Software Architecture for Mobile Computing

Amy L. Murphy
Department of Computer Science
U. of Rochester, NY, USA
http://www.cs.rochester.edu/~murphy/

Who am I?

- University of Rochester, NY, USA
  - Assistant Professor (on leave)
  - Department of Computer Science
- Politecnico di Milano, Italy
  - Visiting researcher
  - Dip. Elettronica e Informazione
- Research Interests
  - Middleware for Mobile Computing
  - Middleware for Sensor Networks
  - Algorithm development for mobile environments

ASK QUESTIONS!!!

Objectives

- Provide an understanding the challenging issues of mobile environments
- Survey significant research efforts in enabling mobile software development, specifically middleware
- Understand one middleware (LIME) in depth, its features, programming environment, etc

Software Architecture???

Outline

- Introduction
- Two major research issues
  - Replication
  - Adaptation
- Approaches to Middleware Development
  - Proxies
  - Publish/Subscribe
  - Shared Memory
- Case Study – LIME and GVDS
- Summary and Open Questions

Distributed Systems

"One on which I cannot get any work done because some machine I have never heard of has crashed"

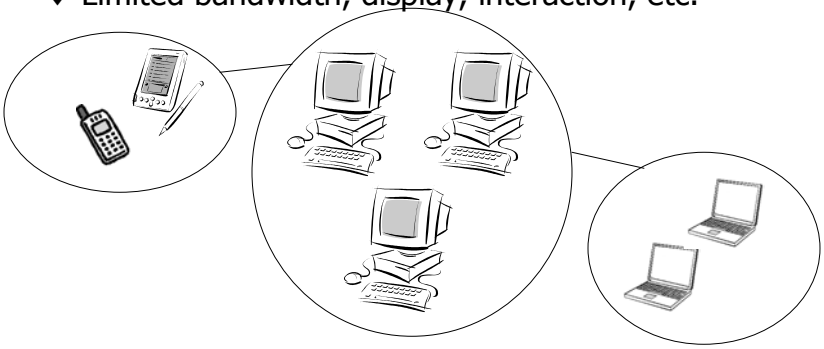
L. Lamport

Traditional Distributed Systems

- Fixed hosts, permanent connection, high bandwidth and stable links, static context
- Motivations and challenges for distribution:
  - Speed: parallelize computation
  - Scalability: accommodate more users
  - Economics: clusters cheaper than mainframes
  - Heterogeneity: different specialized components
  - Fault tolerance: improve management of hardware and software faults
  - Resource sharing: access control, authorization
  - Inherent distribution: e.g., games, mobility

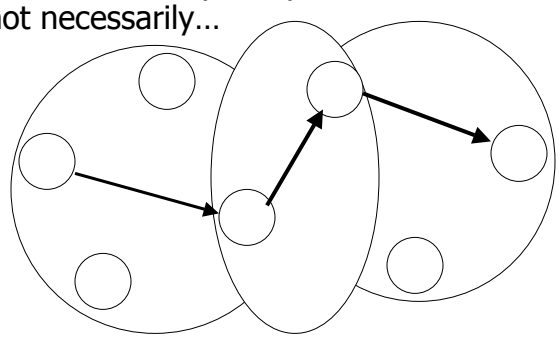
Nomadic Distributed Systems

- More mobile than traditional systems
- Core of fixed hosts
- Wireless base stations, e.g., bridges
- A set of mobile hosts roaming and accessing network from different locations
- Limited bandwidth, display, interaction, etc.



Mobile Ad-hoc Distributed Systems

- Pushing mobility to the extreme, remove infrastructure
- Mobile hosts, intermitted network, non stable links, dynamic environment
- Clusters formed dynamically
- Communication may be symmetric and transitive but not necessarily…



Middleware: Motivation

- Middleware sits between the operating system and the application
- Facilitate the development of distributed applications
- Provide developers with ***abstractions***, hiding details of distribution, enabling rapid, dependable development
- Typical features include communication primitives, replication, concurrency management, etc.

Why Middleware for Mobile?

- Mobile platform requirements are demanding
- Cannot assume stable connectivity
- Need support for handling a dynamically changing context
  - Cannot assume a high degree of coupling between the communicating parties
  - Cannot hide as much context information as before
- Want rapid, reliable application development

Commercial Mobile Middleware

- Beyond Windows "briefcase"
- Web search for "mobile middleware" reveals a wealth of information, research and commercial
- Straightforward, common solution is to exploit a proxy
- Specific systems to consider
  - WAP – Wireless Application Protocol
  - JMS – Java Messaging Service
  - Wireless Corba

Outline

- Introduction
- Two major research issues
  - Replication
  - Adaptation
- Approaches to Middleware Development
  - Proxies
  - Publish/Subscribe
  - Shared Memory
- Case Study – LIME and GVDS
- Summary and Open Questions

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

Key Replication Systems

CODA

- Network File System, caching for performance and accessibility (*disconnected operation*)
- Supports nomadic computing, only servers are trusted
- Provide *application transparent* file access
- Optimistic update policies, conflict resolution performed by users on demand
- CMU

