

# Dagstuhl Event Processing Workshop May 2007

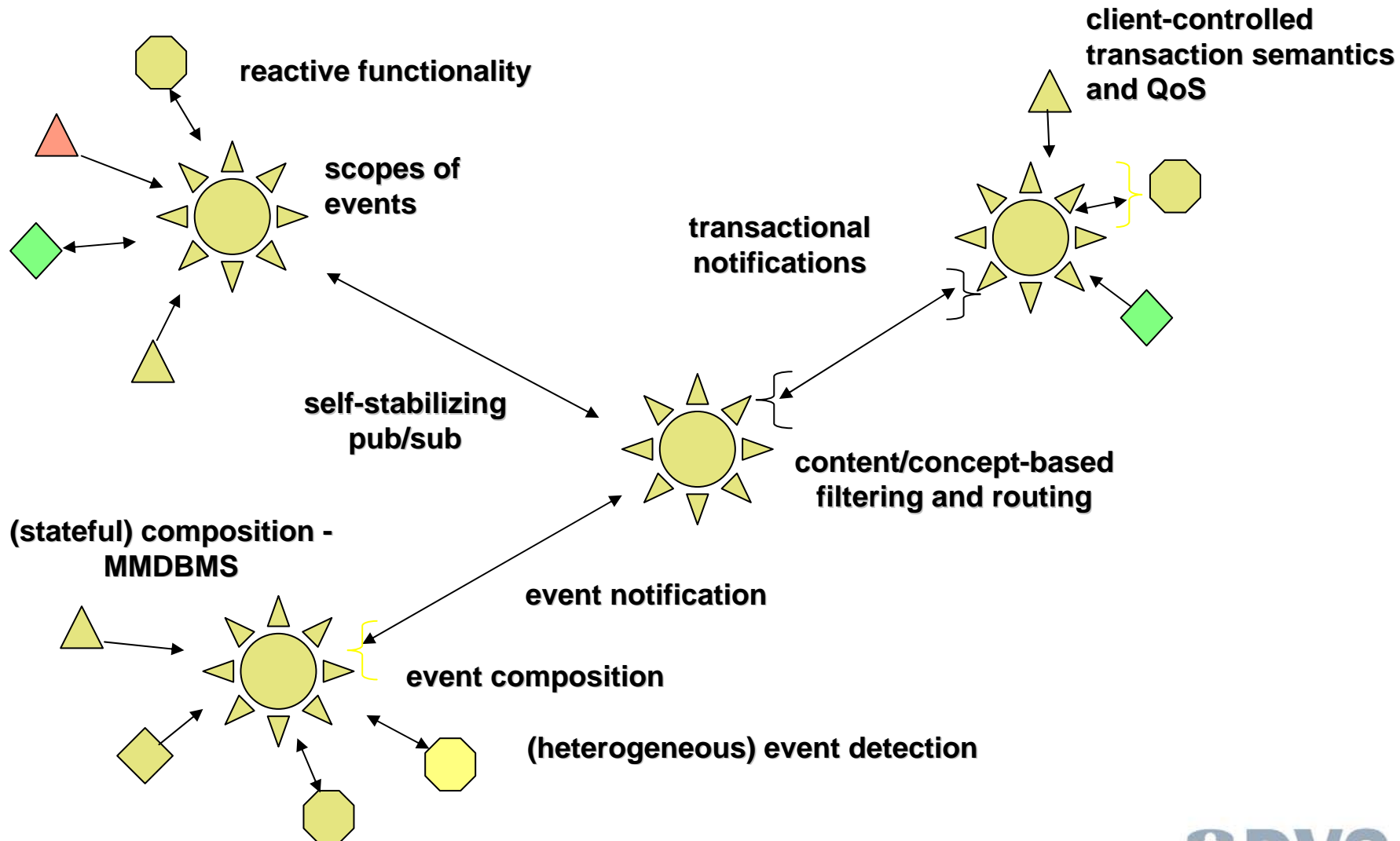
**Alex Buchmann**

*Databases and Distributed Systems Group  
Technische Universität Darmstadt, Germany*

