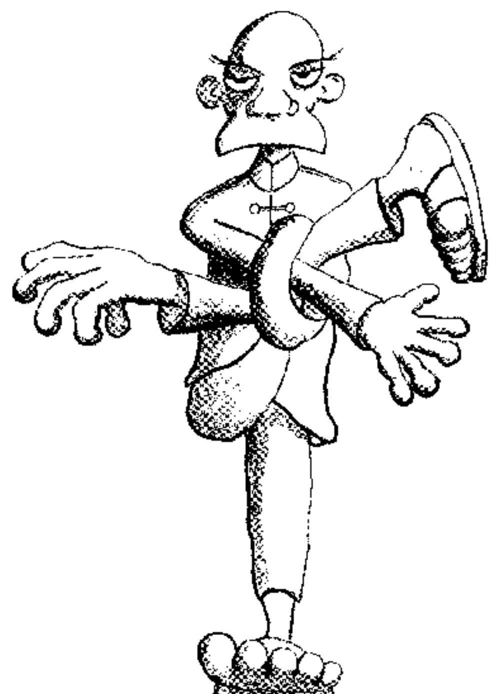
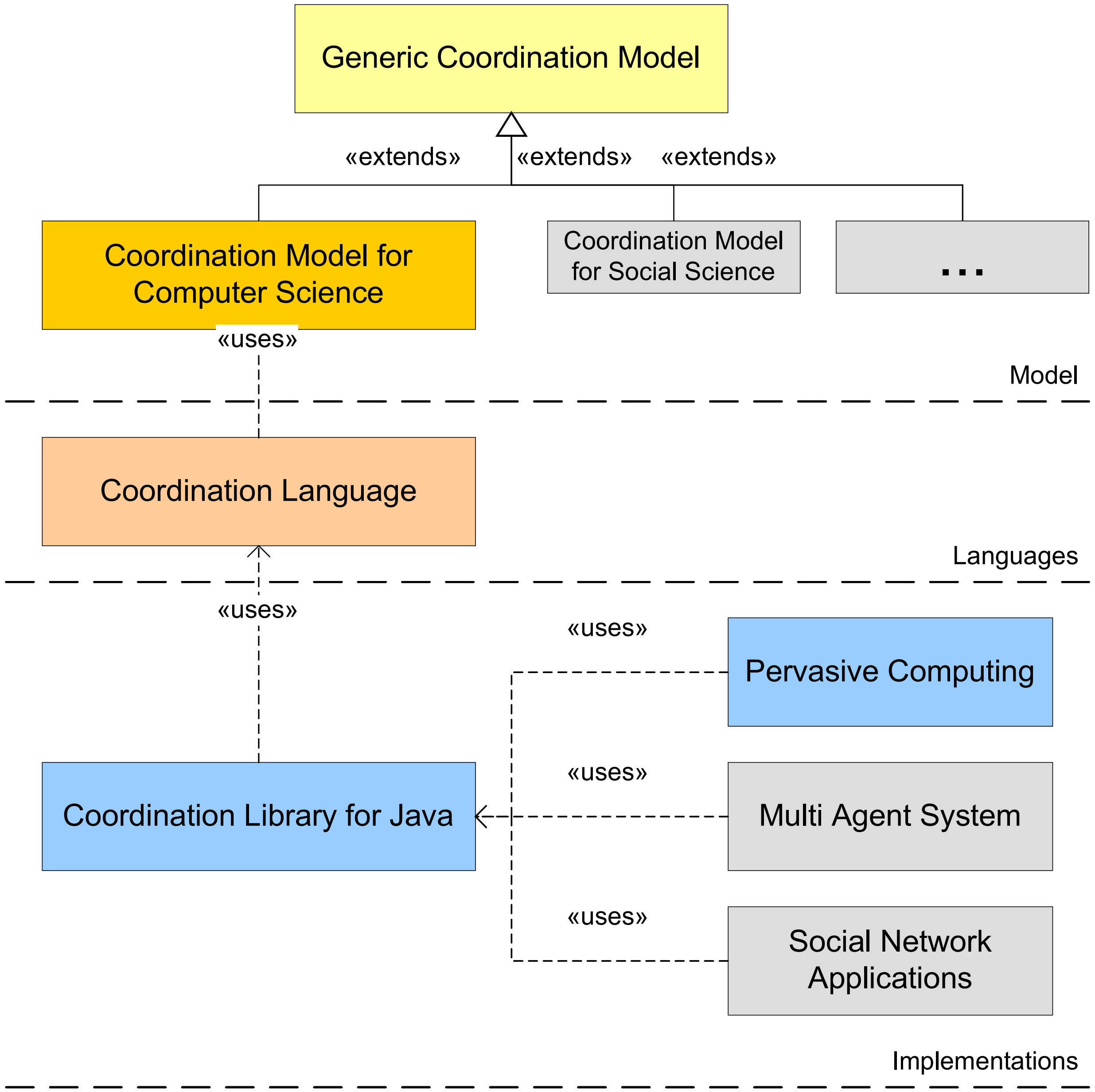
Coordination Language