Bayou

- Distributed database access
- Supports mobile ad hoc networks, and pairwise reconciliation
- Users provide dependency check and reconciliation policies
  - Eventual consistency provided with epidemic algorithm (anti-entropy)
- Sample applications: email, bibliography database, calendar
- Xerox PARC

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

Terminal Adaptation

- Mobile computing exacerbates the problem of handling heterogeneity in a distributed system, since the characteristics of user terminals are extremely different
  - e.g., GUI concerns, e.g., display size, resolution, colors, modes of interaction
- 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 (Apache Consortium) is a platform (relying on Java servlets) for Web content delivery that separates document content, style, and logic totally; transforms XML files 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

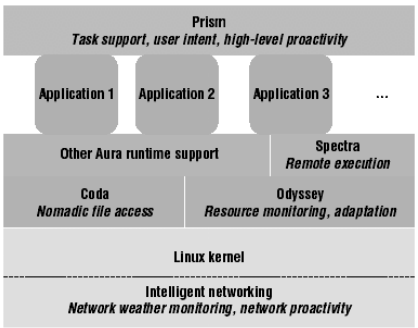
- Odyssey, the "successor" of Coda, supports application-aware adaptation
- Attempts to adjust the quality of data to match available resources, by
  - Defining an application-dependent notion of ***fidelity***
    - Consistency is "permanent" quality of fidelity
    - Fidelity is also data specific, 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

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



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

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

Outline

- Introduction
- Two major research issues
  - Replication
  - Adaptation
- Approaches to Middleware Development
  - Proxies
  - Publish/Subscribe
  - Shared Memory
- Case Study – LIME and GVDS
- Summary and Open Questions

Common Solution: Proxies

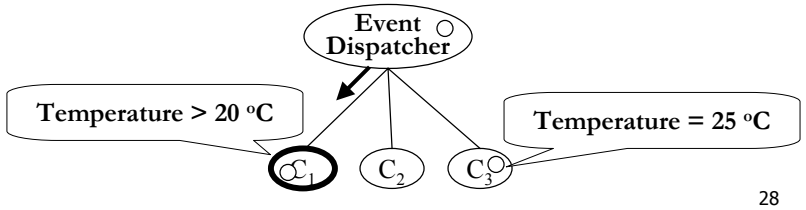
- A mobile client relies on the presence of a proxy on the fixed network, buffering client requests and server replies
  - Allows the client to disconnect, e.g., to save battery power, and gather the results upon the next reconnection
  - Example: Oracle Mobile Agent, and many others
  - Disconnection is made explicit to the end user, and it is assumed that the user can do useful work while disconnected
- Often, a thin client is exploited, essentially providing a remote, mobile user interface
  - Little or no computation takes place at the client
  - Example: InfoPad project

Systems Exploiting Proxies

- Wireless Corba
  - Provide access to mobile object
  - Provide access to static object by mobile clients
  - Proxy forwards messages to current location
- Rover
  - Relocatable dynamic objects, mobile objects moved for efficiency
  - Queued Remote Procedure Call, non-blocking communication
- Java Message Service
  - Reliable, flexible service for the exchange of information
  - Supports synchronous, asynchronous, and publish/subscribe communication paradigms
- WAP
  - Tailored to the design of Web pages that must be rendered on very small screens, without keyboards
  - Sites must be developed in WML, or translated by a server

Publish-Subscribe Events

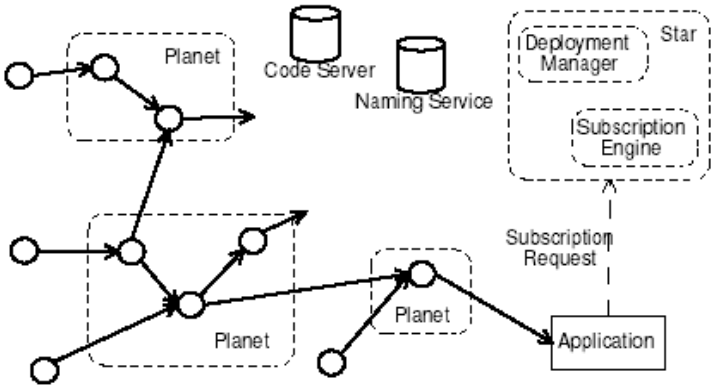
- Publish-Subscribe systems are asynchronous, implicit, multi-point, and peer-to-peer in communication style
- This style is suited to both traditional distributed systems and to mobile systems
- Clients and publishers are decoupled, and the infrastructure can be distributed



Solar

- Context-information collection, aggregation, and dissemination (data fusion)
  - Operator graph representation of computation allows decomposition and reuse of context aggregation primitives
  - Examples: filters, transformers (lookup mechanisms)
- Applications register operations with centralized server "Star" (the centralized dispatcher)
- Computation farmed to available hosts (planets)
  - Directly deliver events to applications

Solar Architecture



Jedi

- Provides a scalable, distributed, content-based event dispatcher
- Supports mobile agents that connect to a dispatcher
  - Clients disconnect
  - The old event dispatcher stores events for disconnected clients
  - When client reconnects, stored events are transferred and delivered
  - Partial order of events is guaranteed

