

A FRAMEWORK FOR CONTEXT REASONING SYSTEMS

Michael Przybilski, Petteri Nurmi, Patrik Floréen
Helsinki Institute for Information Technology HIIT
Basic Research Unit (BRU)
{firstname.lastname}@cs.helsinki.fi

ABSTRACT

Finding a common structure for context reasoning systems is still rather difficult. Most context reasoning mechanisms are implemented in such a way that they support the specific study in mind.

This paper first extends the understanding of context information, based on the definition of Dey and Abowd, [1] and then introduces a structure for a context reasoning framework. This framework takes into account the commonalities amongst different inference and learning mechanisms and allows the re-use of those components. At the same time it allows the combination of different reasoning mechanisms, providing altogether a context reasoning engine that can be used for various context-aware applications.

KEY WORDS

Context Reasoning, Machine Learning, Software Architecture, Software Components

1 Introduction

Context-aware systems are beginning to play an important role in modern software systems, especially in software for wireless information devices. However, the research in this field can still be considered to be in its infancy. Systems that support the study of context-awareness are usually built with the specific purpose of the study in mind (see for instance [2]) and it is usually very difficult to extend those systems for other purposes, or to integrate them with existing applications.

As also Winograd recognizes in his paper [6], so far only very few common architectures, or suggestions towards the architecture of context-aware systems have been developed (see for example [4, 6]).

2 Definitions

For the purpose of this document we will follow the widely accepted definition of context by Dey and Abowd [1]:

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 application themselves.

2.1 Context Information Categorization

Based on the categorization of an entity above, we divide context further, depending on the information that is related to a specific entity (see figure 1). Even though this kind of division is debatable, we especially take into account information regarding

- **the user**,
- **the group**, to which the user belongs (for instance by social status, location, family, interest, etc.),
- **the physical location**,
- **the device**, or set of relevant devices
- **the particular application**, as well as other software and the relevant runtime environment.

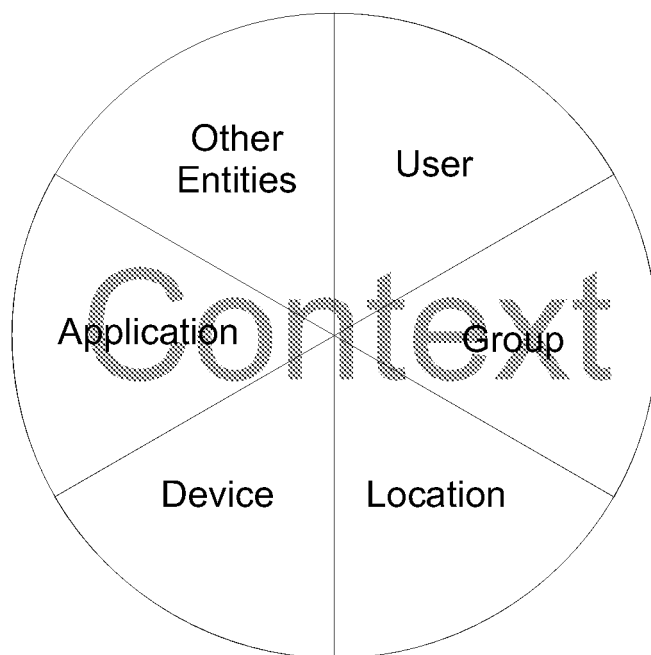


Figure 1. Categorization of Context Information

With regard to this division, it needs to be taken into account that often specific information may belong into several categories. Nevertheless, we consider that this categorization eases further discussion.

2.2 Context Timeline

Even though Dey and Abowd are not specific with regard to the temporality of the information referred to, we also introduce a division with regard to the time of the information.

