Outline

- Introduction and Major Issues
- Commercial Mobile Middleware
- Next-Generation Mobile Middleware
- Case Study – LIME
- Middleware for Wireless Sensor Networks
- Open Issues and Future Directions

Next Generation Middleware

- What is missing from the "commercial" systems?
- Issue Driven Survey

Replication	Object Oriented
Adaptation	Transactions
Service Discovery	Transport Layer Modifications
Event-Based	Underlying Algorithms

Replication

- Goals:
  - Increase accessibility and reliability of data
  - Enable disconnected operation
- Challenges:
  - Limited resources on mobile devices
  - Unpredictability of access to data
- Questions:
  - What to replicate
  - When to replicate
  - How to deal with offline changes that conflict

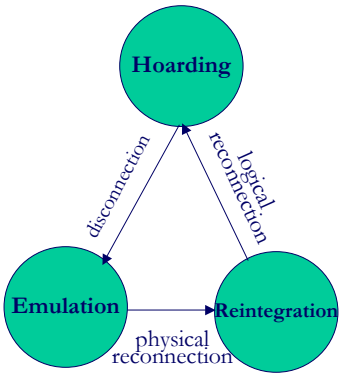
Coda

- A network file system that supports ***disconnected operation***, i.e., allows clients to continue to access critical data in the face of *temporary* server and network failures
  - Only servers are trusted
  - Support only for nomadic computing, not ad hoc
- Based on data caching and replication, not only to increase performance, but also to increase availability
  - Goal: provide an application-*transparent* file access
- It uses optimistic replication, allowing modifications on replicas and detecting (and possibly resolving) them afterwards
  - Pessimistic replication would lock everything upon disconnection
  - Short-term conflicts are rare in practice, but their probability increases with long-term partitions

How Coda Works

CMU — Replication

- Volumes (subtrees) are replicated at several (fixed) servers
- During normal operation Coda fetches files into the local cache
  - The cache is periodically (10 minutes) re-evaluated, thus newer files may be fetched and old files in the cache may be discarded (equilibrium)
  - User-specified *hoarding profiles* allow one to specify priorities over files (files with low priority will be discarded first)
- During disconnection, the client can access only files that are local or in the cache; cache misses appear as failures to the user
- Modifications are logged in a *replay log*, which is used upon reconnection to reconcile the state of client and servers
  - Reintegration is performed one volume at a time: activity on the volume is suspended



Bayou

Xerox PARC — Replication

- Yet another replication system. Existing applications include calendar, email, bib database
  - Support for arbitrary communication topologies
    - Any pair of replicas can update one another
  - Operation over low-bandwidth networks
    - Exchange update operations, not all data
  - Incremental progress
    - If interrupted, update process not total loss
  - Eventual consistency
    - Epidemics ensures consistency
  - Efficient storage management
    - Replicas can discard updates to reclaim storage
  - Propagation through transportable media
  - Light-weight management of replica sets
    - Does not require a server to create a replica
  - Arbitrary policy choices
    - Need only ensure an eventual path between any pair of replicas

Bayou

- Provides application specific conflict detection and resolution
- Based on DBMS repository
  - The API provides `read` and `write` operations
  - Conflicts are generated by `write` after `write` operations
  - Also includes dependency check for data
- Replicas are always accessible to applications, no matter whether they are potentially conflicting with others or not
  - In contrast with Coda, where volumes are locked
  - Cascading conflict resolution is a potential drawback
- A data collection is fully replicated on a set of servers: access to any one of the servers is sufficient for a client to access data
  - Bayou does not require a client to access a specific server: updates are communicated in a peer-to-peer fashion
  - Amenable to MANETs

How Bayou Works

- After a client submits a `write` operation to a server, it has no further responsibility to it
  - In particular, it does not wait for its propagation to other servers
- Whenever a `write` operation is sent, a *dependency check* and a *merge procedure*, written by the programmer, are sent along as well
  - The dependency check contains a query and the expected result, and allows the receiving server to determine whether there is a conflict or not
  - The merge procedure is run by the server if a conflict is detected
- Updates are eventually propagated using an epidemic algorithm: as long as the set of servers is not permanently partitioned, the `write` will eventually reach all servers, and thus be committed
  - Because `write`s are performed in the same order at all sites, and conflict detection and merge procedures are deterministic
- Servers propagate writes among themselves during pairwise interactions (called anti-entropy), during which they bounce `write` operations until they agree on the set of `write`s they have seen and what is the order to perform them

