Interoperation

Interoperation

- What programming model is better suited for interaction between components of volatile systems?
  - Traditional systems: changes in the set of interoperating components are either long-term configuration issues or a runtime error condition to be handled occasionally
  - Try to avoid the lost opportunity problem
  - Global agreement between developers is necessary – minimise what needs to be agreed upon
  - Interface incompatibility problem
    - Heterogeneous interfaces with interface adaptation
      - Semantic interoperability is difficult and error prone
      - Scale for N interface NxN adaptors are needed
      - Acquisition and loading of adaptors?
    - Standardisation (e.g. UNIX pipes, Web – HTTP)

Data-oriented programming (1)

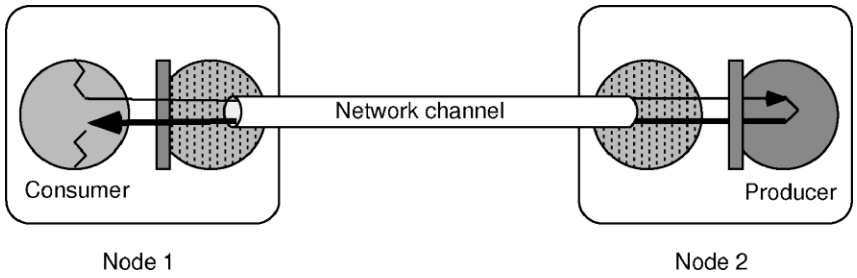
- Data/content oriented programs: programs using an unvarying service interface
  - In contrast to object-oriented
  - Trade flexibility against robustness
    - Compatibility checking restricted to type of data sent

Coupling/Decoupling in communication

- Space coupling: the extent to which interacting partners need to know each other
- Time coupling: the extent to which interacting partners need to actively participate in the interaction at the same time
- Synchronisation coupling: the extent to which interacting partners need to block while the interaction is taking place
  - Interaction in the main flow of control or not
- Removing coupling increases scalability and robustness against volatility

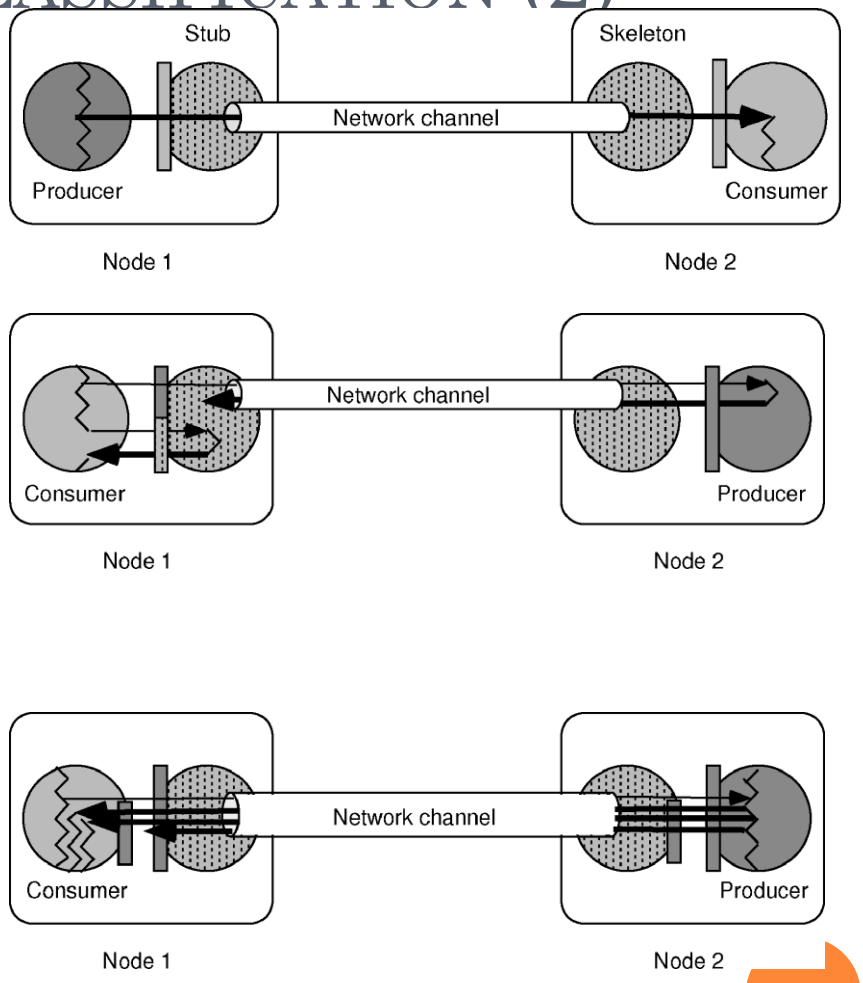
Interaction model classification (1)

