Rethinking Context Models*

Emiliano Pérez[1], Andrés Fortier[1,2,3], Gustavo Rossi[1,3], and Silvia Gordillo[1,4]

[1] LIFIA. Facultad de Informática. Universidad Nacional de La Plata, Argentina
[2] DSIC. Universidad Politécnica de Valencia, Valencia, España
[3] CONICET
[4] CICPBA
{eperez,andres,gustavo,gordillo}@lifia.info.unlp.edu.ar

Abstract. Since the first context-aware applications were designed, context modelling has played a central role. During the last decade many different approaches were proposed to model context, ranging from ad-hoc models to extensions to relational databases or ontologies. In this paper we propose to take a step back and analyse those approaches using the seminal views presented by Paul Dourish in his work (What we talk about when we talk about context). Based on that analysis we propose a set of guidelines that any context model should follow.

Keywords: Context-awareness, context modelling, pervasive computing, software architectures.

1 Introduction

The basic aim of a context-aware (CA) application is to adapt its behaviour in one or more aspects according to its context. Here, the word *adaptation* is used in a broad sense, comprising actions like changing the application's presentation, the displayed content [1] and performing proactive [2] or reactive actions [3]. However, in order to perform some kind of adaptation, we must first have an internal representation of what is considered context by the application, which in other words means having a context model. This last issue is not a simple one, since the context model highly depends on the application's requirements. In the extreme case, each application may need to define what context is and how it is represented to best suit its needs. On top of that, it is not possible to define beforehand what context will be used for; even the same context model can be used by two different applications to perform completely different things. As an example of these two issues, consider modelling a user's location: while a smart home may need a model based on rooms (i.e. in which room the user is in) a friend finder may need a *(latitude, longitude)* model. On the other hand, an emergency system may reuse the context model used in the friend-finder application, but use it to send an ambulance instead of finding known people.

Defining context is not simple job and many authors have already engaged in that assignment. As Paul Dourish states [4] *"Context" is a slippery notion. Perhaps*

* This paper has been partially supported by the SeCyT under the project PICT 32536.

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 78–87, 2009.

appropriately, it is a concept that keeps to the periphery, and slips away when one attempts to define it. However the important part of his article is not the quote, but the two opposite views of context that Dourish describes. In short, while the "technical" view treats context as a representation issue (i.e. *How do I represent context inside a computer program?*), the "social" view treats it as an interaction issue (i.e. *How does context emerge from the interaction?*). Even though both views are presented as contrary, they are of great importance to CA software engineering, since their underlying nature can help us to model context in our programs and understand how that context is generated.

The aim of this paper is to share our ideas regarding context models and to encourage the discussion around this topic. These ideas are the theoretical emergent of our previous works [5, 6, 7]. In this paper our contribution is two-folded:

- We evaluate different context models types according to the concepts presented in Dourish's article.
- We present a set of preliminary guidelines to be considered when defining context models.

2 What We Talk about When We Talk about Context

In this section we will briefly summarise the two views presented by Dourish [4], since they will be referenced throughout the rest of the paper. The *positivist* view is maybe the one that most software developers consider as straightforward, since it attacks the context modelling problem on a concrete level. In this view the main concern is *how to represent* the context information in a computer, thus converting the problem of modelling context in a representational one. *What* context is and *how* will it be represented depends of the application requirements. We next summarise the four main aspects of the positivist view, as stated by Dourish:

1. Context is a form of information. It is encoded and represented as any other application data.
2. Context is delineable. The application requirements define what pieces of information will be considered as context.
3. Context is stable. As the elements that represent the context can be determined once and for all, the structure of the context doesn't need to change.
4. Context and activity are separable. The approach is only concerned with capturing the data, without keeping a relationship to the action that generated it.

The *phenomenological* view takes an opposite position, since it considers context as an interaction problem rather than a representation one. In this approach the information that represents the context of an entity is subject to the current situation and the point of view of the observer. Context becomes a subjective concept and it is no longer a predefined entity; the focus is now shifted to a *contextuality* relationship between two or more entities, where an *entity becomes contextually relevant to the other in a given moment*. In this view the four key aspects are:

1. Context is a relational property that holds between objects or activities. Something may or may not be contextually relevant to other entity or activity at a given time.

2. Context can't be delineated beforehand, since it is constantly being redefined.
3. Context is particular to each activity or action. Contextual information is an occasioned property, relevant to particular settings, particular instances of action, and particular parties to that action.
4. Context arises from the activity. Contextual information is actively produced, maintained and enacted in the course of the activity at hand, thus context can't be separated from the action(s) that created it.

