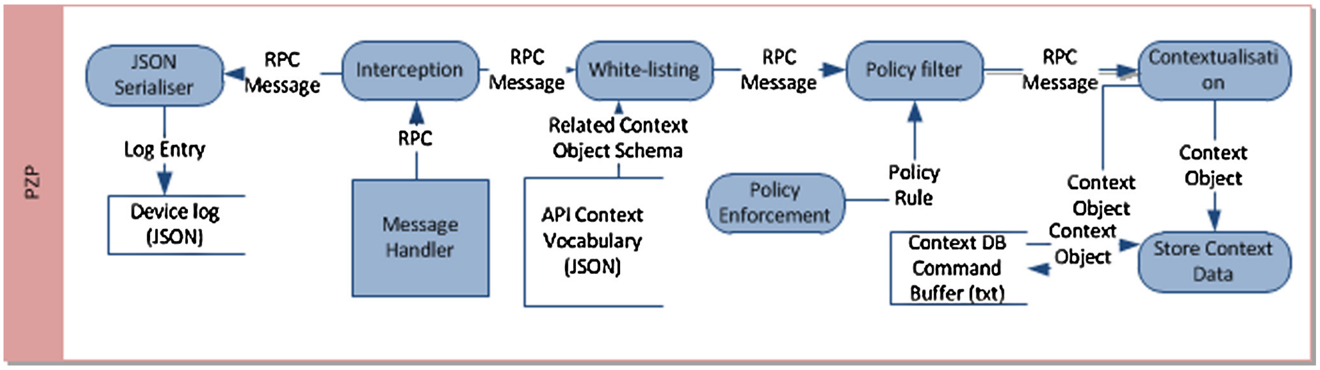

Fig. 1. RPC Interception mechanism.

containing contextual information in the RPC and structuring them as Context Objects, via the extensive API Context Vocabulary. The API Context Vocabulary is a list of structures and rules for the automatic contextualization of intercepted RPC messages.

4.2. Background Service

Context Objects can be registered for periodic background data collection when the Personal Zone Proxy (PZP) is running, by defining Context Rules for the Context Object to be stored, the polling interval, the conditions and the collection method.

4.3. Application Context

Context Objects can be defined and stored independently by any application. An application can request the registration of a new custom Context Object and request permission from the Policy Manager to start storing these Objects to the Context DB. The application can define its own process to obtaining the data for these objects, as well as the frequency and their lifetime. An application developer can make use of the webinos Application Context Vocabulary to define these custom rules and structures for storing application-specific Context Objects, or ones that are derived by any process or combination of preexisting or new contextual data.

5. Context Manager/API Functionality

The Context Manager's key functionality can be summed up in the following points:

- It automatically keeps a log of all RPC calls, stripped from the parameters and results to the PZP in a locally stored log.json file.
- It contains the Webinos API Context Vocabulary, which is a description of the APIs, their methods exposed to the WRT, the expected structure of their parameters and the expected structure of their results, structured under Context Objects that would produce similarly formed contextual information. For example, MyLocation is a the same Context Object irrespective of whether it is produced by Geolocation API's getCurrentPosition method, the watchPosition method or from a call to the Vehicle's API GPS functionality.
- It automatically intercepts RPC calls of API methods.
- It automatically intercepts the registration of API listeners and tracks the callbacks, in order to match them to the corresponding registration call, structuring them as unique RPC calls.
- It automatically looks up every RPC call in the Webinos Context Vocabulary, finds the corresponding match based on the API making the call, the name of the method called and in some cases the parameters used to match the call to a Context Object. The call's input and output data are structured based on the Webinos Context Vocabulary's rules, disallowing data like passwords and other sensitive information that are excluded from any contextual query and are not to be passed to the Policy Manager.
- Context Objects created from RPC calls are stored in a buffer file if the Virtual PZP holding the Context DB is not connected.
- Context Objects created from RPC calls and those included in the context object buffer are sent to the Virtual PZP to be stored in the SQLite3 Context DB.
- It allows the definition of custom Context Objects in a separate Webinos Application Context Vocabulary directly from the WRT.
- Applications can request their own uniquely defined Context Objects to be stored in the Virtual PZP Context DB using the same structure as all other context data.
- The data in the Context DB can be queried from the WRT using a custom Context Querying structure that allows the description of simple or complex queries containing sub-queries as related to Context Objects, treating the database as descriptive of Object Models. The results are returned as Context Objects.
- Polling Rules can be registered. In these, an API call can be scheduled in predefined intervals and the result is automatically intercepted by the Context Manager.
- Context Event Listeners can be registered. A rule can be applied and when a specific condition is met and a Context Query returns a result, that result will trigger an event to the application that registered the listener.
- Integration with the Policy Manager gives permissions to APIs and applications, to store Context Objects in the Virtual PZP Context DB.
- Integration with the Policy Manager gives permissions to applications to run context queries to the Virtual PZP Context DB.
- Integration with the Policy Manager gives permissions to applications to create and enforce Context Rules.