- Message-passing
  - Need for explicit handling of physical addressing, data marshalling and flow control
  - Time and space coupling
  - Consumer synchronisation coupling
- RPC/RMI
  - Network transparency, but …
    - Hides data marshalling, physical addressing and flow control
  - Space, time and synchronisation coupling



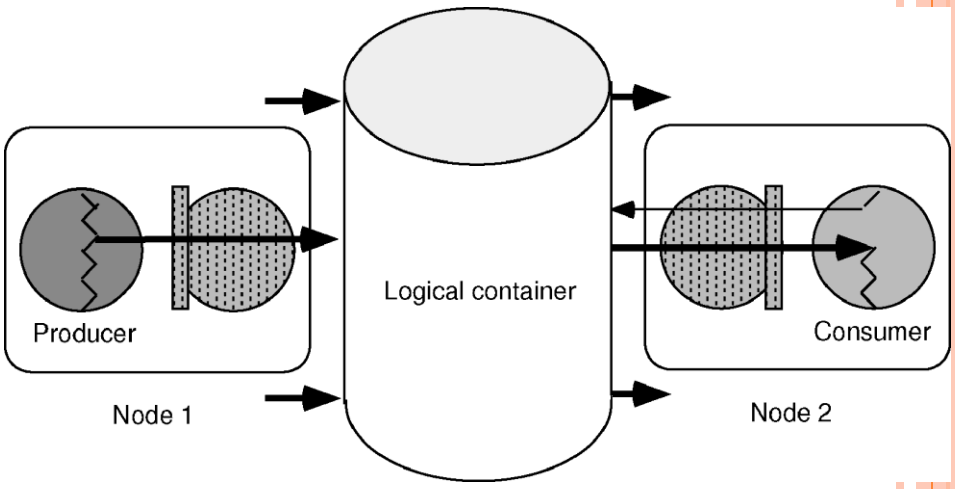
Interaction model classification (2)

- Oneway messages
  - No reply, no success/failure notification
- Futures
  - Reply decoupled from call
- Notifications
  - Use callbacks
  - Observer patterns
  - Time and space coupling
  - Synchronisation decoupling



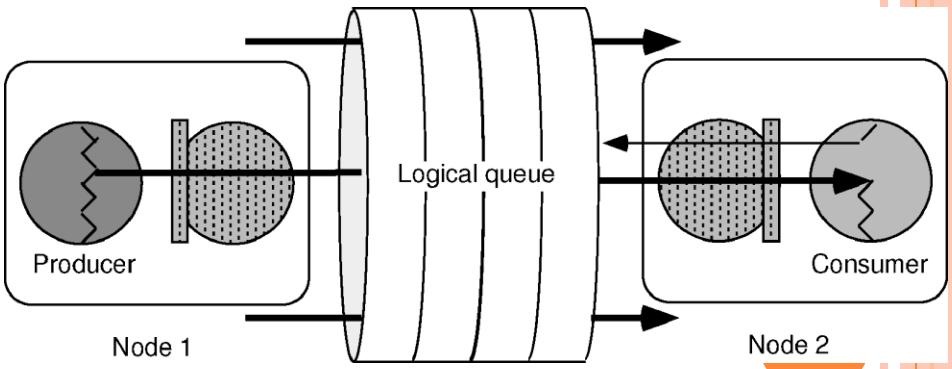
Interaction model classification (3)

- Shared spaces
  - Tuple space introduced by Linda
  - Communication through insertion/removal of ordered tuples
  - Operations: out (put in space), in (remove from space – 1 of n), read (read but not remove from space – 1 to n)
  - Time and space decoupling
  - Consumer synchronisation coupling
    - Addition of asynchronous notifications
- Rendezvous in I3: decouple sending and receiving of messages



