Engineering Context Aware Enterprise Systems

Paddy Nixon[1,2], Feng Wang[1], Sotirios Terzis[1], Tim Walsh[1] and Simon Dobson[2].

[1]*Global and Pervasive Computing Group*
Department of Computer and Information Sciences
TheUniversity of Strathclyde
GLASGOW, Scotland.
{Paddy.Nixon, Feng.Wang}@cis.strath.ac.uk

[2]*Aurium*
Clifton House,
Lower Fitzwilliam Street, Dublin 2
Ireland
Simon.Dobson@aurium.net

1 Introduction

According to Anind Dey[1], context is: "Any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects."

Today's computer systems are unaware of the user's context. They do not know what the user is doing, where is the user, who is nearby and other information related to the user's environment. They just take the explicit input from the user, process it, and then output the result. Regarded as computing for the next generation, pervasive computing will greatly change the way computers behave. The basic idea is to instrument the physical world around us with various kinds of sensors, actuators, and tiny computers. The huge amount of information can then be collected and processed by computer systems, enabling computer systems to deduce the user's situation and act correspondingly with user's intervention.

One demanding challenge for pervasive computing is how to collect and process data from sensors and other sources. Most early researchers built their solution in an ad hoc way to investigate the problem space [2][3]. They had to consider everything, including the details of reading sensor data, distributing sensor data, and transforming sensor data into high-level data as well as application adaptation behaviour. From software engineering perspective, this makes developing applications for pervasive computing very cumbersome.

2 Related Work

Realizing the limitation of ad hoc solution, Anind Dey et al. at Georgia Institute of Technology developed a framework and a toolkit (Context Toolkit) to simplify the collecting and processing context data. In the framework, there are three kinds of objects: widget, aggregator, and interpreter.

- A widget collects sensor data from sensors. Widget provides a unified interface to sensors and a unified interface to applications.
- An aggregator aggregates data related to one entity such as a user from several sources so that the application does not have to contact the data sources individually.
- An interpreter translates low-level sensor data into high-level data that can be used by applications directly.

The Context Toolkit provides common functionality such as communication between context components and encoding of context data. Consequently application developer only needs to add sensor-specific code for each sensor and tailor context-dependent processing code.

The Context Toolkit provides a good starting point. However, it has the following disadvantages. At design time, the developer has many choices when he decomposes the task of gathering and processing context information into a set of widgets, aggregators and interpreters. But after the decision has been made and these context components are built, they become fixed. If the application has requirements for new context information at run time, the function of context components cannot easily be extended even this new context information can be computed based on existent context information. The developer has to redesign the context components. This indicates that the developer has to foresee the requirement of applications at design time, which is unrealistic in pervasive computing environment where changes are frequent. Besides, pervasive computing systems must tackle the problems of scalability. What we need is an infrastructure that supports dynamic composition of modular context components at run time.

Guanling Chen and David Kotz [4] have proposed Solar as an infrastructure for collecting and aggregating data in ubiquitous computing environment. In their infrastructure, all the communication between context components is through events thus address some of connection issues. Solar does support dynamic composition of context components. Dynamic composition results from changes in the environment. For example, one application requires the following information: the status of the printer in the same room as the user. If the user moves from one room to another, the binding to the printer will change from the printer node in one room to the printer node in another room. Besides the system-provided context components, it also supports using user-provided context components. It requires the application developer to explicitly specify the composition graph of context components. The event publishers in the graph can be either the sensor or the output of some subgraph exposed by some applications. The infrastructure will try to find the common part of context processing graph of different applications and reuse them, thus improve scalability.

Since the foci in Solar are scalability and flexibility, they have not considered the robustness issues. In pervasive computing, the same context may come from several sources and the data sources may become available or not available due to user movement or component failure. The requirement that the application developer has to explicitly choose data source, choose context operators and specify the context-processing graph will affect the robustness of the context system. For example, indoor location can be provided by the Active Badge system [9] or the Cricket system [10]. If both location systems work at the same time and provide location information for one user, can the application use both systems to get user's location? In Solar there is a context-sensitive naming service, can the output of two processing graphs register under the same name? If one system acts as backup for the other in case of failure or environment changes, how to instruct the system switch from one data source to the other? Although it is possible to construct a third operator to do this, the solution is not flexible enough in case of more data sources are available. So Solar does not provide an inherent robust solution.

