[simula . research laboratory]

QuA

Service Plans for Context- and QoS-aware Dynamic Middleware

SIUMI'06

Sten A. Lundesgaard (presenter), Ketil Lund and Prof. Frank Eliassen

Simula Research Laboratory

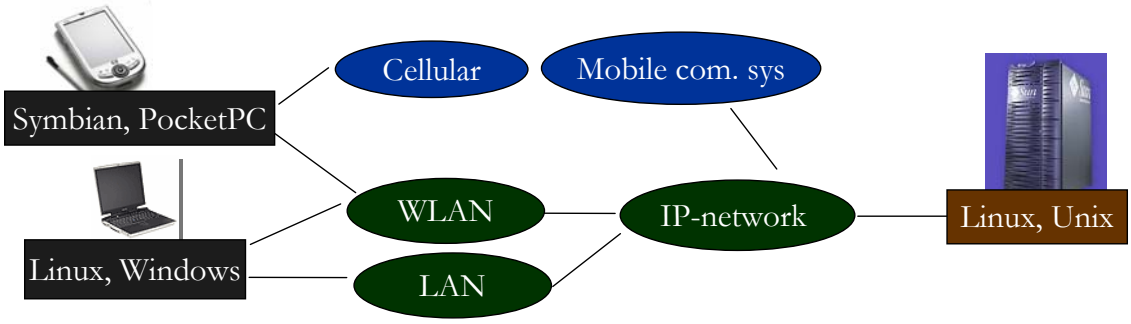
University of Oslo, Norway

Agenda

- Background and Problem
- Approach - to enable mobile middleware support QoS-sensitive applications in a dynamic environment
- Contribution –middleware architecture and service plan concept
- Life-Cycle Phases –application

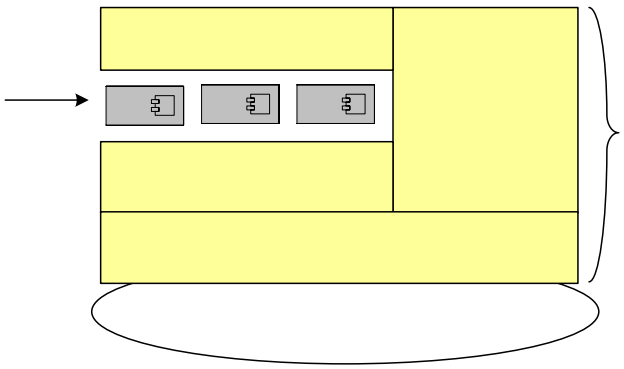
Background (1)

- Currently mobile terminals and mobile communication systems provide best-effort QoS.
- Application provide and maintain QoS-levels:
  - QoS mechanisms integral part of the application logic (e.g., rate adaptation, content processing, error correction).
  - Applications dynamically reconfigure itself (e.g., add, remove, or replace component).



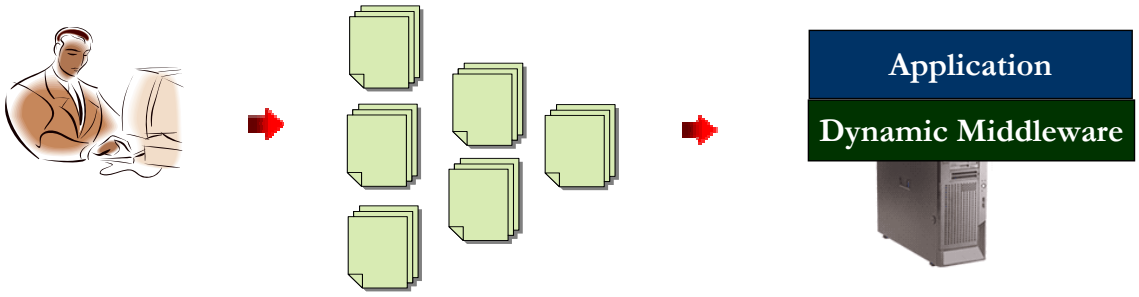
Background (2)

- Academia works on dynamic middleware; employs alternatives of the application together with late bindings (of components) for run-time configuration.
- Management plane and hard-coded rules to control reconfiguration of component composition
- Meta-level for introspection of the running application.



