Decoupling Context Management and Application Logic: a new Framework

Bachir Chihani, Emmanuel Bertin
Orange Labs
42 rue des Coutures,
14066 Caen, France
{bachir.chihani,emmanuel.bertin}@orange.com

Noël Crespi
Institut Mines-Telecom, Telecom SudParis, CNRS 5157
9 rue Charles Fourier
91011 Evry, France
{noel.crespi}@it-sudparis.eu

Abstract—Several frameworks have already been proposed to simplify the development of context-aware applications. These frameworks are focused on collecting and publishing contextual data, and on providing common semantics, definitions and representations of these data. This implies that applications share the same semantics, which limits the range of use cases where a framework can be used since that assumption induces a strong coupling between context management and application logic. This article proposes a framework that decouples context management from application business logic. The aim is to reduce the overhead for applications that run on resource-limited devices while still providing efficient mechanisms to support context-awareness and behavior adaptation. This framework implements an innovative approach that involves third parties in the process of context processing definition by structuring it in atomic functions, and describing it with an XML-based programming language. Its implementation and evaluation demonstrates the benefits, in terms of flexibility, of using trusted design patterns from software engineering for developing context-aware application.

Keywords— Software Engineering; Context-awareness; Adaptation; XML

I. INTRODUCTION

Two architectural approaches [1] have been used to design context-aware applications: architectures based on a broker model and those based on a point-to-point model. Both have context providers for collecting context, and context consumers (i.e. context-aware applications) that use context to adapt their behavior. In the first model, a context broker is used to decouple context providers and consumers, limiting or eliminating direct connections. This broker is in charge of context modeling and inference [1] [2] [3] [4] [5]. However, these tasks are performed in a pre-defined way and are barely customizable by consumers. As a result, the generated information may not fully match the specific needs of consumers. In the second model, context consumers know the providers and send their requests to them directly [6]. This model is less sophisticated, as consumers need to know which provider should be addressed for a given context and should also be aware of their state (e.g. awake or asleep).

In both cases, consumers continuously request contextual information from their sources (either the context broker or directly from context providers) or subscribe to be notified with context updates. Upon reception, this information must then be processed again by the context consumers to determine if it should impact their behavior. When the updating load increases, consumers are overloaded with messages, many of which are not at all relevant to them. Moreover, consumers have to store and handle context information locally to maintain a consistent vision of the user’s situation and to adapt their behavior accordingly.

We have thus identified the need for a better separation within context-aware architectures. The application business logic and the context management operations should be kept separate, so that one of these logics can be modified without having to modify the other. To enable this separation, we propose to host all operations related to context management in the context broker, outside of context-aware applications. Consequently, an appropriate language is needed, in which application developers can specify which actions should be executed by the broker when a specific context is published.

Our contribution is a generic framework, based on the Observer design pattern [7], which is flexible enough to be customized for building context-aware applications in different domains. Herein, flexibility refers to dynamic adaptation, customization and component reusability. The use of the Observer pattern allows the context broker to be treated as an observed component by context consumers. When the broker’s state changes as a result of processing context data, it notifies the consumers interested in these. The framework is able to process context data differently based on the specifications defined by context consumers.

The rest of the paper is organized as follows. Section 2 illustrates our framework architecture and implementation details. In section 3, we present an evaluation of the proposed framework through a real case study and performance evaluation in a simulated environment. Section 4 discusses related works and the proposed framework. Finally, we conclude the paper and present future work in section 5.

II. DESIGN CONSIDERATIONS

The heterogeneity of contextual information (location, presence, temperature, etc.) brings a considerable complexity to their management (provisioning, distribution, use, etc.) which makes the development of context-aware applications a hard task. The pooling of certain management operations (e.g. provisioning) at a specialized element of the overall architecture could reduce complexity of the development of
context-aware applications. However, such an approach would not be enough because the pooled functionality will be performed the same way whatever was the requester (here context-aware application which are represented as context consumers).

We propose a new framework that not only pool some common context management related operations but also provide a way for context consumers to define in a custom way how the pooled function should be performed.