Shared Memory

- Provide distributed shared memory development paradigm in the mobile environment
- MARS/Tucson do this for mobile agents
- KLAIM is a model for physical and logical mobility
- We will talk about GVDS:
  - Global Virtual Data Structures
  - …through a case study: LIME

Outline

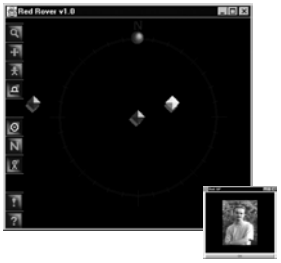
- Introduction
- Two major research issues
  - Replication
  - Adaptation
- Approaches to Middleware Development
  - Proxies
  - Publish/Subscribe
  - Shared Memory
- Case Study – LIME and GVDS
- Summary and Open Questions

Case Study: LIME and GVDS

- LIME
  - Applications
  - Model
  - Extensions
  - Summary
- LIME is an example of a Global Virtual Data Structure. Two other examples are
  - XMIDDLE
  - PeerWare

REDROVER: virtual games in physical space

- ***Distinguishing characteristic:*** An application where transient interactions among mobile users are central (similar to disaster recovery or robot environment discovery)
- Maintains a consistent view of the current system configuration: ***who else is around***
- Players request information on demand from specific connected players, as well as register interest for special data from *any* player



ROAMINGJIGSAW: a multi-player puzzle

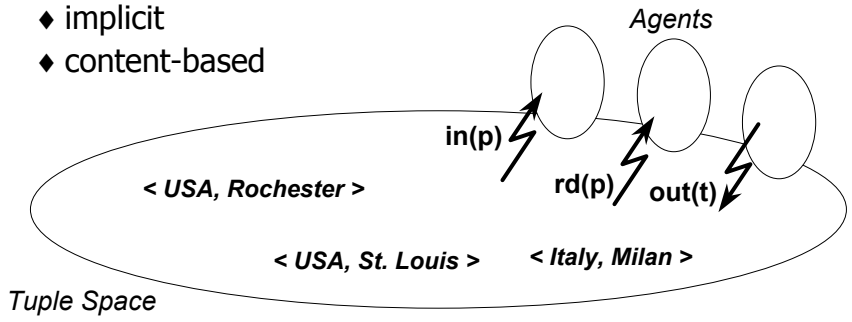
- ***Distinguishing characteristic:*** a mobile application where the limited availability of shared information due to mobility is central (similar to CSCW scenarios)
- Allows players to work while disconnected to assemble parts of the puzzle
- Maintains a ***weakly consistent*** view of global progress toward the overall puzzle solution

Enabling the Rapid Development of Mobile Applications

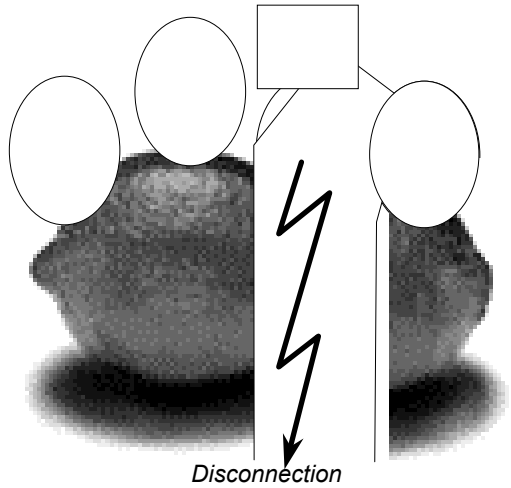
- Embody a ***conceptual model*** to facilitate the design of mobile applications
- Functional characteristics to consider
  - Disconnected operation
  - Context awareness (data and system)
  - Context transparency (data and system)
  - Reactive programming
- Provide ***coordination constructs*** to achieve rapid development of mobile applications through middleware

Linda

- Tuple-based model of coordination
- The tuple space is global and persistent
- Communication is
  - decoupled in time and space
  - implicit
  - content-based



LIME: Linda in a Mobile Environment

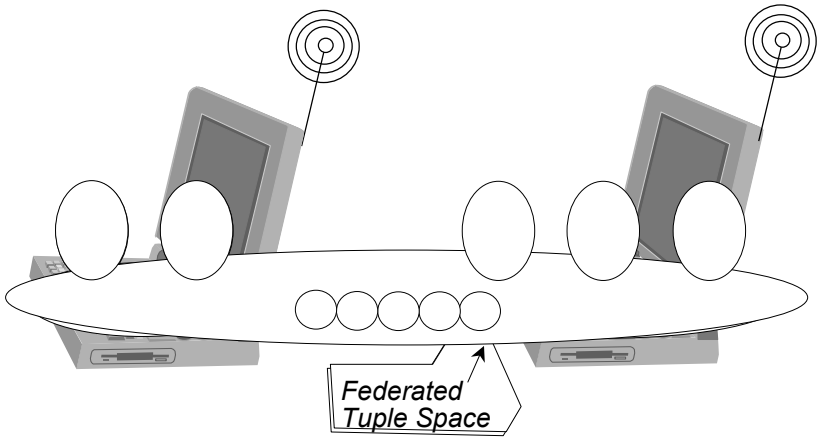


- Maintain simple DSM programming model
- LIME = Linda +
  - Transiently Shared Tuple Spaces
  - Tuple Location
  - Reactions
  - System Configuration Tuple Space
- Result: rapid application development