# Selected papers

- **Publish/Subscribe Grows Up: Support for Management, Visibility Control & Heterogeneity**, Ludger Fiege, Mariano Cilia, Gero Mühl, Alejandro Buchmann, IEEE Internet Computing (Special Issue on Asynchronous Middleware and Services), Vol. 10, No. 1, IEEE Computer Society, January 2006
- **Towards Multi-Purpose Wireless Sensor Networks**, Jan Steffan, Ludger Fiege, Mariano Cilia, Alejandro Buchmann, International Conference on Sensor Networks (SENET'05), IEEE, Montreal, Canada, August 2005
- **Event Handling for the Universal Enterprise**, Mariano Cilia, Christof Bornhövd, Alejandro Buchmann, Information Technology and Management -- Special Issue on Universal Enterprise Integration, Vol. 5, No. 1, Kluwer Int. Press
- **Security Aspects in Publish/Subscribe Systems**, Ludger Fiege, Andreas Zeidler, Alejandro Buchmann, Roger Kilian-Kehr, Gero Mühl, Third Intl. Workshop on Distributed Event-based Systems (DEBS'04), May 2004
- **DREAM: Distributed Reliable Event-based Application Management**, Alejandro Buchmann, Christof Bornhövd, Mariano Cilia, Ludger Fiege, Felix Gärtner, Christoph Liebig, Matthias Meixner, Gero Mühl, Web Dynamics: Adapting to Change in Content, Size, Topology and Use, ISBN 3-540-40676-X, Springer, May 2004
- **Dealing with Heterogeneous Data in Pub/Sub Systems: The Concept-Based Approach**, Mariano Cilia, Mario Antollini, Christof Bornhövd, Alejandro Buchmann, International Workshop on Distributed Event-Based Systems (DEBS'04), Edinburgh, Scotland, May 2004
- **Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware**, Mariano Cilia, Ludger Fiege, Christian Haul, Andreas Zeidler, Alejandro Buchmann, Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03), ACM Press, San Diego, California, June 2003
- **Engineering Event-based Systems with Scopes**, Ludger Fiege, Mira Mezini, Gero Mühl, Alejandro P. Buchmann, Proceedings of the European Conference on Object-Oriented Programming (ECOOP'02), LNCS 2374, Springer-Verlag, Malaga, Spain, June 2002
- **Middleware Mediated Transactions**, Christoph Liebig, Stefan Tai, 3rd Intl. Symposium on Distributed Objects and Applications (DOA'01), IEEE Computer Society, Rome, Italy, September 2001
- **Event Composition in Time-dependent Distributed Systems**, Christoph Liebig, Mariano Cilia, Alejandro Buchmann, Proceedings of the 4th Intl. Conference on Cooperative Information Systems (CoopIS '99), IEEE Computer Society Press, Edinburgh, Scotland, September 1999





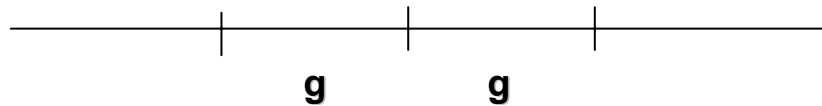
# Event detection and composition in distributed systems

- Events may be simple or composed
- Operators of event algebra depend on order
  - sequence, history, relative temporal events, ...
- Consumption modes also depend on order
  - chronicle, recent, sliding windows, cumulative
- Must accommodate uncertainty
- Middleware should not make false claims ==>  
let higher level resolve uncertainties  
(e.g. by using application semantics)

