# A generic architecture prototype for context-aware systems

Ni Zhang, Chris Todd

Dept. of Electronic & Electrical Engineering, University College London

**Abstract:** This paper summarized main architectural requirements for building a generic context-aware system, and proposed a solution for context collection, storage, aggregation and dissemination, with special focus on satisfying the system requirements of scalability and authorized access to private context information.

## 1. Introduction of context-aware architecture

The concept of context-aware computing [1] has been around for a number of years now, the research focus has been moving on from providing a stand-alone software support for a certain context-aware application to elaborating a more generic architecture that could accommodate various applications and explore the potentials of context-awareness in more aspects, for example, network management in IST's Ambient Networks project [2]. Although many research efforts are endeavoring to construct context-aware architecture with their own purposes and means, which are neither same nor exhaustive, a shared conceptual model of architectural support for context-aware applications includes context collection, storage, process and dissemination.

- **Context Collection**

  Given the diverse and distributed nature of context information, they may be collected from different OSI layers, for example, network QoS information is from OSI Network Layer while channel information is from the OSI Data Link Layer. The cross-layer context collection can be implemented in a number of ways, including: Push mechanism: Context sources periodically push updated context information to the context service; Pull mechanism, Context client request context information via polling or on-demand; Shared-space mechanism: Context sources periodically publish their information in some place context client can read it whenever they need; Persistence subscription mechanism: context consumers subscribe with certain context sources and these sources will provide them with the required context accordingly.

- **Context Storage**

  Some context-aware applications may make decisions based on historical observations and thus may require a repository for history data. Although various solutions addressing where to store raw or processed context information exist, either locally in data sources or somewhere else, most agree on adopting a distributed design for context storage in order to achieve scalability.

- **Context Process**

  Processing of context data manipulates raw context information to create more composite and semantically rich information. Few context-aware applications want to work directly with raw data from information sources, it could be that the application only needs portion of the data, the data is in wrong format, the data is inaccurate, or the data is incomplete and not useful without aggregating other sensor inputs. Thus, raw data from context sources typically needs to go through several processing steps before it becomes meaningful contextual knowledge desired by applications. The processing of context information does not only include aggregating but also inferring, correlating, de-aggregating and other similar functions that can be performed as and when requested and in the most appropriate locations.

- **Context Dissemination**

  Context information can be categorized into several classes based on factors such as real time or non-real time, private or public, etc. Context information in these different classes may require different dissemination policy. For example, public context information can be disseminated to everyone needing the information, but private context information must be securely handled and disseminated only to authorized requests.

A fundamental challenge in building a context-aware system, then is to collect raw data from thousands of

diverse sources, process the data into context information, and disseminate the information to hundred of diverse applications running on various platforms, while scaling to large numbers of sources, applications, and users, securing context information from unauthorized uses, and respecting individuals' privacy. Supporting many applications simultaneously while maintaining efficiency in small as well as in large networks has been attracting many research efforts, while there is surprisingly little literature about access control in context-aware systems although researches often note the important of privacy and security in context-aware computing [3].

[4] proposed a graph-based abstraction for context collection, aggregation and dissemination. It models the context information as events, which are produced by context data sources and flow through a directed acyclic graph of event-processing operators to subscribed applications, by which the low-level raw data from context sources are easily aggregated into the high-level semantically-rich context information desired by applications. Since the abstraction decomposes the context aggregation process of every application into a series of modular and re-usable operators, a context-aware experimental system based on the abstraction , called Solar [5], could achieve a great scalability and flexibility. The graph-based abstraction, however, does not support context storage, making context-aware applications based on tracking history information not available. In addition, the experimental Solar system is not secure and is vulnerable to malicious attacks, since there is no access control enforced to guard information privacy [5].

## 2. An architecture prototype for context-aware system

Building upon the Solar's experience, we proposed a more complete context-aware architecture prototype (Figure 1) for context collection, storage, aggregation and dissemination. The improved architecture could not only maintain the advantages of the graph-based approach in context aggregation, scalability and flexibility, but also enable context storage and access-control mechanism to function.

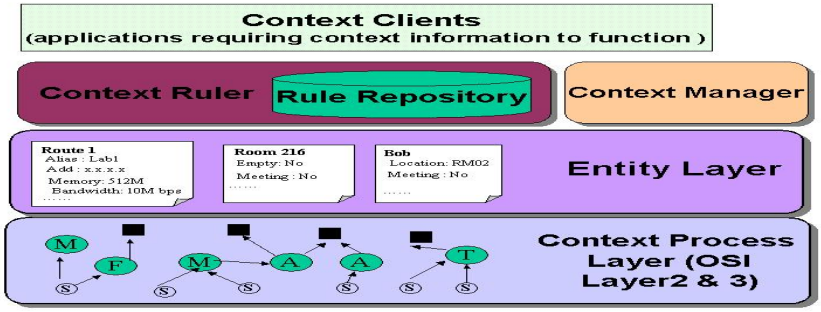


**Figure 1: Context-awareness architecture prototype**

While Context Process Layer basically adopts the graph-based abstraction to collect and aggregate context information, the introduction of two new components, Context Ruler and Context Manager, and a new Entity Layer plays a vital role in implementing context storage and access-control.