(illustrated with Java)

Benjamin Hadorn
PAI research group, University of Fribourg
26.02.2010



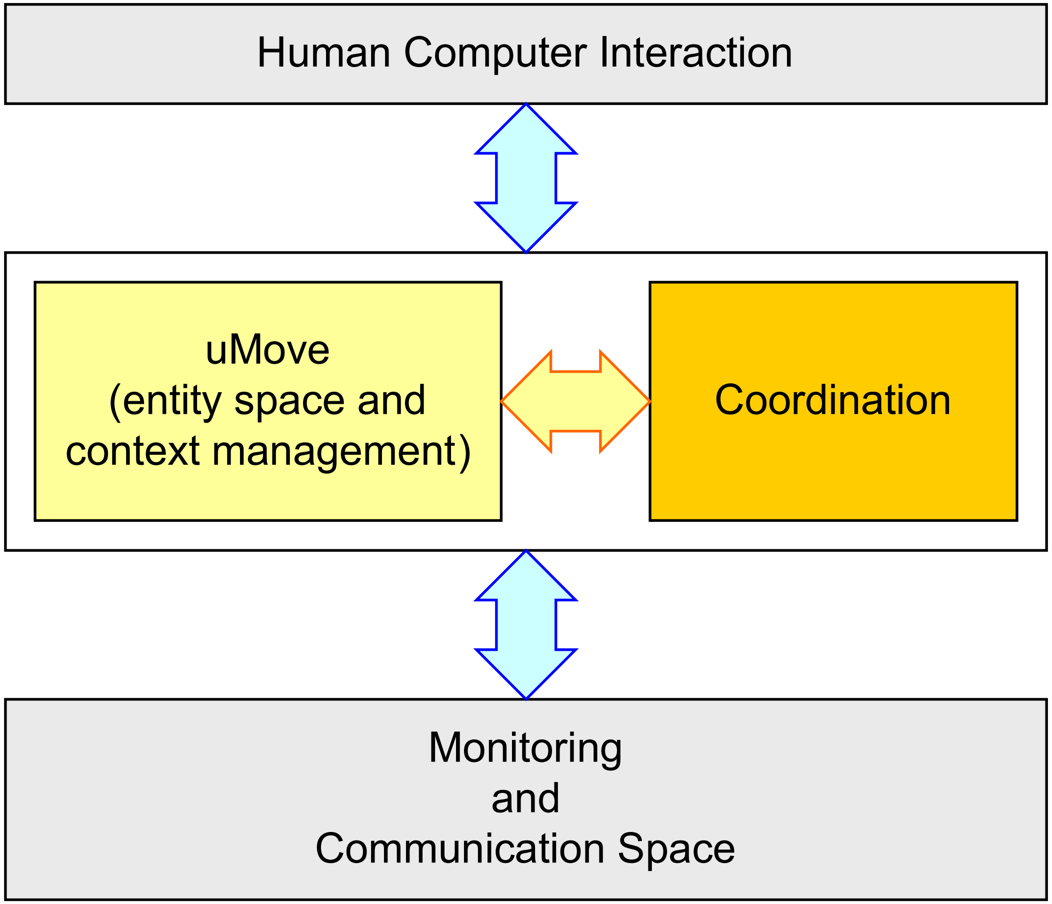
Overview



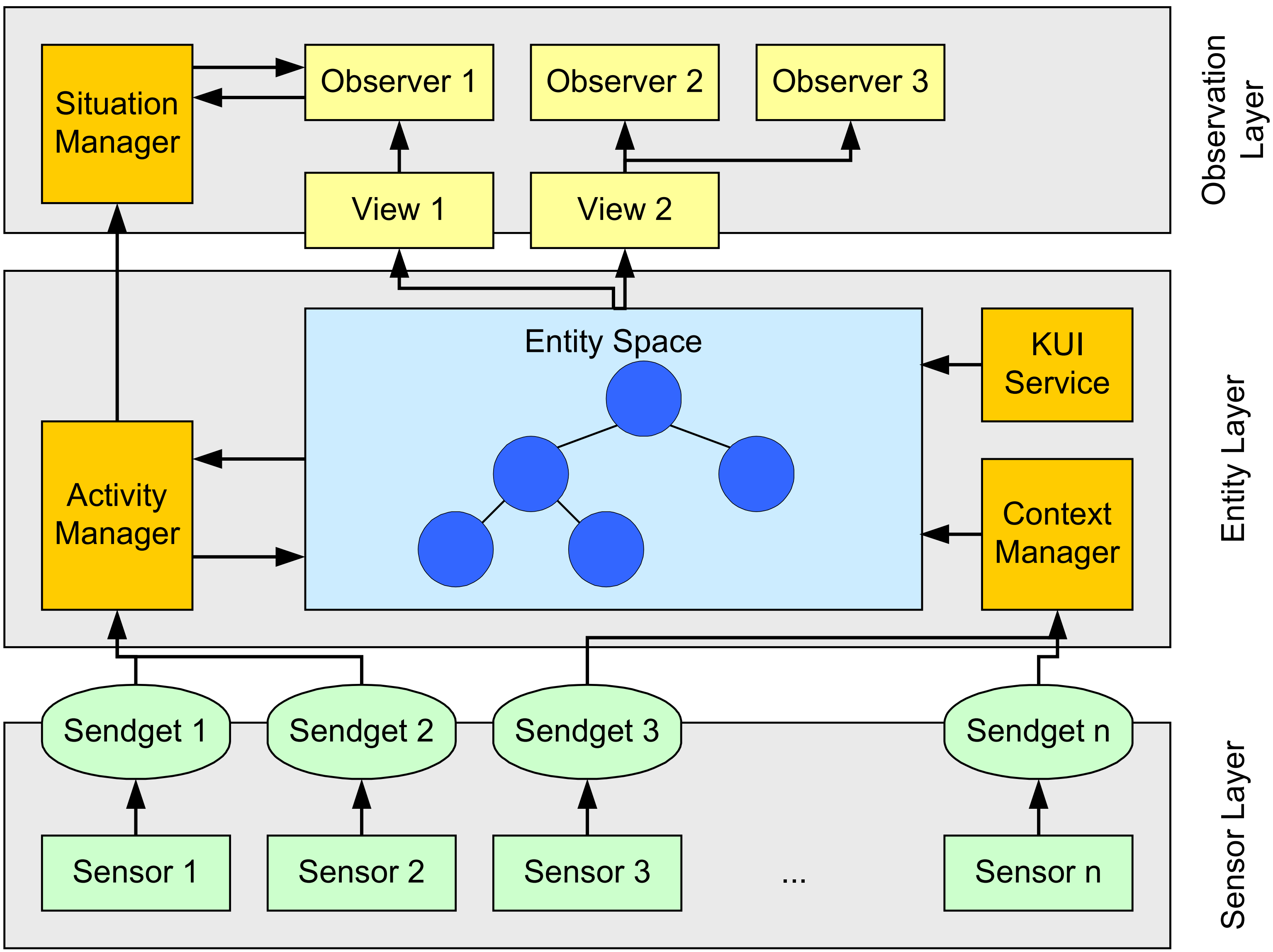
Generic Coordination Model
«extends»
«extends»
«extends»
Coordination Model for Computer Science
Coordination Model for Social Science
...
«uses»
Model
Coordination Language
Languages
«uses»
«uses»
Pervasive Computing
Coordination Library for Java
«uses»
Multi Agent System
«uses»
Social Network Applications
Implementations

Architecture of Middleware for Pervasive Computing

- Java Package extending the computational part of the middleware by coordination