Such framework will take in charge context management to facilitate the development and deployment of context-aware applications. It should address the following challenges:

1. Ease of context manipulation to facilitate the integration of context data with other types of data manipulated by the application.
2. Ease of management of context reasoning to facilitate the updating of already deployed context reasoning mechanisms or replace them.
3. Ease of application development by providing simple mechanisms allowing developers to define how context should be handled.
4. Ease of integration of new elements to allow developers to add/update/remove new context providers/consumers without having to modify their main application code and without redeploying it.

Within this paper we present a framework that addresses efficiently these challenges and compare it to state of the art proposals.

III. CONTEXT MANAGEMENT FRAMEWORK

A. Framework Architecture

We propose a programmable framework (figure 1) for processing contextual information. With the help of an XML language (listing 1) called CPDL (Context Processing Definition Language), context management is decomposed into six primitive functions: produce, filter, abstract, select, aggregate, and consume. Each of these functions corresponds to a specific action from the well-known layered approach [9] in context management.

![Figure 1. A layered framework for context management](image)

The “produce” function consists of producing raw contextual information. It is implemented by context providers that wrap sensors to comply with the framework API for publishing context.

The “filter” function provides signal processing functionalities to reduce noise in context (e.g. temperature). Some examples of the signal processing algorithms that can be selected are Kalman Filter and Simple Moving Average (SMA).

The “abstract” function transforms raw contextual information into a higher level of abstraction. It is implemented via the “FiniteStateMachine” element that defines a finite state machine composed of the different states that context data may have. As a trivial example: the context ‘availability’ may have two states, ‘available’ and ‘not available’. The context ‘availability’ transits between these states following changes occurring at a raw context level, such as ‘presence’, sensed from an instant messaging system. A finite state may transit from one state to another only if some preconditions (on published context, on the state of another finite state machine, or on a timeout) are satisfied. A transition may lead to events, which can be internal, to trigger another finite state machine or an aggregation rule, or external, such as to notify a context consumer.

The “select” function enables the selection of the ‘best’ available context (from multiple sources). It defines the parameters to compare in the selection process (i.e. choosing the maximum or the minimum value). The compared parameter might characterize the quality of a context, such as accuracy and precision. For example, the abstract location (e.g. home, work) of a person can be obtained by abstracting GPS information and also from location information acquired from an instant messaging system. The difference between the two location sources is in the precision, which is high for the first source and low for the second. A tracking application may choose to select a source that provides the maximum available precision.

The “aggregate” function performs an aggregation on a set of contextual information in order to generate a composite context to be exposed to context-aware applications. It is implemented thanks to the ‘Rule’ element that defines a set of conditions to be met by different contextual data, and a set of actions, in the form of events and notifications to a context-aware application. The aggregate function represents an IF-THEN rule that can be triggered by one or more abstract functions when some pre-condition(s) are satisfied.

Finally, the “consume” function is performed by context consumers (i.e. context-aware applications) as they consume the context from any level (raw, abstract or composite) and adapt their behavior accordingly.

Listing 1. Document Type Definition of CPDL

```xml
<DOCTYPE Definition [
  <!ELEMENT Definition (Filter | FiniteStateMachine | Select | Rule)>
  <!ELEMENT Filter EMPTY>
  <!ATTLIST Filter Src CDATA #REQUIRED Type CDATA #REQUIRED Derm CDATA #REQUIRED>
  <!ELEMENT FiniteStateMachine (State*, Start-State, End-State)>
  <!ATTLIST FiniteStateMachine Id CDATA #REQUIRED Name CDATA #REQUIRED>
  <!ELEMENT State (Transition)>
  <!ATTLIST State Name CDATA #REQUIRED>
  <!ELEMENT Transition (Condition | Event)>
  <!ATTLIST Transition Dest CDATA #REQUIRED>]
```
Reasoning rules defined by the context-aware applications to contextual information. As which components (CxC) are allowed to request a user's permission, the Permission Manager (PM) is used to enforce user policies, such as which components (CxC) are allowed to request a user's permission. A configuration manager (CM) manages the configuration file in the application, so that it is uploaded at application installation. Afterwards, the CM component instantiates the reasoning rules corresponding to the configuration file. The rules then become operational and the RE component will apply them every time a new context event is received.