The RPC Interception mechanism is displayed in Fig. 1. When an RPC is sent, the Context Manager intercepts it, stores a copy of selected relevant RPCs to the log and initiates the *Contextualisation* mechanism. The API Context Vocabulary is queried and depending on the method and the API for which the call was made, the appropriate Context Object is created. If there is a policy rule that allows for the specific Context Object to be stored, it is either sent to the vPZP via the PZH (Personal Zone Hub) or a buffer, depending on the connection status, to be stored on the Context DB.

The Application Context Objects are described in a local flat file similar to the API Context Vocabulary, the Application Context Vocabulary (Fig. 2). If an application is requesting to either create a new Application Context Object or retrieve one, the Policy Manager is called and, depending on the permissions, is allowed to do so. In the same figure, the querying mechanism is displayed, where, either when a context event is fired, or an application requests data

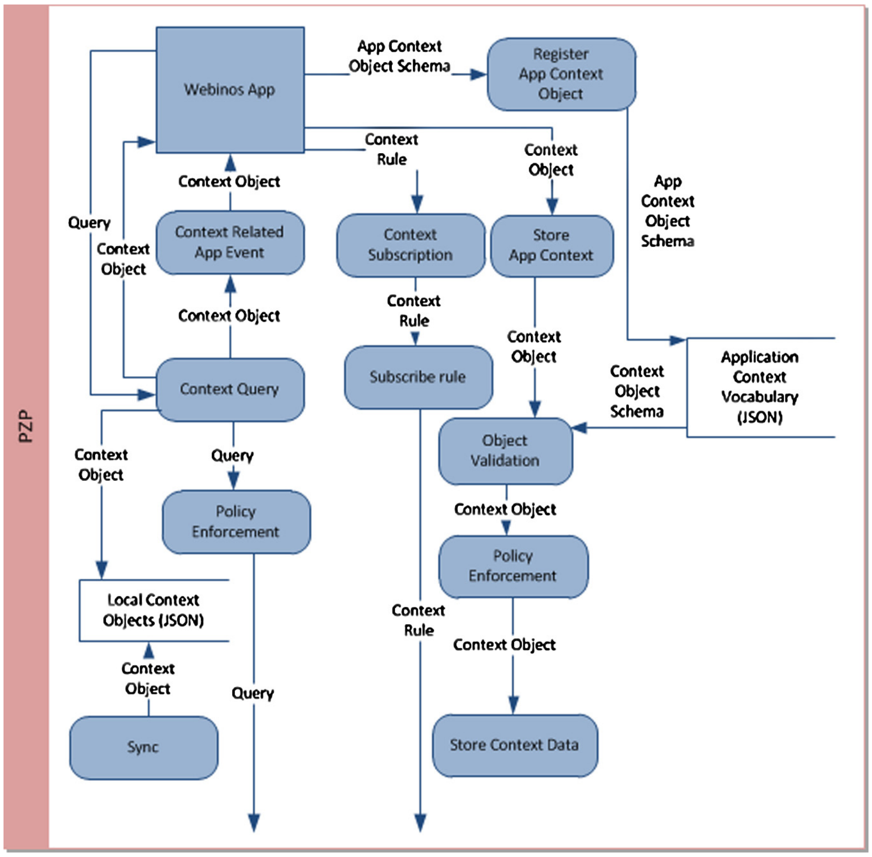

Fig. 2. Application Context, Context Queries and Context Rule subscription.

via a Context Query, the Context DB is queried. Furthermore, the mechanism by which a *Context Rule* is subscribed is displayed.

The Context Background Service (Fig. 3) handles the execution of the polling rules. The service is triggering API calls in intervals described by the rules. With this, it is also triggering the automatic RPC Interception mechanism. These rules are stored in a flat file contained in the file system of the devices for which the rules are applicable. These rules are synchronized with the ContextDB.

The Virtual PZP, which holds the Context DB, is connected via the Personal Zone Hub (PZH) to each PZP in the Personal Zone. The Context Manager service for the Virtual PZP is shown in Fig. 4. The Context Rules, Context Objects, Context Subscriptions and the raw data from the RPC interceptions are stored and retrieved from there. The data that is to be stored locally in each corresponding PZP can be handled by the webinos Sync Manager. These include Subscription rules, polling rules and local Context Objects.