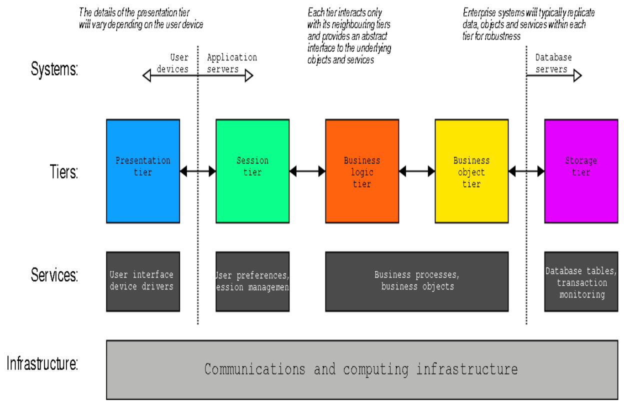
iQueue [5] from IBM research is a pervasive data composition framework. In the framework, the input of each data composer is defined by an abstract data requirement, which is based on date type and conditions over the attributes of the type of data. The data resolver will receive periodic advertisement from available data sources and try to match the input data requirement of a data composer with the data sources available in the environment and binds the composer to the data source. The matching process is based on the comparison of attributes of data source and composer. It happens dynamically. If new data sources appear or existent data sources disappear, the run time will get notification and change the binding dynamically. One data source requirement can be constructed dynamically from other data sources. The binding of a composer's data specification to other composers can result in a hierarchy of composers. So through abstract data specification iQueue supports dynamic binding to different data sources. However, iQueue requires that the data provider must be active before the matching and binding process happen no matter the data provider is a data source or the output from other composers. We observe that the same context can have different representations. We can get other representations after we do some processing on one representation. For example, a person's location may have different representations. It can be represented as coordinates or as room number or floor number. By doing some operation on coordinates, we can get the room representation. More generally, some data sources may not directly match the data requirement. But after being further processed by a sequence of operators, they will match the data requirement. For example, there exist some software components that can return a list of places that are near your current location. One component can return a list of theatres. Another one can return a list of restaurants. Still another one return hotels, etc. Depending on the representation of the location information available and what kind of places the user is interested, the number of combination will become quite big. It would be inefficient to create the graph before hand and advertise it to the data resolver. For data sources like this, iQueue does not provide a good solution.

Some researchers have proposed using automatic path creation [11][12][13] to provide customized networked services. Although their aimed problem domains are a little different, they share the same ideas. Based on the type of data, a system service can build a data flow path from sources to required data automatically. In Iceberg [11] and CANS [13] this path creation process will dynamically adjust the path according to the resources changes. Automatic path creation is suitable for problems when some elements of the processing graph are unknown before run time.

Most of the above supporting infrastructures aim at tackling dynamic context, which change very fast. The changes of dynamic context are usually event driven. However, according to the definition of context, there are also context information that rarely changes, such as the relation between room and a building and context that changes very slowly such as such as personal preference, a person's role in one organization, the relation between printer and room. This value of this kind of context information may be assigned manually by the user or is computed from external data sources. In order to support context-aware applications, the infrastructure should support both static context and dynamic context. There are already some attempts to organize the context information. In the Sentient Computing project [6], both the static environment and the dynamic world are modeled as objects. The static context includes the containment relationship between room and building. The dynamic context information includes user's location. The persistent state of the object is stored in an Oracle database. However, the world model is quite informal and only suitable for location-based systems. In HP Cooltown project [7], each entity such as a person, a place, and a device has a corresponding web address. The web pages are the descriptions of each entity. The information in the web pages is unstructured and intends to be used by the user directly rather than other applications. In Context Shadow [8], cross-references are set up between different entities. Sensors and services related to one person are organized together, creating a searchable topology of context information. Its implementation is based on IBM TSpaces.