SEER - Automatic Hoarding

- Heuristic based hoarding (a variant of caching)
- Comparison metric – miss-free hoard size
  - Minimum space needed by a hoarding system to guarantee that the user will have access to all his files during disconnection
- Complex heuristics
  - Combination not easily understood
  - Makes selection of optimal parameters difficult
    - Depends both on heuristics and on user use patterns

SEER Hoarding Parameters

oftenPercent	Controls which files are omitted from the semantic-distance calculation due to high usage. Such files are always included in the hoard. **Lower values cause more files to be considered high-usage.**
peekThreshold	Controls when a process is thought to be scanning the files in a directory, so that its accesses are not semantically meaningful. **Lower values increase the probability of interpreting process behavior as directory scanning.**
ageWeightFactor	Controls the probability of a file's relationship to another being replaced by a newer one in the relation list (which is of size "keepReferences," below). **Larger values increase the probability of the older relationship being kept for use in clustering.** Interacts with agingThreshold and the last four parameters listed here.
agingThreshold	Controls the probability of a file's relationship to another being replaced by a newer relationship. **Larger values increase the probability of the older relationship being kept for use in clustering**. Interacts with ageWeightFactor and the last four parameters listed here.
keepReferences	Controls the number of **inter-file relationships** kept for each file, which in turn has a complex effect on the clustering behavior
minimumReferences	Controls both overlapping and non-overlapping clustering. If fewer than this number of inter-file references are observed, they are not considered semantically meaningful (because of the probability of noise). **Smaller values cause fewer and larger clusters.**
sharedLooseThreshold	Controls overlapping clustering. If this number of references are shared between two files, their clusters are overlapped. **Smaller values cause more overlaps to be found in clusters**. Values greater than keepReferences will disable overlapping clusters.
sharedTightThreshold	Controls overlapping clustering. If this number of references are shared between two files, their clusters are merged. **Smaller values cause fewer and larger clusters** by making it more likely that clusters will merge with each other. Values greater than keepReferences will disable cluster merging.

SEER – Recent comparison

- Modified LRU, enhanced with "noise filtration" heuristics
  - Ignore "meaningless" programs, e.g., find
  - Force hoarding of critical files, e.g., shared libraries
  - Force hoarding necessary for system initialization
- Experiments compared against hand tuned selection of hoarding parameters
- Results: LRU surprisingly good!
  - 5/6 users achieved best performance with modified LRU
  - 6th user: only 0.81% worse than best parameters
  - Simple heuristics is a great improvement – especially considering movement to devices with less computational capability (e.g., palm rather than laptop)

Collaborative Work Replication

- Target: Mobile Collaborative Ad Hoc Groups
- Replication for availability and fault tolerance
  - Must ensure that the user accesses the most recent data version available in the group
  - Uses a conservative coherency protocol
    - Exclusive writer, distributed token management. Need token management in presence of disconnection/loss
    - Updates propagated when a member tries to access the data, save bandwidth to propagate unnecessarily. Only transmit to requester of data to "save energy"(?)
    - Updates also propagated lazily

Collaborative Work Replication (2)

- Profile based replication
  - Available energy, expected time in group, available local storage space
  - Combined to form "ad-hoc group profile". Used to control the rate of replication
- Work Replicas vs. Preventive Replicas
  - WR generated upon access demands to non-locally cached files. Lazily propagated to other group members. LRU replacement scheme
  - PR serve to maintain an up-to-date copy within the group
- Secure group management
  - Must know third party's public key for authentication
  - Group key exchanged for most interaction, changed when group membership changes

Replication Review

- Replication is performed in a mobile environment for BOTH accessibility and fault tolerance
- Some techniques are tailored to ad hoc environments, others to nomadic
- Most techniques use user profile either implicitly or explicitly
- Resource management is key to effective replication policies

Adaptation to Context

- What is context?
  - Location, proximate devices (and characteristics, e.g., energy), physical environment (e.g., noise level, bandwidth), history of environment
- Operating in a mobile environment means context is always changing, and many applications must adapt to these changes
- Approaches to Adaptation:
  - ***Application-transparent:*** adaptation is the responsibility of the system
    - Applications do not change, simplifies programming
    - Does not accommodate all situations, user must sometimes intervene
  - ***Application-aware:*** applications notified of changes in context, and expected to modify their own behavior

Adaptation (cont.)

