Using Ad-Hoc Services for Mobile Augmented Reality Systems

Asa MacWilliams
Summary

- Concepts and technologies for self-assembling mobile AR systems
- Design of the Middleware for DWARF
- First implementation of the Middleware, validating the design
Outline

- Self-Assembling AR Systems
- Requirements
- System Design
- Results
- Future Work
Context: a user is roaming through an intelligent environment with a mobile AR system.

Goal: his mobile system should automatically take advantage of devices in the environment, such as external trackers.
Self-Assembling AR Systems

- Context: a user is roaming through an intelligent environment with a mobile AR system.
- Goal: his mobile system should automatically take advantage of devices in the environment, such as external trackers.
- Idea:
  - Divide the system into different *Services* for tracking, display, etc.
  - Deploy the Services on different mobile and stationary computers
  - As they come within range of one another, the Services assemble into a complete AR system
- This requires intelligent *Middleware*. 
Functional Requirements

- DWARF consists of self-assembling Services.
Requirements (1)

- Functional Requirements

- DWARF consists of self-assembling Services.
- These Services must be able to find each other...
Requirements (1)

- **Functional Requirements**

- DWARF consists of self-assembling Services.
- These Services must be able to *find* each other...
- ...and *communicate* with one another.
Nonfunctional Requirements

- For convincing AR, we need fast communication with low latency.
- To use ad-hoc Services as they are found, however, we need to choose the communication partners flexibly.
- This is a conflict in design goals.
- The design of the middleware must balance flexibility against speed.
Requirements (3)

Mediator Role of the Middleware

- The DWARF Services are designed as independently of one another as possible, so they can be combined into different applications.
- For them to cooperate, we used the Mediator pattern.
Distributed Mediating Agents

- If the middleware becomes a *central component*, it reduces fault tolerance and flexibility.
- Instead, I used *Distributed Mediating Agents*. 
Subsystem Decomposition

- Communication is fast
- Location is flexible
- Service Manager provides high-level interface
Subsystem Interaction

- Everything off

Diagram:
- TrackingBox: Linux Notebook
  - Tracker
  - Communication
  - Service Manager
  - Location

- DisplayBox: Windows Notebook
  - Communication
  - Display
  - Service Manager
  - Location
Subsystem Interaction

- Everything off
- Display starts
System Design (3)

**Subsystem Interaction**

- Everything off
- Display starts
- Location

![Diagram of system design with boxes for TrackingBox:LinuxNotebook and DisplayBox:WindowsNotebook showing interactions between Tracker, Communication, Service Manager, and Location.]
**Subsystem Interaction**

- Everything off
- Display starts
- Location

![Diagram showing the interaction between TrackingBox:LinuxNotebook and DisplayBox:WindowsNotebook]
Subsystem Interaction

- Everything off
- Display starts
- Location

- Establish communication
- Activate Tracker

Diagram:
- TrackingBox: Linux Notebook
  - Tracker
  - Communication
  - Service Manager
  - Location

- DisplayBox: Windows Notebook
  - Communication
  - Display
  - Service Manager
  - Location
System Design (3)

Subsystem Interaction

- Everything off
- Display starts
- Location
- Establish communication
- Activate Tracker
- Communicate

Diagram:

```
TrackingBox:LinuxNotebook
  Tracker
  Communication
  Service Manager
  Location

DisplayBox:WindowsNotebook
  Communication
  Display
  Service Manager
  Location
```
**Service Manager: Service Descriptions**

- Each Service has a description that can be written in XML
- These describe Services’ Needs and Abilities
- They also specify communication protocols
Location Subsystem

- Various mechanisms are available to locate ad-hoc services: SLP, Jini, UPnP, SDP, etc.
- None address AR, many are for home networking
- The Location Subsystem defines a strategy pattern to use these different protocols
- The current version is designed to use the Service Location Protocol (SLP), a simple and open standard


**Communication Subsystem**

- The DWARF components have different communication needs.
- The Communication Subsystem can encapsulate many different protocols.
- It currently supports:
  - **CORBA** remote method calls
  - **Event-based communication with the CORBA Notification Service**
Summary—Design Challenges

The Middleware needs to be fast, yet flexible
- Decomposition into Communication and Location subsystems

The Middleware should not have to be in the middle
- Distributed Mediating Agents

Services that do not know each other have to cooperate
- Service Descriptions with Needs and Abilities
Complete Design for a flexible, yet fast middleware system
Supports roaming users in intelligent environments
Systems built with DWARF can *spontaneously self-assemble*
First implementation was successfully demonstrated on Linux and Windows, Intel and PowerPC
Future Work

- Further implementation:
  - Full SLP support
  - Full XML support
  - Optimizing of communication resources
  - Graceful handling of network errors

- Visualization tools

- Testing with different types of Services in different application domains
Thank You

Questions?