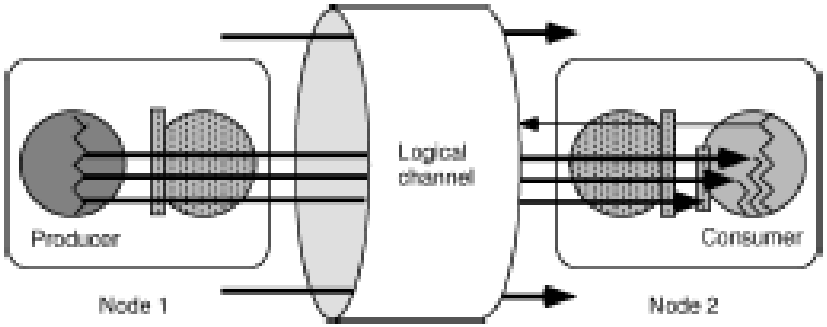
Interaction model classification (4)

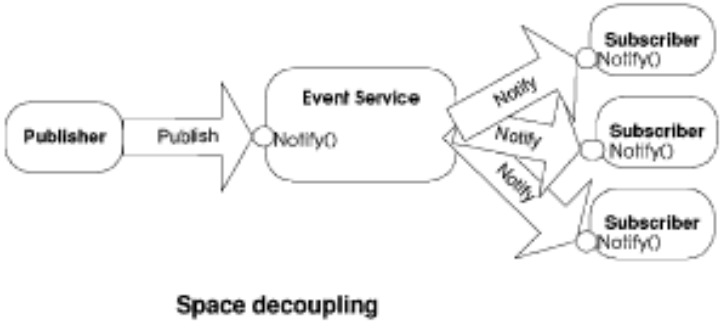
- Message queuing
  - MOM – message oriented middleware
  - Similar to a global tuple space with transactional, timing and ordering guarantees
  - Point-To-Point queuing with 1-of-n semantics
    - Similar to tuple space in but with FIFO retrieval
  - Time and space decoupling
- Consumer synchronisation coupling
  - Limited support for asynchronous message delivery
  - Scale limitations (additional interactions needed for providing the guarantees)



Interaction model classification (5)

- Publish/Subscribe
  - Time, space and synchronisation decoupling





Space decoupling

Time decoupling

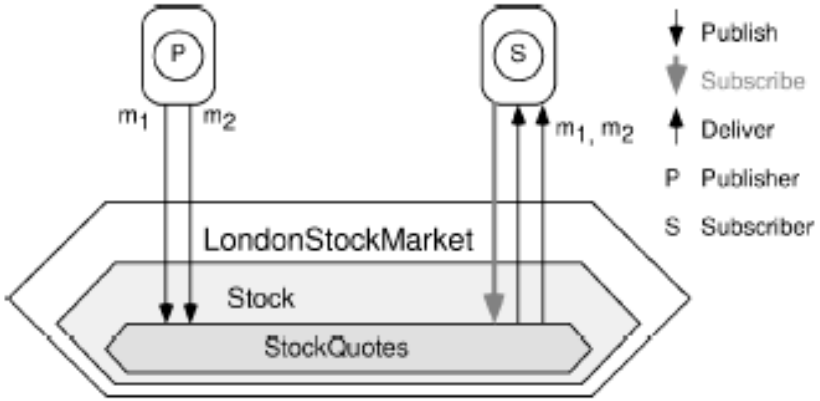
Synchronization decoupling

Interaction model classification (6)

- Push versus pull
- Periodic versus aperiodic
- Unicast versus multicast
- According to coupling

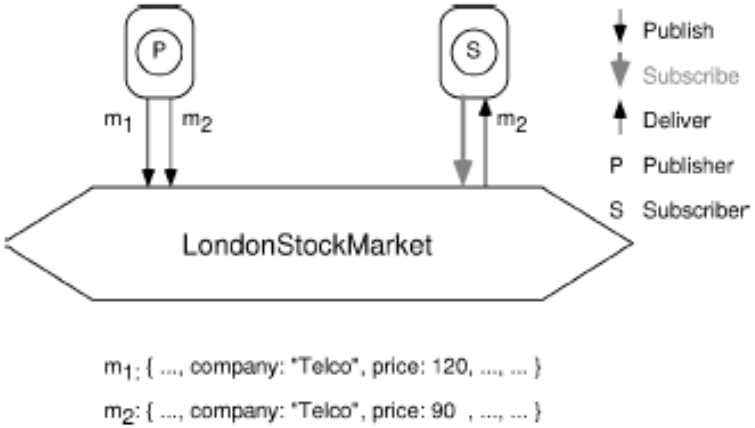
Publish/Subscribe (1)