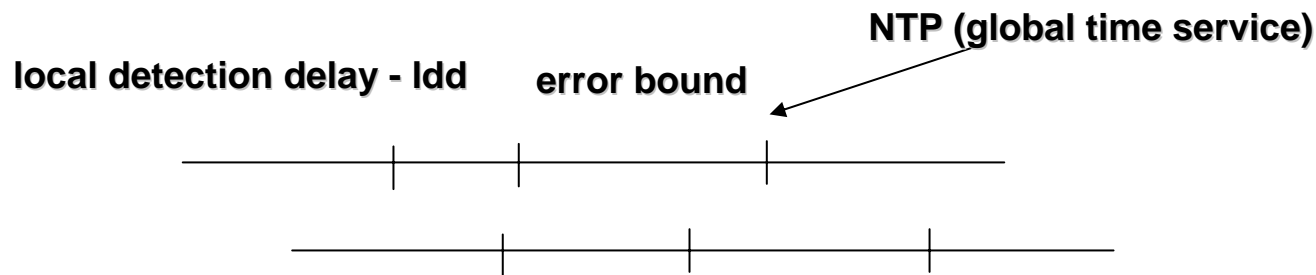


# Event detection and composition in distributed systems

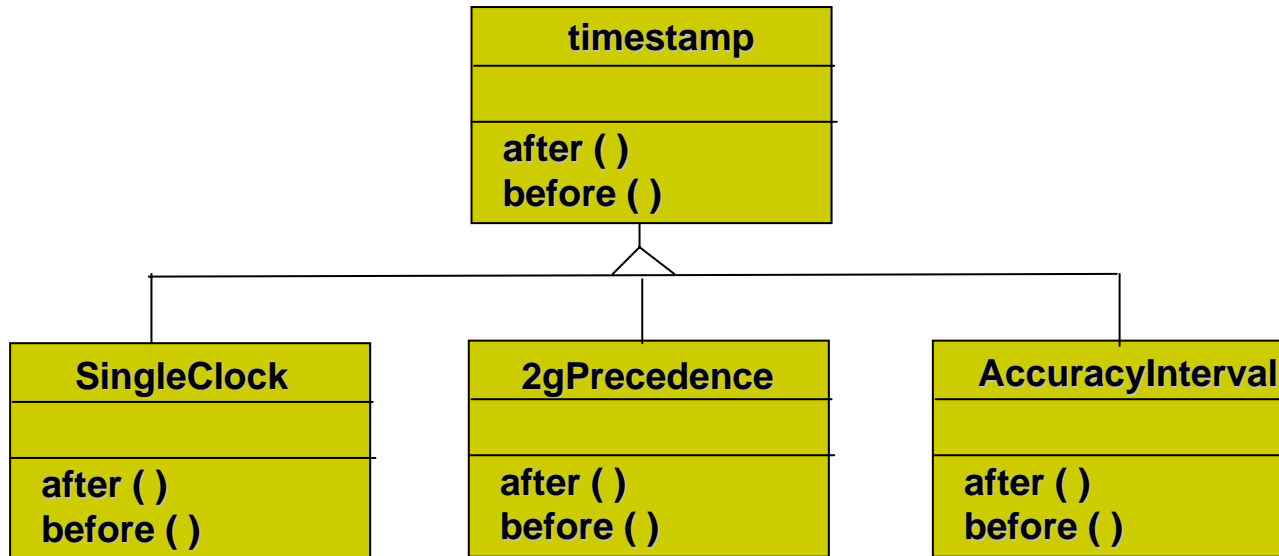
- 2g-precedence for closed networks and bounded imprecision



- Accuracy Interval Approach for timestamping events in large-scale, loosely coupled DS



# Variable Timestamp Representation



- Must provide interval-based time model with explicit (in)accuracy intervals
- Allen's interval semantics ==> indeterminacy
- Stable past vs. unstable past and present
- Middleware must expose indeterminacy, resolve through application semantics

# Problems derived from lack of control

- In a pub/sub system the producers and consumers of events are decoupled through the mediator
  - transactions must include the mediator
  - transactional behavior (if needed) must be controlled on consumer side
- Challenge: how to combine notifications with conventional object requests into middleware mediated transactions (MMT)



# Middleware Mediated Transactions

- MMTs extend atomicity sphere of transactional object requests to include mediators and/or final recipients of notifications and depend on:
  - topology: fixed vs. variable (1:1, 1:n, n:m)
  - binding: reference-based vs. mediator-based
  - life-cycle: time-independent (may not be available at same time) vs. time-dependent
  - synchronicity: blocking vs. non-blocking
  - delivery: reliable (at least once, exactly once) vs. unreliable (at most once, best effort)
  - processing: best effort or atomic transactional
  - recovery: forward vs. backward



# Example

- OO communication and transactions  
1:1, fixed topology, reference-based, time-dependent, synchronous, atomicity only
- (Traditional) Pub/sub  
n:m, variable topology, mediator-based, time-independent, asynchronous, transactional delivery to mediator but not to final recipients



# Coupling Modes in X<sup>2</sup>TS

<b>Visibility</b>	immediate, on commit, on abort, deferred
<b>Context</b>	non, shared, separate top
<b>Forward dependency</b>	none, commit, abort (transitive)
<b>Backward dependency</b>	none, vital, mark-rollback
<b>Production</b>	transactional, independent
<b>Consumption</b>	on delivery, on return, atomic, explicit

