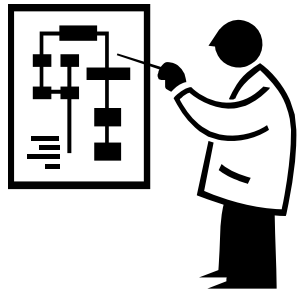
Agent Oriented Software Engineering (AOSE)

PROMETHEUS METHODOLOGY

Software Engineering

- "The computer science discipline concerned with developing large applications."; process, notation, ...
- Typical activities in developing a system:
  - Requirements
  - Analysis
  - Design
  - Implementation
  - …



Agent Oriented Software Engineering (AOSE)

- Software Engineering …
  - "We know how to do SE, how apply to agents?"
- … of Agent Oriented systems
  - "We know about agents and AI, how engineer large (agent) systems?"
- Relatively recent development (first AOSE workshop was in 2000)

Why not traditional SE?

- High level design differs for different programming paradigms, different abstractions:
  - Procedural: What does it do?
  - OO: What objects are there? (data+operations)
  - Agent: What **goals** are there? What are the **relationships** between agents?

Software Engineering

- Some principles
  - Modularity
  - Abstraction
- Process of developing software
  - Requirement analysis, design, testing,…
- Software Architecture

Agents

- Definition

  "An agent is a system situated within and part of environment that senses that environment and acts on it, over time (states), in pursuit of its own agenda (goals) and so as to effect (as in 'modify the environment') what it senses in the future."

- Agent properties
  - Autonomy—has control over its behavior
  - Re-activity—continuously observes and react to changes in its environment in timely fashion
  - Pro-activity—goal oriented
  - Sociality—communicate in a high-level way

Agents

- Contributors to agent research
  - Artificial intelligence
  - Object systems
  - Human-computer interface design
- Multi-agent systems
  - Agent communication languages
  - KQML & FIPA

The Problem

- Agents are a powerful technology with demonstrated potential
- But …
- Agent systems still scarce - why?

➔ Agents hard to develop

AOSE: Milestones

- **Agent-based computing** is a synthesis of both Artificial Intelligence (AI) and Computer Science
- An **agent** is an *encapsulated* ***computer system*** that is situated in some *environment* and that is capable of *flexible, autonomous action* in that environment in order to meet its *design objectives*
- Agents are being advocated as a next generation model for engineering *open, complex, distributed* systems
  - *Open:* components can join or leave the dynamic operating environment and the operating conditions change in unpredictable ways
  - *Complex:* the software has a large number of **components** that interact following complex **interaction protocols**; every agent has a **partial view** of the environment and there is **no centralized control**

We need the right abstractions, methodologies and instruments to correctly engineer applications of this kind

Agent-Oriented Software Engineering [Jen00]

- The case for agent orientation to software engineering
  - **Agent-oriented decomposition** is an effective way of partitioning a problem space
  - **Agent 'mindset'** (agent, interactions, and organizational relationships) are a natural means for modeling complex systems

Objects in OOP vs. Agents

- **Similarities**: Objects (like agents) have
  1. systems with encapsulated states.
  2. Certain methods are associated with the object's state.
  3. Methods support interaction with the environment.
  4. Different objects communicate by message passing.