- Topic-based
  - Message are published in topics or subjects
    - Identifying topics with keywords
  - Quite similar to group communication but with different aim (replication versus communication)
  - Topics may be organised in hierarchies
  - Systems may allow topic names with wildcards



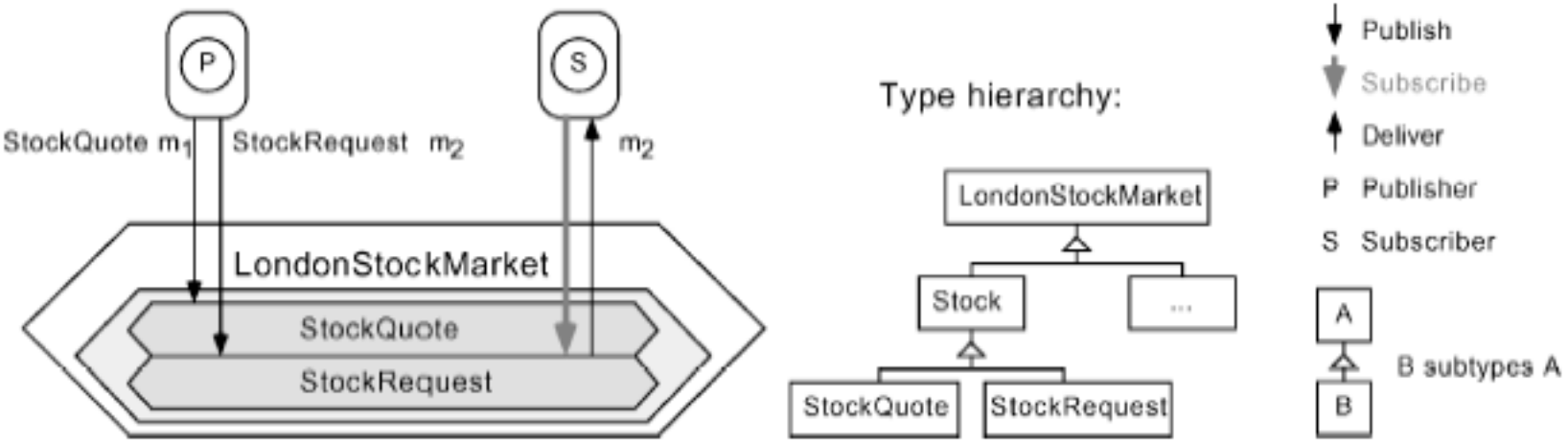
Publish/Subscribe (2)

- Content-based
  - Classifying events according to their actual content
    - Internal attributes or metadata
  - Event subscription with filters (name-value pairs of properties and basic comparison operators)
    - Complex subscription patterns by logical combination of filters
    - Possibly event correlation: combinations of events
    - Subscription patterns: string, template object, executable code



Publish/Subscribe (3)

- Type-based
  - Event matching on the basis of structure i.e. event types
  - Integration with programming language, type safety (may be checked statically)
  - Content filtering on the public attributes



Publish/Subscribe (4)

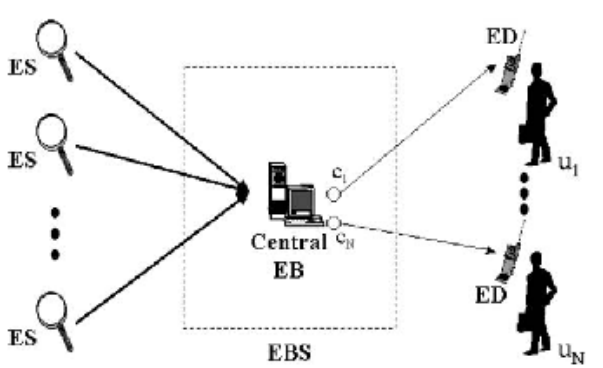
- Topic-based publish/subscribe is rather static and primitive, but can be implemented very efficiently
- Content-based publish/subscribe is highly expressive, but requires sophisticated protocols that have higher runtime overhead
- One should generally prefer a static scheme whenever a primary property ranges over a limited set of possible discrete values with additional expressiveness by content-based filters in the context of statically configured topics

Publish/Subscribe (5)