In research of context-awareness often historical information is taken into account, in order to make predictions of future context changes. Furthermore, applications usually attempt to take into account the current context of the user, or the device, in order to, for instance, change their behavior, or provide a context-specific user interface. Thus, as illustrated in figure 2, the timeline of the context information can be divided into:

- **Past Context:** any context information that characterizes a past situation.
- **Present Context:** any context information that is currently considered valid. The definition of "present", or "current", especially with respect to a practical implementation is of course debatable and should be discussed further.
- **Predicted context:** any context information that has either been automatically predicted (using a prediction mechanism, such as interpolation, or particle filters), or manually "predicted" by the user, e.g. user enters a meeting in his calendar and thus predicts that he will be in that particular context at that particular time. Furthermore, even though it is also debatable, we felt more comfortable with the term "predicted context", then with "future context", as we are unfortunately still not able to foresee the future.

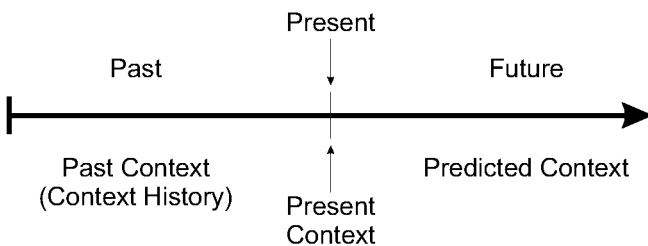


Figure 2. Context Timeline

2.3 Framework for Context Reasoning

Context-reasoning can be defined as deducing new and relevant information to the use of application(s) and user(s) from the various sources of context-data [3]. Since the goal is to provide a framework for context-reasoning, its task is to provide a basic structure for different reasoning mechanisms, as well as those components that a majority of these mechanisms have in common.

We would thus define that the framework consists of components, which are connected (at build-time), and which

result in one reasoning engine. This may internally implement several different reasoning mechanisms, but even though the inner workings of such instances may differ, they all provide a similar set of interfaces and also require the same basic set of interfaces.

Alternatively, we would even go as far as saying that the components can be connected, and even exchanged, at runtime. However, this would require the existence of a proper runtime environment and component model, which have not been chosen or implemented yet.

3 Structure

Based on this information, we describe the framework to be the central context-reasoning mechanism for context-aware applications. It takes as input (raw) context data, i.e. information from either physical sensing-mechanisms (e.g. temperature sensor, GPS-based location coordinates, etc.), as well as (indirectly; thus not marked in figure 3) from applications (e.g. calendar, contacts, user-selected-profile, etc.). As output the framework does not only provide recommendations, but higher-level context information, together with their probability (confidence).

The high-level structure of the framework, together with its connections to sensors and applications, is depicted in figure 3. The picture only denotes the logical parts, and does not state anything regarding the distribution of the different components. These issues are discussed in later parts of this document. Furthermore, the purpose of the black arrows is discussed in more detail later-on.

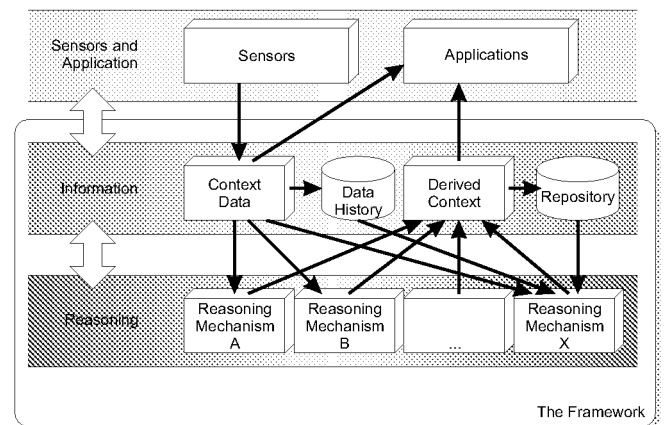


Figure 3. Structure of the framework

As can be seen, the framework itself consists essentially of two layers.

