IEEE distributed systems ONLINE
Expert-authored articles and resources

ds online exclusive

PRINTER FRIENDLY VERSION

June 2003

Decentralized Coordination of Distributed Interdependent Services

Asa MacWilliams, Thomas Reicher, and Bernd Bruegge • Institut für Informatik, Technische Universit ä t München

Current middleware technology for ubiquitous computing connects mobile computers to services in the environment. This is often insufficient because it doesn't address the issue of services depending on other services. The DWARF framework offers a decentralized approach to configuring and coordinating interdependent services.

Typical ubiquitous computing scenarios let mobile users access stationary services. This creates star-shaped dynamic connections between a user device–typically a PDA or a wearable computer–and smart devices in the environment. For this kind of configuration, current middleware can discover services and set up communication.

However, this approach is insufficient for highly mobile and collaborative applications involving several users with diverse mobile and stationary hardware. Services the environment offers to users might depend on services that another user's hardware offers. This poses a new challenge to the middleware: it must set up connections between interdependent ad hoc services.

We have used our research platform–DWARF (Distributed Wearable Augmented Reality Framework[1])–to build distributed mobile multimedia and augmented reality systems consisting of decentralized services. Augmented reality systems must determine the position and orientation of a user's head and other moving objects in real time and use this information to overlay 3D virtual objects onto real scenes.

Consider just a small part of a mobile augmented reality scenario.

Advertise with us

Subscribe to IEEE Computer Society publications

Alice, an architect, has a wearable computer with a see-through head-mounted display. She enters a colleague's architectural studio for the first time. When she approaches the design table, a 3D model of a recently designed building appears on the table, shown in her head-mounted display. It takes several distributed components to implement this scenario: A stationary tracking system within the room tracks an optical marker attached to Alice's head-mounted display, a wireless network transmits the building model and the position of Alice's head to her wearable computer, and the wearable computer renders the model in correct 3D registration.

This arrangement naturally leads to a system architecture with communicating services such as position trackers, filters, and renderers. Services on the client computer and those in the environment form an ad hoc augmented reality system.

Solution using DWARF

DWARF, a framework based on distributed services, contains services for position tracking, 3D rendering, multimodal input and output, and modeling of user tasks. These services support the building of complete augmented reality applications. Flexible applications that run in many different environments and hardware configurations allow users a great degree of freedom. Systems we've built so far[1-3] consist of between 10 and 50 services.

Distributed CORBA-based middleware manages the services. Each DWARF system network node has one service manager; there is no central component. The service manager controls the node's local services and maintains descriptions of them. The service managers cooperate with each other to set up connections between services.

Attributes describe the properties of services. They can also describe a user's context, thereby enabling the service manager to select and configure services according to this context.

Services, abilities, and needs

The basic elements of our framework are services with abilities and needs. A *service* is a piece of software running on a stationary or mobile computer that provides a certain functionality–for example, optical tracking.

Abilities describe the functionality a service provides, such as position data for optical markers. A service can have several abilities–say, an optical tracker that can track several markers simultaneously. Abilities are typed; an example would be PoseData for 3D position and orientation.

Needs describe the functionality required of other services. For example, an optical tracker needs a video sequence and descriptions of the markers it should find, and a 3D renderer needs the position

and orientation of the viewpoint it should render the scene from. Needs are also typed, and only abilities of the same type can satisfy a need.

Connectors describe the communication protocol–for example, CORBA notification service events, shared memory, or remote method calls–that provides the functionality. An example service is an optical tracker that receives a video sequence, detects fiducial markers within the image, and provides the markers' 3D position. This model lets us abstract from communication protocols.

Services for configuration

Services that are useful in different contexts need configuration information. We looked at several approaches and decided to store the configuration data in several separate services, such as a marker description storage service. Such storage provides abilities to the services needing to be configured. To distinguish different configurations, we use context attributes.