The framework is based on a RESTful architecture where components (providers, consumers and brokers) are implemented as web services. The distribution of context is based on the use of channels also called Context URI (Unified Resource Identifier) that uniquely identify contextual information. The context provider (CxP) chooses a channel on which it will publish the produced context. Providers of the same contextual information (e.g. location) can publish content on the same channel. The CxB contains a mapping table that matches context data and corresponding URIs for requests and subscriptions to context, and another lookup table (table 1) is used to track who is publishing to and who is consuming from a given context channel. A consumer (CxC) can request or subscribe to specific contextual information by contacting the CxB to start listening on the corresponding channel and be notified when new information is published.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Context Providers</th>
<th>Context Consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A= &quot;/[user_id]/presence&quot;</td>
<td>CxP₁, CxC₃</td>
<td></td>
</tr>
<tr>
<td>B= &quot;/[device_id]/settings&quot;</td>
<td>CxP₂, CxC₁, CxC₃</td>
<td></td>
</tr>
<tr>
<td>C= &quot;/[POI]/location&quot;</td>
<td>CxP₁</td>
<td>CxC₂</td>
</tr>
</tbody>
</table>

An example of an aggregation rule is a rule that checks a user’s availability when a new call arrives to decide how it should be handled (e.g. reject the call). If an abstract function is used to abstract SIP (Session Initiation Protocol) communication states into ‘start’, ‘initiated’, ‘established’ and ‘terminated’, then the aggregation rule is triggered when the finite state machine corresponding to the SIP abstract function transits from state ‘start’ to state ‘initiated’ (e.g. after reception of a SIP INVITE message). The rule will check the current user context (e.g. availability) and if the current value is available it notifies the context-aware application. Sent notifications correspond to the appropriate messages described in the CPDL (i.e. event element) for the current situation.

B. Implementation

We implemented these concepts with a context management platform, brokering between context-aware applications (installed on smartphones) and sensors (located either on smartphones or on external servers).

Figure 2 illustrates the implementation: CxPs are Context Providers responsible for the retrieval of contextual information from sources and for sending it to the Context Management Platform (CMP) that is in charge of managing context. Context Consumers (CxC) are applications that consume contextual information and adapt their behavior accordingly. The context Broker (CxB) is responsible for distributing context from sources (CxPs) to its consumers (CxC). A local context broker is used at the device level as a Cache to store contextual information locally (and temporarily until expiration) to speed up future context requests. CxPs installed on user devices should publish their produced context to the local CxB that is in charge of forwarding it to other components [8] [9]. A Configuration Manager (CM) manages the reasoning configuration of files and assures their validation. A rules repository is used to store the reasoning rules. A Reasoning Engine (RE) is responsible for applying the reasoning rules defined by the context-aware applications to the context events generated by updates from the CxPs. A Permission Manager (PM) is used to enforce user policies, such as which components (CxC) are allowed to request a user’s contextual information.

The developer(s) upload a CPDL file describing the context-aware application’s reasoning configuration to the CM component. Another possibility is to embed the reasoning configuration file in the application, so that it is uploaded at application installation. Afterwards, the CM component instantiates the reasoning rules corresponding to the configuration file. The rules then become operational and the RE component will apply them every time a new context event is received.

The framework is based on a RESTful architecture where components (providers, consumers and brokers) are implemented as web services. The distribution of context is based on the use of channels also called Context URI (Unified Resource Identifier) that uniquely identify contextual information. The context provider (CxP) chooses a channel on which it will publish the produced context. Providers of the same contextual information (e.g. location) can publish content on the same channel. The CxB contains a mapping table that matches context data and corresponding URIs for requests and subscriptions to context, and another lookup table (table 1) is used to track who is publishing to and who is consuming from a given context channel. A consumer (CxC) can request or subscribe to specific contextual information by contacting the CxB to start listening on the corresponding channel and be notified when new information is published.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Context Providers</th>
<th>Context Consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A= &quot;/[user_id]/presence&quot;</td>
<td>CxP₁, CxC₃</td>
<td></td>
</tr>
<tr>
<td>B= &quot;/[device_id]/settings&quot;</td>
<td>CxP₂, CxC₁, CxC₃</td>
<td></td>
</tr>
<tr>
<td>C= &quot;/[POI]/location&quot;</td>
<td>CxP₁</td>
<td>CxC₂</td>
</tr>
</tbody>
</table>

The provisioned context corresponds to information like user location, proximity (nearby wifi access point or Bluetooth devices), activity, profile information, device status, etc. It is represented in XML for an easy distribution among the framework components. The following listing represents an example of contextual information about a known/detected...
user location that is made available by the context broker on channel "/p000/location".

