BY **Joëlle Coutaz, James L. Crowley, Simon Dobson,** AND **David Garlan**

Context is not simply the state of a predefined environment with a fixed set of interaction resources. It's part of a process of interacting with an ever-changing environment composed of reconfigurable, migratory, distributed, and multiscale resources.

CONTEXT *is* KEY

Since the early 1960s, the notion of context has been modeled and exploited in many areas of informatics. The scientific community has debated definitions and uses for many years without reaching clear consensus [4]. Nonetheless, it is commonly agreed that context is about evolving, structured, and shared information spaces, and that such spaces are designed to serve a particular purpose. In ubiquitous computing, the purpose is to amplify human activities with new services that can adapt to the circumstances in which they are used.

Figure 1. In the Olympic café, Bob and Jane use the objects on the table to illustrate their ideas for the layout of the city they are planning together.

Our purpose here is to illustrate how the challenges of large-scale ubiquitous computing can be tackled with a structured, flexible approach to context. The key lies in providing an ontological foundation, an architectural foundation, and an approach to adaptation—all of which scale alongside the richness of the environment.

Ubiquitous computing embraces a model in which users, services, and resources discover other users, services, and resources, and integrate them into a useful experience. The two critical processes are to recognize users' goals and activities, and to map these goals and activities adaptively onto the population of available services and resources [5]. Context informs both recognition and mapping by providing a structured, unified view of the world in which the system operates. This is an ambitious goal, especially compared to current systems that need extensive manual configuration. The following examples make the challenges concrete.

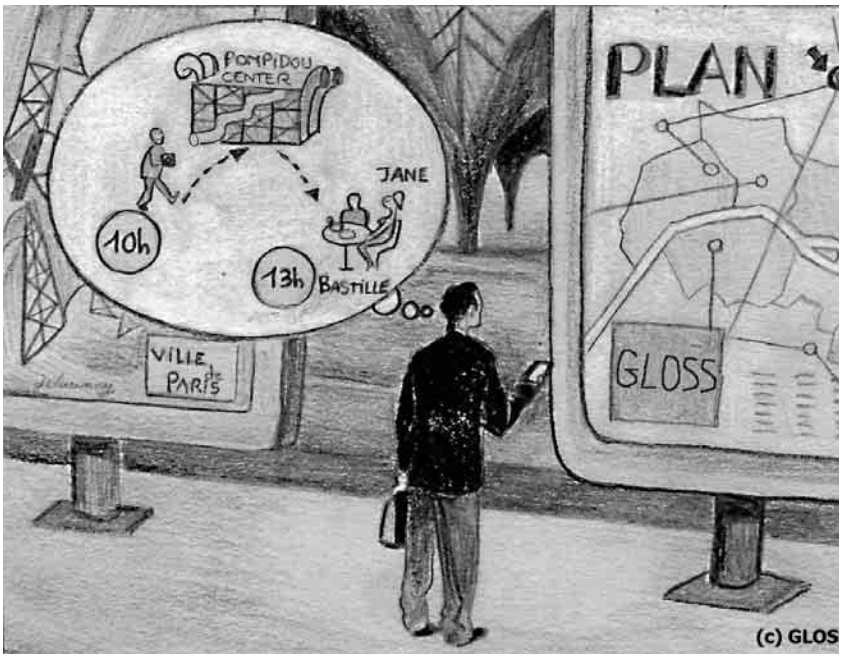


Figure 2. Bob looks for information from an active map in the train station.