uMove: Representation of the World



Entities

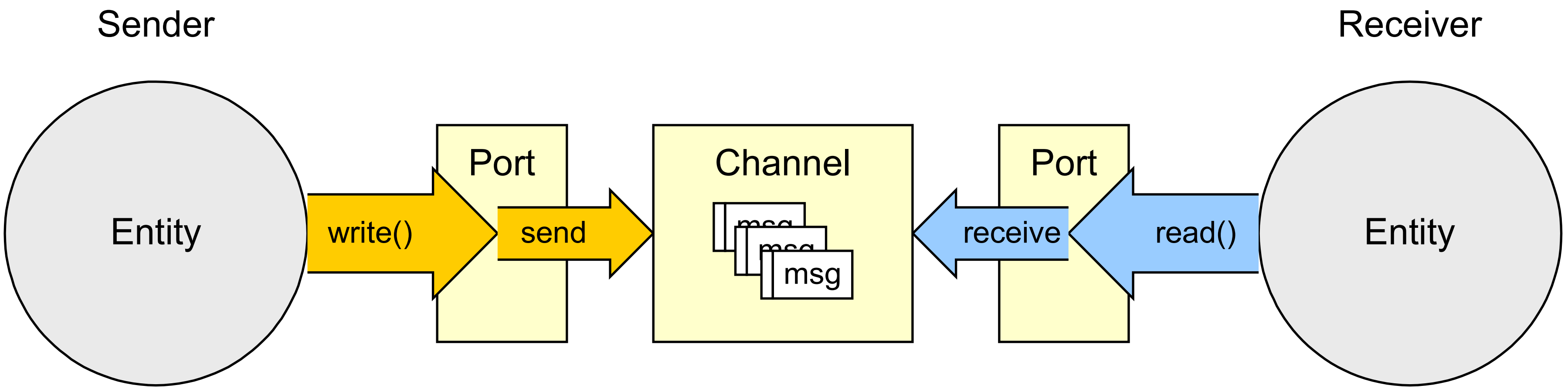
- Entities are described by
  - Identity: unique ID (UUID), Name, RFID Tag, etc.
  - Context: Sensor values, location
  - Motion and Activity
- There exist different types:
  - Actor: observed entities of the physical world
  - Observer: Software observer looking at a part of the world
  - Viewer: Software filter used
  - Sendget: Sensor entity
  - Group: virtual entity used to group physical entities

Ports (I)

- Ports are described by a port descriptor:
  - ID and Name
  - Orientation (input, output or inout)
  - Synchronisation type (asynchron or synchron)
  - Delivery Protocol (FIFO, causal order, total order, ...)
  - Public flag (visible for others)
  - Address

Ports (II)

- An entity can access the communication channel only through ports
  - **write**: The sender puts a message into the channel. The port must have output access.
  - **read**: The receiver reads a message from the channel. The port must have input access.



Ports (III)

- Interface IPort
  - **open**: connects the port to a communication channel
  - **close**: disconnects the port from the channel
  - **peek**: gets a message from the channel without removing it
  - **read** and **readBlocked**: consumes a message from the channel
  - **write**: puts a message to the channel

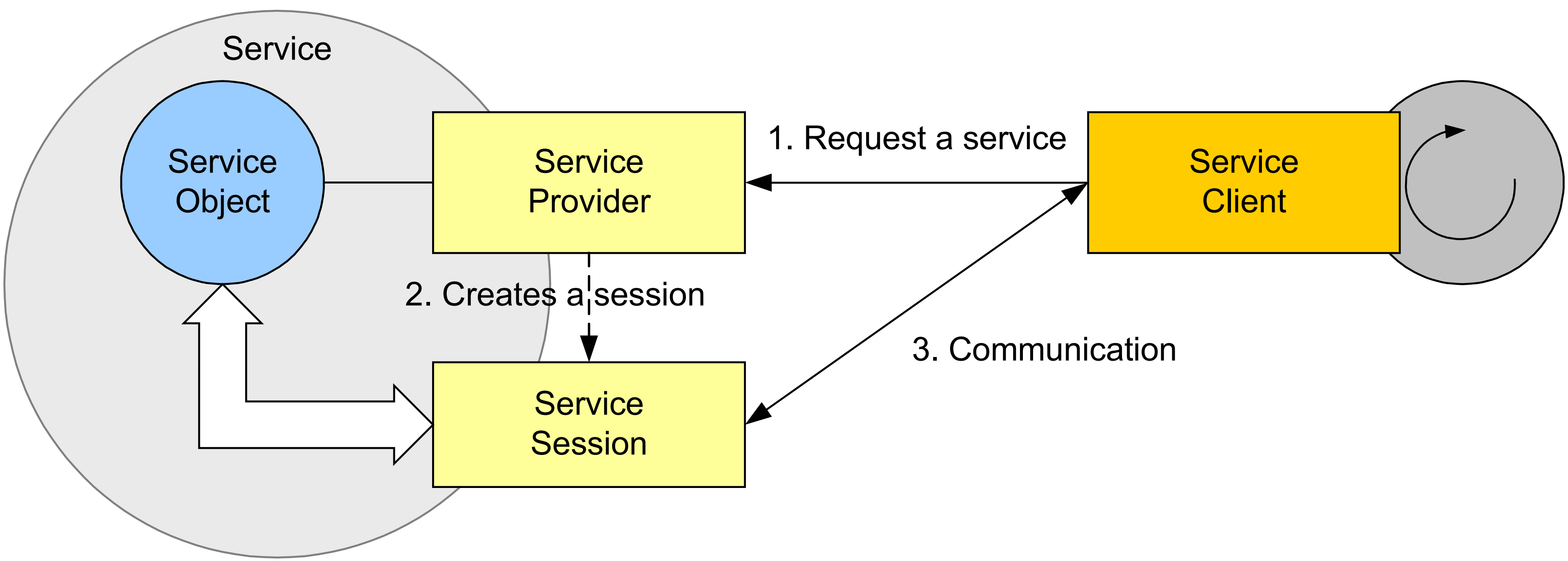
- InputPort, OutputPort and InOutPort classes implement asynchronous port behavior

