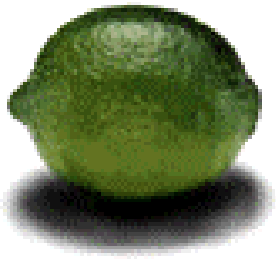
# The Making of LIME

- 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



# 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



# LimeTupleSpace API

**Basic Ops**

**React-ion Ops**

```
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);
}
```

# 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);
}
```



# 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**

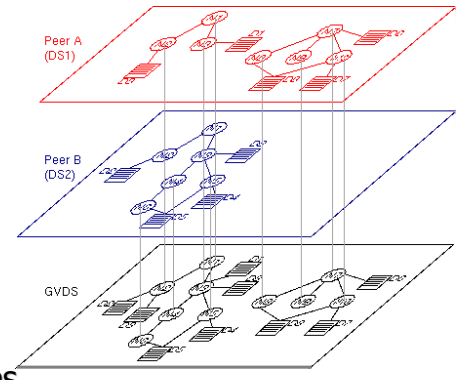
# Global Virtual Data Structures

- The notion of global virtual data structure (GVDS) lifts the previous assumptions:
  - ***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



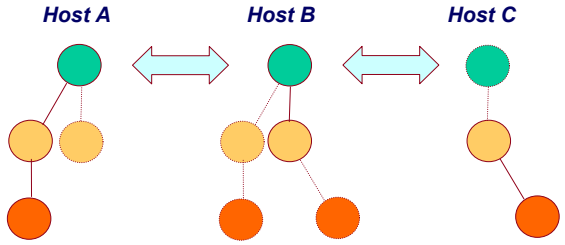
# GVDS Incarnations – 2: PeerWare

- 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