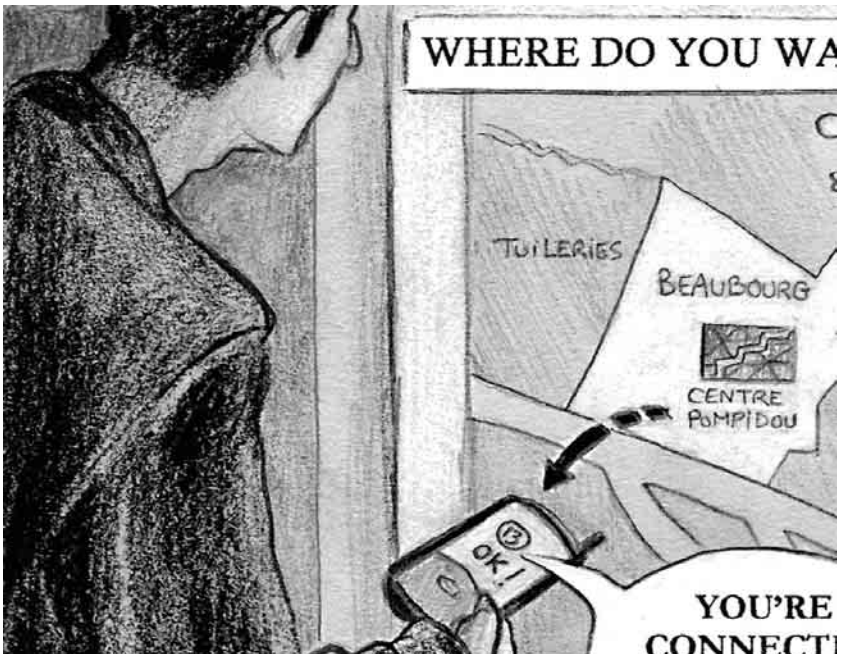
The Hearsay service developed as part of the GLOSS project allows users to pick up small notes left for them in the environment [2]. It makes sure users will find the message only if their context is correct (right person in the right place at the right time). The same approach is applied to other applications, providing a structured link between environment and behavior to improve utility and usability. In Figure 1, the computer-vision system used to identify and track the objects manipulated by Bob and Jane determines what to look at (the objects) and how to interpret (movement, location) based on the context (Bob and Jane's activity).

By approaching the active map, Bob dynamically creates an interactive space from a public hot spot and a private device (Figure 2). Here, the system detects the presence of Bob and his device, it understands that Bob is willing to use the service of the active wall, and it both accommodates the diversity of private devices and provides its own services (for example, printing). In particular, the user interface of the inquiry service must dynamically distribute itself among the resources of the interactive space and dynamically adapt to Bob's private device without creating confusion and distraction.

These simple examples illustrate three issues. Firstly, context is not simply a state but part of a process. It is not sufficient for the system to behave correctly at a given instant: it must behave correctly during the process in which users are involved. Correctness must be defined with reference to the process, not simply the succession of states making up that process. The printing service might be able to select the nearest ("correct") printer at any given point, but if the user is moving while printing a set of documents they may end up dispersed across a bewildering range of printers regardless of the individually correct point choices.

Related to all of this is the issue of a *holistic treatment* of context. The "best" adaptation (by whatever metric) will typically be determined by a fusion of information, often crossing semantic levels. Using the printing service example, the system will select a "correct" printer for documents if it knows the user's eventual location (perhaps derived from a diary entry for a next meeting) and can route documents to the printer nearest to this location. This view of *context-as-process* is more flexible than the simpler view of *context-as-state,* and makes clear the utility and usability of a system are derived from the emergence of information and cooperation rather than the sophistication of its individual components.

The third issue is the risk of engendering a mismatch between the system's model of interaction and users' mental model of the system. For example, how do Bob and Jane know the system understands what they are doing? How can Bob predict (and possibly control) the location of his printed document? In the conventional GUI genre, designers have typically developed prepackaged solutions for a predetermined interaction space. In ubiquitous computing, the interaction space is ill-defined, unpredictable and emerges opportunistically. In



these conditions, new interaction techniques must be devised to minimize system and users model mismatches (see the article by Russell, Streitz, and Winograd in this section). If model mismatches are unavoidable, then another issue is how to detect and correct them. These issues can be addressed using a conceptual framework as a starting point.

A Conceptual Framework for Context-Aware Systems

The conceptual framework we propose includes an ontological and an architectural foundation that structure the adaptation process in a unified way.

Ontological foundation. Context is an information space that can be modeled as a directed state graph, where each node denotes a context, and edges denote the conditions for change in context [1]. Each context is defined by a set of entities (typically including literal values, as well as real-world and information objects), a set of roles (for example, functions) that entities may satisfy, a set of relations between the entities, and a set of situations. Entities, roles, and relations are modeled as expressions over observables captured and inferred by the system at the appropriate level of abstraction. Contexts are defined by a specific set of situations, roles, relations, and entities. For example, figures 1 and 2 denote two contexts: the café context where entity Bob plays the role of an architect, and the train station context where entity Bob plays the role of a traveler. A shift in context corresponds to a change in the set of entities, a change in the set of possible relations between entities, or a change in the set of roles that entities may play (as between the context café and the train station context).

