HUMAN-COMPUTER INTERACTION, 2001, Volume 16, pp. 257–268
Copyright © 2001, Lawrence Erlbaum Associates, Inc.

Context as a Dynamic Construct

Saul Greenberg
University of Calgary

ABSTRACT

Context is a dynamic construct. Although some contextual situations are fairly stable, discernable, and predictable, there are many others that are not. Similar looking contextual situations may actually differ dramatically, due perhaps to people's previous episodes of use, the state of their social interactions, their changing internal goals, and the nuances of local influences. The consequence is that, for all but simple cases, the designer of a context-aware application may find it difficult or even impossible to (a) enumerate the set of contextual states that may exist, (b) know what information could accurately determine a contextual state within that set, and (c) state what appropriate action should be taken from a particular state.

1. INTRODUCTION: REVISITING THE DEFINITION OF CONTEXT

In the anchor paper, Dey, Abowd, and Salber (2001[this special issue]) ponder the definition of *context.* They first give the Webster's Dictionary defini-

Saul Greenberg is a computer scientist with interests in studying individual and group behavior, articulating interface design principles, prototyping novel systems, developing software infrastructures and physical devices for rapid prototyping, and evaluating system effectiveness through user testing; he is a Professor in the Department of Computer Science at the University of Calgary.

CONTENTS

tion—the "whole situation, background or environment relevant to some happening or personality"—and then argue that this definition is too general to be useful in context-aware computing. After reviewing other definitions, they craft a new one that operationalizes the concept in terms of the actors and information sources involved in creating context:

> 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. (Dey et al., p. 106)

From this definition, Dey et al. present a context-aware computing framework. The premise behind the framework is that a set of toolkit components can be combined to determine a contextual state: The components capture, transform, and aggregate the raw information. If the designer can articulate the set of interesting contextual states ahead of time, then programming for context is simply a matter of determining which of these contextual states we are in and then having the system take an appropriate action.

Unfortunately, this definition glosses over one critical point. Context is a dynamic construct as viewed over a period of time, episodes of use, social interaction, internal goals, and local influences. Although some contextual situations are fairly stable, discernable, and predictable, there are many others that are not. The result is that similar looking contextual situations may actually differ dramatically. That is, in all but the simplest cases it may be difficult or impossible for a designer/programmer to list a priori:

- The set of contextual states that may exist.
- What information could accurately determine a contextual state within that set.
- What appropriate action should be taken from a particular state.

This has a severe consequence for context-aware computing, as it implies that the designer/programmer cannot always compose a correct "set of rules" (or program or formal model) for determining context and appropriate actions.

The theme of this essay pursues this idea of context as a dynamic construct. To develop this theme, I first review several theories that suggest the dynamic nature of context as a natural part of human–human–artifact interaction. I then discuss the implications of this in relation to Dey et al.'s context framework, and to context-aware computing in general.

2. THEORIES OF CONTEXT

Three reasonably well-known theories are highly relevant to context, for they describe what comprises context and how people work within these contexts. These are Situated Action (Suchman, 1987), Activity Theory (Nardi, 1997a) and the Locales Framework (Fitzpatrick, 1998; Fitzpatrick, Kaplan, & Mansfield, 1996; Fitzpatrick, Mansfield, & Kaplan, 1996). These theories help us understand the limits of context-aware computing, for all suggest that we must consider context as a continually evolving and highly situation-dependent construct.

2.1. Situated Action

Suchman's (1987) book, *Plans and Situated Action,* is a critique of plan-based approaches to cognition, which described activities as a set of well-defined goals and plans that people determine ahead of time. As an alternative, Suchman offered her theory of situated action, which takes the perspective that "every course of action depends in essential ways upon its material and social circumstances" (p. 50); that is, that human cognition and subsequent action is an emergent property of the moment-by-moment interaction of an individual with his or her physical and social environment. This theory does not imply that plans do not exist. Rather, and as nicely summarized by Dourish and Button (1998), "plans are one of a range of resources which guide the moment-by-moment organization of an activity, rather than laying out a sequence of work which is then blindly interpreted" (p. XX).

Suchman's situations are analogous to context; although unlike the definitions of context provided by Dey et al., she was explicit about its fluid and ever-changing nature. Nardi (1997b) highlighted this when she recapitulated that "the inquiry [of analysis of situated action] is meant to take place at a very fine-grained level of minutely observed activities, inextricably embedded in a particular situation" (p. XX). Nardi continued, "it forces the analyst to pay attention to the flux of the ongoing activity, to focus on the unfolding of real activity in a real setting" (p. XX). Lave (1988) also pointed to the dynamic nature of context as defined by situated action: "[It] emphasizes responsiveness to the environment and the improvisatory nature of human activity" (p. XX).

