Mobile middleware

CARISMA and MADAM

Torkild Retvedt – Martin Øinæs Myrseth

Mobile middleware

- Context changes are frequent
- Mobile devices have limited resources
  - Limits complexity and overhead of context change handling
  - Context awareness has an impact on system resources and service quality

CARISMA

Context-Aware Reflective mIddleware System for Mobile Applications

Introduction to CARISMA

- Provide a context aware layer for mobile platforms
- Handle context changes
  - e.g. variation in bandwidth, battery, network coverage
- Implementation is hidden from both the user and the developer (transparent)
- Applications may have valuable information about contexts

The Reflective Model

- Mobile devices changes context rapidly
- Provide an abstraction of the middleware
- Allow applications to dynamically inspect and/or change middleware behavior
- Context configurations choose what policies are applied to a service

Profiles

- Profiles are passed down to the middleware
- Context configurations decides what policies to apply to a service
- Services are affected by one and only one policy at a time
- Applications may add associations, dynamically changing the behavior of the middleware

Conflicts

- We divide conflicts into two main categories
  - Intraprofile conflicts
    - Conflicts exists in a profile of an application on a single device (local conflict)
  - Interprofile conflicts
    - Conflicts exists between profiles on an application running on different devices (distributed among various middleware instances)

Conference Application

- Reminder of next talk
  - A local service
  - Service requested when attending a talk
  - An alert occur while user is interacting with the device, and attending a talk
  - Conflict! Each service is delivered using only one policy

```
talkReminder
    soundAlert
        location = outdoor
    vibraAlert
        location = conferenceRoom
    silentAlert
        userFocus = on
```

Conference Application cont.

- Exchange of messages
  - Distributed application
  - Alice has battery < 40% and Claire has bandwidth > 50%
    - Everyone agree
  - Alice has battery > 40% or Claire has bandwidth < 50%
    - Interprofile conflict!

```
% Bob
messagingService
  plainMsg
```

```
% Alice
messagingService
  plainMsg
    battery < 40%
  compressedMsg
    battery > 40%
```

```
% Claire
messagingService
  plainMsg
    bandwidth > 50%
  compressedMsg
    bandwidth < 50%
```


Avoiding conflicts

- Dynamicity
  - Not possible to discover conflicts before they happen
  - Ignore conflicts until they are invoked
- Simplicity
  - Cannot take up to much resources
- Customization
  - We don't want to ask applications for solutions each time a conflict occur
  - Applications must be able to favor a solution to a conflict

Conflict resolution

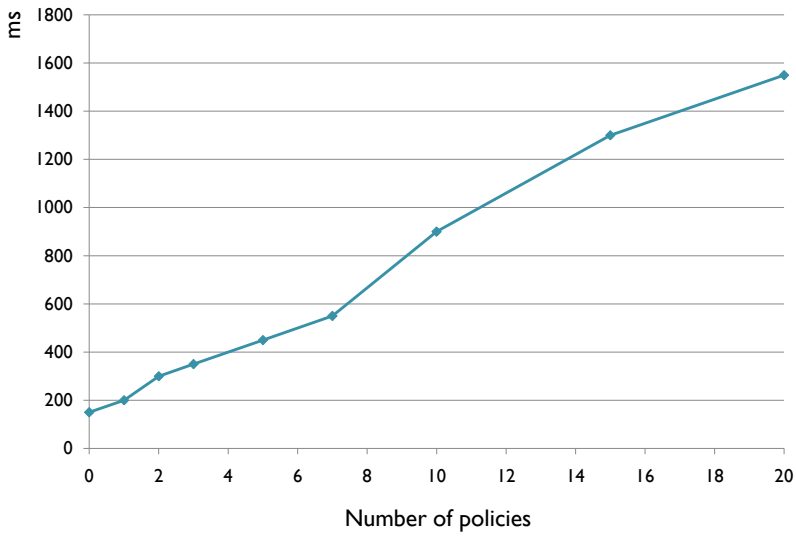
- All participants must take a collective choice to use a single policy
- Use microeconomic techniques
  - The different policies are goods
  - Applications are consumers
  - A good scheme to use is the auction protocol
    - Greater heterogeneity than simpler schemes
    - Parties make decisions independently
  - Middleware is the auctioneer

Conflict resolution cont.

