Ubiquitous Computing Middleware: 
Application Sessions for Dynamic Environments 

Christine Julien 
Director, Mobile and Pervasive Computing Group 
EDGE Distinguished Lecture Series 
April 13, 2006
Outline

- Motivation and Challenges
- An Application-Driven Approach
  - Defining a unique set of application characteristics
- A Pervasive Application Virtual Machine
- Introducing Middleware and Coordination
- The Application Sessions Middleware
  - Formalizing behaviors
  - Programming interactions
- Mobile and Pervasive Computing Group
Motivation

- Simplify application development for complex environments
  - Allow application developers to program directly to applications’ needs
  - Dynamically gather context information
  - Seamlessly adapt to environmental changes
  - Transparently help programmers cope with problems inherent in ubiquitous networks
Challenges in Mobile Ubiquitous Computing

- Transient connectivity to resources
  - Demands opportunistic interactions and limited *a priori* knowledge

- Rapid change due to user and environment dynamics
  - Demands awareness and adaptation

- Lack of guaranteed connectivity to fixed infrastructure
  - Demands local, on-demand communication
An Application-Driven Approach

- Sample Application Domains
  - First Responder Applications
  - Intelligent Construction Sites

- First Responder Applications
  - Surveillance
  - Paramedic
  - Injured
  - Sensors

- Intelligent Construction Sites
  - On-site interactions: safety, maintenance, and alerting
  - Opportunistic mobile data collection
  - On-the-fly integration
  - Local sensor queries
Unique Set of Application Characteristics

- Heterogeneous devices
- Unpredictable and dynamic connectivity and availability
- Data-driven resource requests
- Large scale and distribution
- Increased need for application coordination
- Locality of information and interactions
- Dependence on environmental factors
A Changing Set of Assumptions

- These new characteristics nullify common assumptions
  - No addresses for use in communication
  - Long-lived static overlays not feasible
  - Little control over *hardware* deployment and maintenance

- Top-down approaches address these failings from application abstractions through communication mechanisms
A Pervasive Application Virtual Machine

- Middleware for network and device transparency
  - Hide complexity of underlying communication
  - Actively cooperate with application at runtime to provide dynamic functions

- Algorithms for automatic and adaptive context collection
  - Sense and react to a changing environment

- Protocols for application-sensitive communication
  - Provide local and dynamic connectivity
Why Middleware?

- Speed up application development in complex environments
  - Enhance level of programming abstraction
  - Provide generic and flexible context operations
  - Shift complex and non-application specific tasks to underlying infrastructure
  - Facilitate rapid dissemination and adoption

- Proven solutions for other domains
  - Distributed object systems
    - CORBA, Jini, DENO
  - Mobile agent systems
    - Aglets, D’Agents, μCode
  - Coordination systems
    - LIME, MARS, Jedi
Why *Coordination*?

- Separation of communication from computation into a coordination *language*
  - Enables reasoning about interactions separately from other behavior
  - Promotes reuse, modularity, rapid prototyping
  - Most importantly, eases program understanding
The Application Sessions Approach

- Ubiquitous applications/users need to coordinate with changing set of resources
- Middleware handles these interactions
  - without exposing their complexity
  - without reducing their expressiveness
- Developer focuses on interaction instead of the *implementation* of that interaction
The Application Sessions Approach (2)

- Fundamental Questions:
  - What is the correct programming abstraction for ubiquitous computing?
    - Object oriented programming?
    - Aspect oriented programming?
    - Service/resource oriented programming?
  - What is the most important thing with respect to ubiquitous programming?
    - Interactions

- Session-oriented programming
The Application Sessions Contribution

- **Given session-oriented abstraction:**
  - Identify, formalize, and codify the interactions required by ubiquitous applications
  - Encapsulate similarities and parameterize differences

- Result ➔ a new *programming paradigm* for mobile ubiquitous computing
Application Session: A Definition

- Temporary logical connection among two or more networked devices
  - Over which application data is exchanged
- State maintained for the session involves an application-level dialog
  - Includes the set of operations the application intends to perform
- Functionality provided by session definitions closer to application requirements than other sessions
  - Not just for end-to-end connectivity
A Model