Transiently Shared Tuple Spaces

- Mobile agents are the only active components in the system and are permanently associated with an ***interface tuple space*** (ITS)
  - Mobile hosts are just "roaming containers" for mobile agents
- Through the ITS, the mobile agent perceive a context that may change dynamically
- The shared context, as determined by mobility, is determined through ***transient sharing*** of the ITSs
  - Mobility (agent migration and/or changes in connectivity) triggers ***engagement*** and ***disengagement*** of the tuple spaces, and dynamic reconfiguration of the contents perceived by each agent
- The ITS is accessed using Linda operations

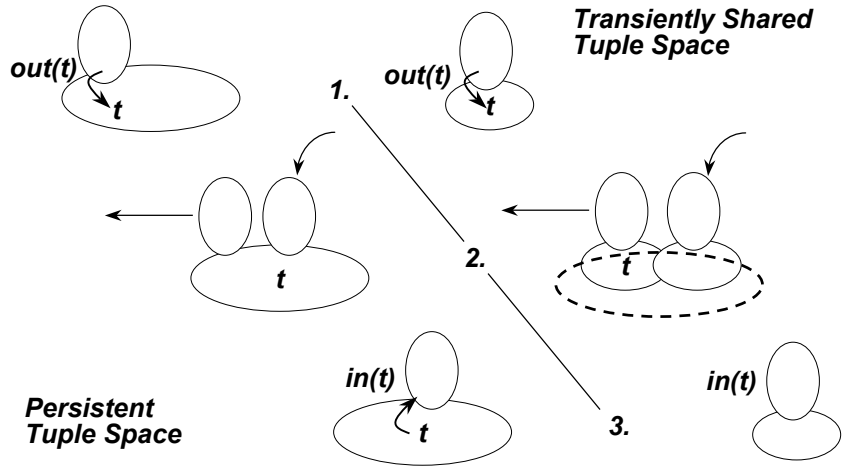
Context Transparency: Transiently Shared Tuple Spaces



Degrees of Context Awareness

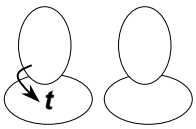
- Thus far, distribution and mobility are hidden in what is perceived as a local tuple space (the ITS)
  - Programming is simplified
- But, this view may hide too much from some applications which may need to:
  - limit the scope of query operations to a part of the context
  - output tuples that are meant to stay with a host different from the producer

Persistent vs. Transiently Shared Tuple Spaces



Binding Tuples to Locations

- A tuple's location is the ITS of an agent
- ***out***[λ](*t*)
  - the tuple *t* is inserted in the caller's ts
  - if λ is connected, *t* migrates to λ's ts; insertion and migration constitute a single atomic step
  - if λ is not connected, *t* stays in the caller's ts and is marked as ***"misplaced"***
- **in**[ω, λ](*p*) and **rd**[ω, λ](*p*)
  - The query for a matching tuple is restricted to a projection of the tuple space, namely to all the tuples whose current location is ω and destination is λ

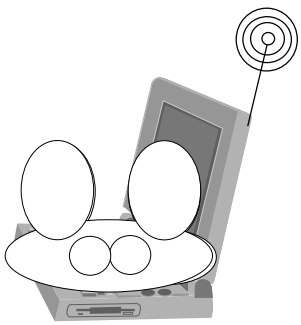


More on Tuple Location

- Upon insertion in a tuple space, a user tuple *t* is augmented with two fields, yielding a new tuple ⟨ *c,d,t* ⟩:
  - *c,* ***current***: the identifier of the agent whose tuple space is hosting the tuple
  - *d,* ***destination***: the identifier of the agent that is the *intended* recipient of the tuple
- If $c \neq d$, the tuple is "misplaced"
- This information is used during ITS engagement and disengagement

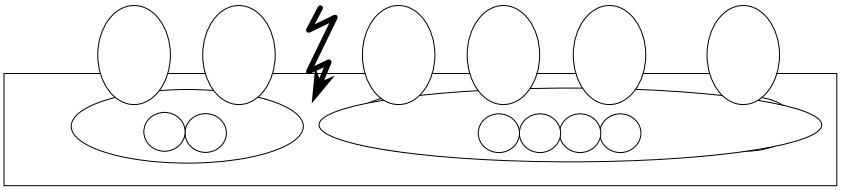
Tuple Space Engagement

- Engagement is triggered by the arrival of a new mobile unit (physical or logical)
  - The contents of the ITSs are merged
  - Misplaced tuples are migrated to destination
  - Engagement operations are perceived as a single, atomic step



Tuple Space Disengagement

- Disengagement also relies on tuple location
  - Transiently shared tuple space are separated as if each mobile agent were alone
  - Separate federated tuple spaces are computed based on the system configuration after disconnection
  - In practice, all the tuples are already with the right agent, and no tuple movement is necessary