3 The Engineering Problem

Modern enterprise systems architecture is built around a "tiered" model. There is much discussion as to exactly how many tiers should be used in an application, the details of what each tier provides, etc. However, the common theme is that each tier represents a *level of abstraction* in the design of the system, allowing developers to focus their attention at one level (for example business logic) without having to worry about the details of other levels (such as storage management). For this approach to function, each tier needs to export an agreed set of interfaces exposing its functions to its neighbouring tiers. The implementation of these interface functions may be changed freely, and these changes will not affect other tiers accessing them through the interface. A typical model is the five tier architecture shown below:



The details of the presentation tier will vary depending on the user device
Each tier interacts only with its neighbouring tiers and provides an abstract interface to the underlying objects and services
Enterprise systems will typically replicate data, objects and services within each tier for robustness
Systems:
User devices
Application servers
Database servers
Tiers:
Presentation tier
Session tier
Business logic tier
Business object tier
Storage tier
Services:
User interface device drivers
User preferences, session management
Business processes, business objects
Database tables, transaction monitoring
Infrastructure:
Communications and computing infrastructure

Figure1: A five-tier architecture

This industry accepted architectural style has evolved, most recently in areas such as webservices, because it supports a stable and appropriately partitioned design approach. However, consider this approach from the perspective of providing dynamically adaptive systems which are:

- User centred – configured on the fly for the user;
- Supporting dynamic aspects;
- Respnsive to the changing environmental characteristics; and
- Responsive to the different networked appliances in the proximity.

Such systems must take account of changing and varying contextual information. In the rest of this paper we consider the industrial needs for architectures that support context as a core concept.

Adding context

As organisations deliver context-enhanced applications to their users, context services will be integrated into the tiered, web-service-based enterprise architectures that are becoming recognised as best practice. Imagine that you can monitor your users physical activity and location and then tailor the response of the local appliances to that users needs. This is what context can do for a system. Context enhancement brings together a collection of information streams - location, diaries, preferences, time, appliances characteristics, access policies and situation - the context - and reacts to combinations of this information based on a simple scripting interface. It is true to say that most context solutions can be achieved by developing point solutions: however these become complex very quickly. In this paper we outline a single platform based development solution that can be used to transform software components and applications into context aware solutions.

The impact of context on architecture

The first question to ask is: how does the use of context affect the *purpose* the architecture is intended to serve? The most obvious impacts of context are in the front tiers - presentation and session. An application may wish to change its presentation based on a user's context, for example by moving automatically from an instant message to an SMS when a user leaves their office.

However, we may also see contextual changes on other tiers. For example, some processes may be intrinsically "easier" in some contexts than others, and could be streamlined. Alternatively a system might adapt the processes available in a session to avoid those that would be inappropriate for whatever reason, or provide monitor service delivery and react to potentially costly breaches in service-level agreements.

This cross-tier impact is what differentiates full-on contextual enhancement from simple location-based services, device transcoding or personalisation. A simple presentation tier add-on such as transcoding, for example, may allow applications to target the user's device but cannot capture and express the business implications of this choice - such as a reduction in security or attention, or the reduction in detail in the services being provided. A context-enhanced application, by contrast, can draw application-level implications from the low-level information about devices, allowing adaptation across the whole application. This removes context enhancement from the "gimmick" category and moves it to the status of an infrastructural service with enterprise impact.

Context: conditioning the layers

In general, one may view context as conditioning the behaviour of the tier in an architecture. That is to say, the way the tier provides its functions will be affected by the context in which those functions will be used.