- Explicit separation of user program (behavior) from session management (interactions)
  - Coordination through shared variables
  - Interactions are defined by the coordination language

- Session is:
  - Defined by \texttt{spec}
  - Accessed through \texttt{s}
Precise Behavioral Specifications

- Different session types provide different coordination semantics
- Each session is defined (partially) by a specification
  - Content-based restriction over resource description
  - Implemented using tuples and patterns
  - Other description languages possible
- Applications code interactions as session declarations

\[
\begin{align*}
\text{spec} &= \text{specification} \\
&[\text{request session}] \\
&\langle \text{await } s \neq \bot \rangle \\
&\text{if } s \neq \varepsilon \text{ then} \\
&\quad \text{[use } s\text{]} \\
&\text{fi}
\end{align*}
\]
The Query Session

- The simplest session: retrieve a data value
  - Very short-lived interaction, but fundamental to many applications
- In programmatic notation, just \( s = \text{spec} \)
- Formally,
  \[
  s = s'.(s' \models \text{spec} \land s'.\text{connected})
  \]
  - Read as: “\( s' \) satisfies \( \text{spec} \)”
  - “\( \text{connected} \)” models communication capabilities
  - Non-deterministic selection says: “any match is as good as any other match”
The Query Session: Some Subtleties

- The $s$ the application can use is just an *instantaneous* view of a matching resource
  - The resource *was* connected at the instant the await statement unblocked
  - No further guarantees regarding the selected resource

```plaintext
spec = specification
s = spec
〈await s ≠ ⊥〉
if s ≠ ε then
  [use s]
fi
```
The Provider Session

- Find and connect to a matching resource
- Maintain the connection as long as possible at all costs
- Programmatically:
  \[ s \leftarrow \text{spec} \]
- Formally:
  \[ s = s'.(s' \models \text{spec} \land s'.\text{connected}) \]
  \[ \text{if } s \neq \varepsilon \text{ then} \]
  \[ \langle \text{await } \neg s.\text{connected} \rightarrow s = \varepsilon \rangle \]
  \[ \text{fi} \]
Using a Provider Session

- After assigning the specification to the handle and receiving a resource
  - Must periodically verify that the resource is still connected
  - Formalized as a loop; implemented as a callback the session user must handle

```
Provider session declaration

spec = specification
s ← spec
⟨await s ≠ ⊥⟩
while s ≠ ε do
  [use s]
  od

Periodic handle verification
```
The Type Session

- Find any matching resource and stay connected to any matching resource
- Programatically:
  \[ s \leftrightarrow \text{spec} \]
- Formally:
  \[ s = s'.(s' \models \text{spec} \land s'.\text{connected}) \]
  \[ \text{while } s \neq \varepsilon \text{ do} \]
  \[ \langle \text{await } \neg s.\text{connected} \rightarrow s = s'.(s' \models \text{spec} \land s'.\text{connected}) \rangle \]
  \[ \text{od} \]
Using a Type Session

- Similar to provider session
  - s is set to $\varepsilon$ only if there does not exist another matching resource

- Maintenance of state across sessions
  - Future work

```
spec = specification
s <-> spec
⟨await s ≠ ⊥⟩
while s ≠ ε do
  [use s]
od
```
The Group Session

- Find and connect to *all* matching resources
  - Under dynamics, automatically update the group
  - Interact with \(s\) using set operations

- Programmatically:
  \[
  s \leftarrow \{\} \quad \text{spec}
  \]

- Formally:
  \[
  s = \langle \text{set } s' : s' \models \text{spec} \land s'.\text{connected} :: s' \rangle \\
  \text{while } s \neq \emptyset \text{ do} \\
  \quad \langle \text{await } \text{group-change} \rightarrow s = \langle \text{set } s' : s' \models \text{spec} \land s'.\text{connected} :: s' \rangle \rangle \\
  \text{od}
  \]
Multiple Matches and Preferences

- Is non-deterministic matching sufficient?
  - Resource preferences
    - lower relative mobility
    - closer
    - lower error rates
    - higher remaining battery power
    - etc.

