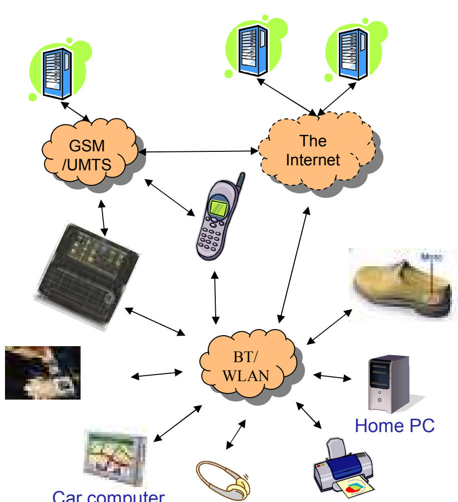
Mobile and ubiquitous computing

INF 5040/9040 autumn 2009

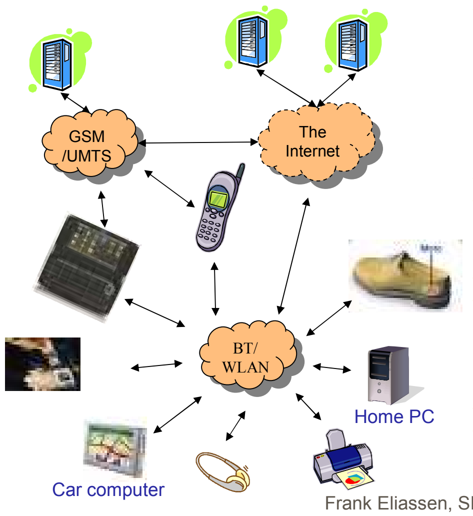
lecturer: Frank Eliassen

Motivation



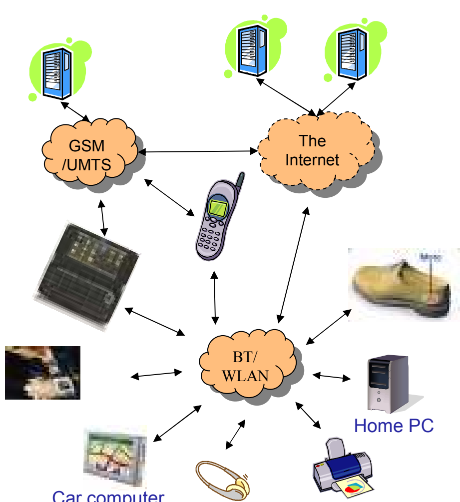
- Mobile computing is concerned with exploiting the connectedness of portable devices
- Ubiquitous computing is about exploiting the increasing integration of services and (small/tiny) computing devices in our everyday physical world
- Mobile and ubiquitous computing require particular solutions in many areas caused by dynamically changing computing environment: users, devices, and software components

Some open questions



- How can software components associate and interoperate with one another while devices move, fail or spontaneously appear?
- How can systems become integrated with the physical world?
- How to adapt to small devices' lack of computation and I/O resources?
- How to handle security in volatile, physically integrated systems?

Fields and subfields of mobile and ubiquitous computing



- Mobile computing
- Ubiquitous computing
- Wearable computing
- Context-aware computing

Volatile systems

- Common system model for mobile and ubiquitous computing (and their subfields)
- Changes (or failures) are considered common rather than exceptional (in contrast to other types of systems where changes or failures are considered to be exceptions)
- Forms of volatility
  - failures of devices and communication links
  - changes in the characteristics of communication such as bandwidth
  - the creation and destruction of associations – logical communication relations – between software components resident on the devices
- Mobile and ubiquitous computing exhibit *all* of the above forms of volatility.