It is interesting to notice that different ways of defining context have been around for some time in the CA community. As a result two main trends appeared: one where the possible context data was explicitly enumerated [8] (e.g. context is location, time and activity) and a more general one, where any information that can be used to describe a subject's or his medium can be considered context (maybe Dey's [9] definition[1] is the most popular in this area). Instead of advocating for a radical view we consider that a lot can be learned from trying to reach a balance between both approaches. The positivist view has the advantage of placing us (the architects, designers and developers) in a field that we are used to, where the requirements are stated and the problem boils down to design and implement an application. On the other hand, this approach looses many aspects of context interactions and becomes too rigid to finally achieve the original ideas behind UbiComp [10]. In this sense, the phenomenological view is better suited, since it focuses on relationships and how those relationships evolve with time. However, this view has a lack of formality, something required to design and implement an application. Thus a deeper analysis must be made to define the requisites for a context model that can be represented in a program while being flexible to easily accommodate changes.

3 Approaches for Context Modelling

Since the first CA applications appeared the problem of modelling context has been attacked from different perspectives, each one with its specific trade-offs. To analyse them we will use the taxonomy presented in [11] and we will show how some of these categories relate to the presented views of context.

3.1 Key-Value Models

Maybe a first step towards creating a structured context model is to represent context as a collection of key-value pairs. When using this approach the context in encoded in a set of pairs, whose key is generally a unique identifier and its value is the context aspect that the developer is trying to capture. Also, even though it is not a restriction, the context "values" are generally simple data types, like numbers, arrays or strings. A typical example of a user location using this approach would be <'location', (50.9584,-1.2192)>.

[1] Context is any information that can be used to characterise 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.

Context tuples can be managed in different ways according to the architecture. In some cases the tuples are passed from the sensors straight to the upper layers (adaptation and reaction) [12] whereas in other cases tuples are sent to a tuple space that is shared among different processes or applications [13].

This approach for context modelling clearly fits the positivist view better than the phenomenological one, since:

- Context is a form of information and is encoded in tuples.
- Context is delineable because it is explicitly represented by tuples.
- Context may *not* be stable. There is no structure of context and its shape may vary freely, especially when different providers feed a shared tuple space.
- Context and activity are separable. Most approaches take this assumption since there is no association between the tuples value and the process where this information emerged from. However, tuples could be tagged with metadata to keep the link between context data and the activity that produced it.

3.2 Markup-Based Models

Evolving from the key-value approach we find several models that use variations of markup languages (such as XML) to represent contextual information [14, 15]. These models present an improvement over the tuple-based models since they allow hierarchical structures and the possibility to add extra information besides the key-value pair by means of specific tags or tag-attributes.

Mostly, the markup documents (often called profiles [11, 15]) are used to store static information about an object. Because of their nature they are highly portable and thus especially adequate for distributed systems that use hybrid technologies (e.g. web services). On the other hand, profiles are defined as static structures, largely used to describe the capabilities of specific hardware or software components.

Although this approach enhances the previous one form the phenomenological point of view, it is still associated to the positivist view:

- Context is a form of information and is encoded in markup tags.
- Context is delineable because we can determine which aspects will be relevant to each profile following the XML schema.
- Although it may be built dynamically, context is well structured and usually stable since it is explicitly represented by serialised XML structures.
- Context and activity are separable. The profiles are independent tagged documents and are configured statically prior to its use.

3.3 Relational-Based Models

Another widely used method for building context models is by using a relational database (RDB). This approach has the advantage of being a well-understood technology that is backward compatible with legacy data.

Current RDB context models are used to store preconfigured preferences [16, 17] but have great capability to produce new context dependent information performing specialized queries. In approaches like [16] context is directly considered in the SQL clauses (using views or specifying it in the WHERE clause), while other models use

enhanced DBMS that support special context-dependent clauses (e.g. [17] uses OLAP operators to process CA queries).

In general, RDBs store context information as attributes in relationships tables [17], which means that the context model structure is defined by the RDB layout. In order to change the context shape the database structure has to be modified and although it may not represent a major rewrite, it certainly cannot be done easily at runtime. Approaches like these are best suited for situations in which context relevancy is predefined (user preferences, device characteristics, etc.) or when the functionality of the application is limited to context-dependent data retrieval.