Attributes and predicates

DWARF uses an XML file format to describe services to the service manager. Figure 1 shows an example fragment.

```
<service name="OpticalTracker">
<attribute name="Room" value="Studio"/>
<attribute name="Lag" value="0.01"/>
<attribute name="Accuracy" value="0.001"/>
<need name="videoStream"
type="VideoStream">
  <connector protocol="RTSPReceive"/>
</need>
<need name="markerData" type="MarkerData"

predicate="((Thing=*)(User=*))"> <connector
protocol="ObjrefImport"/> </need>

<ability name="poseData" type="PoseData"
isTemplate="true">
<attribute name="Thing"
value="$(markerData.Thing)"> <attribute
name="User" value="$(markerData.User)">
<connector protocol="NotificationPush"/>
</ability>
</service>
```

Figure 1. An XML fragment describing an optical tracking service with two needs and one ability.

Abilities can have named attributes–for example, Thing=Head or Accuracy=0.001. Needs can specify predicates, such as (&(Lag<=0.3)(Accuracy<=0.01)). When matching abilities to needs, the service managers ensure that the ability's attributes satisfy the

need's predicate. The developer can specify attributes for the entire service; in this case, all the service's abilities inherit the attribute.

The attributes of a service's ability can change according to how its needs are satisfied. For example, if a description of the marker on Alice's head satisfies an optical tracker's MarkerDescription need, the tracker can determine the position and orientation of Alice's head. To support such dynamically changing abilities, a service developer can specify that an ability's attributes depend on the attributes of a specified need's communication partner, as shown in the Thing attribute in Figure 2.

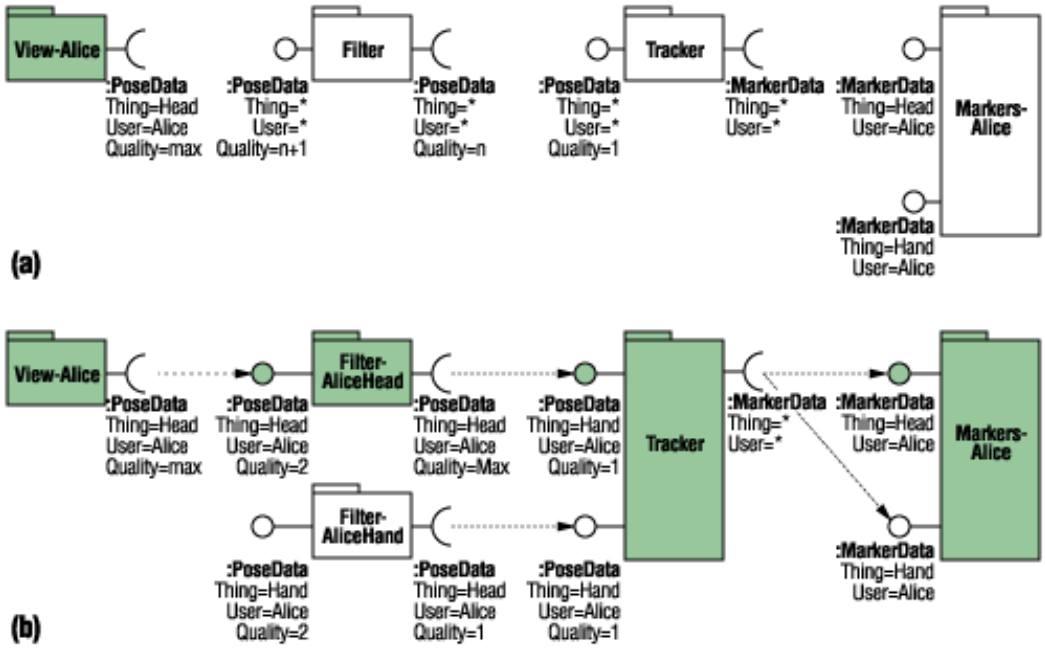

Figure 2. How a chain of services is formed; semicircles denote needs, and circles denote abilities. (a) Unconnected services; (b) a chain of services. Running services are green, potential services are white. In the singleton tracker service, the service manager creates a new ability for each marker. Additionally, it creates a new instance of the filter service when it is needed.