Situated action does not imply complete randomness and unpredictability. Suchman (1987, p. 179) explained that the goal of studies of situated action is to explicate the relation between structures of action and the resources and constraints afforded by physical and social circumstances. However, this is far from easy to do, and she cautioned against the institutionalized (and often simplistic) social norms that are often used to predict and prescribe all actions (e.g., Suchman, 1994). Nardi (1997b) summarized this as "a tension between an emphasis on that which is *emergent, contingent, improvisatory* and that which is *routine* and *predictable*" (p. 84).

2.2. Activity Theory

Activity theory (Nardi, 1997a) claims that activity defines context, where an activity comprises a *subject* (the person or group doing the activity), an *object* (the need or desire that motivates the activity), and *operations* (the way an activity is carried out).[1] Artifacts and environment are seen as entities that mediate activity. Nardi (1997b) then argued that activity-as-context includes not only external resources (people, artifacts, settings) but internal processes (objects and goals) as well.

Activity theory also recognizes the dynamic nature of context. As Nardi (1997b) emphasized, "activity theory holds that the constituents of activity are not fixed but can dynamically change as conditions change" (p. 75). Contributing to this change is the transformative relation between people and artifacts: This transformation can change the internal objectives and thus the course of the activity. Thus context cannot be inferred simply by enumerating the external set of people and artifacts (Nardi, 1997b), for it must also include people's internal (and perhaps changing) states. This sentiment is also echoed in Bellotti and Edward's essay (2001 [this special issue]): "There are human aspects of context that cannot be sensed or even inferred, so context-aware systems cannot be designed simply to act on our behalf."

1. It is this inclusion of *object* or intention that separates this theory from situated action (Nardi, 1997b, p. 89). Also, Activity Theory's use of the term *object* is quite different than how the same term is used in Dey et al.'s definition presented in Section 1: The former uses it as a synonym for intention, the later as a synonym for artifact.

2.3. The Locales Framework

The Locales Framework was developed as a principled approach to help people understand the nature of social activity and work, and how a locale (or place) can support these activities (Fitzpatrick, 1998; Fitzpatrick, Kaplan, et al., 1996; Fitzpatrick, Mansfield, et al., 1996). It is a descriptive theory based on Strauss's (1993) Social Worlds concept, which in turn defines groups of people who share some commitment and collective action. Fitzpatrick observed real-world situations as how people create social worlds and how they work together within them. A main outcome is how she saw *locales* arise as a social world appropriating and using particular sites and means for pursuing work. Locales are not necessarily fixed entities with fixed meanings. For example, two completely separate locales may be realized by a single physical meeting room space and how two different groups use it for the duration of their interaction. A locale does not even have to be associated with a physical space. Rather, it may arise in a virtual domain by the tools that are on hand for the group: e-mail, MUDs, instant messaging services, and so on. The important point is that locales arise from social worlds, and that locales are associated with the site and means for doing work.

Other relevant parts of the Locales Framework describe the subtleties of how locales are used. First are *individual views.* Because individuals can belong to many social worlds, a person may have one's own partial view into multiple locales at any time. As Fitzpatrick, Kaplan, et al. (1996) stated "her involvement, i.e., the current state of her interaction with and in the various worlds' locales, will determine which aspects of locales and indeed which locales are most relevant at any point in time" (p. XX). Second is *mutuality,* a term Fitzpatrick used to describe how people maintain a sense of shared place and that keeps them informed about shared activity. Mutuality includes one person's awareness of others, the artifacts comprising the locale, where things are located, and how things are changing within it. Third are *interaction trajectories.* This recognizes that interactions evolve over time and that considerable effort is spent managing this evolution. Interaction trajectories include a group's control over past, present, and future aspects of routine and nonroutine work; how people coordinate and negotiate plans and activities over time; how people leverage past experiences; how breakdowns are noticed and repaired; and how processes are supported.