6. webinos APIs captured

The webinos APIs (webinos, 2013) that are included in the Context Manager automatic mechanism for capturing contextual information via RPC Interception have been decided to include the following, by careful consideration of several factors. The APIs should not involve overly sensitive or private information, the data captured should not interfere in security or policy sensitive operations and the selection should minimize the amount of data stored, depending on the relevant contextual value and potential usage.

1. *Media Content*: This API provides access to multimedia contents and relative information. The Context Manager captures most of the data involving the display or playback of media on all devices, but not the media content itself.
2. *Generic Actuator*: The webinos Generic Actuator API provides applications with an API to control actuators in the device, connected to the device or in another device. The Context Manager captures the events relating to the type of the actuator called and the action performed.
3. *Web Notifications*: The webinos Web Notifications specification provides an API to display notifications to alert users outside the context of a web page. The Context Manager captures the type of the message as well as the message displayed, if not flagged as confidential.
4. *Device Interaction*: This module provides a mechanism to interact with the end-user through features such as *device vibrator*, *device notifier*, *screen back-light*, *device Wallpaper*. The Context Manager captures all events that involve settings, such as the change of the screen back-light level and exclude irrelevant data, such as the activation of the device vibrator at a specific instance.
5. *Device Status*: This API uses a kind of tree model to allow developers to get at the various bits of data about things on a device. The information retrieved and the data captured by the Context Manager include the battery status, status of the Wi-Fi connectivity, phone state.
6. *Navigation*: The webinos Navigation API provides a mechanism to interact with in-car navigation software. The API exposes the functionality to interact with a satellite navigation service. The Context Manager captures navigation events involving the selection of a point-of-interest as destination and the addresses of selected destinations.
7. *Generic Sensor*: The webinos Generic Sensor API provides web applications with an API to access data from sensors in the device or in another device. As this API is agnostic to the underlying low level methods for sensor discovery and communication with

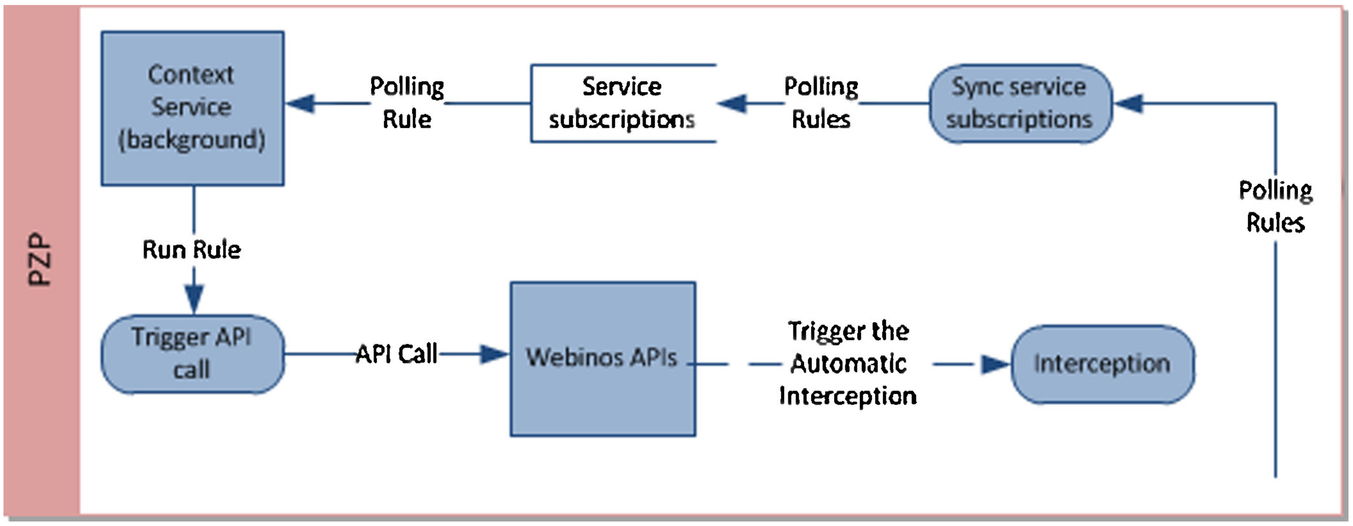

Fig. 3. Context Background Service.

sensors, the Context Manager captures data for any sensor for which a call has been made.