We use a set of well-defined context attributes describing several aspects of the user's context, such as User=Alice, Room=Studio, Application=Architecture, and Role=Architect.

next »

DS Online ISSN: 1541-4922 • Feedback? Send comments to dsonline@computer.org.
This site and all contents (unless otherwise noted) are Copyright ©2003, Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

IEEE distributed systems ONLINE
Expert-authored articles and resources

Communication setup and service instantiation

A *singleton* service exists only once for a given service description; a *template* service has multiple instances. When a service manager finds possible communication partners for a service's need, it creates new instances of template abilities and template services. For example, the optical tracker service gets a new instance of its PoseData ability for each partner of its MarkerData need. The service manager instantiates a new filter service for each tracked marker. The attributes of the newly instantiated ability or service are bound to those of the need's communication partner. Thus, if a tracker finds a MarkerDescription with Thing=Head, it will get an additional ability with Thing=Head.

These new instances remain as descriptions within the service manager and do not start until another service or the user needs them. The service managers use these descriptions of potential service instances to create possible networks of interconnected services. When the user actually selects a service, the service starts and in turn connects to other services.

Chains of services formation

The service manager assigns attributes to mobile services according to the user's context. For example, all services on Alice's wearable computer have the attribute User=Alice. As Alice enters the room, her marker description service becomes available to the tracker in the room. This leads to the creation of appropriate tracking abilities and filter services. The viewer on Alice's wearable computer then connects to the filter service and receives the correct position information. Figure 2 shows a snapshot of a dynamically created chain of services.

This service selection process is entirely automatic and based on user context. Where user service selection is needed, we offer an additional selector service that provides a list of available services, displayable on a PDA.

Related work

Several current approaches and technologies for ubiquitous computing systems share the idea of a space providing services for the user; they include PIMA,[4] Gaia,[5] Oxygen,[6] and Aura.[7] (For more extensive information, see http://devius.cs.uiuc.edu/gaia/html/links.htm.) In contrast to our decentralized approach, most existing systems have a star-shaped architecture with central components. Some approaches include service federation, but we are not aware of any concepts for context and configuration propagation among the services.

The approaches are quite different in detail, but each addresses the problems of component control, lookup, and selection.

Component control

To set up a net of cooperating services at runtime, we must be able to obtain information about them and control them. Component- and service-based systems such as the CORBA Component Model, Microsoft's Component Object Model, and Sun's Enterprise JavaBeans meet these requirements. The runtime environment controls the components and can start/stop and modify them on demand. Web services extend this capability to Web-based access methods. Research projects such as Oxygen, Aura, and Cooltown[8] provide their own elaborate component models as part of their research.

Component lookup and selection

In dynamic systems it isn't enough to have a naming service, such as the CORBA Naming Service, that translates identifiers to references. Binding a component at runtime to a formerly unknown component requires an abstract description, and a lookup service must search for an implementation. Examples are CORBA, with the CORBA Trader Service; Sun's Jini lookup service; Microsoft's UPnP; Sun's JXTA; and the Internet Engineering Task Force's Service Location Protocol (SLP). Most systems use simple attribute value matching, but some efforts seek to reuse well-known techniques from knowledge management research to transport more semantic information.[9,10] In Oxygen, Pebbles find each other through the Goals lookup service.[11,12] Gaia provides a Space Repository component that stores information about available resources in the user's space. DWARF reuses existing lookup services (currently SLP) for preselection but lets the user perform the final selection.

In a sense, a service offers a contract: if its needs are satisfied, it offers its abilities. Thus, the design of cooperating services is a variant of the design-by-contract paradigm.[13]

Future work

Our first step will be to move the configuration data into separate services. Our middleware can already set up the appropriate connections between services and their configuration services. Second, we will create different sets of configuration data for different users and applications. Finally, using the approach described in this article, we will extend DWARF's middleware to dynamically change services' configurations. In this way, we can build mobile augmented reality systems that adapt to a user's context with unprecedented flexibility.