Objects in OOP vs. Agents

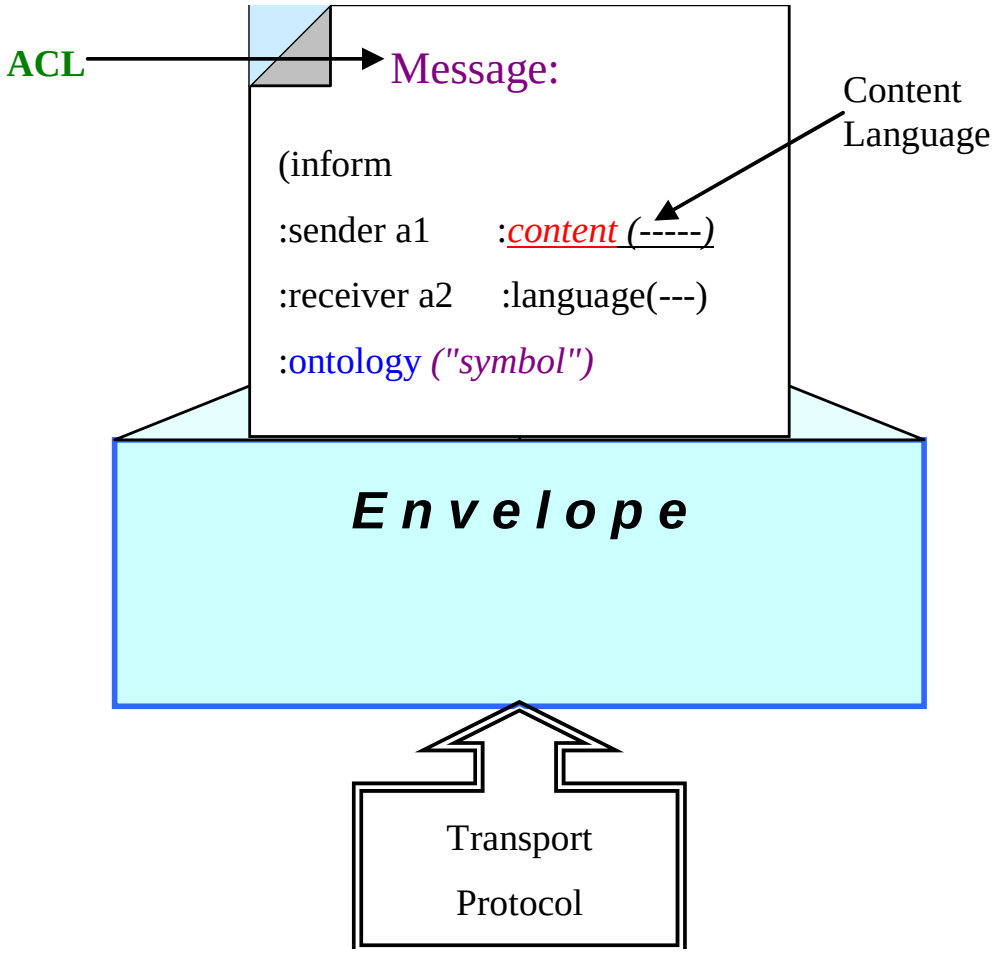
- Differences:
  1. Objects do not usually control their own behaviour.
  2. Agents can initiate their own actions. Object generally do not.
  3. Objects do not have a *social behaviour.*
  4. Agents do not invoke methods in one another.
  5. Interacting agents usually have their own individual thread of control.
  6. Agents can use more than just simple messages to communicate.

1. Objects are associated with their class. Agents can have multiple associations which may also change at any time.
2. Emergence can occur from groups of agents but not from objects.

OOP vs AOP

	OOP	AOP
Basic Unit	Object	Agent
Parameters defining state of basic unit	Unconstrained	Beliefs, commitments, capabilities, choices…
Process of computation	Message passing and response methods	Message passing and response methods
Types of messages	Unconstrained	Inform, request, offer, promise, decline…
Constraints on messages	None	Honesty, consistency…

Message Encoding



Agent Communication Language

- **The information content is communicated through the message types.**
- Ex: (**request** (… a service)
  :sender OrderAgent (service consumer)
  :receiver OperationPlanningAgent (provider)
  :content(action d (produce 50 parts A (product ABC))(due date: Dec 12, 2000)) (service description)
  :protocol fipa-request
  :reply-with order 132
)

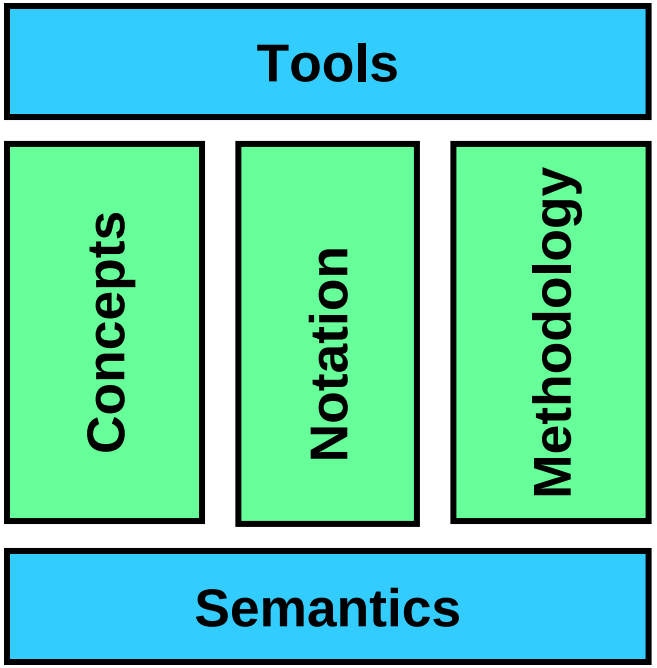
AOSE

To make developing intelligent agents easier (without losing the power of the BDI model).