- Computation of the solution set
  - Peers need to agree on a common policy
  - If no common policy is found, conflic cannot be resolved
  - All peers must bid in the auction
  - The policy with the highest sum of bids wins
  - All auctions are isolated
    - Next time same conflict may have a different winning policy
  - Policies can be favored
    - Applications can tell the middleware that it favors specific goals within a policy

Performance

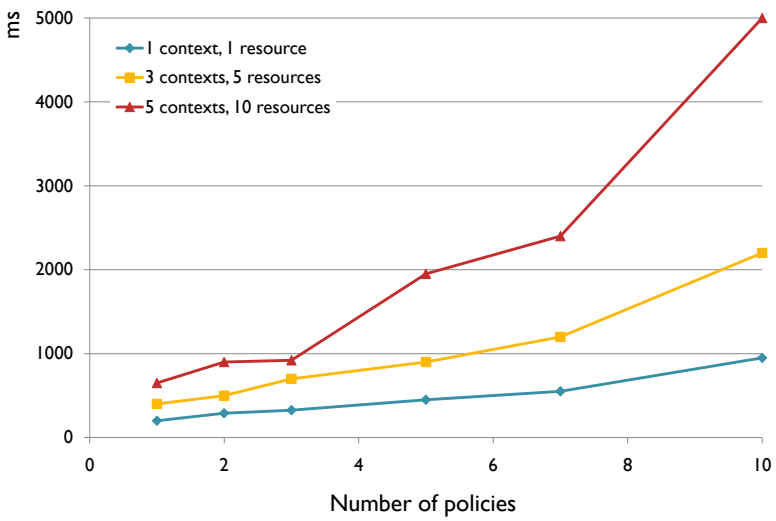
Impact of Reflection



Martin Øinæs Myrseth - Torkild Retvedt

Performance

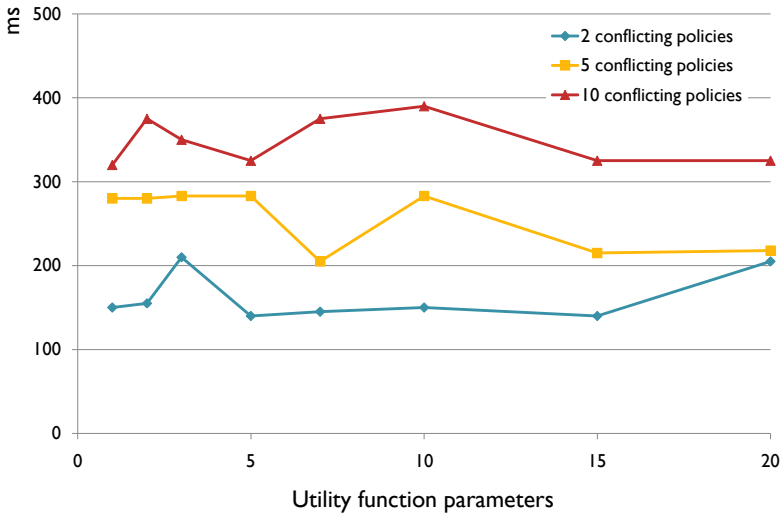
Impact of context-awareness



Martin Øinæs Myrseth - Torkild Retvedt

Performance

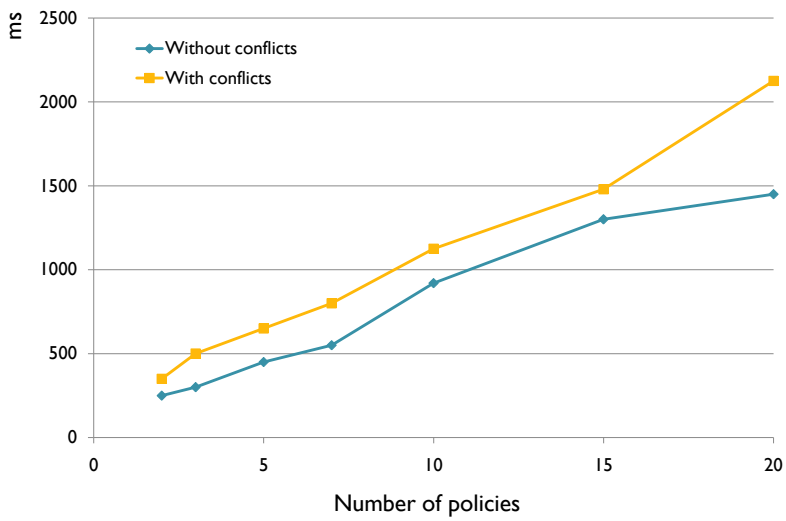
Impact of utility function parameters



Martin Øinæs Myrseth - Torkild Retvedt

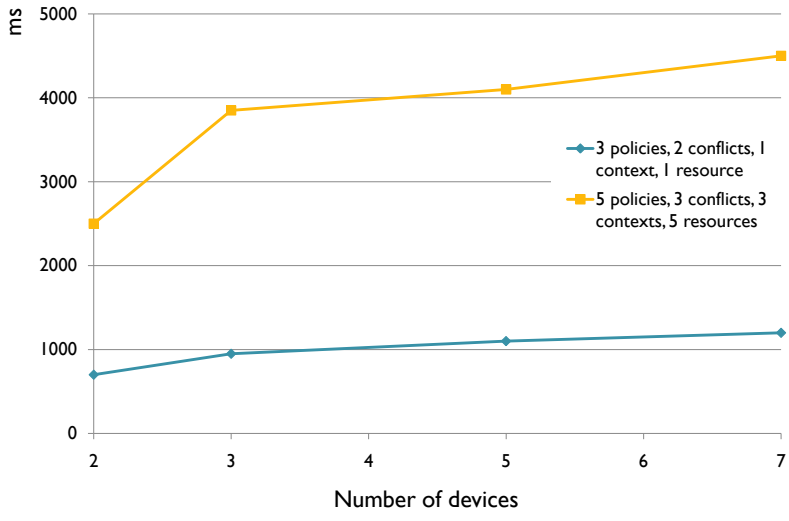
Performance

Impact of conflict resolution mechanism



Performance

Impact of conflicts in a distributed setting



MADAM

Mobility and ADaptation enAbling Middleware

MADAM – Goals

- Provide software engineers with suitable means to develop mobile adaptive applications
  - Modelling language extensions
  - Tools
  - Middleware
- Basis in studies of adaptation requirements of mobile applications
- Provide a set of reusable adaptation strategies and adaptation mechanisms
- Use a dynamically reconfigurable component architecture

MADAM's main functions

- Detect context changes
  - Changes in the operating environment
- Evaluate context changes and make a decision on what adaptation to perform
  - Select the best suited application variant
- Implement the adaptation choices
  - Adapt the running application, invoke the application variant

Variability

- MADAM uses component frameworks
  - Composition of component types
  - Plugging in different component implementations
- Two types of variability
  - Compositional variability
    - Coarse-grained adaptability
    - Structural and algorithmic variability
  - Parameterization
    - Fine-grained adaptability
    - Modify program variables and behavior

Variant properties

- Annotate components with properties discriminate between alternative component implementations
- Qualify the services components offer and needs
- Components interact through ports with attached properties
- Services needed and offered are properties attached to ports

Component type

- Component implementations plug into a component type
- Various component implementations should be comparable
- Component implementations must share a common set of properties, defined by the component type

Variant selection – utility functions

- MADAM uses utility functions for application variant decision making
- Utility functions assign a scalar value to every possible application variant as a function of application properties
- The architect specifies the utility functions, not the user – hard task
- User has the ability to prioritize certain needs to allow some level of user adaptation control

MADAM's architecture

- Runtime models
  - At application launch time the middleware interprets the models the architect specified to generate the *framework architecture model's* runtime representation.