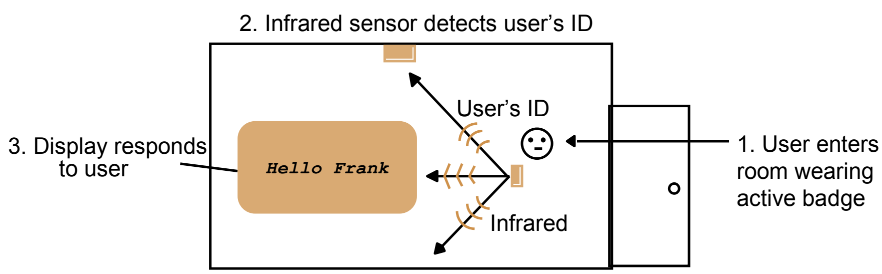
Elements of the volatile systems model

- smart spaces
- device model
- volatile connectivity
- spontaneous interoperation

Smart spaces: environments within which volatile systems subsist

- A physical place/room with embedded services. The services are provided only or principally within that space
- Movements or 'appearance or disappearance' in a smart space:
  - Physical mobility
  - Logical mobilitet
  - Service/device appearence
  - Service/device disappearance
- GAIA: active spaces
  - http://gaia.cs.uiuc.edu/

Smart rooms that respond to a user with an active badge



Elements of the volatile systems model

- smart spaces
- device model
- volatile connectivity
- spontaneous interoperation

Frank Eliassen, SRL & Ifi/UiO 9

Device model

- Limited energy
  - typically use battery
  - energy-preserving algorithms
- Resource constraints
  - relative resource poor (CPU, memory, ...)
  - algorithmic challenge
- Sensors and actuators
  - to make a device context-aware
- Examples
  - Motes
  - Camera phones
  - ....

Frank Eliassen, SRL & Ifi/UiO 10

Elements of the volatile systems model

- smart spaces
- device model
- volatile connectivity
- spontaneous interoperation

Volatile connectivity

- Variation between different technologies (Bluetooth, WiFi, GPRS, etc)
  - Bandwidth, latency
  - Energy costs
  - Financial costs to communicate
- Disconnection
  - More likely in wireless networks
- Variable bandwidth and latency
  - Packet loss due to weak signal
  - Signal strength varies
  - Difficult to determine timeout-values in higher layer protocols due to varying conditions

Elements of the volatile systems model

- smart spaces
- device model
- volatile connectivity
- spontaneous interoperation

Spontaneous interoperation

- In volatile systems, components routinely change the set of components they communicate with
  - take advantage of possibility to communicate with local components in a smart space, or a device may want to offer services to clients in its local environment
- **Association**: a logical relationship formed when at least one of a given pair of components communicates with the other over some well-defined period of time
- **Interoperation:** interaction during an association
- **Spontaneous** interoperation: interoperation that is not planned or designed in!

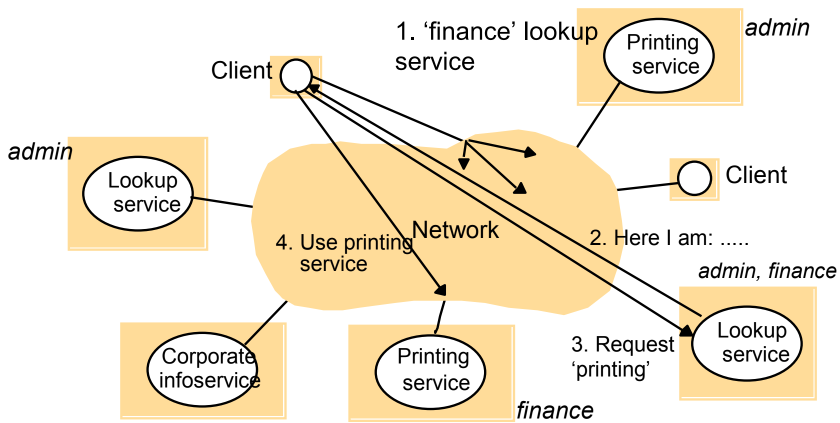
Pre-configured vs spontaneous associations: examples

Pre-configured	Spontaneous
Service-driven: *email client and server*	Human-driven: *web browser and web servers*
	Data-driven: *P2P file-sharing applications*
	Physically-driven: *mobile and ubiquitous systems*

Entering a smart space

- **Requirement:** a device that appears in a smart space needs to bootstrap itself in the smart space
- Two steps for bootstrapping itself:
  - Network bootstrapping (DHCP-server)
  - Establish associations between components on the device and services in the smart space
- The association problem
  - How to constrain the scope to services in the smart space only (e.g., the hotel room)?
  - 'Boundary principle':
    - smart spaces need to have system boundaries that correspond accurately to meaningful spaces as they are normally defined (territorially or administratively)
- **Discovery-services:** one approach to the association problem (e.g., Jini: see Coulouris page 671)