In many ways, the Locales Framework is about the social construction and use of context, and it too recognizes its dynamic properties. Locales (the site and means) are the external contributors to context; although locales can be fixed, most are fluid. The choice and makeup of these sites and their means change with the moment-by-moment needs of individuals and the social world as a whole. That is, the dynamics affect what facets of the setting might be relevant to context. What is relevant at one time may be irrelevant at other times. Mutuality is the mechanism whereby people recognize both context and subtle contextual changes. Individual views acknowledge that people can be involved in many contexts at once and may selectively attend to them. Interaction trajectories highlight that context changes and evolves over time and that part of people's ability is to recognize and manage that change.

3. IMPLICATIONS TO PRACTITIONERS

The aforementioned theories describe how context is created from a variety of internal and external factors, how it changes moment by moment, and how people interpret it to perform actions. These theories make it quite clear that that designers must consider several nontrivial aspects when building systems that recognize and take advantage of context.

1. *Determining an appropriate set of canonical contextual states may be difficult or impossible.* It would be very convenient if a designer of a context-aware application could enumerate a priori a limited set of likely contexts and what comprises them. Programming the application then becomes a matter of determining which contextual state best matches the current real context and then taking appropriate action. Although there may be a variety of settings where such canonical contextual states can be articulated and applied, there are also many settings where this will not be possible simply because no canonical set exists. There is also a temporal aspect: A canonical set that may seem appropriate today may be completely inappropriate tomorrow because of internal and external changes in the social and physical circumstances.
2. *Determining what information is necessary to infer a contextual state may be difficult.* The aforementioned theories suggest that many things contribute to context and that the relevance of any bit of information is highly dependant on the particular situation. External things—the artifacts, the physical environment, the people—are relatively simple to capture (although mapping that information onto a context may be hard). In contrast, internal things—individuals' interests in that contextual setting, their history of interaction, their current objectives, and the state of the activity they are pursuing—are extremely difficult to capture. The problem is that there is likely no guaranteed way for a system to infer a correct contextual state by just collecting external information. At best, the system can only provide an approximation or educated guess of the real current context.
3. *Determining an appropriate action from a given context may be difficult.* The actions that people do, or the desired responses people expect from a

context-aware application, are also highly situation-dependent. The various theories suggest that even if two contextual situations appear almost identical, the desired action in one may differ substantially from the other simply because a different series of events may have led to those situations. People's internal states—their objectives and their social constructs—may differ as well. Consequently, there is a chance that the action performed by the system may be the wrong one.

With these design caveats in mind, let us return to Dey et al.'s framework of generic toolkit components that developers can use to build context-aware applications. Their basic idea is that

- *Context widgets* capture the raw information used to characterize an environment.
- *Interpreters* can transform the raw information captured by one or more context widgets into another more useful abstraction.
- *Aggregators* collect related information into a single context and offer it to the application.
- *Services* execute actions on behalf of the application.
- *Discoverers* maintain a registry of the aforementioned components

Their many examples then suggest how component architectures for context-aware applications are constructed: The context widgets and interpreters are used to collect information, the interpreters determine the contextual state, and the services take actions based on that state. The simplicity of this component architecture is very appealing to designers and programmers: They can use the conceptual framework to design the architecture, and the context-aware toolkit to implement the context-aware application.

Although these ideas are good, the design trap should now be obvious. The framework and toolkit is an elegant way to design and implement context-aware applications for simple and highly routine contextual situations. However, it does not include anything to inform the designer about what contextual situations are appropriate to it (i.e., whether useful canonical contextual states exist, whether information can be captured to infer that state, and whether resulting actions are meaningful).

One can, of course, argue that the purpose of the framework and toolkit is not to inform designers about context, but to speed actual development. Yet the very fact that building context-aware applications becomes easy also means that we will see many uninformed designers build inappropriate ones. We have already seen this happened in other computer application domains. Workflow systems, for example, began with a fairly naïve view of how procedures in an office setting could be captured and automated. A variety of systems were built, where the job of the workflow analyst became a routine matter of recognizing (what appeared to be) canonical workflow states and the activities that should happen from those states. Although this worked for certain types of settings and situations, workflow systems were applied to entirely inappropriate settings as well. People began to fight the system, for the system view of context (in this case the workflow context) did not fit with what was actually happening. It took some time for the community to recognize the problem, and even longer for commercial systems designers to accept the limitations of procedural workflow (Grinter, 2000). Fortunately, newer workflow systems do recognize this problem and are far more flexible in dealing with particular situations that do not fit the canonical workflow norm. Of course, workflow systems are not unique: We have seen similar problems with e-mail-like systems based on speech-act theory (Suchman, 1993).