- Implementation issues
  - Events: messages or invocations
    - Message: header and payload,
    - Header: message identifier, issuer, priority, or expiration time – interpreted by the system or serve as information for the consumers
    - Payload: opaque byte array, or set of message types (text, XML), or self-describing messages
    - Invocations: typically one-way, values for invocation arguments and filtering
  - Media: architecture and dissemination
    - Architecture: centralised, decentralised, distributed network of servers (hierarchical, message broker graph)
    - Dissemination: unicast or multicast
      - Dissemination-based systems – selective event routing
  - Quality of service
    - Persistence (storage and replay), Priorities (in transit messages), Transactions (atomic delivery of groups of messages), Reliability (guaranteed delivery)

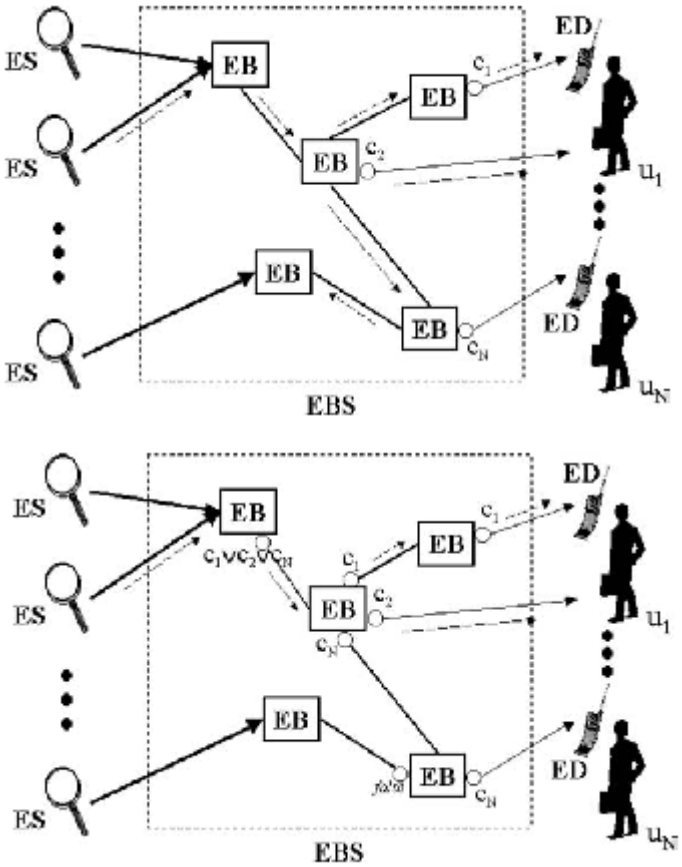
Publish/Subscribe for Mobility (1)

- Centralised architecture
  - Performance bottleneck and single point of failure
  - Mobile adaptation
    - Broker on a separate computer with fixed network connectivity
      - Allows for more resources, Autonomy from subscribers, Allows for storage and repeated transmission of events, Allows for continuous connectivity in the light of disconnection, Keep the system running in the light of network disconnection
    - ED disconnection: EB store and selective discard or aggregate
    - ES disconnection: ES store but selective discard impossible without knowledge of subscriptions
  - Ad-hoc networks
    - EB election and periodic re-subscription
- Quenching



Publish/Subscribe for Mobility (2)

- Distributed architecture
  - With broadcast
    - Subscription divided between EBs
    - EBs connected into a graph (e.g. hierarchical)
    - ES connect to the closest EB that is responsible to pass events to all others (broadcast or forwarding tree)
  - With multicast
    - Pruning of forwarding trees with selective forwarding



Publish/Subscribe for Mobility (3)

- Mobile adaptation
  - ED re-connection
    - Requires EB to update forwarding tree (request subscriptions) and forward stored messages
    - ED resubmits subscription but new EB contacts the old to cancel
      - Avoid duplicate delivery – keep log of connections and event ids
    - Base station handoff may be utilised
  - Ad-hoc networks
    - Wireless connectivity for event propagation
    - Building publish/subscribe trees (one for each EB)
- Replication
  - Consistency problems: duplicates, out of order events
  - Ordered-ness, Consistency, Completeness

Publish/Subscribe for Mobility (4)

- Broadcasting in MANETs
  - Broadcast storm problem: redundant broadcasts, broadcast contention, broadcast collisions
  - Solutions: reduce the possibility of redundant broadcasts and differentiating the timing of rebroadcasts
  - Probabilistic: broadcast with a certain probability, introduce small random delay before rebroadcast
  - Counter-based: wait for a small random period before rebroadcast but count the number of times a message is received and do not rebroadcast beyond a certain threshold