Services and Public Ports (I)

- Service: The service is a special entity providing a functionality or information to the world
  - one way communication: reading or writing only
  - two way communication: exchaning information, querying, interaction
- Public ports are accessible from outside (external systems)
  - Service Provider: The service provider port is used to provide services to entities
  - Service Session: Once connected to the service provider each entity is treated in a private session.
  - Service Client: Port on the client side

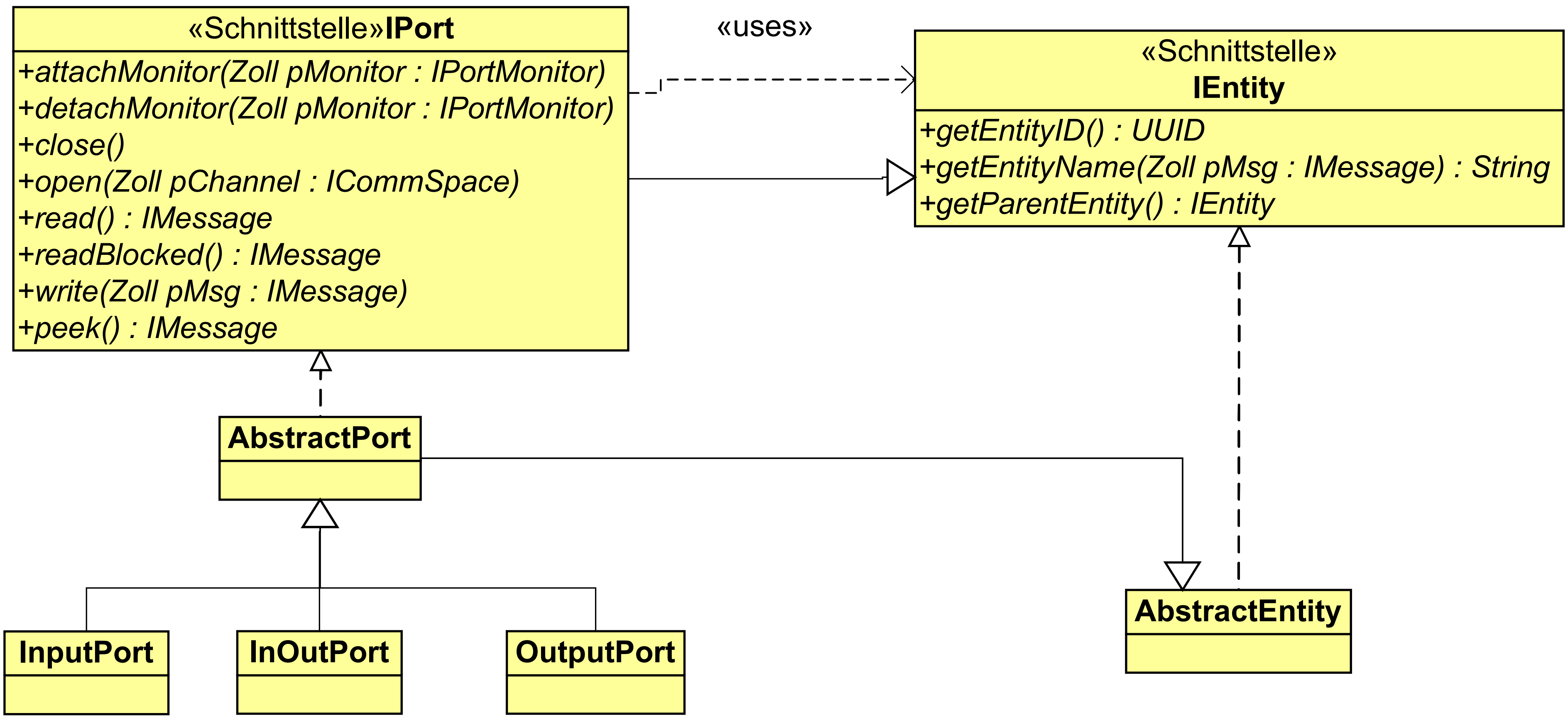
Services and Public Ports (II)

- 1.) requesting the access to service through public port
- 2.) if client is accepted a private session is created.
- 3.) the private session manages the access between client and service