- **The Information layer** serves as a mechanism to collect, provide, and store any context data. It takes as input the (raw) context data, information that comes from either physical sensing-mechanisms, as well as from applications. The derived context is referred to as any other information, that is output from the rea-

soning mechanisms, or that is used as initial data by the reasoning mechanisms (e.g. rules, parameters, or learning data), whereas the term context data, as described above, refers to the information that comes directly from the sensing mechanisms.

Furthermore, this layer contains data, such as learnt models, or rules, either specified by the user, or automatically learned, that can be beneficial to more than one reasoning mechanism. In above picture, this data is considered part of the repository.

In order to ensure data security, the information layer also contains access control mechanisms, which manage the access of different applications and reasoning mechanisms to the context data, as well as the derived context.

- **The Reasoning¹ layer**, which contains the implementation of the actual mechanisms that differentiate the various inference and learning methods. This may also include information, which is specific to one particular mechanism and can or should not be used by other mechanisms (e.g. information which can for certain reasons not be provided to the outside of the mechanism itself).

4 Components

As mentioned earlier, figure 3 serves as an example of the final structure. The arrows denote the flow of information between the different components. However, the structure and functionality of the different components needs to be described in more detail:

- **Context Data:** contains current information, collected from the sensors. This component is responsible for gathering the (raw) data. When the data of one sensor is being updated (overwritten by more recent data), the old data is moved to the Repository. Furthermore this component is responsible for allowing the subscription of other components to subscribe to specific events and their associated data. In addition, the component is responsible for the notification of events, and the provisioning of data.
- **Data History:** contains all, but the most recent data from all sensors.
- **Derived Context:** contains current context information, derived through reasoning. As in the data component, also here a subscription and notification event model is part of the mechanism. The component is also responsible for storing context information in the Repository and the provisioning of current context information.
- **Repository:** contains all previous context information, as well as either user specified, or learned rules

¹We consider reasoning to be the combination of inference and learning.

and models, that can be used by other components. It contains for instance also data that was used by certain learning mechanisms as an initial data-set (e.g. for reasoning).

As mentioned earlier, the mechanism that ensures data security is not denoted in figure 3, but never-the-less part of the information access layer.

- **Reasoning Mechanisms:** These depict the mechanisms that implement and set apart the different inference and learning mechanisms.

5 Repository

As mentioned above, the repository contains a multitude of information. Categorizing this data clearly is rather difficult. However, one possibility is to distinguish between data that has initially been provided (e.g. initial parameters for the reasoning mechanisms, initial, or default rules, etc.) and data that has been used to learn the reasoning mechanisms. Distinguishing this data is rather simple, since it has the characteristic that it will typically not change any further. However even for this data it may be possible to be updated and re-learned, in order to provide more optimal results.

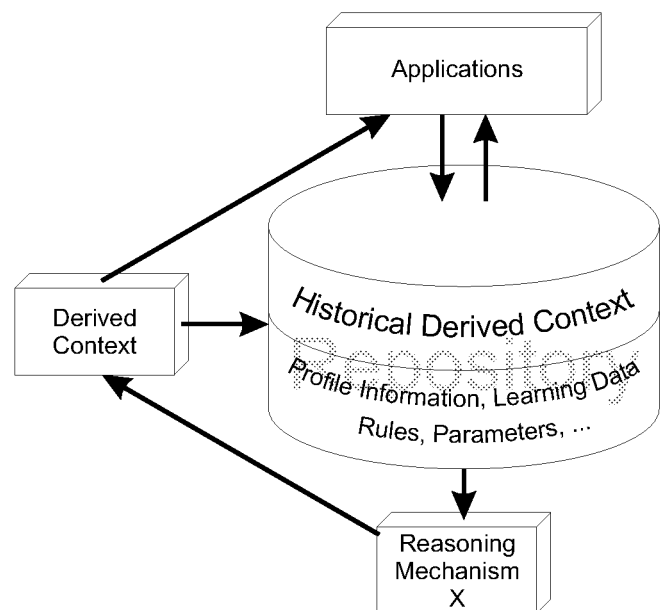


Figure 4. Context Repository

Further distinction, based on the frequency of change becomes increasingly difficult. While for instance device characteristics, such as screen-size or shape, tend not to change very often, it is with the advancements of technology already possible that the display resolution and even the display shape change (e.g. flexible displays [5], etc.). Furthermore, even in those mobile terminals, considered here, this is infrequently the case so far, it has to be considered that the user may be interested in upgrading his device,