Listing 2. A snippet of an XML document representing location context

```xml
<contexts>
  <context type='location' subtype='gps'>
    <meta timestamp='12345' expiration='12347'/>
    <latitude>33.376439</latitude>
    <longitude>6.862087</longitude>
  </context>
  <context type='location' subtype='address'>
    <meta timestamp='23561' expiration='23680'/>
    <street>27 rue aouis</street>
    <city>el-oued</city>
    <postal>39000</postal>
    <country>algeria</country>
  </context>
</contexts>
```

The use of channels in conjunction with an XML-based context representation provides a hierarchical addressing schema allowing the composition of context data. For instance, if Alice’s presence information is published on channel '/alice/presence' and her location information is published on '/alice/location', then Alice’s composite context information (presence and location) can be requested from channel '/alice'. In the same way, if location information is represented in XML and contains an element for GPS coordinates (longitude and latitude) and another element for civil address, then channel '/alice/location/gps' should return the GPS coordinates.

External components have to register within the broker in order to access user context. The registration consists of declaring descriptive information (depending on the type of management operations for the components involved) about the component on a special ‘meta’ channel. For a CxP, the required information includes the type of published context and the entity concerned (e.g. a monitored user or a device) that owns the published context. For a CxC, the required information can be derived from the corresponding CPDL file. The registration information is used by the broker to route context updates from CxP to CxC. Privacy policies are used to regulate this routing. Thanks to this channel-based publish/subscribe messaging mode, the framework has a loosely-coupled architecture in which consumers do not have to know the identity of providers since they get their context directly from the broker.

IV. CASE STUDY

As a case study we consider CoAR (Context-Aware Reachability) an IMS (IP Multimedia Subsystem)-based context-aware communication system [10]. CoAR takes into account user context and preferences in handling incoming calls (e.g. accepting or rejecting a call). Figure 3 depicts a graphical representation of a typical call management rule that can be used by CoAR.

In a call management rule, user context is represented as state machines, and user preferences for handling incoming calls are implemented as aggregation rules. Multiple types of context are used in the management rule: user location is abstracted into work, home, and others or unknown. The activity context is related to the application currently in use (e.g. text editor). It is abstracted to the type of application, such as office (e.g. text editor, -mail programs, etc.), business (e.g. CRM, Customer Relationship Management), or internet (related to internet browsing). The communication context is related to communication activities, whether the user is in communicating or not. Information from the call provider corresponds to the state of a call. When a call is initiated, it triggers the aggregation rule that will check the current situation (of the other state machines) to decide how to handle this call. The action resulting from the aggregation rule will be sent to the call management component to be executed.

A second case study consists of a M2M scenario, where a transportation truck handles a product from one starting point to a destination with some intermediate stop places. The truck reports continuously contextual information about its location and temperature of the transported product. At the monitoring center, an agent is supervising these contextual data to react in time to potential incidents (e.g. a temperature excess).

Figure 4 illustrates an example of rules that can be configured to send a notification in case of incident to the supervisor on the right devices based on his availability information. In the figure, the truck temperature is abstracted to colors that represented a certain severity degree. Whenever, the temperature reaches the red severity the process of deciding on which device to notify is triggered. The latter is represented with an aggregation rule; the condition part checks the user availability and the action part consist of choosing the target device.
V. EVALUATION

A. Efficiency

To evaluate the benefit of abstract-aggregate based reasoning compared to a regular case of the publish-subscribe paradigm where no reasoning is performed, we conducted an experimental study with a test-bed based on OMNeT++. As the main concern of our proposal is to reduce overhead on context consumers which are deployed on resource constrained devices, we choose the throughput of received context messages as a comparison metric.

