Overview

- Context-Aware Computing
- Motivation
- Mobile UNITY Overview
- Modeling Context
- Conclusions
Context-Aware Computing

- In mobile environments, both software and hardware components constantly move and change
- Continuous and rapid adaptation to change
  - Scope that extends beyond the local host
  - Generalized interaction with varied context types
- Context needs vary by task and situation
  - Application specific context definition
  - Asymmetric context interaction
  - Multiple contexts evolving over time
Motivation

- No formal model exists for directly reasoning about context-awareness
  - Focus on mobile nature of context-aware systems
  - Facilitate understanding of program behaviors that are controlled by changes in context

*Build formal model that captures the essential features of context-awareness*
Mobile UNITY Overview

- Mobile UNITY programs are units of mobility
  - Movement reduced to a value assignment
  - Reasoning about motion through standard logic
- Mobile UNITY provides constructs for modeling mobility
  - *inhibit statement*--allows programs to strengthen guards of normal statements
  - *reactive statement*--allows programs code to execute in response to specified conditions
  - *transaction*--provides large-grained atomic state change
- Coordination among components is specified in a special *Interactions* section
Mobile UNITY Overview

Program `Loader(i)` at \( \square = 0 \)
declare y : integer  
initially y = 0  
assign
  load :: y := y’.(y’>0) if y = 0  
end

Program `Unloader(j)` at \( \square = N \)
declare z : integer  
initially z = 0  
assign
  unload :: z := 0 if z ≠ 0  
end
Mobile UNITY Overview

Program Loader(i) at 
  declare y : integer
  initially y = 0
  assign
    load :: y := y'.(y'>0) if y = 0
  end

Program Unloader(j) at 
  declare z : integer
  initially z = 0
  assign
    unload :: z := 0 if z ≠ 0
  end

Program Cart(i) at 
  declare x : integer
  initially x = 0
  assign
    go_right :: l := l + 1
    go_left :: l := l - 1
    inhibit go_right when x = 0
    inhibit go_left when x ≠ 0
    l := 0 reacts-to l < 0
    l := N reacts-to l > N
  end
Mobile UNITY Overview

Components
Cart(1)  Loader(1) at 0  Unloader(1) at N

Interactions
Cart(k).x, Loader(i).y := Loader(i).y, 0
  when Cart(k).x = 0  Loader(i).y ≠ 0  Cart(k).l = 0

Cart(k).x, Unloader(j).z := 0, Cart(k).x
  when Cart(k).x ≠ 0  Unloader(j).z = 0  Cart(k).l = N
Moving to Context-Awareness

- The *Interactions* section is used to capture general rules of coordination
  - Provides system-level support that is uniform across all components
- Context-awareness exhibits asymmetry
  - Suggests the model encompass a more component-centric perspective
  - Components specify what the support structure should provide
  - Interactions should capture the system support for each component
Modeling Context:

Context Variable Definition

- Programs define asymmetric contexts
  - Comprised of **context variables** that hold context information
  - Definitions of these variables are based on context-sensitive conditional assignments
    - Based on information available in the environment
- As context changes, context variables are updated
- Often the context contributors are not known
  - Use non-determinism and existential quantification in context variable definition
Modeling Context:

Context Variable Definition

- Redefining the Cart in terms of its context
  - Part of the Cart’s context would be its destination
    - The Cart is not required to know a priori the locations of the Loader and Unloader
      - The Loader and Unloader could be mobile

\[ d := \text{Loader. when } x = 0 \not\equiv \text{Loader.y} \neq 0 \]
\[ \sim \text{Unloader. when } x \neq 0 \not\equiv \text{Unloader.z} = 0 \]
Many context-aware applications have no a priori information about their environment
- No information about contributors to context
- Non-deterministic assignment with existential quantification
  - Allows programs to select values for context variables based on some conditions
  - Notation: \( x = x' \cdot (\text{condition}(x')) \)

\[
d := d' \cdot (\forall j,y : \text{Loader}(j).y \neq 0 :: d' = \text{Loader}(j).\square) \text{ when } x = 0
\]
Modeling Context:

Using Context Variables

- Like any other variables, the program can use its context variables in its statements
- Local statements can react to changes in the external environment

Program Cart(i) at □

... assign
  inc :: □ = □ + 1
  dec :: □ = □ - 1
  inhibit inc when d ≤ □ 
  inhibit dec when d ≥ □ 

... end
Modeling Context:

Using Shadow Variables

- Context-aware programs often adapt to events in the environment
- To capture this in a state-based model we use **shadow variables**
  - Provide built-in ability to propagate changes in variables
    - e.g., through a macro `changed(variable)`
Modeling Context:

Context Resolution

- Programs may also want to affect their contexts
  - Assignment statements in the program change the context variables
  - A context resolution section projects these changes back onto the appropriate context components
    - Requires the ability to react to changes in the context variables using shadow variables

Loader.y := l  \textbf{reacts-to} \ changed(l) \quad l = 0
Modeling Context:

Context Consistency

- Different context-aware applications will respond to changes differently
  - Some (like the Loader) need immediate notification of changes to maintain consistency
    - Eager reaction to context changes
    - Defined via `reacts-to` construct
  - Others do not use this high level of consistency
    - Lazy reaction to context changes
    - Defined via `when (if)` construct
Conclusions

- Defining individualized contexts
  - Restructuring context definition
- Extending the scope of context
  - Using Mobile UNITY’s generalized view of location
- Generalizing context types
  - Assigning to context variables
- Affecting changes on the context
  - Defining rules for resolving context changes
Questions?

Mobile Computing Laboratory
www.cse.wustl.edu/mobilab