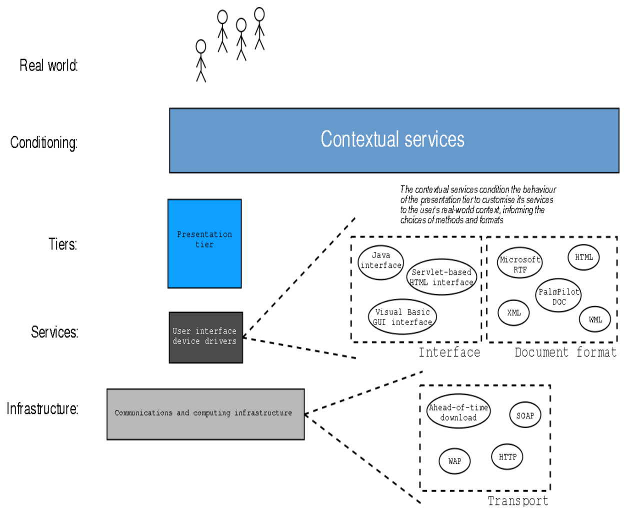

Real world:
Conditioning:
Contextual services
Tiers:
Presentation tier
The contextual services condition the behaviour of the presentation tier to customise its services to the user's real-world context, informing the choices of methods and formats
Services:
User interface device drivers
Java interface
Servlet-based HTML interface
Visual Basic GUI interface
Microsoft RTF
HTML
PalmPilot DOC
XML
WML
Interface
Document format
Infrastructure:
Communications and computing infrastructure
Ahead-of-time download
SOAP
WAP
HTTP
Transport

Figure 2: Context Presentation

The effect of context is to inform the way in which tiers dispatch their exported functions to the objects and services used to provide them. For example, contextualising the presentation tier above may cause a single user interface function (such as "show the user this alert") to be implemented differently whether the user is using a workstation or a small device, or in one location rather than another.

However, context can affect more than simply presentation, and it is here that the differences between context and simple location services or user preferences becomes apparent. An interface might change because of device or location - but also because of task, or the presence of other people, or some other high-level trigger. By maintaining a contextual model of users, applications can leverage contextual triggers across their entire operation rather than simply as add-on personalisation or location adaptation in the front tiers. The model is uniform and can be used, and make use of, elements deriving from anywhere in the architecture, not simply from user interface cues.

Integrating contextual services into a tiered architecture

The philosophy behind tiered architectures is to separate concerns in a system into different levels of abstraction and then encapsulate each level behind its own interface within its own distributed service. There is a degree of "linearity" implicit in the approach, in that tiers interact with their neighbours and do not "jump" to use tiers arbitrarily. This preserves the abstraction boundaries.

Contextual modeling has an end-to-end impact, however, and so is not a "neighbour" of *any* tier. For maximum effect a contextual model should accept information from *anywhere* and be used *everywhere*.

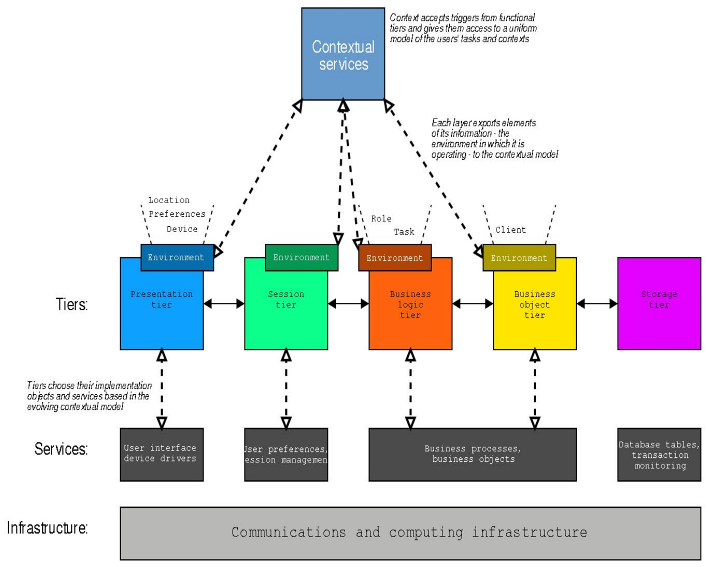

Contextual services
Context accepts triggers from functional tiers and gives them access to a uniform model of the users' tasks and contexts
Each layer exports elements of its information - the environment in which it is operating - to the contextual model
Location
Preferences
Device
Role
Task
Client
Environment
Environment
Environment
Environment
Tiers:
Presentation tier
Session tier
Business logic tier
Business object tier
Storage tier
Tiers choose their implementation objects and services based in the evolving contextual model
Services:
User interface device drivers
User preferences, session management
Business processes, business objects
Database tables, transaction monitoring
Infrastructure:
Communications and computing infrastructure

Figure 3: Integrating Context

One way to resolve this is to situate context modeling outside the normal tier structure. This does not violate the attractive properties of the tiered approach, as the context model can present a uniform, well-encapsulated interface to every tier.