"Easy things should be easy, hard things should be possible"

Target audience: professional developers or senior undergrads

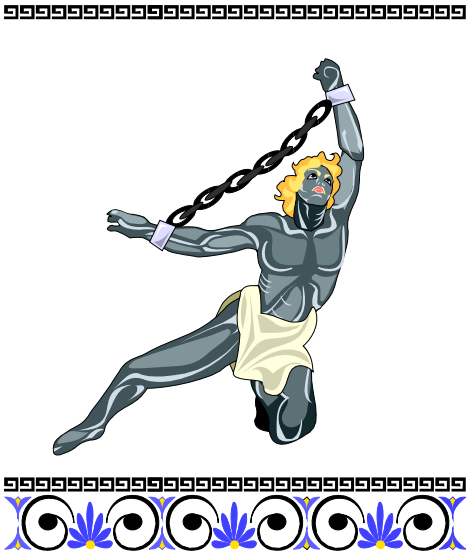
BDI = belief, desire, intention



The Prometheus Methodology

(Padgham & Winikoff)

- Supports the development of *intelligent* agents
- "start-to-end" support
- Detailed process and products
- Evolved out of practical industrial and pedagogic experience
- Has been used
- Hierarchical structuring: scales to large designs
- Cross checking



Prometheus was the wisest Titan. His name means "forethought" and he was able to foretell the future. Prometheus is known as the protector and benefactor of man. He gave mankind a number of gifts including fire. (*http://www.greekmythology.com/*)

Prometheus

- Designed for ease of use, with packaged deliverables:
  - System specification
  - Architectural design
  - Detailed design

Prometheus (Components)

- Prometheus models actors, agents, and resources (data stores)
- Includes BDI percepts, plans, goals.
- Actions, capabilities, messages, and protocols
- Roles
- Scenarios

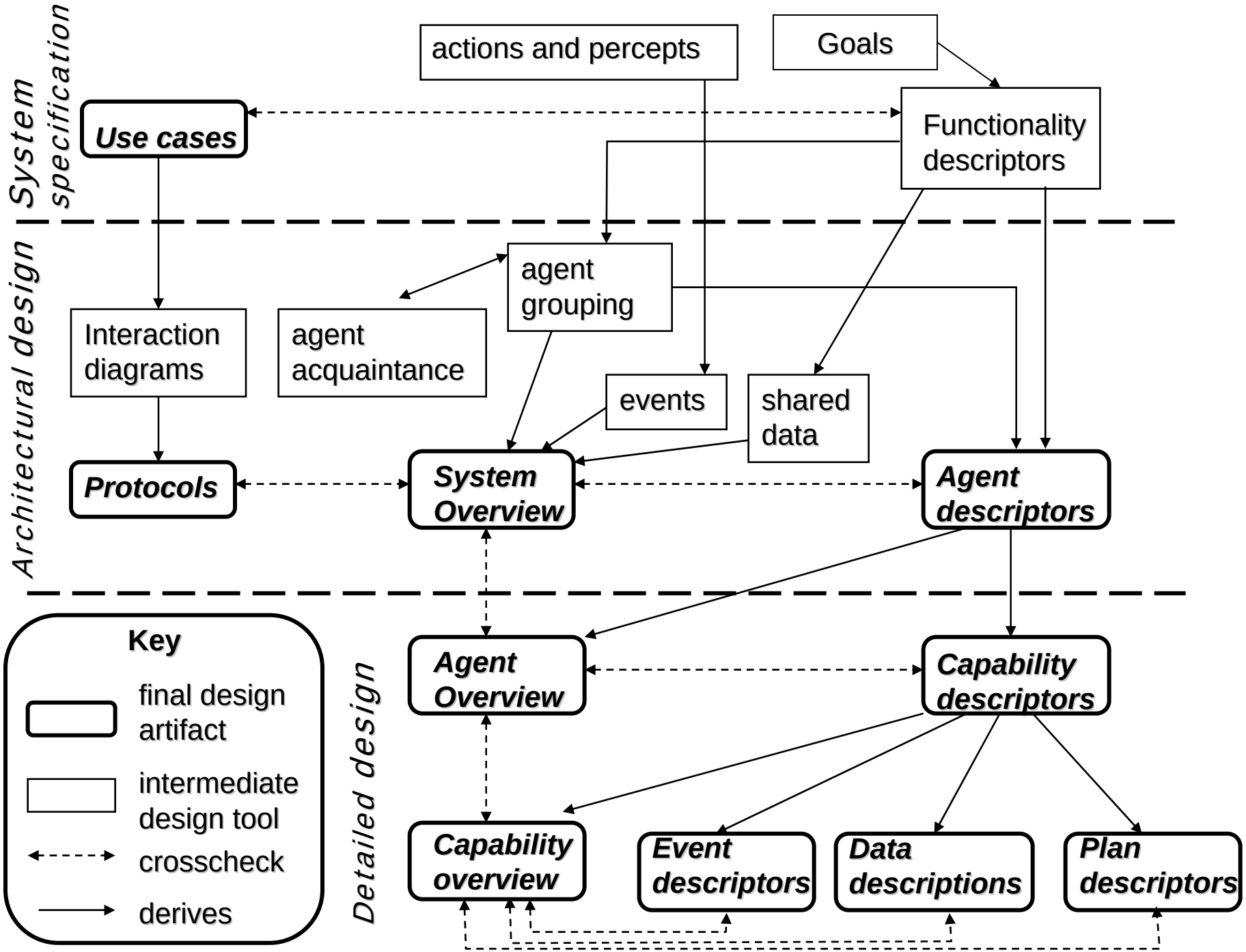
From Prometheus Design Tool Guide

Agency Design

- System Architecture
- **Agent Definitions**
- Knowledge Modeling
- Agent Class Structure