Considering the main aspects of this approach we find that:

- Context is a form of information stored in relational database tables.
- Context is delineable by the table structure that represents the relationship between the context information and the entities.
- Context structure is stable. Changing the context shape implies redefining the RDB structure, which is almost never done at run time.
- Context and activity are separable. Databases can only represent the context data in tables, thus losing the link to the activity that created it.

3.4 Ontology-Based Models

Ontologies are used to define relationships between concepts and later use that information for reasoning. An ontology consists mainly of a knowledge base (KB) that represents the domain concepts and the relationships between them. The information in an ontology is accessed and processed by interpreters (also called reasoners or reasoning engine) [18] independent to the KB itself. Because of this decoupling, a specific KB can be used by different systems for different proposes.

Ontologies support incremental modification of the KB in a collaborative fashion and allow for two main kinds of reasoning [19]. The first one is to infer new relationships from the existing ones (e.g. transitive relationships, inverse relationships, etc.) whereas the second is to express new rules in first order logic predicates (e.g. if a condition is met, a new relationship is created). For instance, an ontology that models the user location can be used to easily convert it between different representations using reasoners (e.g. from GPS coordinates to streets).

The flexibility and benefits of ontologies come at a cost, since the concepts and relationships must be built and agreed by a set of users. Also, conflicts may arise regarding the ontology model. Because of this, to think of an ontology general enough to model any context domain that is effectively usable seems hardly feasible. However, we believe that once defined a particular context domain (such as location, activity, mood, etc) ontologies are of great help to develop CA systems.

Regarding the use of ontologies for modelling context we can summarise it in the following statements:

- Context is information and it is stored in dynamic documents or databases.
- Context is not necessarily delineable because context-relevancy can be determined dynamically by the use of reasoners.

- Context structure is not stable. The evolution capabilities of ontologies allow the structure of the context information to evolve from the use and configuration.
- Context can evolve from activity. This relationship can be expressed using reasoners that react upon the current situation.

3.5 Conclusion

All these models present different characteristics, but in general they all describe the context as data somehow related with the domain model. Although they all aim to solve similar problems, each approach is intended for a particular setting and has a specific scenario for which it was developed.

In Section 2 we presented the phenomenological view as an interesting way to *think* about what context is and how it is relevant to entities, while in this section we made a brief analysis on current ways to *represent* the context information in computing software. Most of the approaches revised take the positivist point of view, being the ontology-based models the ones that are closer to the phenomenological view. In the following section we will aim for a balance between the two interpretations in order to consider the philosophy behind the concept of context, without forgetting that we need to represent it as information usable by a program.

4 A Balance between Positivism and Phenomenology

The phenomenological view seems to characterise context in a more *realistic* way than the positivist one. Consider a typical mobile user who is permanently exposed to social interactions. Such scenario is bound to have unpredictable and constantly changing *contextuality* relationships. However, in order to incorporate CA behaviour in our software we need some sort of formalisation; we must use a context representation that characterises these relationships between objects and situations. Ultimately we must cope with the tension between the phenomenological and positivist approaches, tackling the representational problem of context in a way that is as close as possible to the phenomenological approach.

To perform this analysis we decided to restrict ourselves to the object oriented paradigm and evaluate how the phenomenological ideas could be applied to an OO context model. To keep things simple we use the "pure" definition of the OO paradigm [20], where an application can be seen, in a reductionist way, as a set of objects collaborating with each other by sending messages. Thus the behaviour of an application is scattered in a set of objects, which are responsible for implementing certain responsibilities [21]. This basic statement, which may seem trivial at first, it's actually one of the cornerstones for our proposed guidelines.

From the characterisation of the phenomenological approach we can see that context is not data floating around in our program or a row in a database. When we refer to context, we are referring to what is contextually relevant for someone (or something) at a given point. Here we would like to stress the mention of the subject (*someone* or *something*), which means that context can't exist by itself. Translating this idea to the OO paradigm, modelling context becomes modelling what is

contextually relevant for a *given object*. This idea applies both to domain models that already exist (e.g. knowing the location of a *student* object in a university information system) or to entities that were not conceived before (e.g. to adapt our application to the network's bandwidth we must model the network connection first). We consider this statement so important that is actually the first of our design guidelines:

> 1. Context information is always bound to an object. In order to manage context information, we must first define whose context it is.

By applying this first guideline an interesting characteristic arises regarding how context information is arranged in an application: since the context is part of an object, there is no notion of a context repository or database. In fact, context appears as distributed information and "the context" (as referred to in many publications) is actually the aggregation of each object's context. Thus, our second guideline states:

> 2. Context is not a monolithic piece of data, but information distributed across objects in our application.