The model takes input from the *environment* of each tier affected. For the presentation tier, this might include the user's device, preferences and location; for the business logic tier their tasks and roles within the application. It unifies these environmental factors into a model of the systems users' that may then be used to inform the selection of services within each tier.

5 Implications for tier design

This approach - providing a uniform model accessible across the architecture - does have some implications for tier interface design and function selection.

The most important factor is that context forces abstraction upon tiers. This is most obvious in the presentation tier: if the details of the interface can be changed by context, then the tier interface can only expose abstract functions rather than detailed access to the individual component objects that might be used. The reason for this is simple: if the details are exposed and then change, all tiers accessing those detailed functions are impacted.

More subtly, the tier cannot allow any assumptions about interfaces to propagate across its interface. This is sometimes more difficult. Consider, for example, the case where the application wants to "alert" the user to some event. Not all interfaces have obvious alert capabilities: HTML pages, for example, are not typically "pushed" at the client. Providing the alert function may have implications for the design of the exposed functions in the interface.

6 Conclusion

This paper sets out to provide a guide to engineering context-enhanced services in light of existing IT architectures. We identify that context affects architecture by providing a uniform and well-founded framework within which to control and adapt the behaviour of a system to changing user circumstances. There are implications in the design of tiers and their interfaces needed for these benefits to be fully realised, but these changes reflect good software engineering practices and have other benefits anyway. Many development tools provide the basics of context, mainly focused on the presentation tier of an architecture. However, the major benefits accrue from the end-to-end use of context throughout an enterprise architecture. The architecture described in this paper directly addresses this end-to-end problem.

References

1. Anind K. Dey. Providing Architectural Support for Building Context-Aware Applications. PhD dissertation. Georgia Institute of Technology, November 2000.Baldonado, M., Chang,
2. Long, Sue, Rob Kooper, Gregory D. Abowd and Christopher G. Atkeson (1996). Rapid prototyping of mobile context-aware applications: The Cyberguide case study. In the Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (MobiCom '96), pp. 97-107, White Plains, NY, ACM. November 10-12, 1996.
3. Abowd, Gregory D., Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper and Mike Pinkerton (1997). Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks 3(5): pp. 421-433. October 1997.
4. Guanling Chen and David Kotz. Supporting Adaptive Ubiquitous Applications with the Solar System. Technical Report TR2001-397, May, 2001.
5. Norman H. Cohen, Apratim Purakayastha, Luke Wong, and Danny L. Yeh. iQueue: a pervasive data-composition framework. The 3rd International Conference on Data Management, Singapore, Janauary 8-11, 2002
6. A. Harter et al., “The Anatomy of a Context-Aware Application,” Proc. 5th Ann. Int’l Conf. Mobile Computing and Networking (Mobicom 99), ACM Press, New York, 1999, pp. 59-68.
7. Kindberg, Tim et al. People, Places, Things: Web Presence for the Real World. Technical Report HPL-2000-16, Hewlett Packard Labs.
8. Martin Jonsson. Context Shadow: A Person-Centric Infrastructure for Context Aware Computing. The FEEL project, available at http://www.dsv.su.se/FEEL/zurich/Item_9-Context_Shadow-A_person-centric_Infrastructure_for_context_aware_computing.pdf
9. Nissanka B. Priyantha, Anit Chakraborty, Hari Balakrishnan, The Cricket Location-Support system, Proc. 6th ACM MOBICOM, Boston, MA, August 2000.
10. A. Harter and A. Hopper. A Distributed Location System for the Active Office. IEEE Network, 8(1), 1994.

11. Achieving Service Portability using Self-adaptive Data Paths, by Z. Morley Mao, Randy H. Katz IEEE Communications Magazine special Issue on Service Portability and Virtual Home Environment, January 2002.
12. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment, by Emre Kiciman and Armando Fox. In Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing 2000 (Lecture Notes in Computer Science, Springer Verlag). (HUC2k), Bristol, England, September 2000.
13. CANS: Composable, Adaptive Network Services Infrastructure, Xiaodong Fu, Weisong Shi, Anatoly Akkerman, and Vijay Karamcheti, USENIX Symposium on Internet Technologies and Systems (USITS), March 2001.
14. Nelson G. "Context-Aware and Location Systems". PhD Thesis. Cambridge University Computer Lab, UK, January 1998