- The agents are defined and their responsibilities are presented.
- Example:

Role Schema:	**Manufacturer Agents**
Description: Manufacturers represent providers of components to the Assembly plant consumer of resources (parts) provided by suppliers	
Protocols: FIPA REQUEST and FIPA ITERATED CONTRACT NET	
Responsibilities: To respond to requests to supply the required components, to initiate request for needed parts, to transfer the required components, to receive and consume resources, to be aware of suppliers of parts and their capabilities, to participate in negotiations about the terms of acquiring and/or supplying a specified resource	


System specification
Architectural design
Detailed design
actions and percepts
Goals
Use cases
Functionality descriptors
agent grouping
Interaction diagrams
agent acquaintance
events
shared data
Protocols
System Overview
Agent descriptors
Agent Overview
Capability descriptors
Capability overview
Event descriptors
Data descriptions
Plan descriptors
Key
final design artifact
intermediate design tool
crosscheck
derives

SYSTEM SPECIFICATION

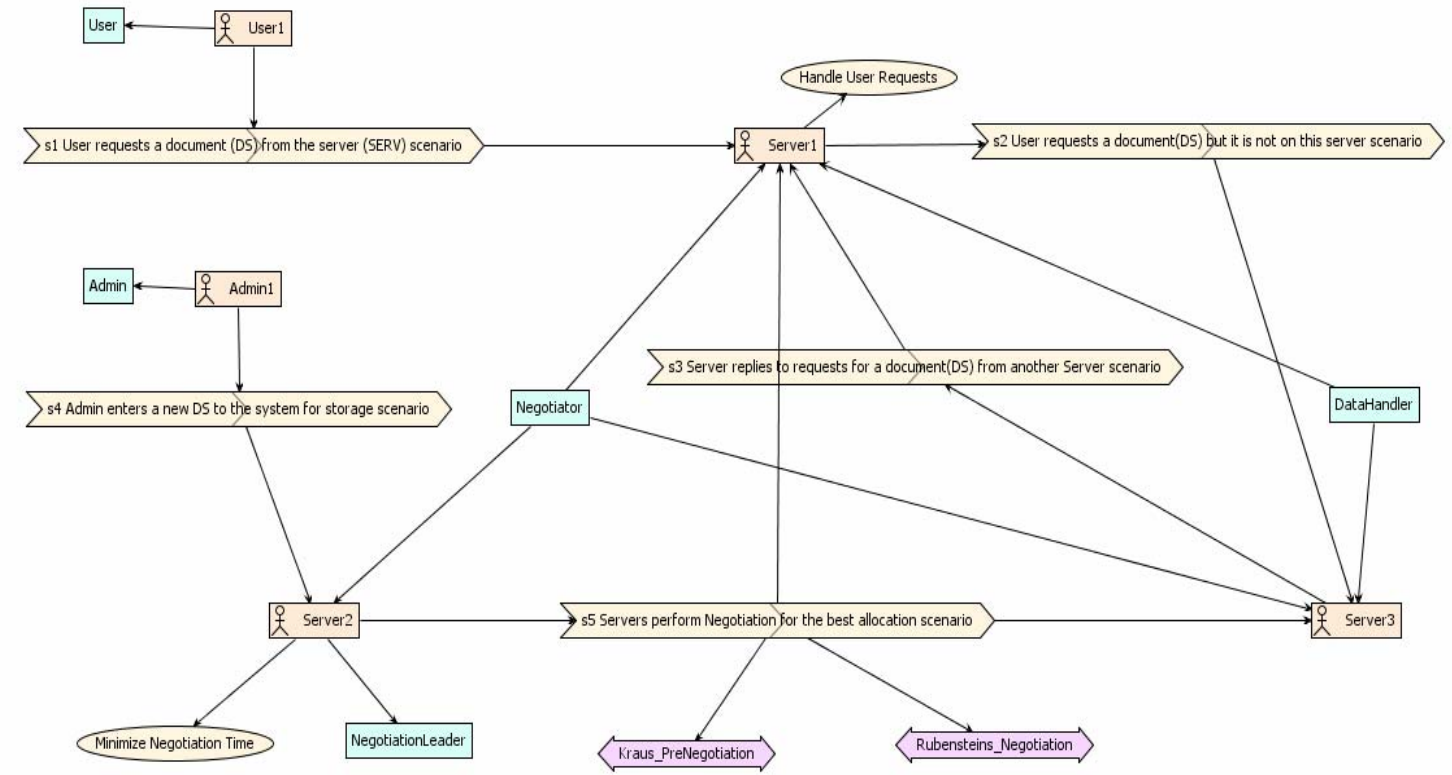
- In this stage "percepts" and "actions" that characterize the agent's interaction with the environment are defined
- **Functional descriptors** that contain a name, description, actions, percepts, data used, and produced and a description of interactions are defined
- **Use cases:** contain an identification number, a natural language overview, an optional field context, the scenario , a summary of all the *information used, and a list of small variants*