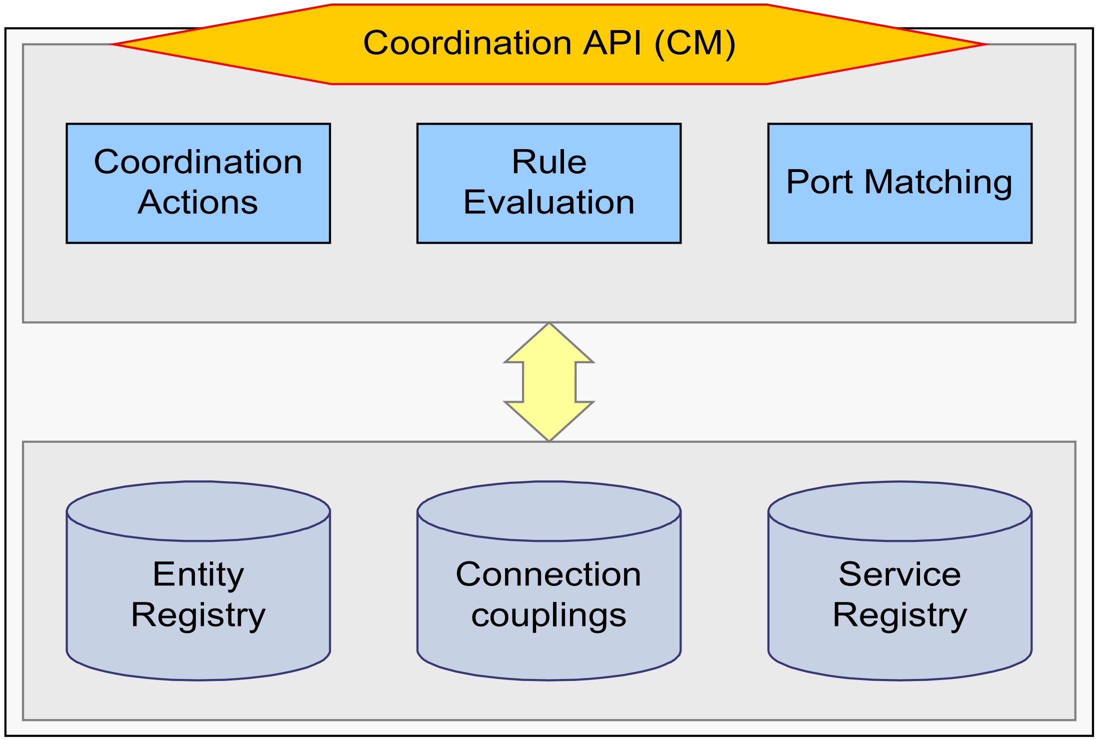
Ports: Classdiagram

- Implementation



Coordination Component (I)

- API for explicit coordination
  - port matching algorithm
  - rule evaluation
  - set of coordination actions



Coordination Component (II)

- The Coordination Manager is used to
  - register all available entities and ports
  - maintain the current connected port (port couplings)
  - clean and remove wasted channels

- The entity space observer is able to detect situation changes of entities within a system
  - evaluates the situation against rules
  - takes actions like
    - creating new channels between entities
    - disconnecting existing channels
    - providing or hiding services