The topology used in the simulation is composed of a CxB, a set of CxPs (n=12) and CxCs (m=8). The CxPs continuously publish context messages to the CxB, in different rates. The context publication is simulated with a Poisson process, they occur independently and at a constant average rate configurable for each provider. Poisson process has been chosen because it allows modeling the distribution of context publications with tuneable parameters. The probability for having $k$ publications during a time interval of length $\tau$ is given by the following equation:

$$P[N(t+\tau) - N(t)] = k! \frac{e^{-N(t)\lambda \tau}}{k!} \quad k = 0, 1, \ldots,$$

Each CxC subscribes to a random number of CxPs in a uniform fashion (all providers have a similar chance for a given subscriber $p=1/n$). The CxB stores all the subscriptions in a routing table; each time it receives a context update from a provider it duplicates the message to all the corresponding subscribers. The following table illustrates the simulation setup parameters.

<table>
<thead>
<tr>
<th>Providers</th>
<th>Rate (publication/s)</th>
<th>Consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CxP1</td>
<td>40 (p/s)</td>
<td>CxC3, CxC8</td>
</tr>
<tr>
<td>CxP2</td>
<td>20 (p/s)</td>
<td>CxC1, CxC3, CxC7</td>
</tr>
<tr>
<td>CxP3</td>
<td>30 (p/s)</td>
<td>CxC3, CxC5, CxC8</td>
</tr>
<tr>
<td>CxP4</td>
<td>20 (p/s)</td>
<td>CxC2, CxC3, CxC5, CxC6</td>
</tr>
<tr>
<td>CxP5</td>
<td>20 (p/s)</td>
<td>CxC1, CxC3, CxC4, CxC5, CxC8</td>
</tr>
<tr>
<td>CxP6</td>
<td>10 (p/s)</td>
<td>CxC1, CxC3, CxC5, CxC7</td>
</tr>
<tr>
<td>CxP7</td>
<td>10 (p/s)</td>
<td>CxC1, CxC3, CxC4, CxC5</td>
</tr>
<tr>
<td>CxP8</td>
<td>50 (p/s)</td>
<td>CxC2, CxC8</td>
</tr>
<tr>
<td>CxP9</td>
<td>20 (p/s)</td>
<td>CxC1, CxC2</td>
</tr>
<tr>
<td>CxP10</td>
<td>40 (p/s)</td>
<td>CxC1, CxC3, CxC5, CxC6, CxC7, CxC8</td>
</tr>
<tr>
<td>CxP11</td>
<td>20 (p/s)</td>
<td>CxC1, CxC3, CxC5</td>
</tr>
<tr>
<td>CxP12</td>
<td>20 (p/s)</td>
<td>CxC1, CxC3</td>
</tr>
</tbody>
</table>

The graphs in figure 5 present the throughput (number of context messages sent or received per second) on the CxB side. The red graph, at the bottom, represents the throughput at reception, i.e., the incoming context updates from the different providers. The graph in green, i.e. the upper graph, represents the throughput at emission when the abstract-aggregate reasoning approach is not deployed (i.e., all incoming messages are sent out to all subscribers). The blue graph, in the middle, represents the throughput at emission when the abstract-aggregate method is used to filter incoming updates and only send out relevant events. The throughput at emission is greater than that at reception because there is more than one subscriber for each CxP, and thus each incoming message has to be duplicated as many times as there are subscribers for the given context information.

From the graphs, the use of the abstract-aggregate reasoning approach allows the reduction of emission throughput, since only relevant events (those relevant to a given application) are sent out. Each context update from a provider is first consumed by the abstraction function that updates the current state of the corresponding finite state machine. The transition between the states may trigger, with a probability ‘p’, an event to the aggregation function that will verify the current situation, and as a result may trigger a notification to a corresponding context consumer with a probability ‘q’. Thus, the probability of an updated context to generate an event out to a consumer depends on both probabilities ‘p’ and ‘q’. The relation between the output throughputs when the abstract-aggregate is used and when it is not is presented in the following equation:

$$\text{Throughput}_{\text{with}} = \text{Throughput}_{\text{without}} \times p \times q \quad (i)$$