An Example: Server Mgmt

- Strategic multi-agent negotiation for data allocation servers:
  - Can only store a single copy of a document
  - Handles user queries for documents
  - Negotiate about which server stores the file
    - Based on
      - Distance
      - Storage space remaining
      - Utility functions (Server-query expectation, capacity, time, etc)
  - Goal is to explore the negotiation protocols
    - Jason + Agentspeak was BDI platform of choice

System Specification

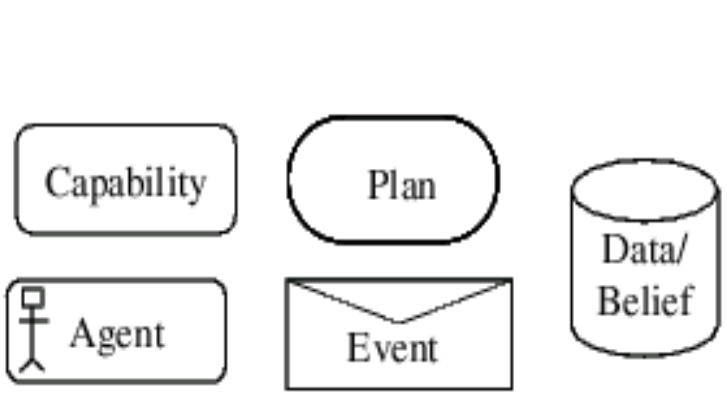
- Artifacts:
  - Scenario Overview Diagram
    - Relates Use cases
  - Analysis Overview Diagram
    - Relates Use cases to users and roles

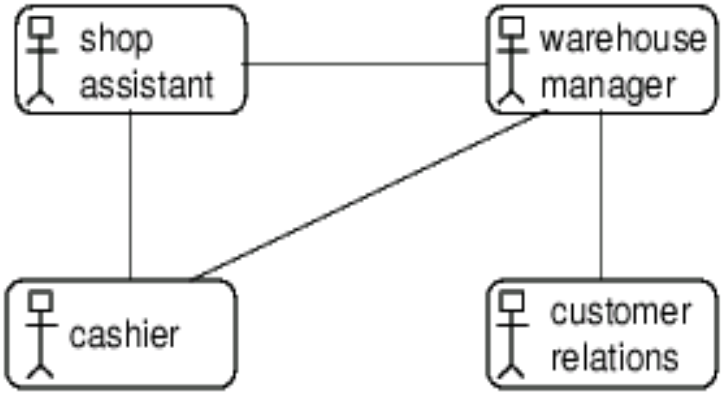


ARCHITECTURAL DESIGN:

⌘ During this stage the following activities are performed**:**

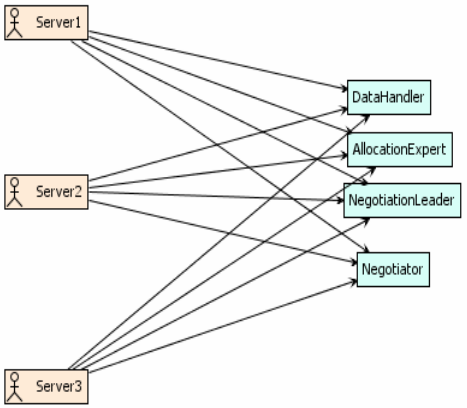
- ↖ Identification of which agents should belong to the MAS
- ↖ identification of groups of agents which share the same functionalities
- ↖ identification of the **agent acquaintance diagram** which defines the links among interacting agents