- **Context Ruler**

Context Ruler serves as an interface to context clients (i.e. context-aware applications), where context information is requested, the type and QoC is negotiated, and possible conflicts are investigated. The Context Ruler maintains a database called Rule Repository in which access-control policies of all entities reside. Entity could be a physical object such as a person, a place, a router, a physical link, or a virtual object such as IPsec tunnel, SNMP agent.

- **Context Manager**

Context Manager manages temporary connections between various context clients and different entities, and notifies the subscribing clients of any changes of requested context information. The Context Manager's responsibilities include: managing associations between context clients and entities; grouping/re-grouping entities associated with context clients; managing context information access: indexing, persistent subscription, scheduling, etc.

- **Entity Layer**

Entity Layer is at the centre of our proposed context-aware architectural prototype. It can be seen as a logical construct representing a distributed repository for context information. To understand, first, each *Entity* has one or more *Attributes* associated with it, for example, an entity of a router could have attributes of alias, address, memory size and available connectivity characteristics. Since entity always populates its attributes with some value (i.e. context information), the Entity Layer as a whole can be seen as a logic repository for context storage. Second, the Entity Layer does not consist of physical body of entities, but the information of entities, which distinguishes "logical" Entity Layer from "Physical" context data sources residing in the Context Process Layer.

The value of *Attributes* might be static, set from the inauguration of entity (e.g. memory size of a router), or dynamic, changing from time to time. Dynamic *Attributes* are filled by requesting the underlying Context Process Layer to collect from data sources and to aggregate to a desirable format.

The introduction of the Entity Layer helps enable access-control mechanism to function, because each entity has an authorization policy that consists of a set of rules that regulate every attribute with respect to privacy-sensitive information such as location information. All access-control rules associated with an entity are periodically (or on demand) updated to and maintained in the Rule Repository of Context Manager.

## 3 An example of a secured context request

Based on the above architecture and discussion, we give an example to show how a context-aware application request for an Entity Bob's location information. While Figure 2 illustrates the sequences of messages and events that characterises a secured context request, Figure 3 presents an operator graph to show how the raw data from information sources flow through the operators to be come usable by the applications.

First, the Entity Bob registers or updates its authorisation policy to Context Ruler; then all the access control rules associated to the Bob Entity are kept in the Rule Repository of Context Ruler. Next, a context client (i.e. context-aware application) asks (1. Fig.2) the Context Ruler the location of Bob. The Context Ruler checks (2. Fig.2) its Rule Repository to see if the client is allowed to access the entity Bob's location information. The query is then resolved, and Context Ruler hands over (3. Fig.2) the client to Context Manager who then takes responsibility for the conversation with the context client.

In the mean time, the entity Bob in the Entity Layer checks (4. Fig.2) whether or not the requested location information is available. If Bob's location information stored in the entity layer is too outdated to satisfy the context client, the entity Bob then issues (5. Fig.2) a SUBSCRIBE request to the Context Process Layer (6. Fig.2) to ask for repopulating (7. Fig.2) its attribute Location. Once finished, the Entity Bob send back (8. Fig.2) the latest location information to the context client. The Context Manager maintains the conversation between the context client and entities until it receives (9. Fig.2) a BYE request from the client or a TIMEOUT message from the system. Once the conversation ends, the context client has to encounter the Context Ruler to go through access-control check if it wants Bob's location information again.