Service discovery in Jini



Discovery services

- A directory service that is used to register and look up services in a smart space
- Requirements to discovery services
  - Service attributes is determined at runtime (hard!)
  - Service discovery must be possible in a smart space without infrastructure to host a service discovery service
  - Registered services may spontaneously disappear
  - The protocols used for accessing the directory need to be sensitive to the energy and bandwidth they consume (cf. device model)

Interface to a discovery service

Methods for service de/registration	Explanation
lease := register(address, attributes)	Register the service at the given address with the given attributes; a lease is returned
refresh(lease)	Refresh the lease returned at registration
deregister(lease)	Remove the service record registered under the given lease
Method invoked to look up a service	
serviceSet := query(attributeSpecification)	Return a set of registered services whose attributes match the given specification

Design choices for discovery services

- Directory server or serverless
- Directory server: clients issue a multicast-request to locate the server (as in Jini)
  - Not all smart spaces have facilities for server implementations
- Serverless discovery: the participating devices collaborate to implement a distributed discovery service
    - **Push model**: servers multicast ('advertise') their descriptions regularly, and clients run their queries against them
    - **Pull model:** clients multicast their requests and devices providing matching services, respond
  - Both approaches are relatively resource demanding (battery, bandwidth) in their pure form

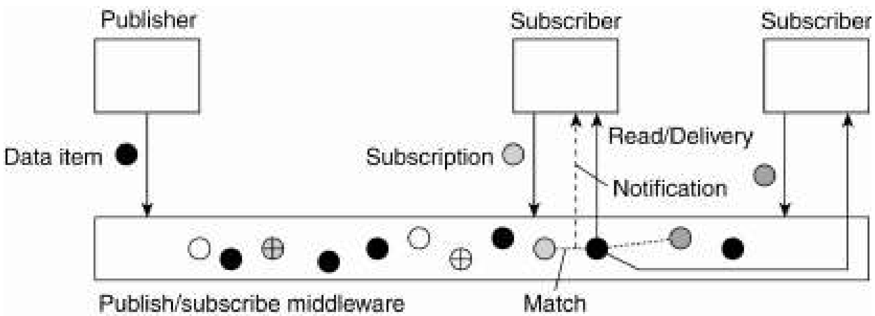
Interoperation

- How can components that want to associate determine what protocol they can use to communicate?
- Main problem is incompatibility between software interfaces (components need not have been designed together)
- Two approaches:
  - Adapt interface to each other (interface adaptation): difficult
  - Constrain interfaces to be identical in syntax across as wide a class of components as possible
    - Example: Unix pipes (read, write)
    - Example: The set of methods defined in HTTP (GET, POST, ...)
    - Such systems are called ***data oriented***
    - Require additional mechanisms to describe type and value of data exchanged (e.g. MIME types), as well as as the processing semantics of the server (difficult!)

Data oriented programming models

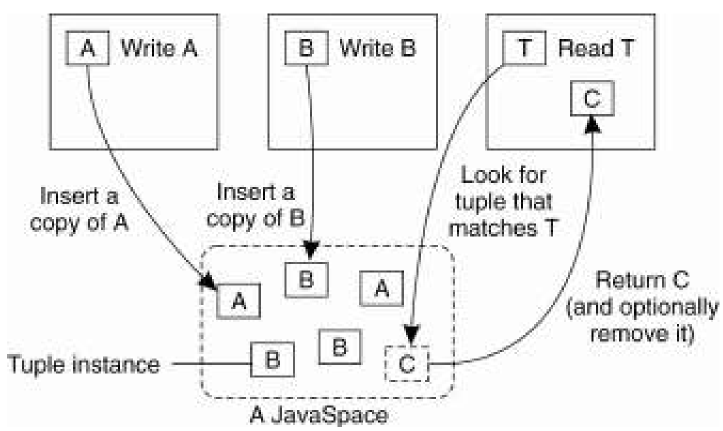
- **Data oriented programming models that have been used for volatile systems**:
  - Event-systems (pub/sub)
  - Tuple spaces
  - Direct device interoperation (devices brought into direct association)

Principle of publish-subscribe (event system)



Frank Eliassen, SRL & Ifi/UiO 23

Example of tuple space: JavaSpaces



Frank Eliassen, SRL & Ifi/UiO 24

Mobile and ubiquitous computing systems

- How can such systems be integrated with the physical world ?