To clarify this first two guidelines consider a system where services are provided to a user. In such system we would find classes like *User* and *Service*. If we want to support location-based services (e.g. showing restaurants near the user) we would need to associate context information to the user object. Now suppose that we also want to support interaction with other users to provide location based services (e.g. sending an invitation for launch to a group of friends and finding the restaurant that is convenient for all of them). In our approach this requirement is satisfied naturally, since the group context is actually the aggregation of the individual context of each user. Both guidelines are addressed in our previous work [6] by *aware objects* and *context features*.

Different applications may have different context requirements, even for the same application object. For example, the user's location is a required feature for a route planner but for a CA agenda it may be an optional feature; since it can be useful to provide a better service, but it is not mandatory. Finally the user's location may be of no use for an application whose adaptation behaviour is to be able to present information on different devices. However, all the applications mentioned before may have as a central actor the same user object (e.g. representing the owner of a PDA). These different ways of viewing a user's context can be related to the work of Gershenson [22], who distinguishes the notions of absolute (a-being) and relative (re-being) being. As defined by the author, the a-being is the absolute and infinite being, independent of the observer. On the other hand, the re-being is how an entity is represented and treated by an observer, shaped by the current context of the observer and the entity. Thus, when modelling an object's context we are choosing a specific view of the subject and deciding what is contextually relevant. This leads to the third design guideline:

> 3. A context model should support different context representation of the same subject, according to the adaptation required.

This guideline is achieved in our prototypes [6, 7] by the use of *adaptation environments*.

If we go back to Dourish's work on context views, a conflicting issue arises: the positivist view assumes that the context "shape" (i.e. what do we consider to be context) is fixed while the application is running, whereas the phenomenological view argues that context is constantly being reshaped. This reshape can be the outcome of losing the means to acquire an existing context feature (e.g. losing a GPS signal tracking a location) or a change in the application functionality (e.g. adding time constraints to the location based services). As a result, we would expect the context of any object to be re-shaped due to different forces (sensor availability, privacy, contextual relevance, etc). From this observation we derive the fourth guideline:

> 4. The context shape associated to an object should be changeable at run time.

From the designer point of view, context modelling is a difficult task since a balance between flexibility and reusability must be met. In other words, we would like to have a structured context model that allows high-reuse while at the same time we would like our context model to be as flexible as possible. To handle this issue (and taking into account our previous guidelines) the context model should allow different context-domains to be modelled with different approaches. Thus, we may find useful to model a user's location with an ontology, while his (static) personal data is stored in a profile:

> 5. Context should be separable and modelled in a domain-basis, allowing each context domain to be realized using a different modelling technique.

Finally, a topic that we must address is the second item in the positivist characterisation of context. This item states that context is delineable for an application and that this can be done in advance. This is another issue that we must balance, since we must have a set of fixed requirements to develop an application but we must be flexible enough to quickly accommodate new requirements. In our approach we consider that it is impossible to define in advance all the possible types of context an application can use. Each application will have its own context requirements and it is very likely that future killer applications make use of context information in novel ways. Thus, instead of trying to build *the* context ontology we rather prefer to mount a skeleton that allows new domains to be defined and quickly prototyped to suite the application's needs. This leads to our sixth guideline:

> 6. A context model should define a basic structure and be extensible to accommodate new context requirements.

In our approach [6] these last three guidelines are addressed by the relationship between the aware objects and the context features, since run-time changes are naturally supported and each context domain is encapsulated inside a context feature.

The guidelines presented so far are the result of trying to reach a balance between the two views presented by Dourish. To end this section we will analyse our proposal in the same way we did with the other approaches:

1. Context is a relationship between objects. An object is, at a given time, contextually relevant to other object(s).
2. Context is delineable by accounting the relationship between objects.
3. Context may not be stable. There is no restriction regarding the lifetime of the relationship between objects.
4. Context and activity are separable. Even though this is true, what is not separable is the context from its subject. If needed, by using a Command [23] pattern we can even associate actions with context.

By specifying our guidelines our aim is to take Dourish views of context to a more concrete level, where the requirements for context models can be stated. Since this is an ongoing work, these guidelines should not be considered as definitive principles, but as a starting point to define what we need to build scalable context models.

5 Discussion and Further Work

In this paper we have presented a set of guidelines for creating flexible context models. These guidelines are not tied to a specific programming language or technology, since we aim to express them as universally as possible. Our only assumption throughout the paper is that the context model will be implemented using the object-oriented paradigm.

