Decentralized Coordination of Distributed Interdependent Services

Thomas Reicher, Asa MacWilliams, Bermd Bruegge
Chair for Applied Software Engineering
Institut für Informatik
Technische Universität München
(reicher,macwilli,bruegge)@in.tum.de

June 19th 2003
The DWARF Framework

• Mobile AR in ubiquitous computing environments
• Already built AR supported scenarios:
  – Navigation (Pathfinder)
  – Maintenance (TRAMP)
  – Multi-Player Game (SHEEP)
  – Collaborative Building Design (ARCHIE)
Problems

• Goal: Seamless integration of local and remote components
  – DWARF uses a distributed approach
  – No separation between local and remote component on architectural layer

• Components offer own services and need other services
  – > service interdependency
  – Applications are not simply star-shaped but build a service graph of interdependent services

• Services are distributed on several deployment units
  – > no centralized coordination and configuration possible

• Usually 10 to 50 services per application
Example

- A View component needs position and orientation data
- A Tracker can provide orientation data in a particular format
- A Filter component must translate between Tracker and View
- The Tracker needs feature information for image processing
Approach

• Service model for interdependent services
  – Service have Attributes and Predicates.
  – They can be variables which are set at runtime.
• Runtime Infrastructure establishes connection between service automatically (management, lookup, connection)
Example Service: Optical Tracker

```xml
<service name="OpticalTracker">
    <attribute name="Room" value="Studio"/>
    <attribute name="Lag" value="0.01"/>
    <attribute name="Accuracy" value="0.001"/>
    <need name="markerData" type="MarkerData"
        predicate="(&(Thing=*)(User=*))">
        <connector protocol="ObjrefImport"/>
    </need>
    <need name="videoStream" type="VideoStream">
        <connector protocol="RTSPReceive"/>
    </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>
```
Service Chains

• Service instantiation
  – Singleton Services exist only once
  – Template Services have multiple instances and can be started on demand by the runtime environment

• Formation of chains of services
  – Services are connected automatically based on context and service-specific attributes
  – Attribute values are handed over from Abilities to Needs

• Services for configuration
  – Service are configured through Configuration Services
  – Selection of the correct one over context attributes
  – Selector Service for user defined connection
Setup of a Service Chain
Conclusion

• Service model used for several AR applications
• Configuration Service and Selector Service are being tested
• Hops of Attribute values from Abilities to Needs work. Particularly needed for selection of correct Configuration Service instance
• There are use cases for the opposite way, from Need to Ability
• Simulations and tests needed to find best set of context attributes for clear service selection