The situations within each context, in turn, may also be organized as a graph of states. Situations denote specific configurations of entities, roles, or relations. Within a context, all situations share the same set of observables, entities, roles, and relations. The current situation is altered by a change in the number of entities (for example, in the "café" context, when Jane enters the café, the discussion can start), or a change in the role assignment to an entity (a lump of sugar is now used to represent a building), or a change in the relations between two entities (when Bob and Jane switch from a face-to-face position to a side-by-side configuration). A graph of situations provides a tool that can be used by software components to predict and adapt according to their contract (for example, the user interface can be rendered differently on the table when both Bob and Jane sit side by side).

Entities, roles, relations, and observables are abstract classes that denote the state dimension of context. They are instantiated within runtime infra-

Context informs both recognition and mapping by providing a structured, unified view of the world in which the system operates. ***This is an ambitious goal, especially compared to current systems that need extensive manual configuration.***

structures that denote the process dimension of context.

Runtime infrastructure model. A runtime infrastructure is middleware that runs reliably and permanently to provide applications with a "public utility" for services. Such a public utility provides economy in program development, and facilitates cooperation among components as well as consistent behavior across applications. An infrastructure for context-aware computing must provide context services accessible from anywhere on the planet, from a fixed network infrastructure to spontaneous interactive islands.

Context services form a fabric structured into multiple levels of abstraction, as illustrated in Figure 3. At the lowest level, the system's view of the world is provided by a collection of sensors that may be physical sensors such as RFID's or software sensors that probe a user's identity and platforms. The *sensing layer* generates numeric observables. To determine meaning from numeric observables, the system must perform transformations. The *perception layer* is independent of the sensing technology and provides symbolic observables at the appropriate level of abstraction.

The *situation and context identification layer* identifies the current situation and context from observables, and detects conditions for moving between situations and contexts. Services in this layer specify the appropriate entities, roles, and relations for operating within the user's activities. They can be used to predict changes in situation or in context, and thus anticipate needs of various forms (system-centric needs as well as user-centric needs). Finally, the *exploitation layer* acts as an adapter between the application and the infrastructure. This is where applications express their requests for context services at a high level of abstraction.

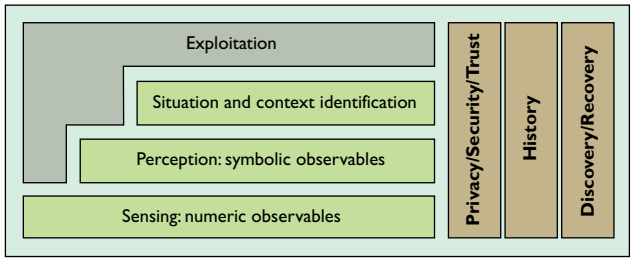


Figure 3. Levels of abstraction for a general-purpose infrastructure for context-aware computing.

These conceptual principles must be tuned and reconciled with practical considerations such as portability (for example, across operating systems), computation cost (for example, memory footprint, latency, bandwidth, and power consumption), and scalability (from microscopic to planetwide). A variety of implementations at different scales and optimized for different targets are needed, united by common interfaces. At every level of abstraction, one must incorporate mechanisms and facilities to support privacy, trust, and security (see the article by Lahlou et al. in this section), as well as history management and discovery/recovery. These protective measures pose the challenge of supporting multiple views of the same information at different levels of abstraction—essentially a truth-maintenance problem over the knowledge base. Indeed, this problem is exacerbated by the dynamic nature of system adaptation and development.

Adaptation and Development

Adaptation allows a system to maintain consistent behavior across variations in operating environments. The environment denotes the physical world (for example, in the street, lighting conditions), the user (identification, location, goals, and activities), social settings, and computational, communicational, and interactive resources. Development refers to the automatic acquisition of situation and context, and ultimately the acquisi-

How can context models evolve and develop without introducing disruption? ***The challenge is to find the appropriate balance between implicit and explicit interaction for providing the feedback required for development.***

tion of the entities, roles, and relations from which situations and contexts emerge.