The danger of building similar misguided context-aware applications is not just an academic cautionary note, for we can already find many instances of inaccurate context-aware systems in even very simple settings. For example, one can now buy off-the-shelf lights that turn on when they detect someone within a room, or outside security lights that turn on when any motion is detected, or toilets that flush when someone stands up, or automatic sink taps that turn on when they detects something in the sink. Other examples are listed in Bellotti and Edwards' essay (2001 [this special issue]). It would be humorous to recount the variety of times these systems get it wrong, but I am sure that the reader has experienced these and similar annoying situations themselves.

Our own experiences (by myself and my group) in building more sophisticated context-aware applications also echo how easy it is to get it wrong. For example, we built an always-on media space that tries to balance privacy and distraction concerns between distance-separated users of these spaces (Greenberg & Kuzuoka, 2000). The idea is fairly simple: What people see through the video channel is a reciprocal function of how far away people are from the displays. If both are close together, they see and hear each other in full fidelity. If one moves away, sound is turned off and image fidelity is reduced through blurring (Boyle, Edwards, & Greenberg, 2000). If both move away, the image and frame-rate are further reduced. That is, we wrote a state table of possible contexts, instrumented the environment to determine which context the actors were in, and had a programmed action that the system would take on the actors' behalf. Although the system worked very well in certain settings (e.g., to connect two distant offices whose occupants wished to work together), we found it inappropriate and inaccurate for other settings. For example, when one camera was situated in a home office, it was possible for the system to inadvertently capture and present in full fidelity the image of another home occupant (perhaps in a state of undress) as he or she walked by the open door. Also, the simple rules did not work as well: There were times when the home workers did not want others to see them in full fidelity, perhaps because they had just woken up and were working behind the computer in their pajamas. People's internal states also affected this desire: Sometimes a person just wanted some solitude.

Michael Boyle, one of our lab members, suggests that context-aware systems such as ours should be designed with the premise that there is a strong likelihood the system will get things wrong. Consequently, context-aware systems should be fairly conservative in the actions they take, should make these actions highly visible, and should leave "risky" actions to user control (Bellotti & Edwards, 2001 [this special issue], take a similar point of view in their essay). For example, consider how we are redesigning our reactive media space (Boyle, 2001). First, to minimize privacy risks, the system is now far more cautious about revealing video than masking it. When a full privacy condition is inferred, the system physically rotates a motorized camera away from the person to point to a blank wall. If this inference is incorrect, the parties are inconvenienced but not put at risk. Because the opposite action of inferring that a person wants to regain the full video connection is far riskier; we make this a manual action; the system does not rotate the camera back to face the person automatically. Second, the system presents noticeable feedback to the actor. We use camera rotation instead of a lens shutter to make the camera position highly visible, and we attach a small display to the camera so one can always see exactly what is passed on to others (i.e., whether full or blurred video is being transmitted). Third, the system presents simple mechanisms for people to adjust and/or override its context awareness properties; that is, a person can momentarily place a thumb over the camera lens can to block the video completely (Boyle et al., 2000).

The main point behind the earlier examples is that building inappropriate context-aware software and hardware is already happening. The challenge is to educate designers about the subtleties of context and to build software that can cater to the changing nature of context.

4. CONCLUSION

Context-aware computing is an important evolutionary step in computer use. Understanding context is vital if we are to build effective ubiquitous computing systems, information appliances, reactive environments, and computer devices that fit within architectural settings. However, we as designers must realize that it is all too easy to trivialize context, and as a result we will end up building inappropriate applications.

We are still coming to grips with what we mean by context, and it will take some time before we have good design principles for context-aware computing (see Bellotti & Edwards, 2001 [this special issue], for a starting point). For now, we present three high-level ideas about what is needed to get context right.

Getting context right means that we must study the expected context-of-use carefully. One approach is to use ethnographic methods to observe and analyze many contextual episodes, and to determine (using Nardi's, 1997b, words) what contexts are mostly emergent, contingent, and improvisatory and what are routine and predictable. Similarly, we must carefully understand the effects of getting context wrong: In some situations, a single inappropriate system action may be enough to preclude people from using it further. This implies that a contextual system should be fairly conservative in the act it takes: "Risky" actions should be taken only if there is compelling evidence of correctness.

Getting context right also means that our toolkits must incorporate flexibility. I hope that systems that infer context will get contextual guesses correct most of the time, but failure is inevitable. Because the information used to infer context might change, people should be able to adjust what information is collected and how it is interpreted so that context is inferred more accurately. Because expected responses to a context may change, people should be able to adjust the actions the system takes so that they are appropriate. Because some contexts cannot be inferred (perhaps because they appear only once), people should be able to override the system altogether.