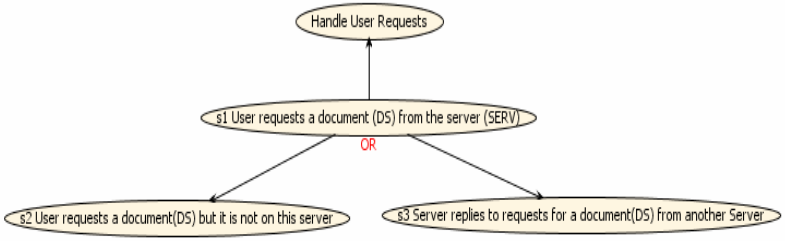


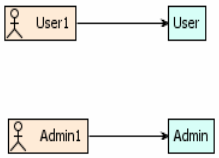


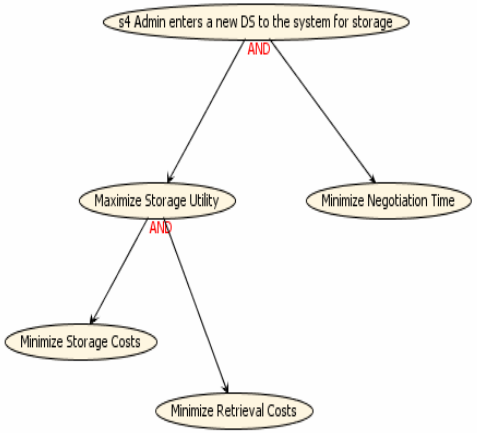
Architectural Design

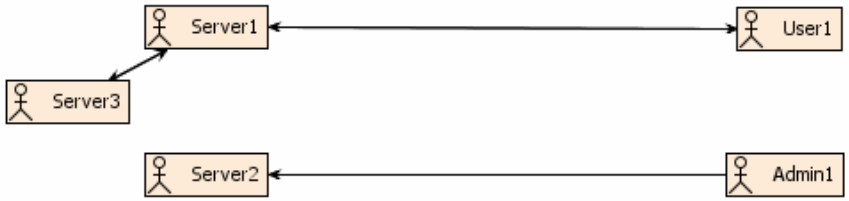
- Artifacts:
  - Agent Role Grouping
  - Goal Overview Diagram
  - Agent Acquaintance Diagram