Frank Eliassen, SRL & Ifi/UiO 25

Sensing and context awareness

- Systems can be integrated with the physical world through sensing and context awareness
- Sensing: use sensors to collect data about the environment
- Context aware systems: can respond to its (sensed) physical environments (location, heat, light intensity, device orientation, presence of a device, etc.) and the context can determine its (further) behaviour
- **Context** of an entity (person, place or thing): an aspect of its physical circumstances of relevance to system behaviour

Frank Eliassen, SRL & Ifi/UiO 26

Sensors

- Combination of hardware and software
- Sensors are the basis for determining contextual values
  - Location, velocity, orientation, ...
  - Temperature, light intensity, noise, ...
  - Presence of persons or things (e.g., based on RFID – electronic labels - or Active Badges)
- An important aspect of a sensor is its failure model
  - Some are simple (e.g., a thermometer often has known error bounds and distribution), some are complicated (e.g., accuracy of satellite navigation units depend on dynamic factors)

Sensor architectures

- Applications normally operate on more abstract values than sensors can produce
- Sensor abstractions are important to avoid application level concerns with the peculiarities of individual sensors
- Therefore common to build a software architecture for sensor data as hierarchies
  - Nodes at a low hierarchical level provide sensor data at a low level of abstraction (longitude/latitude of a device)
  - Nodes at higher hierarchical levels (closer to the root node) provide sensor data at higher levels of abstraction (the device is in Frank's Cafe)
- Nodes at higher levels combine sensor data from lower levels both to abstract and to increase reliability

Context Toolkit: Example of sensor software architecture

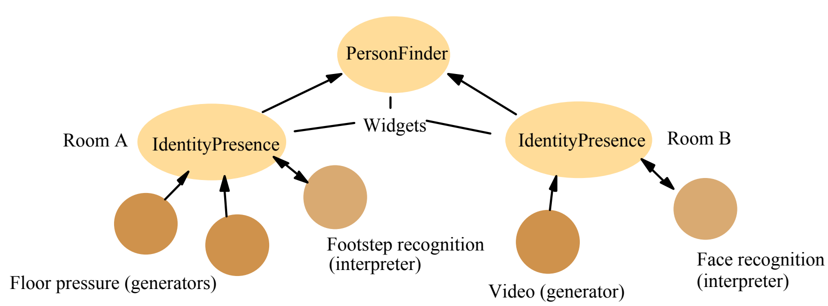
- System architecture for general context aware applications
- Based on 'context-widgets': resuable components that abstract over some types of context attributes (hide low level sensor details)
- Example: Interface to a IdentityPresence widget class

Attributes (accessible by polling)	Explanation
Location	Location the widget is monitoring
Identity	ID of the last user sensed
Timestamp	Time of the last arrival
Callbacks	
PersonArrives(location, identity, timestamp)	Triggered when a user arrives
PersonLeaves(location, identity, timestamp)	Triggered when a user leaves

Frank Eliassen, SRL & Ifi/UiO 29

Context Toolkit: Example of use of IdentityPresence widget

- A *PersonFinder* widget constructed by using IdentityPresence widgets ...



Frank Eliassen, SRL & Ifi/UiO 30

Wireless sensor networks (WSN)

- Network consisting of a (typically high) number of small, low-cost units or *nodes* that are more or less arbitrary arranged (e.g., "thrown out" in high numbers in a certain geographical area)
- Self-organising (ad-hoc network), functions independently of an infrastructure
- The nodes have sensing and processing capacity, can communicate wirelessly with a limited range (save energy), and act as routers for each other
- Are volatile systems because nodes can fail (battery exhaustion or otherwise destroyed (e.g., fire)), connectivity can change due to node failures