- ***Long-term adaptation:*** the adaptation, extension, and replacement of the controlling middleware and application specific software
- ***Medium-term adaptation:*** sensitivity to context changes that occur relatively infrequently such as the appearance of new devices, location changes, etc.
- ***Short-term adaptation:*** adaptation to short timescale fluctuations such as changes in network conditions, device failure, etc.

Terminal Adaptation

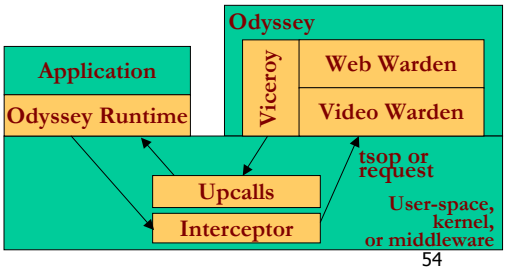
- Mobile computing exacerbates the problem of handling heterogeneity in a distributed system, since the characteristics of user terminals are extremely different
  - This is particularly evident for what concerns the GUI, e.g., display size, resolution, colors, modes of interaction
- A lot of research focused on providing some sort of terminal adaptation, defining languages and mechanisms that allow to reduce (or eliminate) the amount of rework needed
- Examples of related technologies (both are XML-based):
  - Cocoon, developed by the Apache Consortium, is a platform (relying on Java servlets) for Web content delivery that aims at keeping document content, style, and logic totally separated; the ability to process and transform XML files allows for on-the-fly adaptation
  - MoDaL is a language developed by IBM to describe user interfaces for palmtop devices; communication primitives are automatically translated into the corresponding operation of the TSpaces middleware

Odyssey

- Odyssey, the successor of Coda, supports application-aware adaptation
- Attempts to adjust the quality of data to match available resources, by
  - Defining an application-dependent notion of ***fidelity***
    - Consistency is "permanent" quality of fidelity
    - Fidelity is also data specific, e.g., video data fidelity includes frame rate and image quality; map fidelity includes minimum feature size and resolution
  - Providing an API that allows:
    - Applications and the system to talk about salient features of the environment
    - Mechanism that enables applications to track their environment
    - Mechanism through which applications request policy changes

How Odyssey Works

- The interceptor redirects file system operations to Odyssey
- The viceroy contains generic, type-independent functionality related to resource control, for which it is a single point of control
- Wardens contain application-specific adaptation policies
  - Notifications to applications are made through upcalls, and only when resource values fall out of a window of tolerance

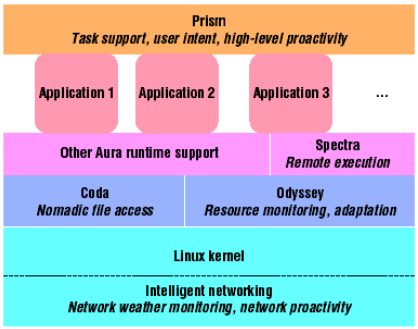


Aura

- Distraction-Free Pervasive computing
- Move computation and data as the user moves
  - Operations represented by tasks
  - Tasks can be accomplished by different services in different environments.
- Anticipate the movement of the user: accomplished with the "Prism" monitor
- Current components include:
  - Cyber-foraging – exploiting computation and storage of nearby devices
  - Bandwidth advisor – predict future bandwidth, advise user about where
  - WaveLAN-based people locator – not triangulation based, but instead uses bootstrap process to collect signal strengths

Aura Architecture

- Note inclusion of Coda (file replication support) and Odyssey (adaptation to context)
- Spectra
  - adaptive remote execution mechanism that uses context to decide how best to execute a remote call
  - e.g., decides where to do speech recognition
- Prism sits above everything and provides advice base on observations