B. Scalability

Another concern is to keep the framework response time reasonable in a way to support scalability of the architecture.
and resist to the increasing number of providers and/or consumers. From the context consumer, the response time corresponds to the amount of time needed for the provider to publish context to the broker plus the time needed for the broker to reason on the published context, plus the time needed by the broker to send a notification to the consumer. The transportation time (for publication and notification) depends on the network conditions and cannot be controlled. It is reasoning time that is more interesting to study as it depends on the quantity of managed context and the number of hosted configuration files. To evaluate the overhead added onto the context broker when using the abstract-aggregate reasoning approach, we used an implementation of a M2M scenario (section IV).

Figure 6 depicts a comparative summary of reasoning time distributions in relation to the number of Abstract-Aggregate configuration files (that complies to the DTD presented in listing 1) uploaded to the context management framework. This figure utilizes box plots, a convenient means of data visualization and especially useful for detecting the presence of extreme values in a distribution.

![Figure 6. Distribution of Abstract-Aggregate reasoning time](image)

The different distributions are generated by sending approximately one hundred sample context publications to the framework and measuring how much time the framework spent processing each publication through the functions’ Abstract-Aggregate. In a box plot, a darkened line is used to represent the median (50th percentile) of a distribution, the 25th percentile corresponds to the bottom side of the corresponding box, the 75th percentile is indicated by the box’s top side, the distribution’s 10th percentile corresponds to the box’s bottom line, and the 90th percentile corresponds to the box’s top line [11]. Additional points or lines on the top or bottom of a box correspond to the distribution outliers, i.e. points showing the extreme values for a given distribution.

These box plots show that the median for 10 configuration files is about 15 ms, while it is about 30ms for 50 and 100 configurations, indicating a possible convergence in reasoning time that should be confirmed by further studies. The spread (the box height, i.e. the difference between the 75th and 25th percentiles) of the different distributions is very similar. Also, except for a few outliers, the reasoning time increases as the quantity of configuration files being managed increases.

The graph in figure 7 illustrates the variation in response times, in milliseconds, of the framework for multiple context publications. Data used to generate this graph are collected by sending multiple context publication requests to the framework and measuring the time it takes to receive a response for each request. This response time includes the time required to establish the HTTP connection as well as the time needed for the framework to deal with this published information.

From the figure, we can easily see that the response time oscillates around 150ms; reaches a maximum of 400ms in extreme cases and a minimum of 50ms for the best cases.

![Figure 7. Variation of response time](image)

We can conclude from figures 6 and 7 that the response time of the proposed framework depends mostly on the time required for exchanging HTTP messages among the different elements of the framework. The reasoning time is minor compared to the connection time. An important cause of this latency in HTTP connections is due to the repetitive phases of TCP handshakes, as shown in [12]. The framework performance, as far as context transportation, can be improved by the use of HTTP-persistent connections [13]. The use of permanent connections will improve memory and CPU utilization related to the establishment of HTTP connections, as fewer connections will need to be opened.

VI. RELATED WORK

Our framework relies on well-established design principles (separation of concerns and events processing) and meta-information (context quality) consideration to provide a better context management enabling third party developers to build flexible context-aware applications. Most of earlier works lacked flexibility and did not take advantage of these principles as a result of relying separately on one of them. Readers can refer to our earlier work [10] to have a detailed discussion of our framework compared to existent frameworks. Here is a selection of earlier works gathered based on the considered principle:
A dynamic approach to deal with context characteristics is to manage context lifecycle or validity duration to provide an efficient support to context-aware applications. In [14], the authors proposed a simple context state transition to represent context lifecycles. Context can be ‘active’ as soon it is updated, ‘suspended’ when the activity performed by a user is disturbed, or ‘terminated’ when the user ends his/her current activity.

In some cases, the change in values of contextual information may not imply a change in the current situation. Thus it is interesting to wait for relevant events (relevance depends on the context consumer) that represent changes in a situation. In [15], the authors propose a UML-based and event-driven model to represent static content such as context data and events triggered by changes in context. That study also represents the dynamicity of a context-aware service by representing the different scenes (situations), the transitions between them, and the service behavior corresponding to a scene that could be initiated as a response to a transition. UML classes modeling contextual information are mapped to objects in the target programming language (e.g. Java) which makes context easily manipulated by context-aware applications. However, the framework has the control on how context is handled, and it cannot be customized for a given context-aware application.

