Pervasive Computing and the Future of CSCW Systems

A Position Paper for the CSCW2000 Workshop on Software Architectures for Cooperative Systems

David Garlan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
garlan@cs.cmu.edu

Abstract: The use of largely isolated workstations and laptops as the primary carrier of personal computing is giving way to a world of pervasive computing populated by ubiquitous networks of heterogeneous computing and information facilities. In such a world the boundaries between a user's personal computation space and other people's is no longer well defined. As a result, collaboration becomes the dominant mode of computing – in essence, all systems become the basis for on-going computer-supported cooperative work.

1. Introduction

Traditionally CSCW has been viewed as an occasional activity that we might engage in from time to time when we are not doing personal, private computing. This model fit well in the PC era: most of our computation centered on personal, isolated uses of applications that we bought, installed, and ran on dedicated (or virtually dedicated) processors. In that world collaborative work therefore represented a major shift in our day-to-day computational activity and seemed to require specialized facilities and capabilities.

The world is changing. In the future, most computation will not occur on personal computing platforms, but instead exploit a pervasive and ubiquitous computational and informational infrastructure. We will use resources on demand for communication, information retrieval, and computation. We will carry on on-going tasks that persist across physical and temporal boundaries. We will share information more freely, and transparently. We will be able to exploit myriad modes of communication between people.

In such a world, I believe that personal computing will become the exception: the primary mode of computer-supported activity will be cooperative. Thus, I would argue, pervasive computing and CSCW are intimately and inextricably connected: the former enables the latter, and propels it to center stage. Therefore to understand what are good architectures for future CSCW systems is to understand what are good architectures for pervasive computing.

2. Facets of Pervasive Computing

From an architectural perspective, there are a number of important facets of pervasive computing:

1. *Service-based computation:* Traditional computer use centers on PCs, using local applications and data that are installed and updated manually. Increasingly, the PC and other interfaces (handheld devices, laptops, phones) are used primarily as a user interface that provides access to remote data and computation services. This trend has a number of consequences for software architecture. First, it forces us to consider architectures for open systems. In a service-based environment applications may well be composed out of parts that are not under central control. Second, it forces us to consider architectures that scale to the size and variability of the Internet.
2. *Many heterogeneous devices:* A related trend is the emergence of a computing universe populated by a huge number and variety of heterogeneous computing devices: toasters, home heating systems, entertainment systems, smart cars, etc. There are a number of consequent challenges. First, we will need architectures that are suited to systems in which resource usage is a critical issue. Second, architectures for these systems will have to handle reconfiguration dynamically, while guaranteeing uninterrupted processing of the parts that don't change.
3. *Mobile computing:* Increasingly people expect to carry on their computing tasks in any environment. This trend implies a

need for architectures that will handle user mobility transparently. From an architectural perspective, the problem is largely one of finding ways to reconfigure high-level tasks to match local resources. This will require new architectures that can self-adjust dynamically to a changing computing base.

3. Architecture Research Themes

A systematic solution to these challenges will only come about through research along at least three dimensions.

1. *Correct software => Utility-based software:* Traditionally researchers have focused on correctness as the dominant issue. A software system that does not compute the right result is deemed inadequate. But in many cases less than optimal solutions are preferable if they come at lower costs—both in terms of system development and execution costs. Currently we are ill prepared to design and analyze systems whose overall utility is based on such tradeoffs. A key ingredient for improving the situation will be to have architecture-based analyses that make risks and costs explicit, and then allow one to perform tradeoff analysis within the space of alternatives. This in turn will require ways to document costs, to characterize "approximations to correctness," and to tune utility functions over these values.
2. *Applications => Service Coalitions:* In a world of mobile users, with pervasive access to information and services, we need to be able to describe and reason about systems that are dynamically composed of services, operating in a distributed environment. Research challenges include the ability to document these services and reason about their compatibility. This becomes a problem (both technically and sociologically) if the services are not under the control of any particular organization. The need to support mobility in this setting also introduces a new challenge: to make computations truly portable it is critical for the system as a whole to have a higher-level understanding of users' tasks. This will lead to a new notion of "task-driven computation," supported by run time infrastructure that can represent, instantiate, and adapt tasks as appropriate collections of services.
3. *Static Systems => Dynamic Systems:* Current research focuses on systems whose runtime configurations are largely fixed. In a world of dynamic service coalitions, mobile users, and changing resources, software systems will have to be much more adaptable at run time. This leads to a number of architectural challenges. The first is to be able to describe and reason about these systems. The second is to find architectural styles that are better suited to dynamism. The third is to develop components and integration schemes that support autonomy and self-adaptation. In a world where services are scattered across the net, it becomes critical that the components themselves take more responsibility for monitoring their own behavior and the behavior of their environment and adapting to changes without requiring user intervention.

Acknowledgements

This paper draws on material published in "Software Architecture: a Roadmap," published in *The Future of Software Engineering*, ACM Press, 2000. Research in this area is supported by DARPA under contracts F30602-97-2-0031 and N66001-99-2-8918.