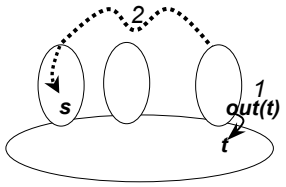
Awareness of System Configuration

- Details of the system configuration context remain partially hidden
  - If a probe **inp**[ω, λ](p) fails, it may be that ω is around and does not have tuples matching p, or that ω is not around
  - Only awareness of the ***data context*** is provided
- Many applications require knowledge of the context determined by the ***system configuration***
  - This is presented to the user in a read-only tuple space named `LimeSystem` is provided
  - The same abstraction is used to represent both data and system configuration context awareness

Reacting to Changes in Context

- Mobility is a highly dynamic environment, where reacting to changes is fundamental
- Linda provides a ***pull*** mechanism; with LIME we want to ***push*** data to applications:

reactsTo (s,p)



- ***Strong*** and ***weak*** reactions provide different atomicity guarantees

Strong Reactions

- Strong reactions derived directly from Mobile UNITY reactive statements
  - after each non-reactive statement, a reaction is selected non-deterministically and its guard evaluated
  - if the guard is true, the action is executed, otherwise the reaction is a skip
  - the process continues until there are no enabled reactions
- The state change and the corresponding action are tightly coupled
  - Implementing strong reactions in a distributed system involves a distributed transaction
  - Strong reactions are mostly exploited within a single host, typically to support logical mobility

Weak Reactions

- A much looser coupling is provided between the state change and the action *s*
  - The action *s* is guaranteed eventually to execute
  - Implementation does not require a distributed transaction
- Similar to event-based systems, or notification mechanisms for tuple spaces (e.g., TSpaces' `eventRegister`, or JavaSpaces' `notify`)
  - … but a LIME reaction is triggered by the state of the system, not by the occurrence of an event

Reacting to System Configuration

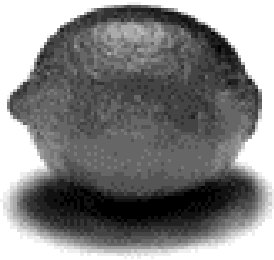
- System configuration is another component of mobile context
- Present ***"who is around"*** as a tuple space called LIMESYSTEM
  - Accessed with same primitives as data context
  - Read only by user, updated by system
- Augmented with system information, e.g., host configuration, link state (QoS)

The Making of LIME

- LIME is the result of a development process integrating formal modeling, implementation, and application development

Transiently Shared Tuple Spaces

Context Transparency
Tuple migration
Location transparent ops



Context Awareness
Tuple location
Location aware ops

Reactivity
Strong
Weak
ONCE/ONCEPERTUPLE

System Configuration Access
LIMESYSTEM tuple space

LimeTupleSpace API

```
public class LimeTupleSpace {
  public LimeTupleSpace(String name);
  public String getName();
  public boolean isOwner();
  public boolean setShared(boolean isShared);
  public static boolean setShared(LimeTupleSpace[] lts,
                                  boolean isShared);
  public boolean isShared();
  public void out(ITuple tuple);
  public void out(AgentLocation destination, ITuple tuple);
  public ITuple in(ITuple template);
  public ITuple in(Location current, AgentLocation destination,
                   ITuple template);
  public ITuple inp(Location current, AgentLocation destination,
                    ITuple template);
  public ITuple rd(ITuple template);
  public ITuple rd(Location current, AgentLocation destination,
                   ITuple template);
  public ITuple rdp(Location current, AgentLocation destination,
                    ITuple template);
  public RegisteredReaction[]
         addStrongReaction(LocalizedReaction[] reactions);
  public RegisteredReaction[] addWeakReaction(Reaction[] reactions);
  public void removeReaction(RegisteredReaction[] reactions);
  public RegisteredReaction[] getRegisteredReactions();
  public boolean isRegisteredReaction(RegisteredReaction reaction);
}
```

Basic Ops

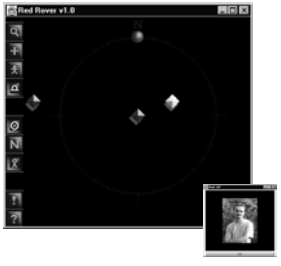
React-ion Ops

Reaction API

```
public abstract class Reaction {
  public final static short ONCE;
  public final static short ONCEPERTUPLE;
  public ITuple getTemplate();
  public ReactionListener getListener();
  public short getMode();
  public Location getCurrentLocation();
  public AgentLocation getDestinationLocation();
}
public class UbiquitousReaction extends Reaction {
  public UbiquitousReaction(ITuple template,
                            ReactionListener listener,
                            short mode);
}
public class LocalizedReaction extends Reaction {
  public LocalizedReaction(Location current,
                           AgentLocation destination,
                           ITuple template,
                           ReactionListener listener,
                           short mode);
}
public class RegisteredReaction extends Reaction {
  public String getTupleSpaceName();
  public AgentID getSubscriber();
  public boolean isWeakReaction();
}
public class ReactionEvent extends java.util.EventObject {
  public ITuple getEventTuple();
  public RegisteredReaction getReaction();
  public AgentID getSourceAgent();
}
public interface ReactionListener extends java.util.EventListener {
  public void reactsTo(ReactionEvent e);
}
```