or set of devices, with additional resources such as memory, or potentially even a new processing unit, with add-on devices with additional capabilities, or even upgrade or replace a physical device completely. Thus, also the characteristics of a device cannot be considered static. The same applies also to other entities, such as specific applications, which may change due to an (over-the-air) update or even to the characteristics and preferences of the user.

All this information, which is not considered present context, is located in the Repository and made available to the other components. The possible distinction, based on the time, into past and predicted context, as well as they distinction based on the entity in question, has been described above, in the Definitions section.

6 Distribution

In the previous description, the physical distribution amongst the devices of one user, or a set of users, as well as servers has intentionally not been described. The reason is that the distribution of the different logical components, as above, as well as the distribution of one the subcomponents of one such logical component cannot easily be specified. The factors primarily influencing the distribution are the limited resources of the mobile devices. In particular, we are considering

- **the processing power**, which most of the reasoning mechanisms require, as well as
- **the availability of data**, without which the reasoning is rather useless. Furthermore, since we are considering, at least potentially, a distributed mechanism, we also have to take into account
- **communication issues**, such as availability, communication overhead, and delays.

Especially on mobile devices, the limited available power is problematic. However, this issue, as well as the costs that arise from the distribution and the involved additional communication are not focus of this paper.

In our opinion, the framework will at least potentially reside both, on the terminal (or terminals), as well as on servers (devices with specialized or magnitudes higher levels of resources). Depending on the terminal capabilities, different levels of distribution can be envisioned. We consider that either whole logical components, as described above, reside on the terminal(s), or that even those are taken further apart, and structured into subcomponents, which are distributed.

The connection between different components is handled through a common component model (e.g. web services). For example, it would be possible to structure a reasoning mechanism into different software components, and implement the same component in Symbian, MIDP, and Java. Then, depending on the terminal capabilities, one component can run as a Symbian version on a Symbian terminal,

while for another user a MIDP version of the same component runs on a MIDP terminal. Another component is being executed also on the Symbian terminal of the first user, while for the second user, this same component resides on a server. Finally, a third component runs only on the server, for both users. All three components together constitute one reasoning engine. The distribution of other logical components of the framework, as well as the components making up the application logic, etc. follow a similar way. We believe that bringing these components together using a common component model is feasible.

Furthermore, from the perspective of data availability, we envision a system that collects as much data on the user device, as possible. Once the available memory is consumed, the least relevant data (presumably the oldest datasets), is pushed to a server mechanism for storage, and can then be overwritten with the latest data. This kind of caching mechanism could be used by all the different components of the framework.

In case the terminal requires data that has already been pushed to a storage mechanism and overwritten, it needs to be retrieved again, in order to allow the reasoning mechanisms on the terminal to function. On the other hand, in case the data is required on a server mechanism, or in the fixed network, old data would already be available, while the latest data could be provided together with the service (reasoning) request. For this kind of implementation, many caching strategies can be used. Which one appears to have the most benefit in which situation, or the use of several, concurrent caching mechanisms, and in that case also consistency mechanisms, should be studied before the implementation, and would have to be verified once the system is being tested.

As mentioned above, another problem is caused by delays due to the communication between the different components. For this purpose, an efficient event-model, or a set thereof needs to be found and if necessary integrated. Furthermore, security aspects that arise from the distribution need to be taken into account as well.