Problem

- Each alternative explicitly (and completely) specified, i.e., many configurations equal number of specifications.
- Dynamic (Re)Configuration according to changes in the environment, i.e., context-awareness only.
- Specification specialised for platform, no reuse of specification and thus the alternative application.



Approach (1)

- To solve the problem, we advocate that mobile middleware should be both context- and QoS-aware.

  ➔ ***Context- and QoS-aware architecture***

- Use components as software entity for late-binding and -configuration:
  - Selects and combines components for a given context and resource QoS characteristics and user QoS requirements.
  - During mobility the middleware dynamically reconfigure the application to context changes and to maintain QoS:
    - Parameter settings of individual components
    - Composition of components
    - Implementation of component type

Approach (2)

- Each alternative application configuration is prepared at design time and deployed onto the mobile middleware.
- Specifications of the alternatives and internal representation within the middleware.

  ➔ ***Service Plan Concept***

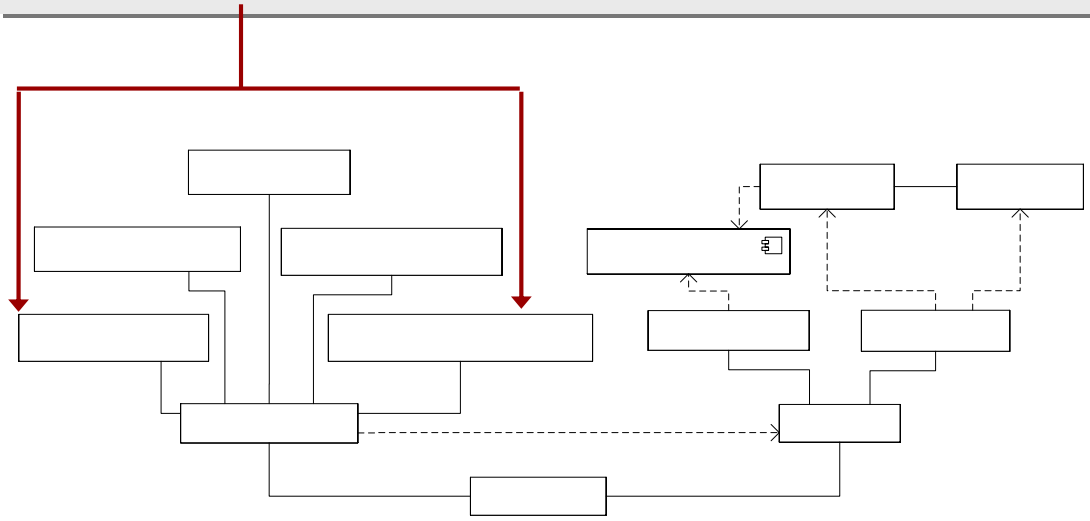
Agenda

- Background and Problem
- Approach - to enable mobile middleware support QoS-sensitive application in a dynamic environment
- Contribution –middleware architecture and service plan concept
- Life-Cycle Phases –application

Context- and QoS-aware middleware architecture (1)

- Traditional mechanisms supported, e.g., context and reconfiguration management

➔ In our QoS-aware middleware architecture employ hooks for QoS management mechanisms.



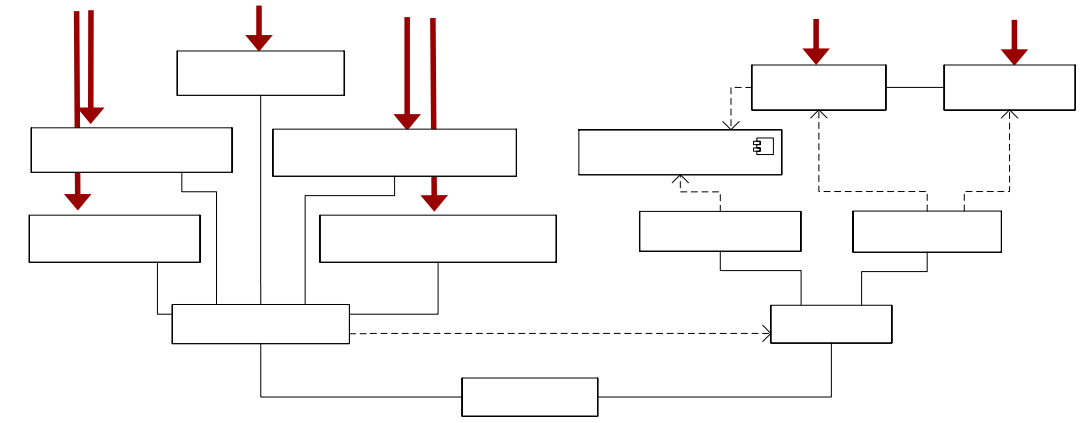
Context- and QoS-aware middleware architecture (2)

➔ *Service type* and *Service Plan* explicitly defines meta-level in architecture.