Frank Eliassen, SRL & Ifi/UiO 31

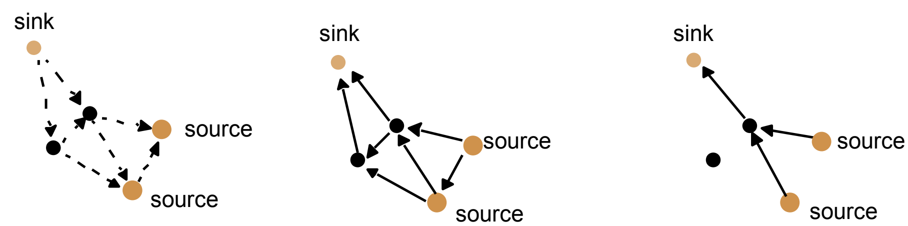
Three architectural features of wireless sensor networks

- Features driven by requirements of energy conservation and continuous operation
- *In-network processing*: The nodes have processing capabilities because processing is much less costly in energy consumption than (wireless) communication. Can be exploited to reduce the need for communication (only communicate when there is a need for it)
- *Disruption-tolerant networking*: based on store-and-forward transfer of data (not end-to-end)
- *Data oriented programmeng of nodes*: since nodes can fail, we can not rely on programming techniques for sensor nodes that refer to single nodes

Frank Eliassen, SRL & Ifi/UiO 32

Directed diffusion

- A programming technique that takes the three architectural features of wireless sensor networks into account
- Nodes (sources) have sensing capabilities (e.g., can measure temperature) or properties (e.g. location) that they can compare to needs for sensor information that they receive (as messages)
- Nodes that have a need for sensor information (sinks) declare this in ”interest messages” that they send to neighbour nodes.



A. Interest propagation B. Gradients set up C. Data delivery

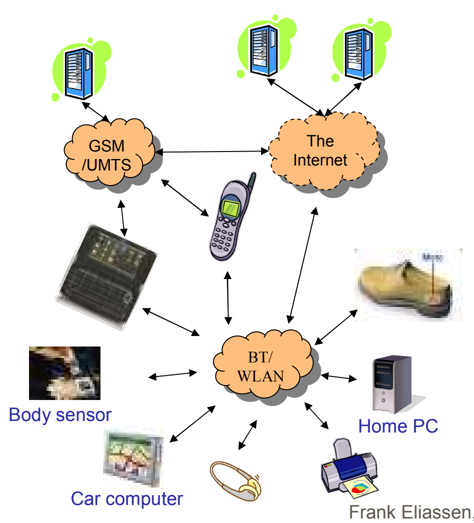
On the need for adaptation

- Run-time conditions of mobile applications (applications running on mobile devices) vary dynamically
  - Varying capabilities of different devices
  - Varying resource availability
  - User needs and wishes
- Need for adapting the application to a dynamically varying context
  - Adapt application to resource situation (battery, bandwidth, memory)
    - Example: Dynamically adapt media quality (e.g., video) to available bandwidth and/or to user preferences, and/or to device capabilities
  - Dynamically adapt user interface to situation of user or the orientation of the device ...
  - Adapt application to availability of devices and services in the environment (ubiquitous services)

MUSIC - Middleware for context-aware mobile application

- Mobile USers In ubiquitous Computing environments
  - An EU-funded project (FP6, Integrated Project)
  - Started October 2006, running through March 2010
- Project outcomes
  - Development studio (modeling and transformation tools)
  - Middleware architecture (prototype implementation)
- Licensing
  - Open Source project (Licensed under LGPL 2.1)
- Website
  - http://www.ist-music.eu

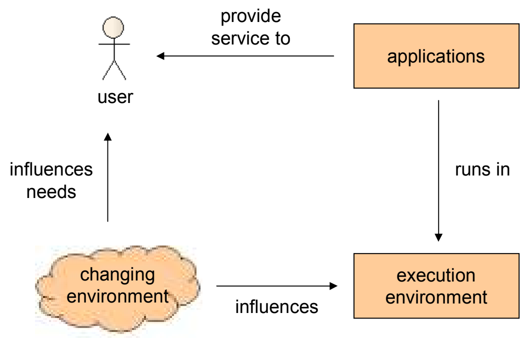
MUSIC - Motivation



When handheld devices are carried by users moving around in ubiquitous computing environments