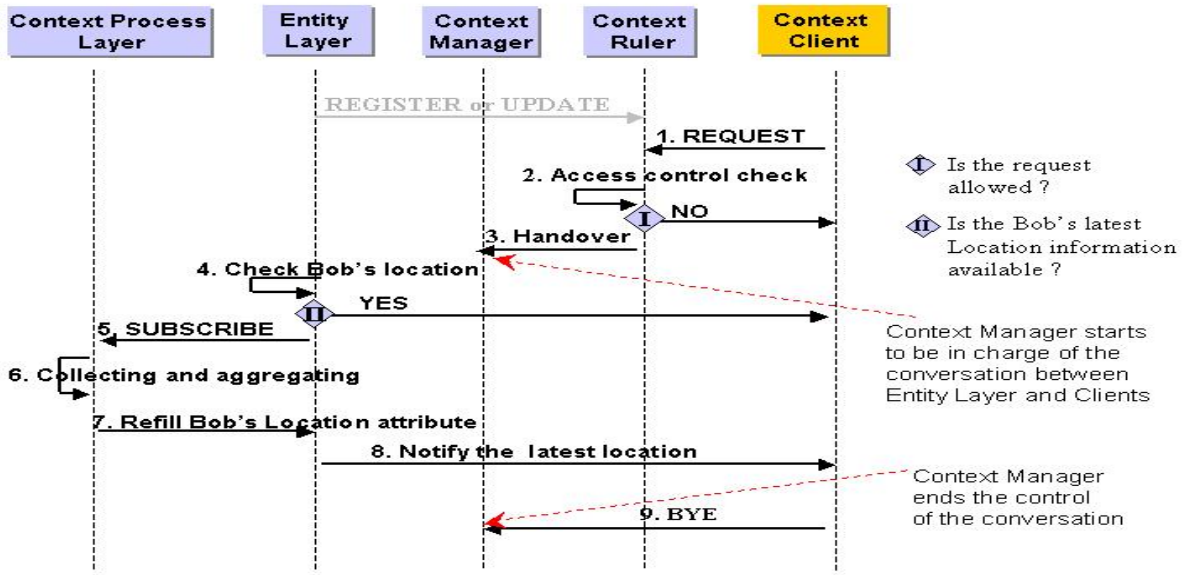


**Figure 2: Example Message Flow of a secured context request**

Figure 3. illustrates how the entity Bob collects its location information through a graph-based aggregation.

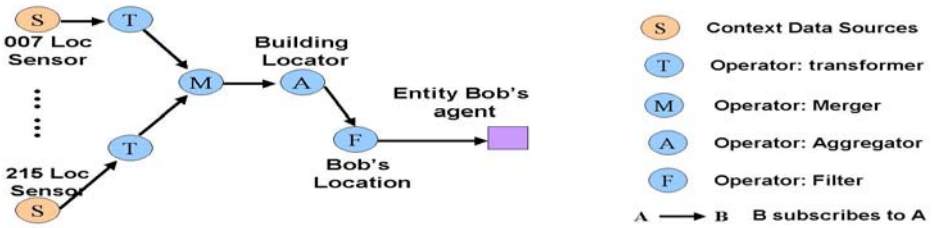


**Figure 3: An example operator graph**

Suppose there are location-tracking sensors installed in each room and badges attached to people and devices. Each time a sensor detects a signal from a badge, it sends out an event containing the badge ID and the timestamp. In the figure these sources are labeled "Loc Sensor" with a room number; each has a transforming operator to map the badge ID to the person or device's name associated with it.

The *Building Locator* operator subscribes to the current location of every badge, based on the transformed and merged events that originate from the location sensors. It records the current location and generates a "location change" event whenever it sees a badge change location. Then, this output event stream can be used by a subscriber, *Bob's Locator*, filtering for changes in Bob's location.

## 4. Conclusion and Future work

This paper mainly proposed a generic context-aware architecture for context collection, storage, aggregation and dissemination, with special focus on satisfying the system requirements of scalability and authorized access to private context information. The requirement for an aggregation capability to create more composite and semantically rich information through interpretation of low-level information available is highlighted in our proposed architecture. This transformation capability indeed opens a whole new space where elicitation of privacy policies might become very complex considering that context processors or aggregators might have to deal with conflicts from the sources of low level context information. By introducing a key Entity Layer and context manage components: Context Ruler and Context Manager, our solution provide a way to address the issue. The adoption of a graph-based abstraction which is based on a non-centralized and re-use logic secure the architecture a scalable one.

The proposed context-aware architecture is neither complete nor entirely satisfactory, though. Further work includes detailing access-control rules, categorizing entities and their attributes and investigating the applicability of the proposed architecture prototype in a more dynamic and heterogeneous system environment.

## 5. Acknowledgement

This paper has been written in parallel with author's involvement in the Ambient Network Project (the EU's Information Society Technologies (IST) 507134 project). Insights gained through the work undertaken in the context of the Ambient Networks help delivery this paper, however, the views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the Ambient Network project or the Context Aware Networks work package.

## 6 References

[1] B. N. Schilit, N. I. Adams, and R. Want. (1994) "Context-Aware Computing Applications", the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, Dec, 1994. P.85-90. IEEE Computer Society
[2] Ambient Networks Project, http://www.ambient-networks.org/
[3] Maria R. Ebling, Guerney D. H. Hunt, and Hui Lei. (2001) "Issues for context services for pervasive computing", the Workshop on Middleware for Mobile Computing 2001, Heidelberg, Germany, Nov. 2001.
[4] G. Chen and D. Kotz (2002), "Context Aggregation and Dissemination in Ubiquitous Computing Systems", the Fourth IEEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society Press
[5] G. Chen and D. Kotz (2004), PhD thesis: "Solar: Building A Context Fusion Network for Pervasive Computing", tech. report TR2004-514, Dept. of Computer Science, Dartmouth College