➔ *ServicePlanner* chooses the application configuration based upon information from *ContextManager* and *ResourceManager* and the *service plans.*

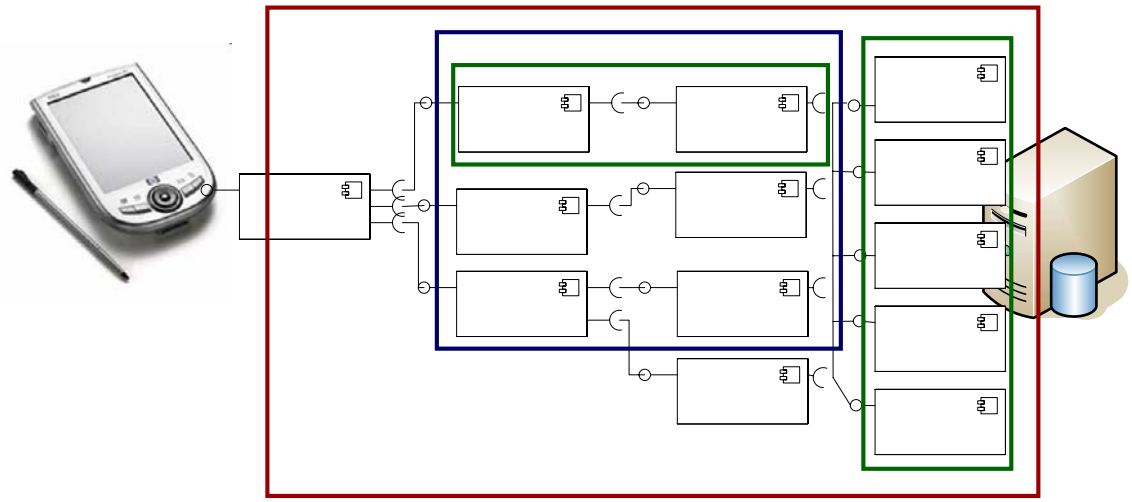
➔ *ServicePlanner* decides when and what to reconfigure, based upon *service plans*.

➔ *Configuration and Reconfiguration* managers use meta-level to (re)configure the application.



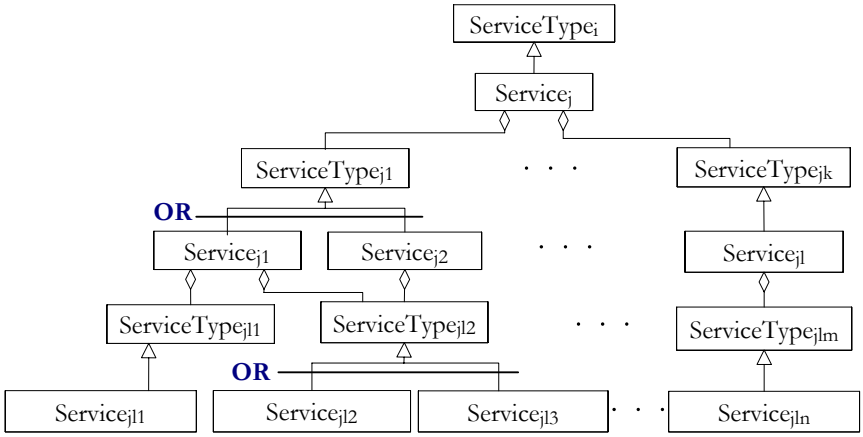
Service Plan Concept (1)

- An application is formed by many components.
- Each one offers a service, which is grouped into compositions of services.



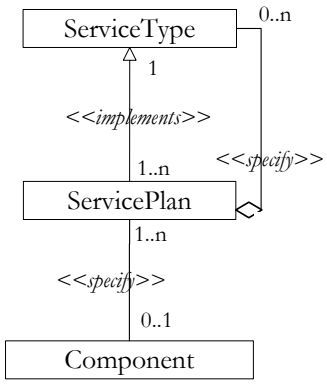
Service Plan Concept (2)

- To be able to define alternatives of these compositions:
  - Make the interface of each service explicit, i.e., service type
  - Introduce a recursive structure of services.
- Variation points are placed in the structure and service type.