The separation of concerns as a design pattern aims to capture redundant functionalities into a set of specialized components for better modularity and reusability. Few works have proposed the use of this concept to decouple context management from application logic. In [1], the authors presented a telecom platform that relies on a layered context management approach and uses modules supplied by third-party developers to infer knowledge from context data. An inner representation based on ontologies is used for modeling context, which may complicate the integration of context data with the application own data. Context reasoning is performed by modules that can be provided by third party developers. However, new components cannot be integrated to an existent application easily at runtime due to registration policies used by the platform to preserve user privacy.

In [5], the authors propose a layered infrastructure to allow context-aware mobile applications to delegate the execution of some context-dependent actions to the infrastructure. Context processing relies on ontological reasoning for deriving new facts, and machine learning to generate high-level models from low-level context. The approach implies to context-aware applications to be able to handle ontological data to. Context reasoning is provided by the infrastructure and applications cannot interact with the reasoning mechanisms. External elements (providers or consumers) can be added but can only be used by the corresponding context-aware application. Also, the kind of context data that can be used by an application is restricted to what the infrastructure can handle with its ontology-based representation.

In [16], the authors presented a framework that aims to decouple context management from the application layer to simplify application implementation. The framework relies on an RDF-based model to which developers have to comply to define their specific context-management modules. It gives developers the ability to modify context management at run time but lacks support for handling a considerable number of components as stated in their article.

Information that is characteristic of a context, also called the quality of context (e.g. accuracy, precision), is additional information that may be of great value to some context-aware applications, since it can be used to assess context-aware actions (e.g. the automatic handling of incoming calls). In [17], the authors proposed a UML-based model to augment contextual information with meta-information describing their quality (e.g. precision, frequency, confidence) and then to propagate this quality after abstracting contextual information.

In [3], the proposed framework relies on the probability of correctness to select a context source from multiple providers in order to reduce the propagation of errors resulting from abstracting low-level context and then basing reasoning on it. Specific ontology is used to model managed contextual information. The framework is integrated with the context-aware application and as a result it is hard to reuse if by another application. Rules for reasoning on context are defined at the application design time without support for an easy modification at runtime.

CAMEO [18] is a context management middleware that enable the development of mobile context-aware applications. It is implemented as a background service and third party applications interact with CAMEO through low level inter-process communication (IPC). A key-value representation is used for exchanging context between applications, and a P2P approach for exchanging content between users’ devices. This framework is oriented for usage in opportunistic social network applications like sharing user generated content (e.g. pictures, videos) among nearby users participating in the same events (e.g. football game). Developers should implement an interface to define how key-pairs representing context should be processed. The only way to modify the implementation of such an interface is to replace the whole application with a new version as it is tied to the application code.

<table>
<thead>
<tr>
<th>TABLE III. COMPARAISON SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of context manipulation</td>
</tr>
<tr>
<td>P. Guthem [1]</td>
</tr>
<tr>
<td>J.B. Filho et al. [3]</td>
</tr>
<tr>
<td>M. van Sinderen et al. [5]</td>
</tr>
<tr>
<td>W. Jih et al. [14]</td>
</tr>
<tr>
<td>T. Mo et al. [15]</td>
</tr>
<tr>
<td>J. Zhu et al. [16]</td>
</tr>
<tr>
<td>S. McKeever et al. [17]</td>
</tr>
<tr>
<td>CAMEO [18]</td>
</tr>
<tr>
<td>Our proposal</td>
</tr>
</tbody>
</table>

We summarize this discussion in the above summary table (Table III) where evaluation criteria are based on the design
considerations introduced in section II. The ‘+’ symbol means
the existence of a support for the corresponding consideration,
whereas the ‘−’ symbol indicates its absence, ‘+/-’ indicates that
support exist but need to be enhanced.