8. *TVControl*: The interface provides a means to acquire a list of TV sources, channels and their streams. The TV channel streams can be displayed in an HTMLVideoElement object. Alternatively the API provides a means to control channel management of the native hardware TV, by allowing to set a channel or watch for channel changes that are invoked otherwise. The Context Manager captures channel change events and changes in sources and streams.
9. *Vehicle*: The webinos vehicle API provides access to specific vehicle data that are captured by the Context Manager. These events include fuel status, steering wheel position, gear changes, climate settings, lights, park sensor position, wiper status, engine oil status and seat position.

The W3C referred APIs used by webinos that are captured by the Context Manager are:

1 *DeviceOrientation Event*: This specification defines DOM events that provide information about the physical orientation and motion of a hosting device. The Context Manager captures the events that involve changes in the device orientation, when these are called by an application.

2 *Media Capture and Streams*: This specification defines a set of JavaScript APIs that allow local media, including audio and video, to be requested from a platform. The Context Manager captures most of the data involving the display or playback of media on all devices, but not the media content itself.

3 *Geolocation*: This specification defines an API that provides scripted access to geographical location information associated with the hosting device, either via a GPS receiver, IP address, RFID, Wi-Fi and Bluetooth MAC addresses, and GSM/CDMA cell IDs, as well as user input. The Context Manager captures events that relate to the current location, speed, altitude, direction and accuracy of the data.

7. Storage of Context Objects

Context Objects are primarily stored securely in the Context DB located in a Virtual PZP, which is reached via a channel opened by the PZH. A Virtual PZP is a very simplified PZP with no WRT, but only

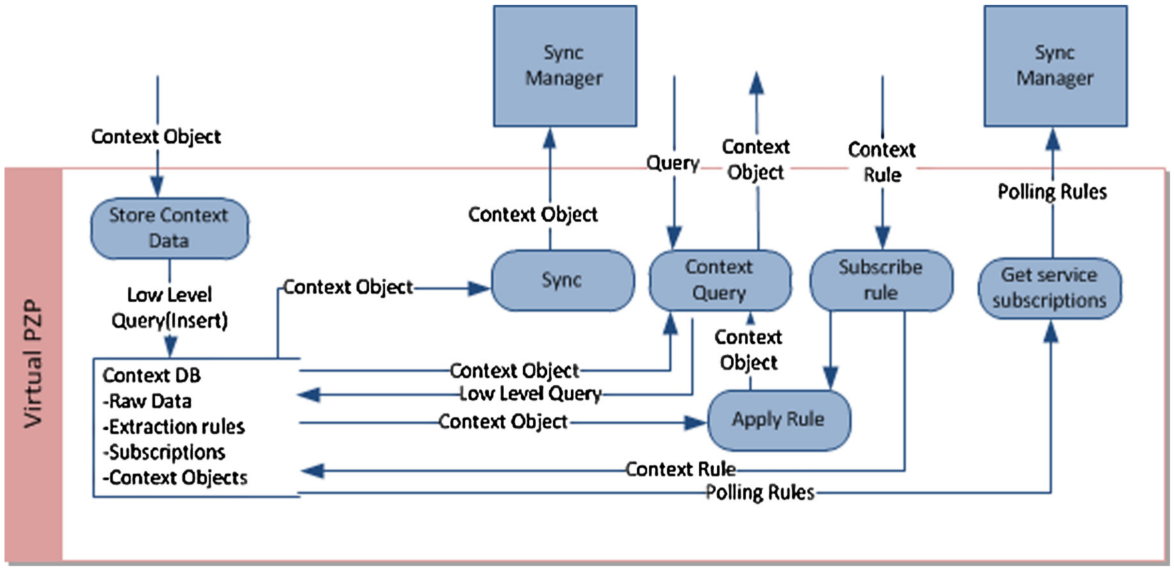

Fig. 4. Context DB and Virtual PZP Context Service.

```
{ "APIname" : "Name of API",
  "URI" : "http://www.webinos.org/APIName",
  "ContextObjects" : [{
      "objectName" : "MyContextObject",
      "methods" : [{
          "Errors" : [{
              "logged" : false,
              "objectName" : "Error",
              "type" : "object",

              "values" : [{
                  "logged" : false,
                  "objectName" : "Error Field",
                  "type" : "data type"}]
      }],
      "inputs" : [{
          "logged" : true,
          "objectName" : "",
          "required" : false,
          "type" : "array",
          "values" : [{
              "logged" : true,
              "objectName" : "",
              "required" : true,
              "type" : "Object",
              "values" : [{
                  "logged" : false,
                  "objectName" : "timeout",
                  "type" : "long"}]
            }]
      }],
      "objectName" : "method name",
      "outputs" : [{
          "logged" : true,
          "objectName" : "output name",
          "type" : "object",
          "values" : [{
              "logged" : true,
              "objectName" : "An output value",
              "type" : "double"}]
        }]
    }],
   "otherMethods" : [{
      "Errors" : [],
      "inputs" : [],
      "objectName" : "A non context method",
      "outputs" : []}]
}
```