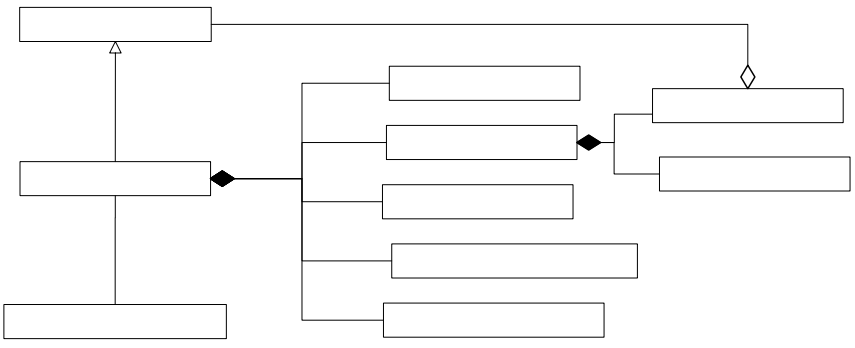
Service Plan Concept (3)

- Need some means to associate the implementation of a service to the service type ➔ the service plan.
- Plan specifies
  - a composition
  - a component

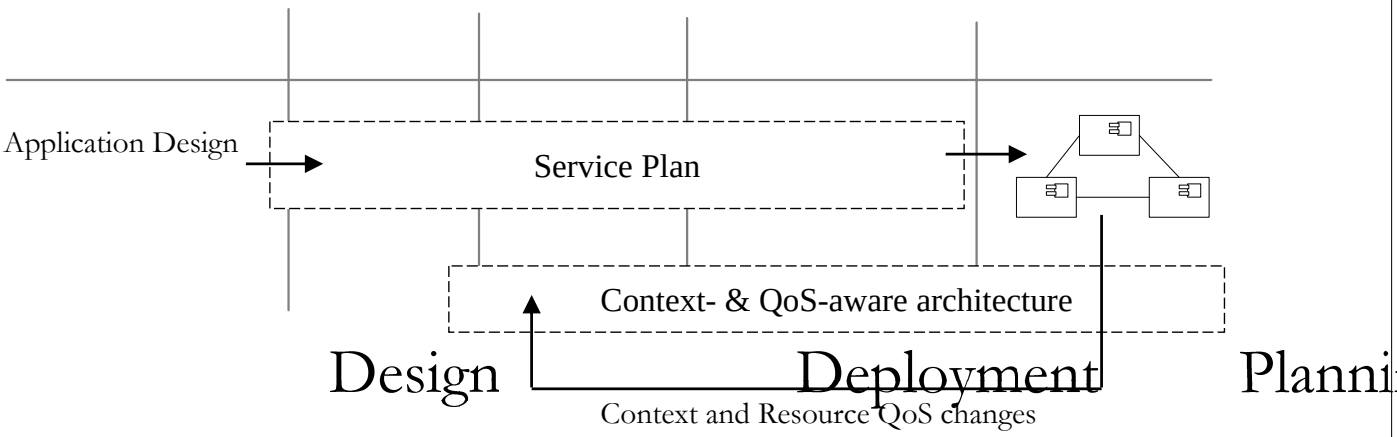


Service Plan Concept (4)

- Extends the plan to include information elements about:
  - Dependencies to the context
  - Any parameter configurations
  - **QoS characteristics**



Application Life-Cycle

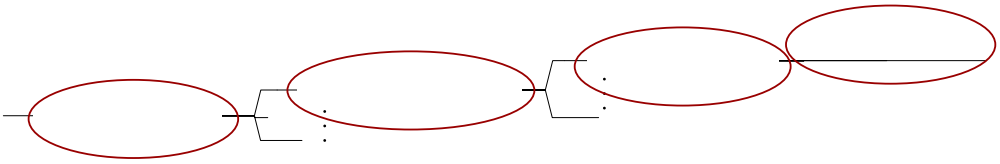


Life-Cycle –Deployment (1)

- Service types and Service plans are deployed using WSDL and XML respectively.
  - Open standards, Human readable, and supported by software design tools.
  - Service types and plans are parsed and analysed using existing (J)DOM API since generic and small total foot-print.
- In case of a service composition the WSDL file (i.e., the service type to the sub-service) imports the name spaces of the services in the composition.
  - Avoid duplication of information.
  - Reuse of to/from message definitions.
- Service Plans employ a tree structure.
  - Elements (among other things) for specifying dependencies to context elements in environment and calculating QoS at application and user level.