- Specified through a preference function $f(R)$
  - Formalization: evaluated as part of the matching algorithm
  - Implementation: separated from specification and (at times) used to parameterize communication
Multiple Matches and Preferences (2)

- An example: type session
  
  \[
  \begin{align*}
  s &= \langle \text{max } s' : \text{spec } \land s'.\text{connected} \equiv f(s') \rangle \\
  \text{while } o \neq \varepsilon \text{ do} & \\
  \langle \text{await } \neg s.\text{connected} \lor \\
  \langle \exists s' : s'.\text{connected} \land s' \equiv \text{spec } \land f(s') > f(s) \rangle \rightarrow \\
  s &= \langle \text{max } s' : \text{spec } \land s'.\text{connected} \equiv f(s') \rangle \\
  \text{od}
  \end{align*}
  \]

- Different types of preference:
  
  - \( f_n(r,a) \): network preference—defines the cost of a network path (static definition, dynamic values)
  
  - \( f_r(a) \): resource preference—defines the cost to a resource of servicing a request (resource load, battery power, intended length of usage, etc.)
  
  - \( f_a(r) \): application preference—defines the cost to an application of using a particular resource (latency of response, resource fidelity, proximity, availability, etc.)
Application Sessions: A Middleware

- Middleware architecture
  - Requests arrive via underlying communication protocols
  - Session factory determines match in comparison to local repository
  - Session implementations handle maintenance (and, if necessary) tear-down of sessions
Handling Dynamics: Aspect-Oriented Techniques

- Goal: hide resource discovery, migration, persistence, etc. behind single session interface
  - AOP techniques allow us to dynamically “plug in” functionality best suited to the application and environment
    - Dynamic object construction and run-time bindings
  - Allows application to perceive the session as a single (persistent) “object”
    - Implementation of method calls on that object may change over time and space
An Application Example

- **Group session**
  - **Resource:** trajectories of all people within 100 m
  - **Interaction:** periodic position updates displayed on the screen

```java
// provide no preference
Rater peopleR = null;
// match all people
Filter peopleF = new Filter("people");
// find anyone within 100m
Region peopleRegion = new Region(100);
IPeopleService peopleService =
    factory.createGroupSession(peopleF, peopleR, region);
```
Application Sessions: Conclusion

- A new programming paradigm targeted for ubiquitous computing
  - *Session-oriented programming*

- Provides formal specifications of target behavior
  - Includes ability to formally define preferences of network, resource, *and* application

- Realized in a middleware that simplifies application development
Application Session Acknowledgements

- Joint work with Drew Stovall
- Appearing in:
  - *Proceedings of the 8th International Conference on Coordination Models and Languages* (Bologna, Italy, June 2006)
Mobile and Pervasive Computing Group

- Middleware abstractions for simplifying development
- Mobile communication and coordination
- Adaptation and context-awareness
- Collaborative ad hoc groups
- Incorporation of RFID and sensors in ubiquitous environments
SMASH: Secure Mobile Agent System

- Facilitate secure mobile agents
  - Movement of autonomous software components
- Enable a combination of trusted and anonymous interactions
  - Careful balance of admission control and an open system
CDR: Application-Aware Communication

- Address-less discovery and routing combined
- Integrated reactive and proactive routing
  - Dynamic definition of optimal advertisement radius

\[ R_{res} = \rho \times \frac{r}{s} - \mu \times m + \pi \times p \]
DAIS: Middleware for Immersive Sensor Networks

- Ubiquitous environments entailing heterogeneous devices
  - Allow direct data queries of local sensors
    - Resource constraints
  - Heterogeneous abstraction and aggregation using virtual sensors
  - Streamlined programming model that hides TinyOS
Questions?

http://mpc.ece.utexas.edu
Persistent (Active) Sessions

- The polling function may not provide the best semantics for the application
- Persistent sessions make null s values legal over time
  - Require applications to be careful about handling null pointers
- Active type session:
  
  \[ s = s'.(s' \models \text{spec} \land s'.\text{connected}) \]

  \[
  \text{while} \ \neg \text{stop} \ \text{do} \\
  \quad \langle \text{await} \ o = \varepsilon \lor \neg s.\text{connected} \rightarrow s = s'.(s' \models \text{spec} \land s'.\text{connected}) \rangle \\
  \text{od}
  \]