Draft submitted for publication

Component-Based Software Engineering in Pervasive Computing Environments

David Garlan
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
(412) 268-5056
garlan@cs.cmu.edu

Bradley Schmerl
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
(412) 268-5889
schmerl@cs.cmu.edu

ABSTRACT
Being able to find, adapt, and incorporate disparate components to form working, reliable applications is the goal of component-based software engineering. To date, there has been a lot of research, among other things, on locating components, reasoning about component compatibility, and methods for interoperability. Pervasive computing raises a number of new challenges for component-based software engineering that heretofore have been given little attention, such as mobility, adaptability, and resource awareness. In this paper we motivate and discuss the need for research in these areas, and discuss how our work in the Aura group at Carnegie Mellon University helps to address these issues.

Keywords
Component-based software engineering, software architectures, pervasive computing.

1 MOTIVATION AND REQUIREMENTS

The world of software development and the contexts in which software is being used are changing in significant ways. One of the emerging trends that promises to have a major impact on software development is that of ubiquitous, or pervasive, computing. Pervasive computing comprises a computing universe populated by a rich variety of heterogeneous computing devices: toasters, home heating systems, entertainment systems, smart cars, etc. This trend is likely to lead to an explosion in the number of devices in our local environments – from dozens of devices to hundreds or thousands of devices. Moreover these devices are likely to be quite heterogeneous, requiring special considerations in terms of their physical resources and computing power.

Developing software for such an environment becomes quite challenging. Users will expect to interact with the environment to support them in every-day tasks, no matter where they are. Not only will they expect this, but if pervasive computing is to succeed and be accepted by the community at large, the instantiation of these tasks will need to be achieved with minimal support from a user – if the average person has difficulty performing a simple programming task such as recording a television program on their VCR, then it is unrealistic to expect them to be able to choose and integrate components for different computing devices, depending on their location and task at hand.

To be successful in a pervasive environment, components need to exhibit the following characteristics:

Mobility: As a user moves from one environment to another, they will expect their tasks to logically "follow" them around so that they are available when needed. Such mobility requires a refocusing of a software system from large, monolithic applications to collections of components cooperating to achieve a user's task Furthermore, a user will expect the computing environment to take advantage of resources in different environments. For example, in an office environment, a user may be able to take advantage of keyboard entry and high resolution display components; the task may take advantage of speech recognition and synthesis to continue a task while the user is leaving work; while driving home, the user may interact through a heads-up display and speech recognition components. Such mobility needs to be achieved seamlessly and transparently, with little or no intervention from the user.

Adaptability: In a ubiquitous environment, it is highly likely that the resources available to a user in an environment will change as resources and users move in and out of that environment. The tasks and software should adapt to take advantage of these new resources. For example, consider the scenario in which we have a presentation being given to a multi-national audience, some of whom have Portuguese as their native language. The task of listening to the presentation might involve displaying the slides to each

user's personal device. If, partway through the presentation, a user walks in with a wearable computer that contains English to Portuguese translation software, the task should take advantage of this to translate the slides to Portuguese and display them to the Portuguese listeners.

Resource awareness: In order to effectively achieve mobility and adaptability, a ubiquitous environment will need to make optimal use of the available resources to support user tasks. For this to happen, components will need not only to publish their interfaces and protocols for interaction, but also make known their resource requirements (such as required bandwidth, computing power, memory and battery consumption). For example, if the preferred input method for performing tasks in a car environment is speech, the environment used to compose the task will need to know whether the resources available in the car will provide optimal performance. The car may have the ability to execute the requisite component, but it may be too slow to be optimal. In this case, the environment should be able to decide to use voice recording and communication using a cell phone to a server to conduct the recognition, for example.

In addition to the environment being able to ascertain components' resource requirements, components should also be aware of the resources offered them, and adapt their performance accordingly. Components should therefore be able to offer *multi-fidelity* services. In the example above, the speech recognition component may be able to offer its services within a car environment with sufficient timeliness, but with a reduced vocabulary.

2 THE AURA APPROACH

The Aura group at Carnegie Mellon University is investigating new architectures for ubiquitous environments that support task execution by building the tasks from cooperating components.

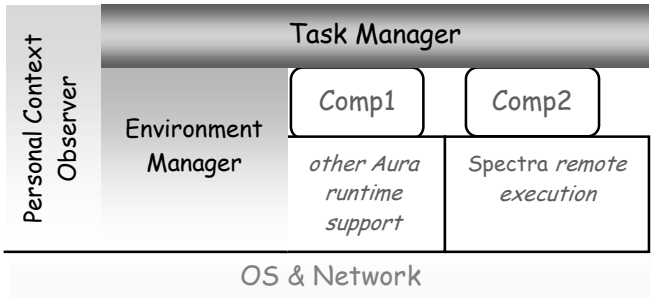

Figure 1. The Aura Architecture

Figure 1 depicts the Aura architecture for coordinating components. The *Personal Context Observe*r interprets the physical context of the user, and is responsible for identifying such things as the user's location, focus of attention, anticipated movements, etc. The *Task Manager* maintains a representation of tasks, and a mapping between context and user preferences. Finally, the *Environment Manager* has knowledge of the computational environment, and can discover, match, and assemble components to complete a task. It is also responsible for recognizing what resources are available and noticing when there is a resource change in the user's environment.