Our proposal addresses most of the challenges described
previously (section II). In fact, ease of context manipulation
is addressed by providing a flexible information representation in
XML as well as a hierarchical addressing schema that allows
handling different granularities of context. Ease of application
development is provided via the use of an XML-based
language for managing context that can be used by application
developers to customize the way context should be processed
before being pushed to application. Ease of management of
core reasoning is addressed by enabling application
developers to define the context reasoning of their application
through an XML-based specification file at design time. This
specification file can be modified at run time by uploading a
new version of it. The ease of new components integration is
the result of the distributed architecture of the proposed
framework.

VII. CONCLUSION

Context information is valid for a limited time, which
makes it suitable to be handled as a series of events. We
propose to provide context-aware applications with the ability
to program a context-aware framework to handle context
events and only notify an application with relevant
information. An XML-based language is provided for this
purpose. This approach is especially useful for applications
running on resource-constrained devices that cannot process
the many events generated by context changes. The benefits
from the developed framework are reductions of both cost and
complexity related to development of context-aware
applications, since the parts of functionalities related to context
management do no longer fall under the responsibility of
applications. Another benefit is the adaptation efficiency of
context-aware applications; they receive more accurate events
of interest related to context changes and therefore only need to
implement the logic that will respond to these events. The use
of a local caching system enables response time enhancement.
Nevertheless, the use of this approach does not deliver
substantial advantage to all types of applications. For example,
a real-time supervision application that tracks the localization
of a truck to visualize it into a map needs to receive all of that
truck’s GPS related events in order to maintain a correct
visualization of the truck. As future work, we plan to build a
Cloud-based context management platform that relies on the
approach presented here to handle context events in large-scale
distributed environments. We are also considering enhancing
the framework with self-management capabilities to provide
context-aware applications with the ability to modify how
context should be processed based on user feedback.

REFERENCES

Applications in Next-Generation Networks,” IEEE Communications
Magazine, volume 50, issue 1 (2011)
Ontologies for Pervasive Computing,” Sixth International Workshop on
Semantic Media Adaptation and Personalization (SMAP'11), VIGO,
Spain (2011)
Context Conflicts in Context-Aware Systems,” IFIP 9th International
Conference on Embedded and Ubiquitous Computing (EUC'11),
Melbourne, Australia (2011)
scalability: Application to the Pervasive Computer Supported
Collaborative Work.” In proceeding of International Conference on
Machine and Web Intelligence (ICMW'10), Algiers, Algeria (2010)
E.H. Eertink, “Supporting context-aware mobile applications: an
infrastructure approach,” IEEE Communications Magazine, Volume: 44,
Issue: 9 (2006)
[6] Y. Oh, J. Han, W. Woo, “A context management architecture for large-
scale smart environments,” IEEE Communications Magazine, Volume:
48, Issue: 3 (2010)
[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides. “Design Patterns:
Elements of Reusable Object-Oriented Software,” ISBN: 0-201-63361-
2, Addison-Wesley (1994)
Context-Aware Communications Systems,” ICIN, Berlin, Germany
(2011)
Case Study,” Proceedings of International Conference on Digital
Information and Communication Technology and its Applications
(DICTAP'11), Dijon, France (2011)
[10] B. Chihani, E. Bertin, N. Crespi, I.S. Suprapto, J. Zimmermann,
“Enhancing existing communication services with context-awareness.
Journal of Computer Networks and Communications (JCNS),” Volume
2012, Special Issue of Context-Enriched and Location-Aware Services
(2012)
American Statistician, Volume 32, Issue 1, pages 12–16 (1978)
Open,” The 7th International Conference on emerging Networking
EXperiments and Technologies (ACM CoNEXT'11), Tokyo, Japan
(2011)
(2005)
Smart Space Environments,” AUPC’09, London, UK (2009)
Aware Service,” 2011 IEEE International Conference on Web Services
(ICWS'11), Washington, DC, USA (2011)
framework for ubiquitous applications with runtime support,” 
Support Transparent Reasoning with Uncertain Context,” 1st
International Workshop on Quality of Context (QuaCon), Stuttgart,
Germany (2009)
context-aware middleware for Opportunistic Mobile Social Networks.”
IEEE International Symposium on a World of Wireless, Mobile and
Multimedia Networks (WoWMoM'11), Pisa, Italy (2011)