Listing 1. API Context Vocabulary.

exposes specific webinos services. The Virtual PZP that holds the webinos Context DB exposes the context services that are related to CRUD and more complex operations through the Context API. The Context DB contains data from across the devices and applications in a Personal Zone (PZ) and each database is unique for that PZ. Querying the Context DB is achieved through a simple to use dedicated query builder that allows the treatment of the Context DB as an Object-Oriented Database, focusing on its main construct, the Context Object. The developer can perform queries directly to the Context API, with the prospect of acquiring any type of Context Object, created by any API, any application and any device across the user's PZ.

7.1. Context Vocabulary

The Context Vocabulary is based on a structure that defines context objects in relation to the APIs they belong to. In this sense, the structure starts from the definition of the API and the data required to recognize it, then, the name of the Context Object captured. Finally, it ends with the structure of the methods and the data relating to either the capturing of data, as part of a Context Object, or the expected values for validation of a specific method. A generic example of this structure is as follows (Listing 1).

The Context Vocabulary holds the name of the API from where the Context Object was captured, the *URI* of that API in order to recognize the method that was called and the methods that describe the specific Context Object. As with all the method calls in webinos, they include the *inputs*, the *outputs* and the *errors* that are produced by the method. The name of the method is identified by the *ObjectName* data value, while the objects inside "*values*" are described with the same structure of the objects, as they appear in the RPCs of these calls. Any of the outputs not meant to be saved carry the "*logged*": *false*, flag.

8. Context Queries

Querying Context Data is performed via a querying mechanism that translates a high lever Context Query to a low level DB query intended for the context DB. The query builder allows the creation of queries based on the following clauses:

- *eq*: Equals
- *lt*: Less than
- *le*: Less or equal than
- *gt*: Greater than
- *ge*: Greater or equal than
- *starts*: Starts with
- *ends*: Ends with
- *in*: In the given list. The value must be and array.
- *contains*: Value contains the given value, only applicable to DOMString

These clauses are provided as an API Database Abstraction layer over any webinos supported database installed on the Virtual PZP holding the Context DB. The abstraction layer can be accessed via the Context API by any application that is authorized by the Policy Manager to either store or extract data from the Context DB.

9. Policy enforcement

It is very clear that users are not very good at understanding the future value of keeping personal information private and are often quick to share the ownership of such information without evaluating the impact of its possible uses (Norberg et al., 2007). With this in mind, the webinos platform ensures that the ownership of contextual information stays with the user, while access rights to applications to store, extract or query context data can be given by the user to the application and not the application developer. This allows the developer to build applications that can utilize Context Objects that are stored by the webinos platform or other applications in the same PZ. In order to further secure the privacy of personal data, all transactions with the Context DB are monitored by the webinos policy manager and specific access rights to read/write, to and from the Context DB are provided per application, per type and per source of Context Objects.

Additionally, with the advancement of the Policy Manager and the ability to create more complex policy rules, each Context API call will have a number of policy events relating to it in the following axes:

- Context API Method
- Read/Write
- Application making the request
- Data Object (Context Object, rule to be added/updated, etc)

10. Scenario: extending the Cardio Hills Remote Sensors Demo

The webinos enabled heart rate demo shows how a user can download a webinos application on a smartphone and by wearing

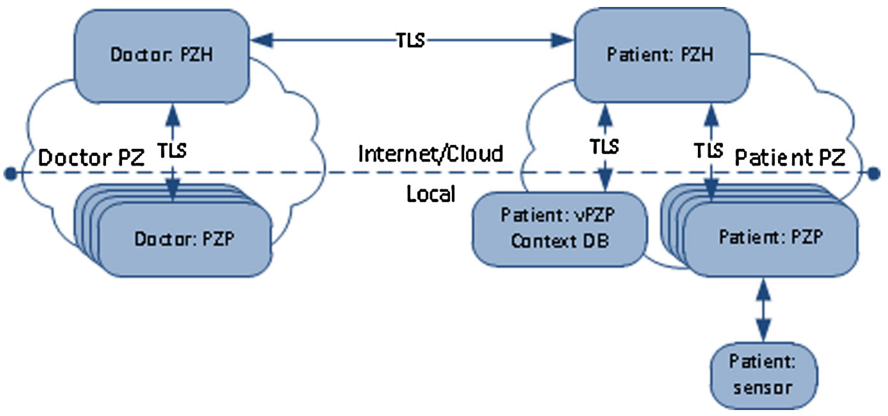

Fig. 5. Heart rate monitor sharing data flow.

a heart rate monitor track his health throughout his day (webinos, 2012).