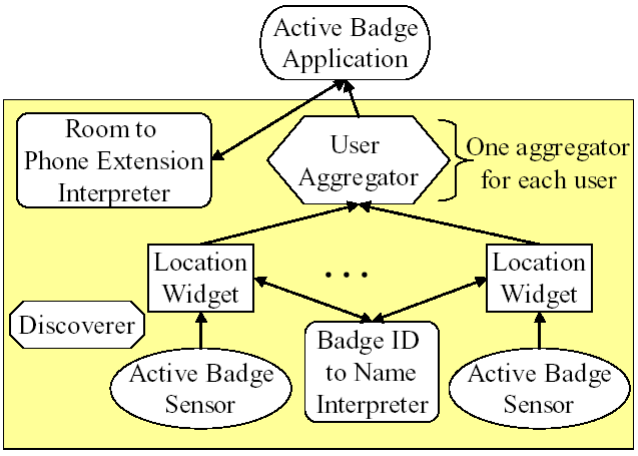
Aura – Sample Applications

- "Aura Vision" scenario
  - Working on presentation at the last minute
  - Details available in Aura paper
- Portable Help Desk
  - Find other users, display map of current area
  - BOTH visual and audio interface
- Idealink
  - Virtual collaboration environment, facilitates ad hoc collaboration
  - Shared whiteboard with replay/reorder of marks
  - User studies showed meetings more effective with Idealink than traditional whiteboard

Context Toolkit

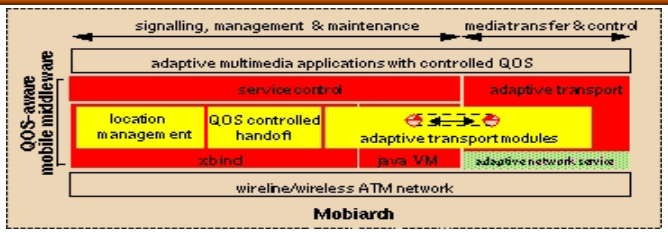
- Facilitate development of context aware applications
- Main components
  - Context widgets
    - Software components providing access to context information, e.g., location or activity
    - Hide details of context sensing
    - Wrap sensors, provide poll/subscription access
  - Context Aggregators (meta-widgets)
    - Hide more complexity of environment
  - Interpreters
    - Extract high level features
    - E.g., identity, location and sound level information can be interpreted to mean that a meeting is taking place
  - Services
    - Execute actions on behalf of applications
  - Discoverers
    - Track capabilities that currently exist

Context Toolkit (2)



To display a list of the building occupants, their location and the nearest phone extension, the application just has to subscribe to all location updates in each User Aggregator and convert the room location to a local phone extension.

Mobiware



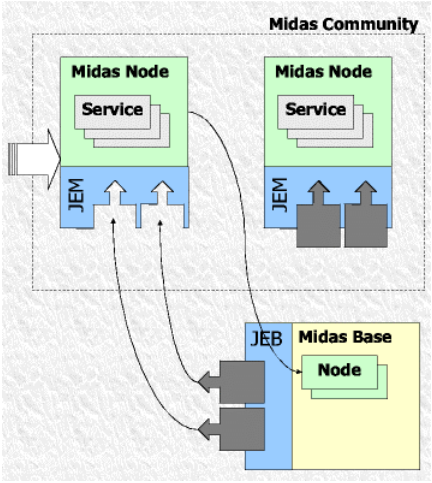
- Goal: provide programmable signaling, transport, and adaptive QoS management algorithms for mobility
  - Targets the transport, network, and data link layer (e.g., a programmable MAC is also provided)
- Handoff – seamlessly transfer multiple (ATM) streams during micro movement. Interacts with installation of filters and media scaling.
- Location Management – maintains mapping between logical name and location in network
- Active Transport modules – e.g., filters, error control, snooping
  - Exploits mobile code for "active networking"
  - Assumes network devices are programmable

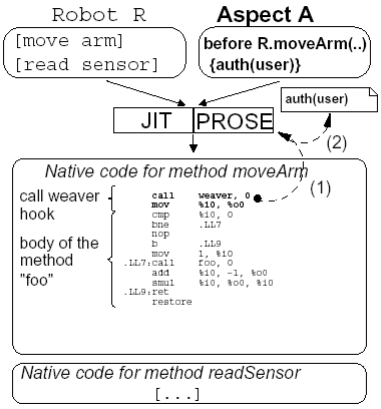
MIDAS – MIddleware for ADaptive Services

- Environment has proactive role in adaptation of applications
- Example: robot moving through space.
  - CS Department wants to log its movement.
  - ECE department wants to prevent it from entering some rooms
- Adaptation achieved with techniques from aspect oriented programming

  before methods-with-signature 'void *.send*(...,byte[] x,...)'
  do encrypt(x)
- Aspects loaded/unloaded dynamically as environment changes

MIDAS(2) - Runtime Adaptation





Adaptation Summary

- Mobility demands that programs be able to adapt to their environment
- Providing adaptability is application specific. Middleware either:
  - Allows applications to be notified of changes or
  - Tries to do the adaptation on behalf of the application