Publish/Subscribe for Mobility (5)

- Distance-based: when receiving a message check the distance of the broadcaster and if below a threshold then do not re-broadcast, otherwise follow above but keep note of receiving it again
  - Distance from signal strength
  - Distance threshold = signal strength threshold
- Location-based: similar as above but keep track of area of coverage being is below a threshold
  - Location from GPS
  - Difficulty in calculating coverage areas, but there are efficient approximations using convex polygons

Publish/Subscribe for Mobility (6)

- Cluster-based: organise nodes into clusters and only cluster heads or gateways rebroadcast using one of the above schemes
  - Clusters formed on basis of unique node ids and broadcast
  - Lowest id is cluster head, nodes that can see multiple cluster heads are gateways
- Counter-based schemes are good when node density is high
- Location-based schemes work best

Publish/Subscribe for Mobility (7)

- Beside physical mobility there could also be logical mobility, i.e. location dependent subscriptions
- Supporting logical mobility with location-dependent filters
  - Over "normal" publish/subscribe – keep track of location and un-subscribe/subscribe when there is a change, but suffers from blackout periods
    - Flooding can remove blackout periods but is very inefficient
  - Over "normal" publish/subscribe – using a movement graph to put predictive subscriptions for possible future locations and virtual counterpart broker to accommodate physical movements
    - Adapt the scope of locations considered according to movement speed for improved performance

Data-oriented programming (2)

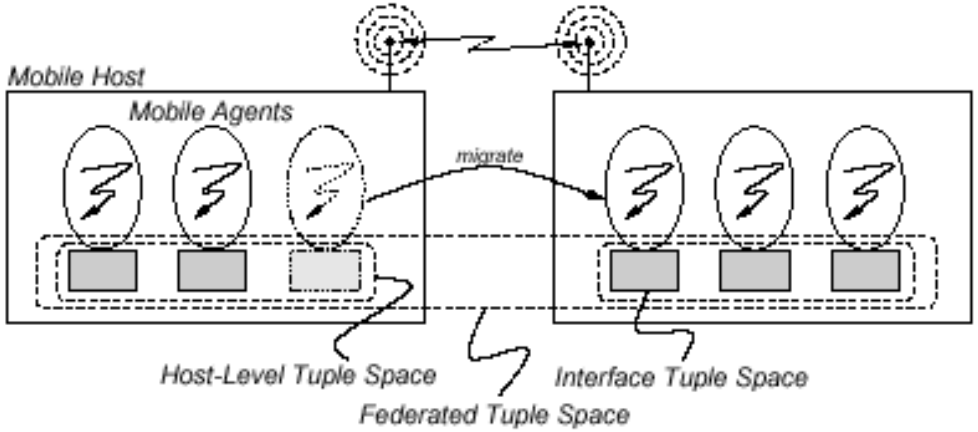
- Events systems
  - Fixed generic interface for publishers to publish structured data (events) and subscribers to receive events
  - Physical or logical scope for event delivery
  - Natural paradigm for announcing and handling changes in volatile systems
  - Composite events
  - Requires agreement on the event service and the syntax and semantics of events
  - Indirect association
  - Event service scoping

Data-oriented programming (3)

- Tuple spaces
  - Fixed generic interface to add and retrieve structured data (tuples) from a tuple space
    - Agreement about tuple structure and values
  - Matching with wildcards (*, ?)
  - Systems
    - Event heap per iRoom
    - LIME (no infrastructure)
      - Maintenance of consistency with unrealistic assumptions (serialised and orderly connections and disconnections, uniform multicast connectivity during tuple space aggregation)

Tuple Spaces (1)

- Basic Linda model
  - out(t), in(p), rd(p) and eval(dynamic process creation and deferred evaluation of tuple fields)
  - Extension with probes: inp(p), rdp(p)
- Lime (Linda In a Mobile Environment)
  - Interface Tuple Space engagement/ disengagement into Federated Tuple Space
  - Private tuple spaces



Tuple Spaces (2)

- Context aware operators
  - out[l](t): (1) out(t) in ITS w, (2) when l engages, then move t from w to l
  - in[w, l] & rd[w, l]
  - LimeSystem tuple space
  - Reactions R(s, p): execute code s when a tuple matching p is found
    - At every operation try all reactions in an arbitrary order – fixed point execution
    - Strong reactions R[w, l](s, p) (only on host or agent to ensure atomicity and serialisation)
    - Weak reactions for the whole tuple space – eventual execution
    - Once or once per tuple reactions