Getting context right is also about good interface design. People should be able to see what context the system thinks it has inferred. Through feedback, actions taken by the system should be clearly linked to that context. Through lightweight interface mechanisms, people should be willing to adjust contextual information, override the system, or adjust the way actions are taken—if people consider any of the previous too hard to do and if consequences of wrong contextual guesses are annoying, they will likely just turn off the system altogether. Finally, observations of how the context-aware application performs in a real-world setting should inform its further design and redesign.

NOTES

Acknowledgments. Dey, Abowd, and Salber wrote a fine anchor paper that helped provoke thoughts about context in computing: This essay would not be possible without them. As well, the systems produced in my laboratory by Michael Boyle, Michael Rounding, and Hideaki Kuzuoka, and our group's discussions about the successes, compromises, and failures when using these systems, contributed significantly to the thoughts expressed in this essay. Revisions between the draft and the final version of this essay were influenced by discussions with Michael Boyle, as well as by comments made by Tom Moran and Paul Dourish.

Support. Partial funding of provided by the National Sciences and Engineering Research Council of Canada (NSERC), the Alberta Software Engineering Research Consortium (ASERC), and Microsoft Research.

Author's Present Address. Saul Greenberg, Department of Computer Science, University of Calgary, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4. E-mail: saul@cpsc.ucalgary.ca

HCI Editorial Record. First manuscript received January 14, 2001. Accepted by Tom Moran and Paul Dourish. Final manuscript received March 1, 2001. — *Editor*

REFERENCES

Bellotti, V., & Edwards, K. (2001). Intelligibility and accountability: Human considerations in context-aware systems. *Human-Computer Interaction, 16,* 193–212.

Boyle, M. (2001). *Ubiquitous awareness spaces* (Yellow Series Report 2001–682–05). Department of Computer Science, University of Calgary, Alberta, Canada.

Boyle, M., Edwards, C., & Greenberg, S. (2000). The effects of filtered video on awareness and privacy. *Proceedings of the CSCW'00 Conference on Computer Supported Cooperative Work.* ACM.

Dey, A. K., Abowd, G. D., & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction, 16,* 97–166. [this special issue]

Dourish, P., & Button, G. (1998). On "technomethodology": Foundational relationships between ethnomethodology and interactive system design. *Human–Computer Interaction, 13,* 395–432.

Fitzpatrick, G. (1998). *The locales framework: Understanding and designing for cooperative work.* Unpublished doctoral dissertation, Department of Computer Science and Electrical Engineering, The University of Queensland, Brisbane, Australia.

Fitzpatrick, G., Kaplan, S., & Mansfield, T. (1996). Physical spaces, virtual places and social worlds: A study of work in the virtual places for collaboration. *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work,* 334–343. ACM.

Fitzpatrick, G., Mansfield, T., & Kaplan, S. (1996). Locales framework: Exploring foundations for collaboration support. *Proceedings of the OzCHI'96 Sixth Australian Conference on Computer–Human Interaction*, 34–41. Hamilton, New Zealand.

Greenberg, S., & Kuzuoka, H. (2000). Using digital but physical surrogates to mediate awareness, communication and privacy in media spaces. *Personal Technologies, 4*(1).

Grinter, R. (2000). Workflow systems: Occasions for success and failure. *Computer Supported Cooperative Work, 9,* 189–214.

Lave, J. (1988). *Cognition in practice.* Cambridge, England: Cambridge University Press.

Nardi, B. (1997a). *Context and consciousness: Activity theory and human computer interaction.* Cambridge, MA: MIT Press.

Nardi, B. (1997b). Studying context: A comparison of activity theory, situated action models, and distributed cognition. In B. Nardi (Ed.), *Context and consciousness: Activity theory and human computer interaction.* Cambridge, MA: MIT Press.

Strauss, A. (1993). *Continual permutations of action*. New York: Aldine De Gruyter.

Suchman, L. (1987). *Plans and situated actions: The problem of human–machine communication*. Cambridge University Press.

Suchman, L. (1993). Do categories have politics? The language/action perspective reconsidered. *Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, 1–14. Kluwer.

Suchman, L. (1994) Speech acts and voices: Response to Winograd et al. *Computer Supported Cooperative Work, 3*(1), 85–95.