REDROVER: virtual games in physical space

- ***Distinguishing characteristic:*** An application where transient interactions among mobile users are central (similar to disaster recovery or robot environment discovery)
- Maintains a consistent view of the current system configuration: ***who else is around***
- Players request information on demand from specific connected players, as well as register interest for special data from *any* player



Using LIME in REDROVER

- The ***reactive model*** employed varies according to the required consistency
  - Strong reactions definitively show who is connected
  - Weak reactions allow tracking of location with a reasonable threshold of accuracy
- The style of ***data access*** varies according to the type of data
  - Location-independent access for general data (e.g. flags)
  - Location-specific access or data whose source is known (e.g. player picture)

ROAMINGJIGSAW: a multi-player puzzle

- ***Distinguishing characteristic:*** a mobile application where the limited availability of shared information due to mobility is central (similar to CSCW scenarios)
- Allows players to work while disconnected to assemble parts of the puzzle
- Maintains a ***weakly consistent*** view of global progress toward the overall puzzle solution

Using LIME in ROAMINGJIGSAW

- Transient sharing of tuple spaces allows transparent access to the set of puzzle pieces that changes according to connectivity
- A *single* LIME weak reaction is sufficient to maintain weakly consistent view of the puzzle
  - while connected, updates are propagated
  - upon reconnection, disparate views are reconciled

Some Lessons Learned …

- ***"Most computation exploits Linda operations"***
  Instead, most of programming exploits reactions
- ***"The `LimeSystem` is nice, but not essential***
  Instead, the `LimeSystem` was key in developing REDROVER
- ***"We are forced to introduce weak reactions"***
  Weak reactions on the federated tuple space are an extremely powerful tool, and a good compromise between expressiveness and overhead

… and Some Reflections

- Is there a programming style induced by LIME?
  - Proactive vs. reactive programming
  - Tuple space as data repository vs. coordination mechanism
- What are the right atomicity constraints?
  - Do we need a separate notion of transaction?
- Are tuple spaces the right abstraction?
  - Other kinds of ***"global virtual data structures"*** may be useful as well
- Can the model be applied back to a wired setting?
  - Sharing abstractions for large-scale networks

Some LIME Extensions

- Tuple space code repository
  - Extend Java class loader to look into the tuple space
- Event distribution
  - Modeling pub/sub on top of tuple spaces
- Service provision
  - Providing service discovery in MANET
  - Service repository is a tuple space
  - Lookup is a query
- Secure tuple space sharing
  - Protect tuple spaces with passwords
  - Provide password protection for individual tuples
  - Reuse passwords to secure the communication

Future Directions of LIME

- ***Cache*** data for improved access, both during connection and when disconnected, extending data context
- Reduce reliance on announced disconnection, weakening the guarantees provided by the model, increasing ***fault tolerance***
  - Initial work on "Safe Distance"
- ***Agent-centered*** view of context
  - Instead of all agents seeing the same federated tuple space, build each agent's context independently
  - Incorporate location into context definition and queries, e.g., agent sees other agents within 1 mile radius or queries for data within a given radius

LIME: Summary

- LIME adapts the ***coordination*** primitives provided by Linda to the domain of physical and logical mobility
- Application programmers found it easy to think about mobility in terms of these abstractions
- LIME ***balances*** the ease of programming with the ability to control the environment
- LIME is the result of a development process integrating formal modeling, implementation, and application development

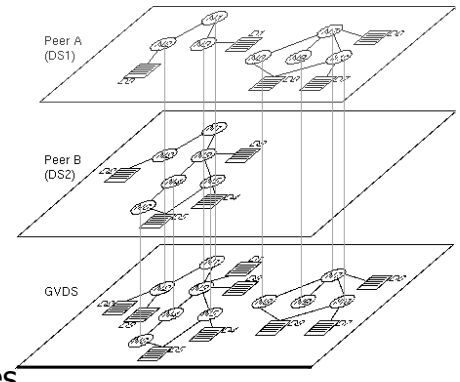
`http://lime.sourceforge.net`

Global Virtual Data Structures

- The notion of global virtual data structure (GVDS) lifts the previous assumptions of distributed shared memory:
  - ***the data structure is no longer indivisible:*** each of the coordinated agents is associated with a fragment of the global data structure
  - ***the data structure is no longer persistent:*** it is transiently and dynamically reconstructed by sharing the fragments contributed by the coordinated agents
  - ***the data structure is no longer globally available:*** only some of the coordinated agents (typically based on some notion of connectivity) are allowed to participate in the transient sharing of the data structure

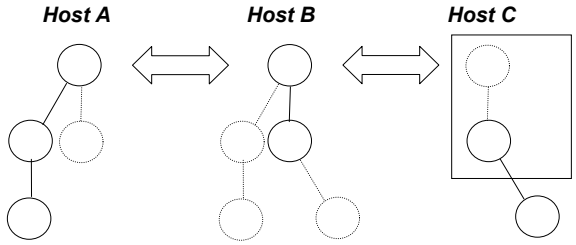
GVDS Incarnations – 2: PeerWare

- Based on trees
  - Nodes provide scoping
- No direct support for location-aware primitives
  - Delegated to traditional middleware, e.g., RMI
- Explicitly separates the local data structure from the GVDS
- Weak atomicity guarantees