Acknowledgments

This work was partially supported by the High-Tech Offensive of the Bayerische Staatskanzlei and the compound project Forsoft II of the Bayerische Forschungsstiftung. We thank all members of the DWARF team, especially our students, who built real working systems with our framework.

References

1. M. Bauer et al., "Design of a Component-based Augmented Reality Framework," *Proc. IEEE and ACM Int'l Symp. Augmented Reality* (ISAR 01), IEEE CS Press, pp. 45 - 54.

2. C. Sandor et al., "SHEEP: The Shared Environment Entertainment Pasture," *Proc. IEEE and ACM Int'l Symp. Mixed and Augmented Reality* (ISMAR 02), IEEE CS Press.

3. G. Klinker et al., "Fata Morgana–A Presentation System for Product Design," *Proc. IEEE and ACM Int'l Symp. Augmented and Mixed Reality* (ISMAR 02), IEEE CS Press, pp. 76 - 85.

4. G. Banavar et al., "Challenges: An Application Model for Pervasive Computing," *Proc. 6th Ann. ACM/IEEE Int'l Conf. Mobile Computing and Networking* (MobiCom 00), ACM Press, New York, pp. 266 - 274.

5. M. Román et al., "Gaia: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, Oct.-Dec. 2002, pp. 74 - 83.

6. "MIT Project Oxygen: Overview," MIT Laboratory for Computer Science, 2003, http://oxygen.lcs.mit.edu/Overview.html.

7. D. Garlan et al., "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, special issue on integrated computing environments, vol. 1, no. 2, pp. 22 - 31.

8. T. Kindberg et al., *People, Places, Things: Web Presence for the Real World*, tech. report, Hewlett-Packard Laboratories, 2003.

9. A. Arkin et al., *Web Service Choreography Interface (WSCI) 1.0*, tech. report W3C Note 8, World Wide Web Consortium, 2002.

10. J. Hendler, T. Berners-Lee, and E. Miller, "Integrating Applications on the Semantic Web," *J. Institute of Electrical Engineers of Japan*, vol. 122, no. 10, 2002, pp. 676 - 680.

11. S. Ward, C. Terman, and U. Saif, *Goal-Oriented System Semantics*, tech. report, MIT Laboratory for Computer Science, Mar. 2002.

12. S. Ward, C. Terman, and U. Saif, *Pebbles: A Software Component System*, tech. report, MIT Laboratory for Computer Science, Mar. 2002.

13. B. Meyer, *Object-Oriented Software Construction*, 2nd ed., Prentice Hall, 1997.

Asa MacWilliams is a PhD candidate at the Institut für Informatik of the Technische Universität München, Germany. His research interests include middleware and architectures for dynamic systems and a continual development process for ubiquitous computing. He is a member of the German Gesellschaft für Informatik. Contact him at macwilli@in.tum.de.

Thomas Reicher is a PhD candidate at the Institut für Informatik of the Technische Universität

München, Germany. He is doing research on the design and development of dynamic service-based systems. His current focus is on architectural patterns for augmented reality and ubiquitous computing systems. Contact him at reicher@in.tum.de.

Bernd Bruegge is university professor of computer science with a chair for Applied Software Engineering at the Technische Universit ä t München and adjunct professor at Carnegie Mellon University. His research interests include software architectures for dynamic systems, agile software development processes, and software engineering education. He received a PhD in computer science from Carnegie Mellon University. Contact him at bruegge@in.tum.de.

DS Online ISSN: 1541-4922 • Feedback? Send comments to dsonline@computer.org.
This site and all contents (unless otherwise noted) are Copyright ©2003, Institute of Electrical and Electronics Engineers, Inc. All rights reserved.