- ✓ devices come and go
- ✓ network connections come and go and QoS varies, and therefore
- ✓ services available for use come and go
- ✓ service quality varies
- ✓ user tasks vary and are interleaved with tasks related to movement and social interaction
- ✓ computing resources and power are limited

MUSIC - *Motivation*



Frank Eliassen, SRL & Ifi/UiO 37

MUSIC - Motivation

- In such environments applications and users will benefit a lot from context awareness and self-adaptiveness
- The demand for applications exhibiting such properties is accelerating
  - Mobile computing
  - Ubiquitous computing
  - Service oriented computing
- Developing such applications with existing methods and technology is difficult, time-consuming and costly

Frank Eliassen, SRL & Ifi/UiO 38

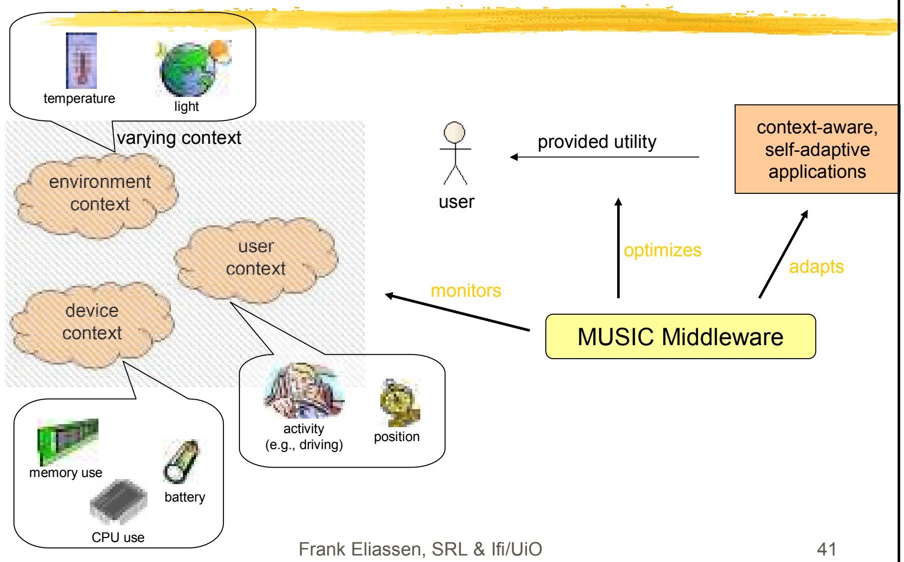
MUSIC - Vision

- "Provide methods, tools and runtime support for developing, deploying and maintaining context-aware, self-adaptive applications aiming for mobile and pervasive computing environments"

MUSIC - Approach

- Disciplined development methodology
  - Separately develop the context sensing and self-adapting logic from the business logic of the applications (Separation of Concerns)
- Use a middleware layer to transparently manage the extra-functional aspects of the application
  - Single, centralized context management (i.e., sensing, reasoning, storing, accessing)
  - Cross-application adaptation reasoning (utility function based)