Tuple Spaces (3)

- MARS (Mobile Agent Reactive Spaces)
  - Agent coordination models
    - Client-Server Model (time and space coupling)
    - Meeting-Oriented Model (time coupling)
    - Blackboard-based Model (space coupling)
    - Linda-like model (time and space decoupling)
      - Each host has a base level tuple space and an associated meta-level tuple space
  - Programmable Tuple Spaces: tuple space access events trigger certain computational activities
    - Tuples are Java objects whose instance variables represent tuple fields, each tuple field is a reference to an object

Tuple Spaces (4)

- Operations: write (out), read, take (in), readAll and takeAll
  - Transaction and timeout parameters, lease parameters for write
- Reactions (Rct, T, Op, I): Rct operation invoked when agent I carries out operation Op on tuple matching T – meta-level tuple space
  - Reaction execution in order of insertion
  - Pipeline of matching reactions for each matching tuple
  - Access to base tuple space does not issue reactions
- Security model
  - Agent roles: Reader (only read), Writer (read, write, only take its own), Manager (all operation including the meta-level tuple space)

Tuple Spaces (5)

- TeenyLIME: a wireless sensor network middleware
  - Each node has a tuple space and can also see the tuple spaces of all nodes one-hop away
  - Operations: out, in, rd, rdg (rd all), ing (in all) – all future-like invocations (tupleReady event)
  - Sensed data and actuated commands as tuples
  - tupleReady can also trigger reactions
  - Flag to indicate reliable or best-effort operations
  - System data also take the form of tuples
  - Custom matching semantics on a per-field basis for data filtering
  - Time divided into epochs attached to tuples (freshness as a parameter in matching, check the epochs passed since tuple creation, tuple expiry of a number of epochs)
  - Capability tuples to reduce unnecessary data tuple creation
  - Remote reaction managed with soft state
  - Matching of capability tuples with reactions
  - Reliable operations with retransmission and acknowledgements piggy-backed on application messages

Tuple Spaces (6)

- Evolving tuples (a tuple space for pervasive computing)
  - Decoupling fields from their order by using names for the fields
  - Associate formulas to fields with access to tuple context (evolution context tuple – externally provided information of the current environment)
    - Formula: names of sibling fields, names of evolution context fields, executable behaviour (common arithmetic and boolean operators and if else statement)
  - Evolution performed by evolve operations in a specific evaluation order that produces new tuples

Tuple Spaces (7)

- Geo-Linda: geometry aware distributed tuple space
  - Geometric addressing: associate a volume (sphere, cone, cylinder, box, sector and point) to each tuple and a volume to each read operation
    - Matching requires intersection of volumes that requires precise location information
  - Detecting movement patterns
    - out (s, t), drop(t), read(s, p), take(s, p), readOnce(s, p), lostOne(s, p)

Data-oriented programming (4)

- Event – Tuple, Specification of interest – Tuple matching template
  - Space decoupling, Time(?)
    - Tuples are persistent (may grow uncontrollably, but expiration after activity time span)
  - Synchronisation decoupling versus coupling

Data-oriented programming (5)

- Direct device interoperation
  - JetSend
    - No need for special drivers
    - Synchronize: central, generic operation – transferring state in a negotiated format
      - Only supports simple data transfer
      - More complex interactions supported with user input through target device interfaces rendered on source device
        - Web like interoperation
  - Speakeasy
    - Similar functionality utilising mobile code
      - Devices can send user interface to PDAs (input validation, interactivity) – security and resource implications
      - Data transfer optimisation (e.g. Compression)

Indirect associations and soft state

- Explicit association only makes sense for well resources services
- It is useful for programmers to know which services are highly available and which are volatile
  - Space decoupling helps
  - Intentional Name System: requests specify the attributes of the required service, the operation to be invoked and its parameters, not the name or address of an instance of it
    - Automatic routing
    - Assumes stateless servers!
- Maintaining state in volatile systems
  - Replication – yes, but assumes redundant resources and extra communication
  - Algorithms that rely on access to persistent store
  - Soft state: data that provides a hint and is automatically updated by the sources of soft state
    - Discovery services: registrations as hints with automatic updates through multicast
- XML? – Semantic Web!