# Visibility

- Visibility determines when an event can be seen
  - immediate: as soon as they arrive at consumer and independent of outcome of triggering TX
  - on commit: event is only notified (delivered) when event-generating TX committed
  - on abort: event is only notified (delivered) when event-generating TX is aborted
  - deferred: consumer is notified as soon as producer starts commit processing



# Coupling Modes in X<sup>2</sup>TS

<b>Visibility</b>	<b>immediate, on commit, on abort, deferred</b>
<b>Context</b>	<b>non, shared, separate top</b>
<b>Forward dependency</b>	<b>none, commit, abort (transitive)</b>
<b>Backward dependency</b>	<b>none, vital, mark-rollback</b>
<b>Production</b>	<b>transactional, independent</b>
<b>Consumption</b>	<b>on delivery, on return, atomic, explicit</b>

# Transaction Context

- Transactional context determines whether a reaction executes in the same transaction as the event generator
  - none
  - shared: executes on behalf of triggering TX
  - separate: executes as its own separate top TX



# Coupling Modes in X<sup>2</sup>TS

<b>Visibility</b>	<b>immediate, on commit, on abort, deferred</b>
<b>Context</b>	<b>non, shared, separate top</b>
<b>Forward dependency</b>	<b>none, commit, abort (transitive)</b>
<b>Backward dependency</b>	<b>none, vital, mark-rollback</b>
<b>Production</b>	<b>transactional, independent</b>
<b>Consumption</b>	<b>on delivery, on return, atomic, explicit</b>



# Forward Dependency

- Forward dependency determines when a triggered reaction may commit
  - none: it will execute independently
  - commit: it executes only if triggering transaction commits
  - abort: it executes only if triggering transaction aborts



# Coupling Modes in X<sup>2</sup>TS

<b>Visibility</b>	<b>immediate, on commit, on abort, deferred</b>
<b>Context</b>	<b>non, shared, separate top</b>
<b>Forward dependency</b>	<b>none, commit, abort (transitive)</b>
<b>Backward dependency</b>	<b>none, vital, mark-rollback</b>
<b>Production</b>	<b>transactional, independent</b>
<b>Consumption</b>	<b>on delivery, on return, atomic, explicit</b>

# Backward Dependency

- Backward dependency constrains the commit of the triggering transaction
  - vital: the triggering transaction may only execute if triggered TX executed successfully
  - mark-rollback: triggered TX is independent of triggering TX but if triggering TX is marked-rollback it will abort if the event could not be delivered



# Coupling Modes in X<sup>2</sup>TS

<b>Visibility</b>	<b>immediate, on commit, on abort, deferred</b>
<b>Context</b>	<b>non, shared, separate top</b>
<b>Forward dependency</b>	<b>none, commit, abort (transitive)</b>
<b>Backward dependency</b>	<b>none, vital, mark-rollback</b>
<b>Production</b>	<b>transactional, independent</b>
<b>Consumption</b>	<b>on delivery, on return, atomic, explicit</b>

# Event Production

- Event production may occur in a transactional context that includes the notification to the mediator or it may occur independently



# Coupling Modes in X<sup>2</sup>TS

<b>Visibility</b>	<b>immediate, on commit, on abort, deferred</b>
<b>Context</b>	<b>non, shared, separate top</b>
<b>Forward dependency</b>	<b>none, commit, abort (transitive)</b>
<b>Backward dependency</b>	<b>none, vital, mark-rollback</b>
<b>Production</b>	<b>transactional, independent</b>
<b>Consumption</b>	<b>on delivery, on return, atomic, explicit</b>

# Event Consumption

- Event consumption determines when an event is considered to have been delivered.
- Once an event has been consumed the notification is considered to have been delivered and will not be replayed if the consumer crashes subsequently
  - on delivery: consumed by accepting notification
  - on return: consumed when returning from reaction
  - atomic: bound to consumer's atomicity sphere
  - explicit: by explicit action of consumer

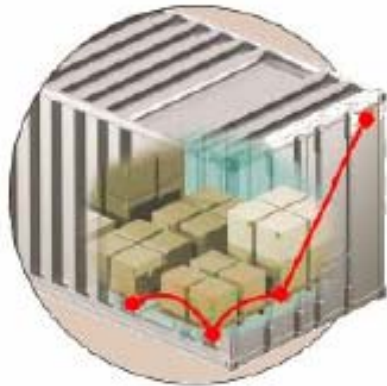


# X<sup>2</sup>TS: An Event-Driven Transaction and Notification Service

- Integration of notification service (NOS) and transaction service (OTS) for CORBA
- X<sup>2</sup>TS realises an event-action model with
  - transactional behavior
  - flexible visibilities of events
  - coupling modes and asynchronous reaction to events
- Prototype based on COTS pub/sub MOM, current version open source
- Quality of service determined by consumer rather than producer ==> support for multiple requirements
- XA-adapters



# The Need for Multipurpose WSNs



Example: Sensors in freight containers

- Monitor air conditions, velocity, tampering, RFID-tags, ...
- Span network between containers
- Used by cargo owners, shipping company, dock workers, customs,...