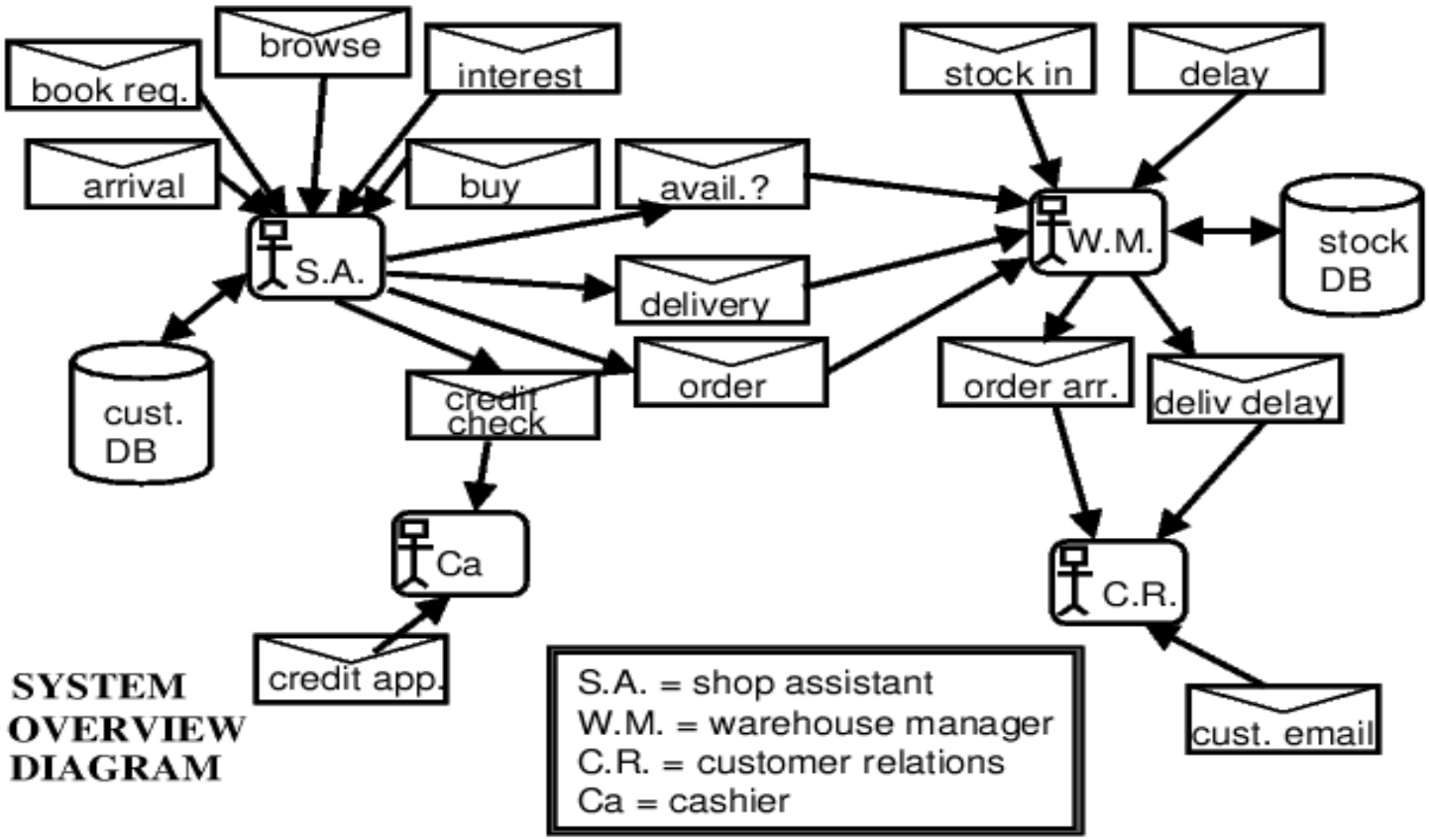




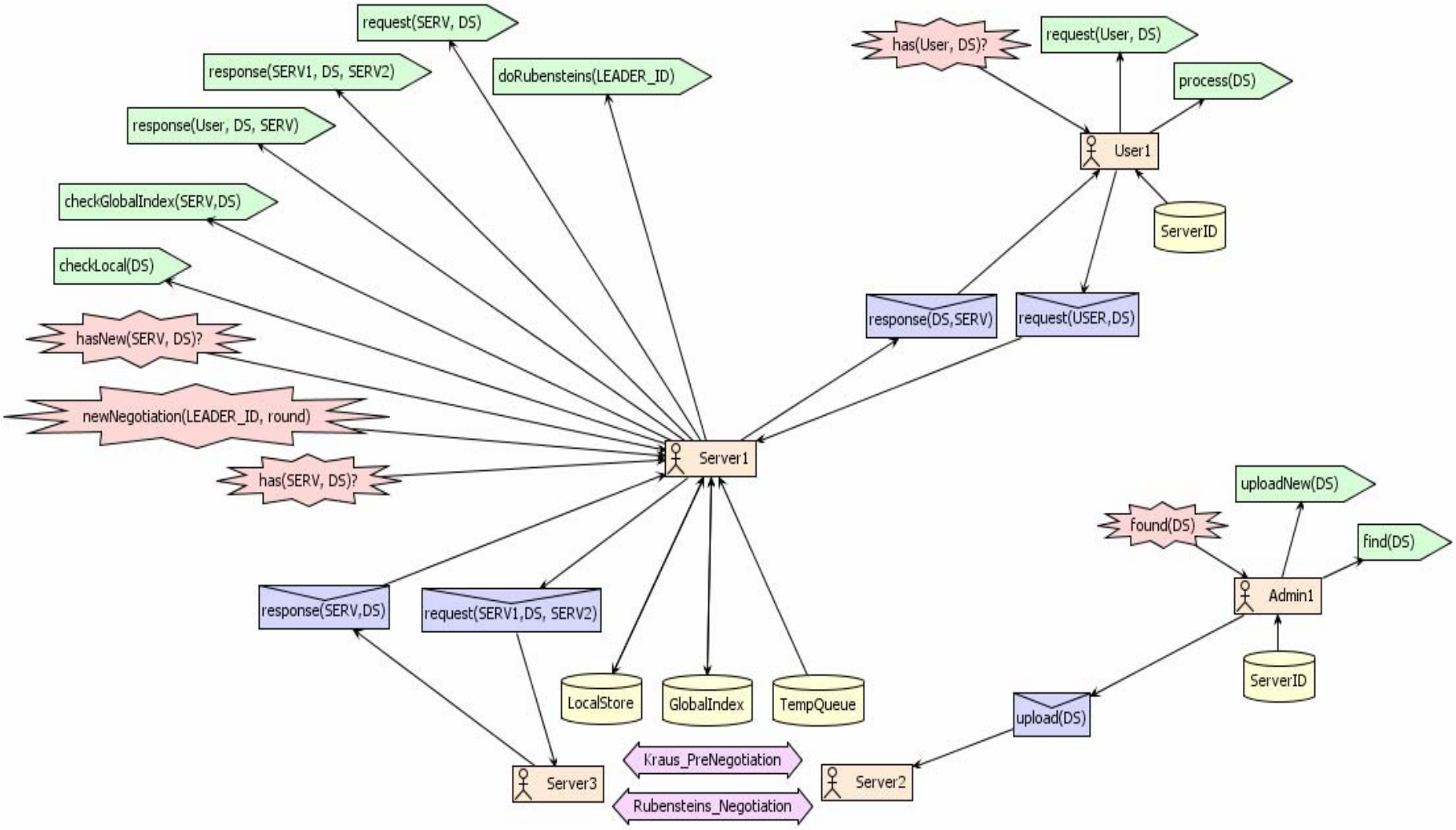
System Overview Diagram

- ⌘ definition of the **agent descriptors**, characterized by name, description, cardinality, functionalities, reads data, writes data, interacts with
- ⌘ definition of the **events**, messages and shared data objects