To illustrate how these parts work together, consider the following scenario:

Fred is in his office, frantically preparing for a meeting at which he will give a presentation and a software demonstration. The meeting room is a ten-minute walk across campus. It is time to leave, but Fred is not quite ready. He grabs his PalmXXII wireless handheld computer and walks out of the door. Aura transfers the state of his work from his desktop to his handheld, and allows him to make his final edits using voice commands during his walk. Aura infers where Fred is going from his calendar and the campus location tracking service. It downloads the presentation and the demonstration software to the projection computer, and warms up the projector.

The task that Fred is performing might be called "Prepare for a meeting." Inside this task is the task step called "Prepare my presentation."[1] A set of task primitives is combined to achieve this step. These task primitives might be: text input, document editing, and document viewing. The task primitives must be carried out by a set of services, a choice that Aura makes based on Fred's preferences and context. For example, the service that could fulfill the text input primitive might include keyboard entry or speech entry. When the Context Observer recognizes that Fred is in his office, the Task Manager notes that his preferences dictate keyboard entry. The Aura Environment Manager is then responsible for discovering a component that provides the service (such as a particular piece of presentation software) to supply this input. However, when Fred starts walking to the meeting (as noted by the Context Observer), the service that best fulfills his need for text input changes to speech entry (as noted by the Task manager). Aura once again faces a choice: recognize the speech entry on the palmtop, which has limited computing power and vocabulary, or transmit the audio to a remote server, which is more powerful, has a larger capacity, but may become unavailable when Fred is out of range. The Aura Environment Manager chooses the latter component, given that the meeting is on campus and the server is unlikely to become unavailable as Fred walks toward it.

The Aura architecture provides the infrastructure that allows components to be coordinated to support a user in a task, regardless of environment or locality.

[1] "Prepare my presentation" could itself be decomposed into other task steps, depending on the granularity required. For the purposes of this example, we consider it a task step.

3 RESEARCH CHALLENGES

Although conceptually simple, capturing and using user intent to configure tasks as sets of components will require a number of significant research advances. In particular, four fundamental areas must be investigated:

Task inference*:* Can user intent be inferred, or does it have to be explicitly provided? In the latter case, is it statically specified (from a file, for example), or obtained on demand through dynamic interactions?

The pervasive computing infrastructure must be able to strike a balance between user involvement in task specification and system inference of user intent. In Aura we plan to strike this balance by employing a mixed strategy that combines both observation of a user's activities, as well as explicit user specification. To minimize overhead in specifying tasks, we expect to provide a set of task templates that can either be filled in automatically by the system, or directly by the user.

Task representation*:* How is user intent represented internally? How rich must this information be for it to be useful? When and how is it updated? How does one characterize accuracy of knowledge in this area? Is incomplete or imprecise knowledge of user intent still useful?

Current task representation languages, such as those found in workflow management systems or robotics applications are inadequate for the purposes of Aura, since their primary purpose is to constrain the behavior of a user (or system) to predefined paths. In contrast, a task description in the context of Aura is used to enable the system to compose appropriate components in the environment in an optimal way, and to anticipate the needs of a person in the future. This suggests that task descriptions and their execution models should be stochastic in nature, and should allow the system to accommodate task steps that are not known a priori.

Resource allocation*:* How can a system instantiate tasks as collections of components to achieve optimal use of resources? How and when should that configuration change to accommodate changing resource pools?

As a user moves from one location to another, tasks must be reconstituted in ways that satisfy user needs but live within the resource constraints of an environment. This will require the development of algorithms for determining the utility of a given configuration of services taking into account user preferences, computational needs, and projections of future use. The problem is even more complex in the context of multiple users, when there are competing demands for the same resources and issues of privacy/security [2]. Our solution will build upon the network and operating system support discussed in [3]. We also expect to build upon [1, 4, 5] to support the dynamic reconstitution of components depending on resource availability, and the user's environment and preferences.

Architecture*:* What kinds of interactions should be supported between lower-level system capabilities (such as service location and discovery, operating system functions, and communication infrastructure) and the task layer?

System level functions should be able to query task management infrastructure to determine future computing needs, for example. At the same time, lower-level facilities must provide notification capabilities so that tasks can be readjusted as resource availability changes. Accommodating these kinds of interactions will require new systems research to determine appropriate interfaces and protocols for tying task representations effectively to lower level system infrastructure.

We believe these challenges will require substantial rethinking of component specification and implementation, as well as the supporting infrastructure for component location, composition, and adaptation.

ACKNOWLEDGEMENTS

DARPA, under Grants N66001-99-2-8918 and F30602-00-2-0616, supports this work. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA. We would like to thank the members of the Aura and ABLE groups.

REFERENCES

1. Allen, R., Douence, R., and Garlan, D. Specifying and analyzing dynamic software architectures. In *Proceedings of the 1998 Conference of Fundamental Approaches to Software Engineering (FASE98),* Lisbon, Portugal, March 1998

2. Garlan, D. Pervasive Computing and the Future of CSCW Systems. In *Proceedings of the Workshop on Architectures for Cooperative* Systems, Philadelphia, PA, December 2000.

3. Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, to appear.

4. Schmerl, B., and Marlin, C. Versioning and Consistency for Dynamically Composed Systems. In *Proceedings of the 7th International Workshop on Software Configuration Management*, Boston, MA, May 1997. Published as Lecture Notes in Computer Science, Vol. 1235, pp. 49-65, Springer, Berlin, 1997.

5. Wang, Z., and Garlan, D. Task-Driven Computing. *Carnegie Mellon University School of Computer Science Technical Report CMU-CS-00-154,* May, 2000.