Challenges:

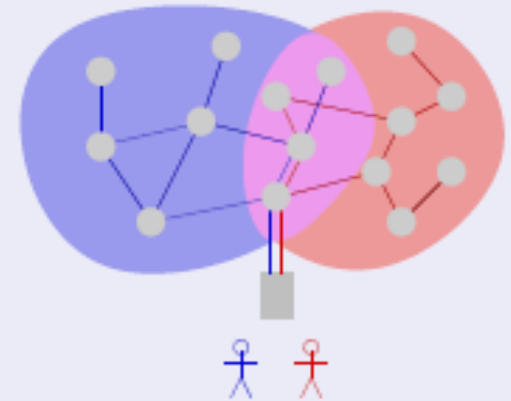
- Shared infrastructure
- Multiple independent tasks
- Deployed on subsets of nodes
- Frequent reconfiguration
- Appropriate access control

# Multi-Purpose Sensor Networks

What we want to achieve:

Basically:

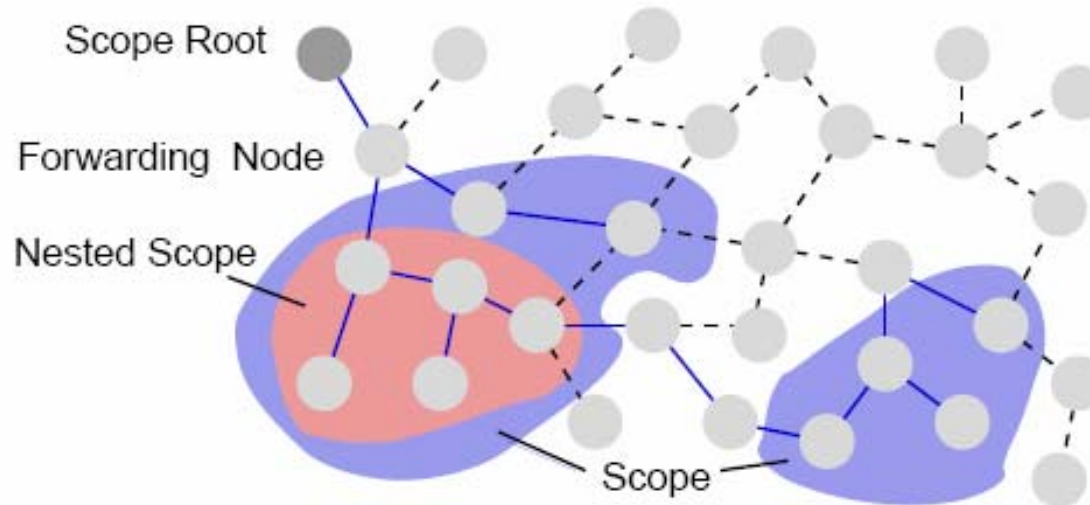
- Multitasking:  
Run multiple concurrent tasks
- Scoping:  
Restrict tasks to certain subsets of nodes



More precisely:

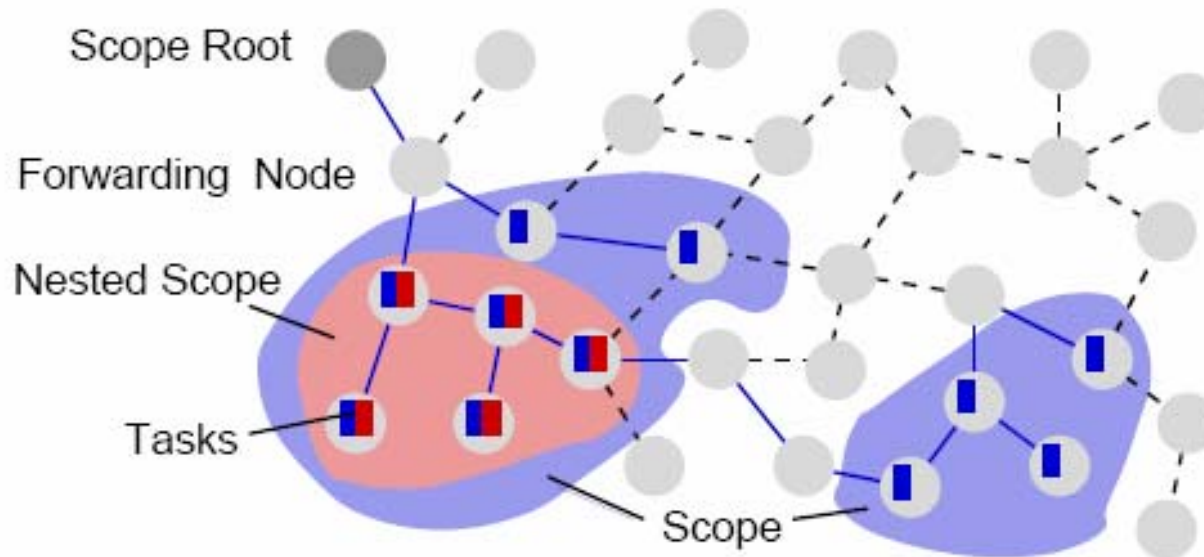
- Manage scopes independently of tasks
- Access control based on scopes
- Scope definition based on scenario specific criteria
- Flexibility with regard to kind of tasks

# Main Concepts of the Model: Scopes



- **Scopes** describe a subset of nodes matching a **membership condition**
- Created and managed through a **root node**
- Two-way communication **root node** ↔ **member nodes**
- **Nested** within each other

# Main Concepts of the Model: Tasks



- **Tasks** are pieces of **application logic** to be executed on nodes
- Created and managed **independently of scopes**
- Deployed to the **member nodes** of a scope

# Benefits of the Model

## Separation of scopes and tasks:

- Scopes can be used for multiple tasks  
(Costly discovery of member nodes only once)
- Allows scope and task management by different roles
- Independent implementations of scopes and tasks

## Nesting of scopes:

- Delimit costly flooding of scope creation requests
- Base for hierarchical access control

# Layered Architecture for Nodes

Application  
Layer

Scheduling of Tasks  
Life Cycle + Task Execution

Scope  
State

Scope–membership decision

Routing

Discovering potential scope members  
Maintaining path to members

Hardware  
Abstraction

OS: Network, Sensors, Timers, etc.

# Multitasking at the Application Layer

We use the SOS Operating System  
developed by the NESL group at UCLA  
<http://nesl.ee.ucla.edu/projects/sos/>



SOS provides:

- Loosely coupled modules  
(binaries compiled from pure C)
- Loading and unloading of modules over the air at run-time
- Scheduler, memory management,  
access to sensors, timers etc.



# Scope Specification and Membership

Each node has static or dynamic properties:

- Static: available sensor types, meta-data,...
- Dynamic: location, distance, number of neighbors,...
- Scenario-specific properties as modules

Scope membership condition:

- Boolean expression over node properties
- Represented in postfix notation as byte-code
- Evaluated locally on each potential member node

## Example (Static and Dynamic Conditions)

Static: `(meta.cargo = Bananas) AND (sensors.temp = true)`

Dynamic: `(sensors.tampering = true) AND (neighbors.count < 5)`

# Evaluation of Scope Membership Conditions

## Example (Expression in Postfix Notation)

```
((sensors.tampering TRUE =) (neighbors.count 5 <) AND)
```

- 1: **for all** *operators* in *expression* **do**
- 2:     fill in static properties
- 3:     simplify to boolean or dynamic expression
- 4: **end for**
- 5: **if** *expression = true* **then**
- 6:     *membership[scopeID]* ← *true*
- 7: **else if** *expression = false* **then**
- 8:     *membership[scopeID]* ← *false*
- 9: **else if** *expression* contains dynamic property **then**
- 10:     start event handler for each dynamic property
- 11:     update *membership[scopeID]* when event fires
- 12: **end if**

◀ ◻ ▶ ◀ ◻ ▶ ◀ ◻ ▶ ◀ ◻ ▶ ◀ ◻ ▶ ◀ ◻ ▶ ◀ ◻ ▶

## Example (Expression in Postfix Notation)

```
(neighbors.count 5 <)
```



# Architecture