  - All components that can plug into the component framework are identified by the middleware (described by compile-time models).
  - The runtime model might be generated only at launch time for software needing few updates.
  - Dynamic applications must update the runtime model while the application is running.

MADAM's architecture

- Context manager
  - Determines properties of interest in evaluation variants.
  - Assigning values to properties requires monitoring of the context since properties relate to context elements.
  - Handles context reasoning:
    - Aggregation
    - Derivation
    - Prediction
  - Passes relevant context information to the adaptation manager – when appropriate.

MADAM's architecture

- Adaptation manager
  - Evaluate the impact of changes on the application – changes reported from the context manager
  - Select an application variant that best suits the current context and user needs – Utility functions

MADAM's architecture

- Configurator
  - Reconfiguring an application.
  - Compares the application instance with new variant models to derive the reconfiguration steps.
  - Might
    - Bring components into safe state.
    - Delete or replace component instances.
    - Instantiate components.
    - Transfer states.

MADAM's architecture

- Core component
  - Provides platform-independent services for managing
    - Applications
    - Components
    - Component instances
  - Includes operations for
    - Publication and discovery of component frameworks and implementations
    - Loading, unloading and connecting components
  - Provides platform-independent access to execution platform's resources
    - Memory etc.

MADAM in action

- Two simple case examples
  - An information service to support janitor inspections.
  - A video streaming application.
- Two industrial pilot services
  - Executed in a simulated context environment.
  - Contained development of architecture models
  - Implementation adjusted the implementation of existing product components to support the reconfiguration interfaces the middleware requires.
  - Lacking good support for defining properties and utility functions

MADAM results

- Prototype middleware
  - 3,000 variations evaluated within one second (iPAQ 5550).
- Two industrial pilot services
  - Many fewer variations evaluated in the same time-span.
  - Less relevant variants than the number of variants obtained by exploring the whole variation set.

MADAM and beyond - scalability

- Scalability
  - More extensive use of parameterization
    - Effectively models and implements variability
    - Can lead to larger sets of variants with only small differences in component properties
  - Concurrently running applications
    - Competing for the same resources
    - Reason over a set of concurrent applications

CARISMA vs MADAM

- Context aware
- Profiles, policies
- Conflicts and conflict resolving
- Utility functions
  - Customizable
- Generalization of reification
- Transparent

- Context aware
- Component types
- No concrete conflict resolving
- Utility functions
  - Customizable
- Architecture runtime models
- Transparent

Questions?