GVDS Incarnations – 3: XMIDDLE

- Based on tree, but with a different partitioning
- The goal is to support offline computation:
  - Emphasis on pairwise communication rather than global access
  - Replication provides some degree of access to GVDS data in absence of connectivity



GVDS Assets

- In coordination models exploiting the notion of GVDS:
  - The association between the coordination context contributed by a given agent and the agent itself is now made explicit
  - The resulting style of coordination draws a distinction between the information immediately available to an agent and the one that can be requested from others
  - Still, the benefits of coordination, e.g., the decoupling of communication from behavior, are retained
- Hence, GVDS fosters a coordination style where:
  - coordination is defined entirely in terms of the coordinated agents, without reliance on some external entity
  - the coordination context is automatically and dynamically reconfigured
  - coordination is achieved through local actions that have a global effect
- The conjecture is that these characteristics are going to:
  - simplify the task of building (and reasoning about) applications that …
  - … are built out of autonomous components …
  - … whose relationships are dynamically and frequently reconfigured

Design Alternatives

- Choice of the data structure
  - Sets, bags, trees, graphs, matrices, …
  - May affect the efficiency and/or complexity of the implementation
- Choice of operations
  - Local vs. global
  - Query vs. manipulation
  - Proactive vs. reactive
  - Synchronous vs. asynchronous
- Choice of the partitioning/merging criteria
  - Superposition, union, composition, …
- Choice of the enabling condition for sharing
  - Based on connectivity
    - connectivity over space vs. connectivity over time for physical mobility
    - co-location for logical mobility
  - Possibly augmented by application constraints
    - e.g., to deal with security, or with specific application constraints

Design Alternatives – cont'd

- Degree of symmetry and transitivity
  - Is everybody "seeing" the same content?
- Degree of atomicity
  - Strikes in when determining the semantics of operations, and their relationship to sharing
  - Determines the extent to which one can treat the GVDS as a "local" data structure
  - Simplifying the programmer's chore vs. delivering an efficient implementation
- Degree of consistency
  - Given two agents, how far can their perception of the GVDS drift?
  - The answer to this question often implies the use of caching and replication schemes
- Degree of knowledge about the system configuration
  - System information can be represented in a GVDS, too
- Degree of persistency
  - If a portion of the system is known to be stable, how can we exploit it?

Research Issues

- What is the good balance to strike among the design alternatives?
  - Relationship with other middleware approaches and results
- Is there a "unifying theory" of GVDS?
  - Is it possible to separate the issues related with distribution from those intimately connected to the data structure chosen?
  - A positive answer could lead to a middleware supporting instantiations of GVDS with different data structures
- What is the relationship between GVDS and security?
- What is the impact of the GVDS abstraction on formal reasoning and verification?

GVDS Summary

- Global virtual data structures are a novel coordination paradigm targeted at highly dynamic environments
- GVDS is not meant to be a new model by itself: instead, it is meant to be the driving concept behind a new family of coordination models
- While some incarnations of GVDS are already available, only a fraction of the design space has been explored so far

Outline

- Introduction and Major Issues
- Two major research issues
  - Replication
  - Adaptation
- Approaches to Middleware Development
  - Proxies
  - Publish/Subscribe
  - Shared Memory
- Case Study – LIME and GVDS
- Summary and Open Questions

Summary

- Middleware for mobile computing provides abstractions for easing the development process
- Commercial middleware is targeted toward the first step of mobility, providing service access to mobile devices
- Major issues and approaches in research include
  - Replication, Adaptation, Service Discovery, Event-based, Object-Oriented, Transactional, Transport Layer, Algorithmic, Data Sharing
  - Not discussed issues include reflection, security, …

Questions:

- Will there be / should there be a single middleware for mobile computing?
  - Not all environments have the same demands and needs
  - Can there be a composable middleware that allows designers to pull in only the aspects that they need?
- Will the need for mobile middleware diminish as wireless networks become faster?
- Can middleware be shared among applications for:
  - Nomadic computing
  - Mobile ad hoc computing
  - Sensor networks

Questions?