The webinos heart rate demo shows one simple example of how the webinos technology and APIs can allow the user to extract data from many different types of sensor devices, such as motion sensors, remotely; in this case the sensor being a heart rate monitor.

Lots of mobile phone applications exist that take advantage of heart rate monitor data, combining them with GPS data to represent the user's sporting activities. The main problem with this kind of applications is that the data remains in the mobile phone. The other option is to export the data manually, share it with a trainer who will have to import it in his application and then make his recommendations (Botsikas, 2012). If an application developer wishes to simplify this process without using webinos, he has to implement a storage system, for both the patient and the doctor/trainer, and the communication of the application with other devices using protocols like Bluetooth, NFC or TCP/IP. Even in this scenario, other applications will not have access to that data and any access to them will be restricted to the application having to be executed.

The webinos Cardio Hills application makes use of both *Application Context* and the *API Context* functionalities of the webinos Context API. The application defines a set of Application Context Objects (e.g. MyPulses), containing data from the external sensor, including a timestamp. It also sets up the capture of geolocation data via the Background Service, which is polling the mobile device's GPS sensor through the webinos Geolocation API. The resulting Context Objects from the contextualization via both the API and Application Vocabularies are synchronized with the Context DB on the Virtual PZP. These data are then available on their generic, fully platform-independent form of Context Objects to the Cardio Hills application across all PZPs on the same Personal Zone. This means that plotting a map with the route an athlete ran, alongside his heart rate, on any device requires only the query of the Context API for these two Context Objects for the device where the heart rate monitor was connected, between two timestamps (using the *le* and *ge* clauses) and simply putting them on the map.

Using the webinos *Notification API*, the *Context API* allows the application developer to set up an event trigger that leads to the display of a native notification (depending on the platform) that has a contextual significance. For example, the training program a user follows can have a caloric goal. By combining Context Objects referring to the user (e.g. age, sex, height, weight) and others that refer to the activity (e.g. activity start time, Geolocation data, including coordinates, terrain undulations and timestamps, heart rate sensor data and weather conditions), the application developer can create a background collection for an Application Context Object (e.g *MyTrainingSession*), using the Background Service. This Context Object contains a real-time calculation of the total calories burnt during the session. Then, a Context Event Trigger can easily be set up to post a notification when the session's caloric goal has been reached. Tapping on the notification can open the Cardio Hills application, using the webinos *AppLauncher API*, and immediately display the results of the session, since the *AppLauncher* executes the Cardio Hills application using the *MyTrainingSession* as part of the execution arguments.

Furthermore, Application Context Objects can be shared between different applications on the same Personal Zone, if the user chooses to allow it through the Policy Manager. This means that, given the appropriate permissions, two separate applications can have access to the sensors' data, one (or more) to capture the data and one to analyze them when at home.

The Cardio Hills application takes this one step further. Using the PZH to PZH communication and the cross-PZ functionality provided by webinos, the user can choose to share these Context Objects with any people he selects. webinos gives the user the ability to control his own privacy profile and to allow only those people he wishes to be monitoring this part of his contextual data for specific time spans. A doctor with a webinos-enabled device that aggregates and analyzes biosensor data can collect all relevant data, including multiple Application Context Objects from different sensors apart from the heart rate monitor (e.g. blood pressure, oxygen saturation), other wearable health care monitoring systems, Geolocation API data, etc. The data can then be used to enrich a patient's health profile by using preexisting historical data already on the user's Personal Zone.

Data stored on the Context DB and served via the Context API in the Personal Zone cloud are completely open to the user. This has allowed webinos to create a new paradigm of shared knowledge between different applications and even different vendors, by minimizing the complexity of application interoperability within webinos. Simultaneously, by centralizing the collection of Context Objects on each Personal Zone, the data from the webinos APIs can be stored without being tied on any specific application or device. Then, given the user's permission, these data can be queried by any application that can make use of them, from any device in the user's Personal Zone, or any other user's Personal Zone that has been explicitly permitted to do so. Due to the fact that this functionality is embedded in webinos, all these can be achieved with very few, highly intuitive calls to the Context API, the App2App Messaging API and the Event Handling API (Fig. 5).