MUSIC - *Approach*



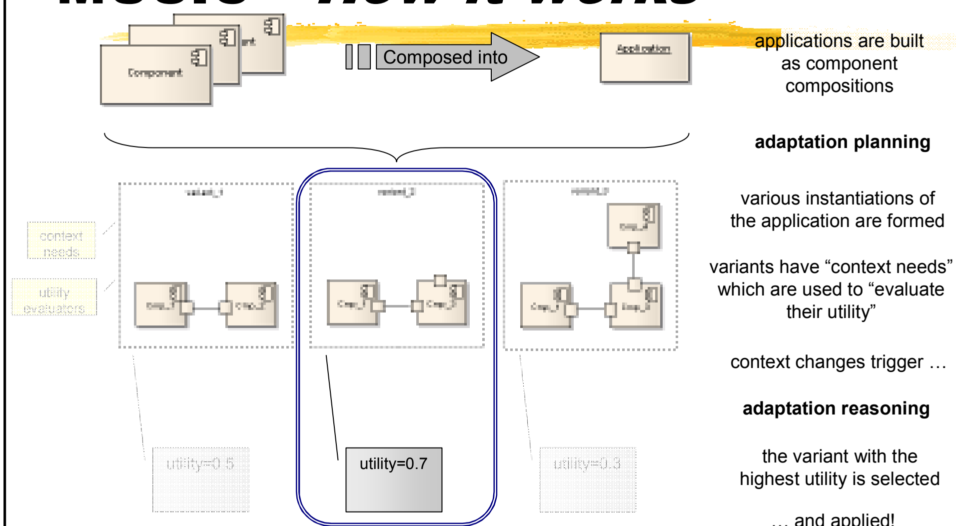
Frank Eliassen, SRL & Ifi/UiO 41

MUSIC – How it works

- Component-based applications
  - Components can be added/removed at runtime (flexibility)
  - Application variants (configurations) are determined at runtime
- Centralized context management
  - Context sensing, storing, reasoning, and access are all performed in a centralized way inside the middleware
- Adaptation reasoning
  - Based on the varying context (environment, execution context, user needs) the utility of each variant is evaluated dynamically
  - Adaptation occurs when the variant found to be offering the optimal utility is different from the selected one

Frank Eliassen, SRL & Ifi/UiO 42

MUSIC – *How it works*



applications are built as component compositions

adaptation planning

various instantiations of the application are formed

variants have “context needs” which are used to “evaluate their utility”

context changes trigger …

adaptation reasoning

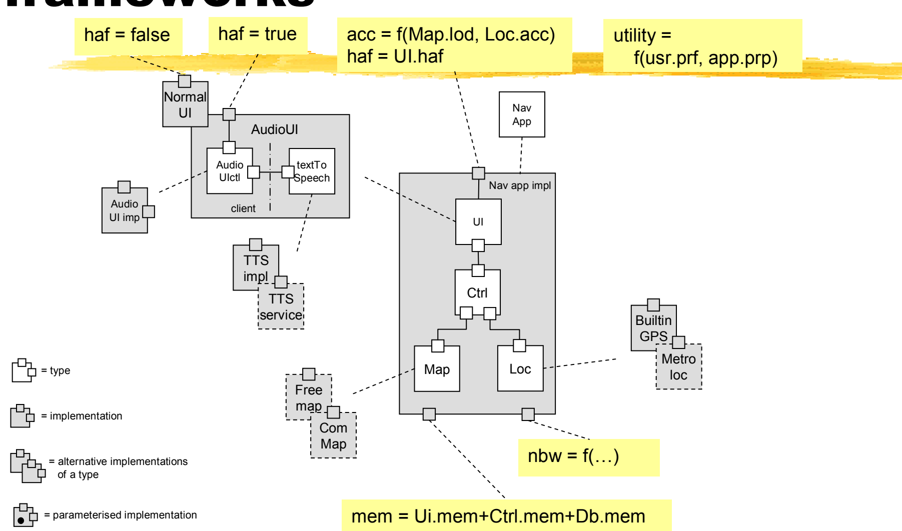
the variant with the highest utility is selected

… and applied!

Frank Eliassen, SRL & Ifi/UiO

43

MUSIC – role of component frameworks



Frank Eliassen, SRL & Ifi/UiO

44

Summary

- Most challenges to mobile og ubiquitous systems are caused by their volatile nature
- In such environments applications need to be context aware and adaptive
  - Integrated with the physical world through sensing and context awareness
  - Adapt to changes in the physical circumstances by changing behavior (e.g. component reconfiguration)
- There are many challenges, but yet only few (comprehensive) solutions
  - MUSIC: an example of a comprehensive solution

Announcing INF5360/9360: Seminar on Dependable and Adaptive Distributed Systems

- First time: Spring 2008. Running again spring 2010
- Seminar content
  - Explores state-of-the art principles, methods, and techniques for devising adaptive and dependable distributed systems.
- The seminar covers
  - Architectural and infrastructural principles for adaptive and dependable distributed systems.
  - Adaptivity and dependability in service-oriented architectures, data dissemination systems, P2P systems, mobile and wireless environments.
  - Approaches to improve the scalability of dependable and adaptive systems.
  - Evaluation and experience reports on dependable and adaptive distributed systems and services.
- Assumes knowledge at INF5040/9040 level