References

- A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. 15th International Conference on Distributed Computing Systems. 1995.
- C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. citeseer.nj.nec.com/bettstetter00comparison.html
- M. Boulkenafed and V. Issarny. A Middleware Service for Mobile Ad Hoc Data Sharing, Enhancing Data Availability. ACM/IFIP/USENIX International Middleware Conference. Rio de Janeiro, Brazil. 16-20 June 2003.
- Braam, P. J. The Coda Distributed File System. Linux Journal, #50 June 1998.
- M.A. Butrico, H. Chang, A. Cocchi, N.H. Cohen, D.G. Shea, S.E. Smith. Gold Rush: Mobile Transaction Middleware with Java-Object Replication. Proceedings of the Third USENIX Conference on Object-Oriented Technologies (COOTS), Pp. 91--102, 1997
- A.T. Campbell, M.E. Kounavis, and R.R.-F. Liao, Programmable Mobile Networks, Computer Networks and ISDN Systems, Elsevier Science, Vol 31., No 7, 1999,
- Campbell A.T., Mobiware: QOS Aware Middleware for Mobile Multimedia Communications, 7th IFIP International Conference on High Performance Networking (HPN) White Plains, New York, April 1997.
- G. Chen and D. Kotz. Solar: An Open Platform for Context-Aware Mobile Applications. In Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), pages 41-47, Zurich, Switzerland, June, 2002. Short papers.
- Margaret H. Dunham, Abdelsalam Helal, Santosh Balakrishnan, A Mobile Transaction Model That Captures Both the Data and Movement Behavior. Mobile Networks and Applications. vol 2 no 2. pp. 149-162, 1997.
- M.R. Ebling, G.D.H. Hunt and H. Lei, Issues for Context Services for Pervasive Computing. In Proc. Workshop on Middleware for Mobile Computing, IFIP/ACM Middleware 2001.
- D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing IEEE Pervasive Computing, April-June 2002

References (2)

- M. Haahr, R. Cunningham and V. Cahill. Supporting CORBA Applications in a Mobile Environment. MobiCom '99: 5th International Conference on Mobile Computing and Networking. Seattle, August 1999.
- M. Haahr, R. Cunningham and V. Cahill. Towards a Generic Architecture for Mobile Object-Oriented Applications. SerP 2000: Workshop on Service Portability. San Francisco, December 2000.
- C. Intanagonwiwat, R. Govindan and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00), August 2000, Boston, Massachussetts.
- A.D. Joseph, J.A. Tauber, and M. Frans Kaashoek. Mobile Computing with the Rover Toolkit. IEEE Transactions on Computers: Special issue on Mobile Computing, 46(3). March 1997.
- A.D. Joseph, J.A. Tauber, and M.F. Kaashoek. Building Reliable Mobile-Aware Applications using the Rover Toolkit, in Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MobiCom'96). November 1996.
- G. H. Kuenning and G. J. Popek. Automated Hoarding for Mobile Computers. Proceedings of the 16th ACM Symposium on Operating Systems Principles, (SOSP-16) St. Malo, France, October 5-8, 1997.
- G.H. Kuenning, W. Ma, P. Reiher, and G.J. Popek Simplifying Automated Hoarding Methods. 5th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2002), Atlanta, GA, September, 2002.
- S. Maffeis. Communication Middleware for Mobile Applications - A Comparison, August 2001.
- C. Mascolo, L. Capra, and W. Emmerich. Middleware for Mobile Computing (A Survey). In Advanced Lectures in Networking. Editors E. Gregori, G. Anastasi, S. Basagni. Springer. LNCS 2497. 2002.
- C. Mascolo, L. Capra, S. Zachariadis and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. In Personal and Wireless Communications Journal 21(1), Kluwer. April 2002.
- A.L. Murphy, G.P. Picco, G.-C. Roman, Lime: A Middleware for Physical and Logical Mobility. in Proceedings of the International Conference on Distributed Computing Systems (ICDCS'01), Phoenix, AZ (USA), pp 524--533, April 2001.

References (3)

- A.L. Murphy, G.-C. Roman, G. Varghese, Tracking Mobile Units for Dependable Message Delivery. IEEE Transactions on Software Engineering (May 2002).
- B. Noble, M. Price, and M. Satyanarayanan. A Programming Interface for Application-Aware Adaptation in Mobile Computing. Proceedings of the Second USENIX Symposium on Mobile & Location-Independent Computing Apr. 1995, Ann Arbor, MI
- K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), Saint Malo, France, October 5-8, 1997, pages 288-301.
- G.P. Picco, A.L. Murphy, and G.-C. Roman, Lime: Linda Meets Mobility in Proceedings of the 21st International Conference on Software Engineering (ICSE 1999), Los Angeles, CA (USA), D. Garlan and J. Kramer, eds., May 1999, ACM Press, ISBN 1-58113-074-0, pp. 368-377.
- A. Popovici, A. Frei and G. Alonso. A Proactive Middleware Platform for Mobile Computing. ACM/IFIP/USENIX International Middleware Conference. Rio de Janeiro, Brazil. 16-20 June 2003.
- X. Qu, J.X. Yu, and R.P. Brent. A Mobile {TCP} Socket. TR-CS-97-08, Canberra 0200 ACT, Australia. 1997
- M. Satyanarayanan. Fundamental Challenges in Mobile Computing. Fifteenth ACM Symposium on Principles of Distributed Computing. May 1996, Philadelphia, PA
- M. Satyanarayanan. Coda: A Highly Available File System for a Distributed Workstation Environment. Proceedings of the Second IEEE Workshop on Workstation Operating Systems. Sep. 1989, Pacific Grove, CA
- M. Satyanarayanan. Coda: A Highly Available File System for a Distributed Workstation Environment Proceedings of the Second IEEE Workshop on Workstation Operating Systems Sep. 1989, Pacific Grove, CA
- J.P. Sousa and D. Garlan. From Computers Everywhere to Tasks Anywhere: The Aura Approach. Submitted for Publication