Message Passing

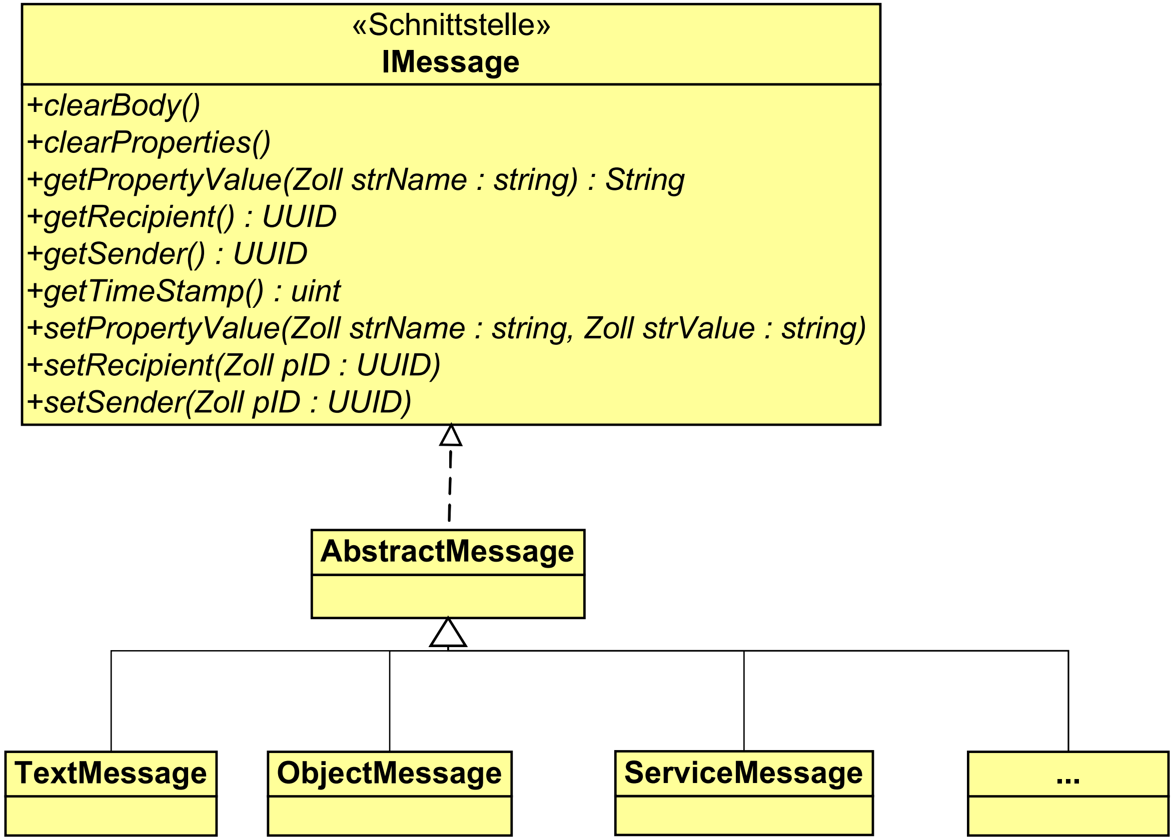
- The communication between entities is realized with message passing technique
  - Transparent for local and remote communication
  - sending through permanent channel
    - a channel is installed permanently between two entities
  - or sending through temporary channel (like e-mail)
    - a message is addresses for different recipients. To each recipient a temporary channel is opened. As soon as the recipient takes the message the channel is closed.

Message Passing Protocol (I)

- Basic construct is of transmission is a message object
  - Header: contains Message ID, Sender ID, Receiver IDs, logical timestamp, properties
  - Body: each concrete message implementation implements its own body
    - TextMessage: String message
    - ObjectMessage: Streamed object
    - ...
- A message is sent from one sender to one or many recipients

Message Passing Protocol (II)

- On top of the message protocol, application specific protocols can be defined

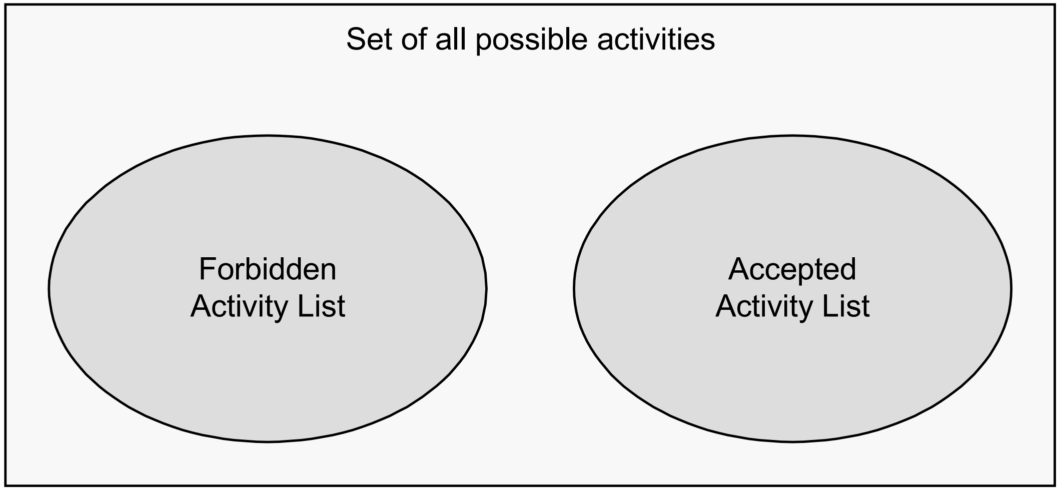


Rules

- Rules are used to
  - Check if an activity is allowed in a place
  - evaluate situations of entities
  - coordinate the interaction between two or more entities
- 3 different rules
  - activity rules
  - situation rules
  - social rules

Activity rules

- If an activity is recognized
  - the activity status is checked
  - the activity rules are attached to the place entity
- Activity lists
  - forbidden: not allowed activities.
  - accepted: approtiated activities.
  - everything else is negotiable.