11. Conclusions/discussion of future work

The Context Manager and Context API for the webinos platform have been described, as they stand at the moment, in this paper. Both the available literature on the subject of pervasive computing as well as various predicaments faced by the webinos team have been taken into consideration. A notable limitation is the innovative nature of webinos as a whole, whereas the potential of making use of existing technologies is limited by the features of the devices and the OSes that webinos is compatible with. Limitations have also been risen by the overall aim for a very lightweight platform, the lack of compatibility of many solutions with the webinos ecosystem and the platform instability as it has only recently entered its beta stage. Many of the potential advancements that have been proposed are being halted in order to be reevaluated when necessary components have matured to a fully functional state.

Future research is aimed at the distinction according to relevance of contradicting Context Objects and the automatic garbage collection policy for the Context DB. The relevance of Context Objects should be defined by recognizing the identity of the active device, namely the device currently operated by the user (e.g. the Geolocation API call for the device currently or lastly operated by the user, the tablet when operated instead of the mobile phone in the pocket), or the preferred device for a specific task for a specific context (e.g. the living room TV, instead of the bedroom TV, the vehicle navigation system instead of the mobile phone when driving). The relevance of each device should itself be deduced by use of the Context Manager before returning data or triggering an event to a specific device. Furthermore, the automatic garbage collection policy should delete or compact data in the Context DB, for example the summation of Geolocation data to single Context Objects defining the locations and the timestamps between which the user remained at these locations, after a specific number of days have passed, or the aggregation of TV channels the user was tuned in for more than half an hour, instead of holding all ChannelChange events.

Another future development is the potential use of MongoDB (Mongodb, 2013) or other NoSQL databases for more easily scalable and lightweight database management than the currently used SQLite3 database, while at the same time providing a more developer-friendly approach to the dynamic horizontal scaling required by the ambiguity of the Context Objects stored.

Overall, the webinos Context Manager and Context API have been described, in their current state, in this paper. The main objectives reached by this approach is an innovative and intuitive interface by which a webinos-enabled application's developer can easily access the user related contextual information relevant to his application, by minimizing the effort for context acquisition and filtering in this highly innovative platform-independent personal application and storage cloud ecosystem.

References

10gen, 2013. Mongodb. http://www.mongodb.org/

Bardram, J.E., 2004. The java context awareness framework (jcaf) – a service infrastructure and programming framework for context-aware applications. A service infrastructure and programming framework for context-aware applications., pp. 98–115.

Botsikas, A., 2012. Heart Rate Monitor - Sharing Data & Services with your Friends. http://www.webinos.org/blog/2012/08/17/heart-rate-monitor-sharing-data-services-with-your-friends

Capra, L., Emmerich, W., Mascolo, C., 2003. Carisma: context-aware reflective middleware system for mobile applications. IEEE Transactions on Software Engineering 29 (10), 929–944.

Dey, A., Abowd, G., Salber, D., 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human–Computer Interaction 16 (2), 97–166.

Dey, A.K., 2001. Understanding and using context. Personal and Ubiquitous Computing 5 (1), 4–7.

Athanasopoulos, Dionysis, Zarras, Apostolos V., Issarny, Valerie, Pitoura, Evaggelia, Vassiliadis, Panos, 2008. Cowsami: interface-aware context gathering in ambient intelligence environments. Pervasive and Mobile Computing 4 (3), 360–389 http://www.sciencedirect.com/science/article/pii/S1574119207000740

Gu, T., Pung, H.K., Zhang, D.Q., 2005. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications 28 (1), 1–18.

Han, B., Jia, W., Shen, J., Yuen, M.-C., 2005. Context-awareness in mobile web services. In: Cao, J., Yang, L., Guo, M., Lau, F. (Eds.), Parallel and Distributed Processing and Applications. Vol. 3358 of Lecture Notes in Computer Science. Springer, Berlin Heidelberg, pp. 519–528.

Henricksen, K., Robinson, R., 2006. A survey of middleware for sensor networks: state-of-the-art and future directions. In: Proceedings of the International Workshop on Middleware for Sensor Networks, pp. 60–65.

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., Retschitzegger, W., 2003. Context-awareness on Mobile Devices – The Hydrogen Approach, 10 pp.

Kim, H., Cho, Y.-J., Oh, S.-R., 2005. Camus: a middleware supporting context-aware services for network-based robots. In: IEEE Workshop on Advanced Robotics and its Social Impacts, 2005, pp. 237–242.