The guidelines presented are the theoretical emergent of different applications and case studies we developed. We are currently working on a context model that follows these guidelines, which is based on a reflective layer that allows us to attach context information to any object in the application model and then build adaptation modules for different context-aware applications.

On a more theoretical side we are currently analysing the relationship between the underlying application model and those objects that account as context. As Dourish states *"The participants may change the subject or otherwise turn a matter from one of middling relevance to central relevance […]. They might turn the location of the conversation from "context" to "content" by remarking that it's too public a place and perhaps they should move elsewhere.* This means that, what is considered context at a given point, may be later considered as core application model (i.e. that the context has gained enough relevancy to become core behaviour) and vice versa. Coping with these changes is still and open issue for us.

References

1. Pascoe, J.: Adding generic contextual capabilities to wearable computers. In: IEEE International Symposium on Wearable Computers (1998)
2. Leonhardt, U.: Supporting Location-Awareness in Open Distributed Systems. PhD thesis, Dept. of Computing, Imperial College (1998)
3. Lamming, M., Flynn, M.: Forget-me-not: Intimate computing in support of human memory. In: Proceedings FRIEND21 Symposium on Next Generation Human Interfaces (1994)

4. Dourish, P.: What we talk about when we talk about context. Journal of Personal and Ubiquitous Computing 8(1), 19–30 (2004)
5. Rossi, G., Gordillo, S., Challiol, C., Fortier, A.: Context-Aware Services for Physical Hypermedia Applications. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1914–1923. Springer, Heidelberg (2006)
6. Challiol, C., Fortier, A., Gordillo, S., Rossi, G.: Architectural and Implementation Issues for a Context-Aware Hypermedia. Journal of Mobile Multimedia (2008)
7. Challiol, C., Fortier, A., Gordillo, S., Rossi, G.: A flexible architecture for context-aware physical hypermedia. In: DEXA 2007: Proceedings of the 18th International Conference on Database and Expert Systems Applications, pp. 590–594. IEEE, Washington (2007)
8. Brown, P.J., Bovey, J.D., Chen, X.: Context-aware applications: from the laboratory to the marketplace. Personal Communications, 58–64 (1997)
9. Dey, A.K., Abowd, G.D., Wood, A.: Cyberdesk: a framework for providing self-integrating context-aware services. Knowledge-Based Systems 11, 3–13 (1998)
10. Weiser, M.: The computer for the 21st century. In: Human-computer interaction: toward the year 2000, pp. 933–940. Morgan Kaufmann, San Francisco (1995)
11. Strang, T., Linnhoff-Popien, C.L.: A context modelling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management. UbiComp, Nottingham, England (2004)
12. Samulowitz, M., Michahelles, F., Linnhoff-Popien, C.L.: Capeus: An architecture for context-aware selection and execution of services. In: New developments in distributed applications and interoperable systems (2001)
13. Schmidt, A., Van Laerhoven, K.: How to build smart appliances? Personal Communications, 66–71 (2001)
14. Han, J., Cho, Y., Choi, J.: A Workflow Language Based on Structural Context Model for Ubiquitous Computing. In: Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J. (eds.) EUC 2005. LNCS, vol. 3824, pp. 879–889. Springer, Heidelberg (2005)
15. WAPFORUM, User Agent Profile (UAProf), `http://www.wapforum.org`
16. Bolchini, C., Curino, C.A., Orsi, G., Quintarelli, E., Rossato, R., Schreiber, F.A., Tanca, L.: And what can context do for data? Communications of ACM (to appear)
17. Stefanidis, K., Pitoura, E., Vassiliadis, P.: On Supporting Context-Aware Preferences in Relational Database Systems. In: International Workshop on Managing Context Information in Mobile and Pervasive Environments (2005)
18. Chen, H., Finin, T., Joshi, A.: Using owl in a pervasive computing broker (2003)
19. Hong, M., Cho, D.: Ontology Context Model for Context-Aware Learning Service in Ubiquitous Learning Environments. International Journal of Computers 2(3), 172–178 (2008)
20. Kay, A.C.: The early history of smalltalk. SIGPLAN Not. 28, 69–95 (1993)
21. Wirfs-Brock, R., Mckean, A.: Object Design: Roles, Responsibilities and Collaborations. Addison-Wesley, Reading (2002)
22. Gershenson, C.: Contextuality: A Philosophical Paradigm, with Applications to Philosophy of Cognitive Science, POCS Essay, COGS, University of Sussex (2002)
23. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley Professional, Reading (1995)