7 Evolution of the Framework

It is considered that different applications will initially use existing mechanisms., instead of implementing all reasoning and learning mechanisms from scratch. Furthermore, as we gain more experience with all the different existing inference and learning mechanisms, the above structure and the implementation will be re-evaluated and adapted, together with the interfaces between the different components.

An example can be seen in figure 5. While the mechanism in figure 5 is for instance using simple rules, which specify how an application should act under certain conditions, and applies them to a demo-application, based on previously acquired data, another, more advanced reasoning mechanism may take current data into account, and adjusts the application accordingly.

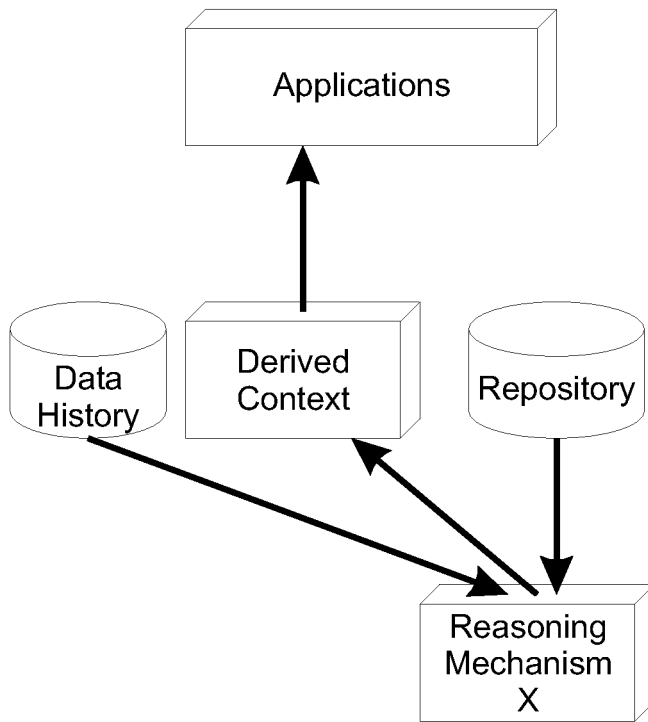


Figure 5. Evolution of the Framework

8 Conclusions

In this paper a framework for context reasoning mechanisms has been introduced. It is based on the commonalities that were found in different inference and learning mechanisms and should provide a solid structure for a reasoning engine that may be part of context-aware applications.

While the framework provides components that different reasoning mechanisms have in common, certain aspects need to be studied further. Amongst those are the suitability of different distribution mechanisms and software-component models. This includes a suitable event and data provision model, that can satisfy requirements, set by applications and their usability. Furthermore, security issues and their influence in this framework need to be investigated further, in order to insure the privacy of the users data, while at the same time providing the data, required for reasoning mechanisms and applications or services.

The implementation of an initial version of this framework, based on a set of sample applications will follow shortly, in order to evaluate the described approach, and refine it further.

References

- [1] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on the What, Who, Where, When, Why and How of Context-Awareness at Conference on Human Factors in Computing Systems (CHI 2000)*, Apr. 2000.
- [2] T. P. Moran. Context-Aware Computing. In *Human-Computer Interaction (HCI)*, volume 16, 2001.
- [3] Petteri Nurmi and Patrik Floréen. Reasoning in Context-Aware Systems; a position paper. <http://www.cs.helsinki.fi/u/ptnurmi/papers/positionpaper.pdf>, 2004.
- [4] M. Wallbaum and P. Dornbusch. Design Considerations for a Platform Supporting Location-Aware Services. In *Proceedings of Multimedia, Internet Video Technologies 2001 (MIV 2001)*, Malta, 2001.
- [5] John. L. West, Grea R. Novotny, Michael R. Fisch, and David Heinman. New Techniques for Fabrication of Flexible Plastic LCD's. *Journal of Information Display*, 2(4), 2001.
- [6] T. Winograd. Architectures for context. In *Human-Computer Interaction (HCI)*, volume 16, 2001.