Kofod-Petersen, A., Mikalsen, M., 2005. Context: Representation and Reasoning: Representing and Reasoning about Context in a Mobile Environment. Revue d'Intelligence Artificielle on Applying Context-Management.

Ngo, H.Q., Shehzad, A., Liaquat, S., Riaz, M., Lee, S., 2004. Developing context-aware ubiquitous computing systems with a unified middleware framework. In: Yang, L.T. (Ed.), In: Embedded and Ubiquitous Computing, vol. 3207. Springer, Berlin and Heidelberg/New York, pp. 672–681.

Norberg, P., Horne, D., Horne, D., 2007. The privacy paradox: personal information disclosure intentions versus behaviors. Journal of Consumer Affairs 41 (1), 100–126.

Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., Nahrstedt, K., 2002. A middleware infrastructure for active spaces. IEEE Pervasive Computing 1 (4), 74–83.

Truong, H.-L., Dustdar, S., 2009. A survey on context-aware web service systems. International Journal of Web Information Systems 5 (1), 5–31.

Vergori, P., Ntanos, C., Gavelli, M., Askounis, D., 2013. The webinos architecture: A developer's point of view. In: Uhler, D., Mehta, K., Wong, J.L. (Eds.), In: Mobile Computing, Applications, and Services, vol. 110. Springer-Verlag/New York Inc, pp. 391–399.

Voulgaris, G., 2011. webinos Factsheet. http://www.webinos.org/content/webinos-factsheet.pdf

webinos, 2012. Cardio Hills – Remote Sensors Demo. http://www.webinos.org/wp-content/uploads/2012/02/webinos_demo_flyer_cardiohills.pdf

webinos, 2013. webinos Device apis - Working Repository. http://dev.webinos.org/deliverables/wp3/Deliverable34/

Yamabe, T., Takagi, A., Nakajima, T., 2005. Citron: a context information acquisition framework for personal devices. In: Proceedings. 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2005, pp. 489–495.

Yau, S.S., Karim, F., 2004. A context-sensitive middleware for dynamic integration of mobile devices with network infrastructures. Journal of Parallel and Distributed Computing 64 (2), 301–317.

C. Ntanos holds a Bachelor in Electronic and Computer Engineering and a Master in Electronic and Computer Engineering obtained at the University of Birmingham, UK and also a Master in Business Administration (MBA) acquired after successfully completing an Interdisciplinary Graduate degree between and the Department of Business Administration and the Department of Marketing and Communication of the Athens University of Economics and Business, Greece. He is currently a PhD candidate in the Decision Support Systems Laboratory of Electrical Engineering, National Technical University of Athens (NTUA), Greece.

C. Botsikas received his MSc in Mechanical Engineering from University of Thessaly, Greece. He has been working as Software Engineer and Information Technologist on various projects, since 2003. He has specialized in web technologies spreading from the open-source PHP and MySQL stack to the proprietary Microsoft and Oracle stacks while he has been a strong supporter of the Javascript client and server programming. He has also been involved in various infrastructure projects on both Microsoft and Linux server systems. He is currently working for the National Technical University of Athens, Greece.

G. Rovis is an Electrical and Computer Engineer currently working at Ericsson Hellas S.A. He graduated from the National Technical University of Athens in 2013 with a Master of Engineering (M.Eng.), specializing in Power Electronics. His thesis, titled Modeling Context Data of Personal Information Systems, involves the modeling of contextual information used in the context-awareness framework for the open-source computing platform webinos.

P. Kakavas is a final year graduate student (M.Eng.) of the School of Electrical and Computer Engineering at the National Technical University of Athens, Greece specializing in Power Electronics. His thesis, titled Personal Data Modeling of Information Systems, involves the modeling of the context- awareness framework for the open-source computing platform webinos.

Prof. D. Askounis is an Assistant Professor of Management Information and Decision Support Systems in the School of Electrical and Computer Engineering at the National Technical University of Athens (NTUA). Dr. Askounis has been involved in numerous IT research projects funded by the EU since 1988 (ESPRIT, BRITE-EURAM, FP5, FP6, FP7, CIP) in the thematic areas of eGovernance (CROSSROAD, COCKPIT, LEX-IS, FEED), eBusiness interoperability (e.g. IMAGINE, GIC, PLUGIN, GENESIS, webinos), decision support, knowledge management, quality management, computer integrated manufacturing, enterprise resource planning, etc. Dr. Askounis has published over 80 papers in scientific journals and international conference proceedings.