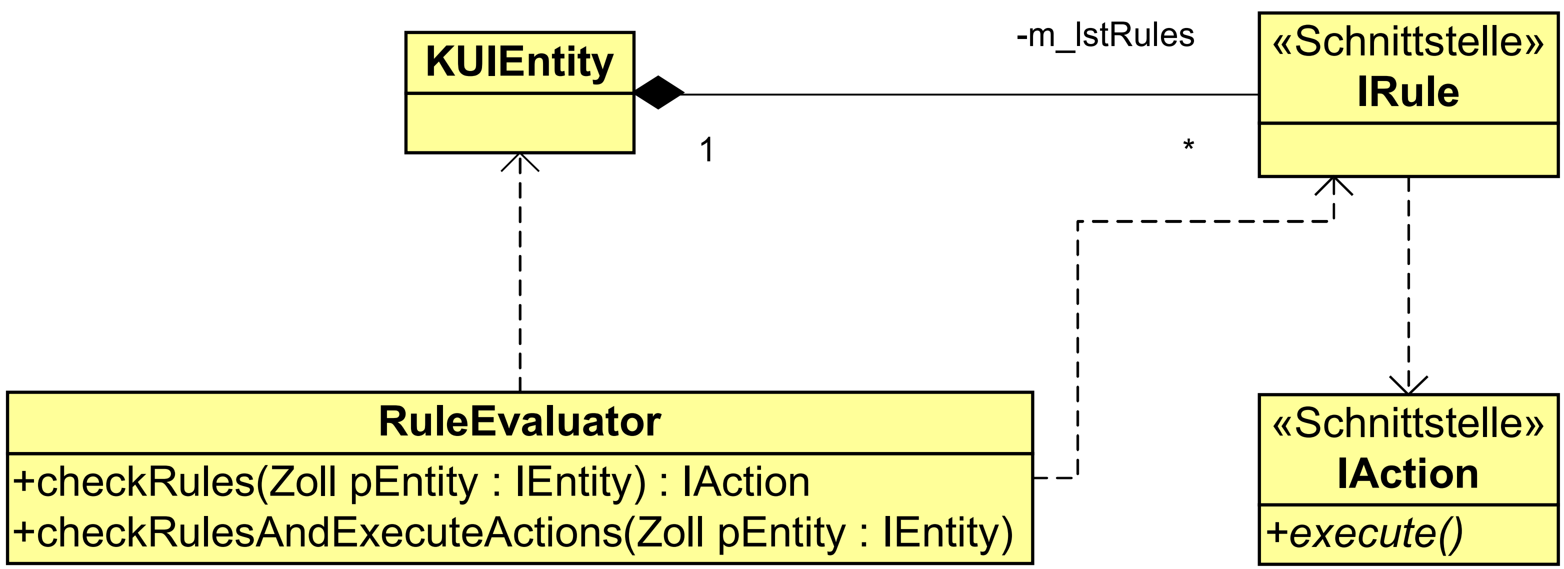


Situation rules

- Situation of each entity is treated in dedicated situation analysis
  - Situation Manager are attached to software observer
  - At each context, activity or structure change the situation is reevaluated
- The Situation Status is reported to the application
  - **Normal**: normal status
  - **Critical**: entity is in a critical situation
  - **Dangerous**: entity is in danger

Social Rules (I)

- Control the interaction between entities
  - sub entities inherit the rules from their parent
  - sub entities can overwrite and extend the rules given by the parent entities
  - after the rule evaluation a list of actions is returned



Social Rules (II)

- Rules inherit the interface IRule
  - **check()**: Checks if the rule is respected. If something has to be done the method returns an action object
  - **getName()**: returns the name of the rule. The name is needed to identify the rule
  - **isFinal()**: Final rules can not be overwritten by rules from child entities.

Merging Social Rules

- Mergin rules using the rule-union operator Û:
  - R_1 and R_2 are set of rules from entity e_1 and e_2 (e_1 = parent of e_2)
  - if a rule exists in both sets the rule of the child entity e_2 is taken

$$R_1 \hat{U} R_2 = R_{xy}(R_1, R_2) \cup R_x(R_1, R_2) \cup R_y(R_1, R_2)$$

$$R_{xy}(R_1, R_2) = \{x \in R_1, y \in R_2 : id(x) \neq id(y)\}$$

$$R_x(R_1, R_2) = \{x \in R_1 : \exists y \in R_2 : id(x) = id(y) \wedge isFinal(x)\}$$

$$R_y(R_1, R_2) = \{y \in R_2 : \exists x \in R_1 : id(x) = id(y) \wedge \neg isFinal(x)\}$$