Correctness in ubiquitous computing may be understood as compliance with a contract, whether the contract stands between software components, or between the system and the human being. A typical contract is that the gross behavior of the system remains the same; for example, the inquiry system used by Bob remains about seeking information in whatever context it executes. However, the detailed behavior may change with context; for example, the filtering criteria based on time and location, as well as the rendering of information based on the resources available in the interactive space. At least from a user's perspective there is often a well-defined behavioral envelope within which adaptation makes sense [3].

Adaptation and development are fundamental to providing useful and usable services to a variety of users in the presence of large variations in resources and activities. Context is too complex to be preprogrammed as a fixed set of stable variables: worse, the contract itself, which defines "correct behavior," is not always precisely specifiable in advance. Thus, the context model, contract, and adaptation process must develop through observation and interaction with the environment. At the same time, this development process must not be disruptive. This creates a dilemma: How can context models evolve and develop without introducing disruption? The challenge is to find the appropriate balance between implicit and explicit interaction for providing the feedback required for development. We must determine the appropriate degree of autonomy, and this problem can impact every level of abstraction.

One attractive solution to these constraints is to observe that the correct selection of elements—including adaptation strategies, interfaces, devices, information—can only be made in the infrastructure. This suggests replacing explicitly coded responses to situations and contexts, which can only accommodate a fixed set of predicates, with a higher-level, more knowledge-intensive use of machine-readable strategies coupled with reasoning and learning. This goes beyond the usual distinction of closed- versus open-adaptive systems [6].

Current learning technologies require large sets of training data—something difficult to obtain for an extensible environment. Nondisruptive development of context models will require new ways of looking at learning, and may ultimately require a new class of minimally supervised learning algorithms that will need to be studied explicitly as part of semi-autonomous systems.

Conclusion

Context is key in the development of new services that will impact social inclusion for the emerging information society. For this to come true, we must find the proper balance between contradictory features. If context is redefined continually and ubiquitously, then how can users form an accurate model of a constantly evolving digital world? If system adaptation is negotiated, then how do we avoid disruption in human activities? We believe that clear architecture and a well-founded, explicit relationship between environment and adaptation are the critical factors; indeed, they are the key that will unlock context-aware computing at a global scale. C

References

1. Crowley, J., Coutaz, J., Rey, G., and Reignier, P. Perceptual components for context-aware computing. In *Proceedings of the Fourth International Conference of Ubiquitous Computing,* (Göteburg, Sweden, Sept./Oct. 2002). Springer, 117–134.
2. Dearle, A., Kirby, G., Morrison, R., McCarthy, A., Mullen, K., Yang, Y., Connor, R.C.H., Welen, P. and Wilson, A. Architectural support for global smart spaces. *Mobile Data Management* (2003), 153–164.
3. Dobson, S. and Nixon, P. More principled design of pervasive computing systems. In *Proceedings of Engineering for Human-Computer Interaction and Design, Specification and Verification of Interactive Systems.* (Hamburg, Germany, July 2004). Springer-Verlag.
4. Dourish, P. *Where the Action Is: The Foundation of Embodied Interaction.* MIT Press, Cambridge, 2001.
5. Garlan, D., Siewiorek, D., Smailagic, A. and Steenkiste, P. Project Aura: Towards distraction-free pervasive computing. *IEEE Pervasive Computing 21*, 2 (Apr.-June, 2002), 22–31.
6. Oreizy, P., Taylor, R., et al. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems 14,* 3 (1999), 54–62.

The GLOSS project was funded by the EU Future and Emerging Technologies initiative as part of the 5th Framework IST program.

Joëlle Coutaz (Joelle.Coutaz@imag.fr) is a professor at Universitè Joseph Fourier, Grenoble, France.
James L. Crowley (James.Crowley@inrialpes.fr) is a professor at Institute National Polytechnique de Genoble, and the director of the Laboratoire GRAVIR at INRIA Rhone-Alpes, Montbonnet, France.
Simon Dobson (simon.dobson@computer.org) is a lecturer in the Department of Computer Science at the University College, Dublin, Ireland.
David Garlan (garlan@cs.cmu.edu) is a professor in the Department of Computer Science at Carnegie Mellon University, Pittsburgh, PA.