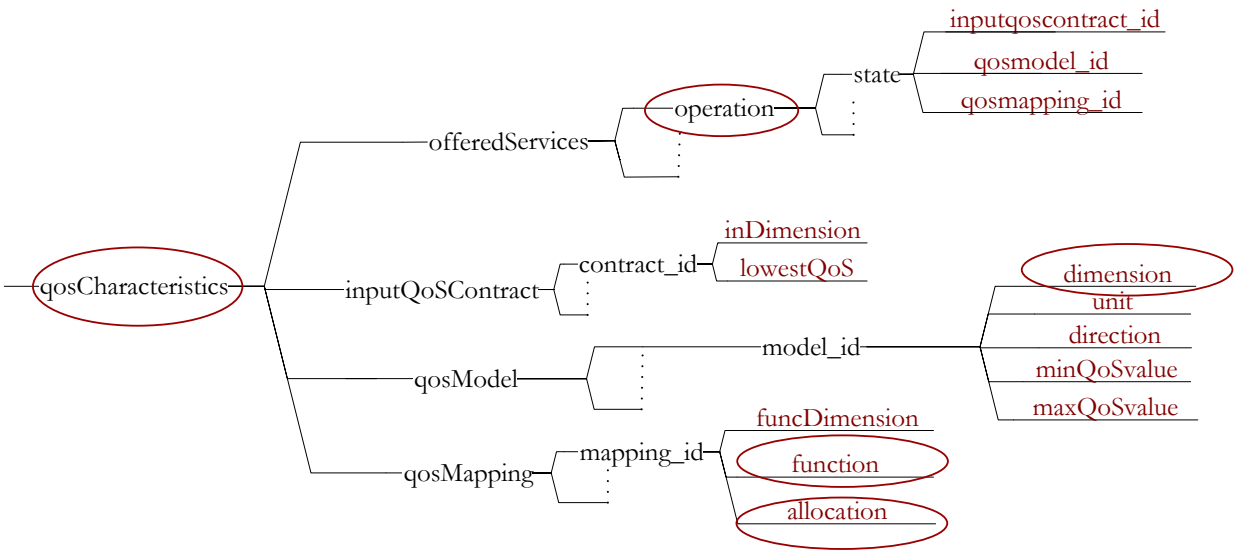
Life-Cycle –Deployment (2)

- Specifying Context dependencies:



Life-Cycle –Deployment (3)

- Specifying QoS:



Summary

- Context- and QoS-awareness can be achieved in the mobile domain.
- Require means to specify alternative application configurations independent of target platform.
- Our solution:
  - New context- and QoS-aware middleware architecture.
  - Concept for specifying and representing in the middleware the application configurations and their QoS characteristics.

Final Discussion

New services, design and rapid prototyping

- The idea of services and service composition is not new. Still they are useful the terms, since enable us to discuss:
  - software without having to take into consider implementation technologies.
  - reuse of existing fully functional software that can be accessed by users.
- Increased proliferation of Internet and mobile terminals

  + ERP and e-commerce and m-commerce systems

  => engineering methods and tools and middleware that allow for shorter development time at a lower cost and self-configuration.
- Hence, concepts and mechanisms for service (oriented computing) should be a natural evolution of existing OOD and target platforms.

Middleware, significant requirements for best support

- Today, main motivation for middleware is to reduce the volume of code and testing needed (i.e., fewer engineers in the project ➔ lowest possible development cost).
- Additional libraries (with advanced functionality) are added to the system on a needed basis.
- Middleware should be possible to use on as many types of computers as possible, open for configurations of its functionality, and at deployment time and run-time add mechanisms needed according to:
  - Technical domain (mobile, grid, embedded)
  - Application type (transaction, streaming, content, logistics, control)
- Dynamic middleware have a possibility for self-configuration, which will avoid high cost even when systems spans across more computers.