- identification of the **system overview diagram** which ties together agents, events and shared data objects
- definition of the **interaction diagrams** and **interaction protocols** using AUML



- Artifacts:
  - System Overview Diagram
  - System Roles Diagram
  - Data Coupling Diagram
  - Agent Detail
  - Agent Capability Diagram

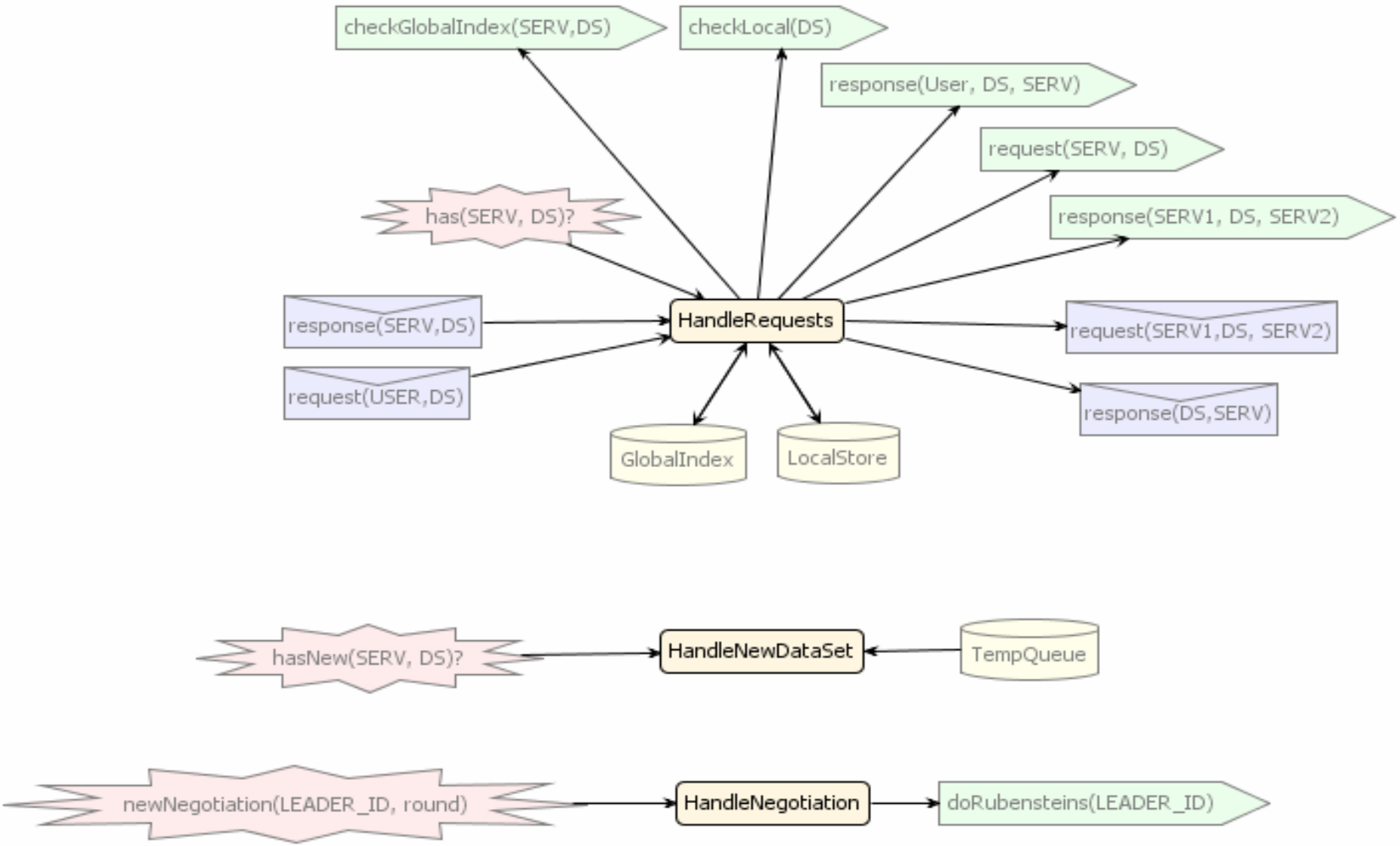


DETAILED DESIGN

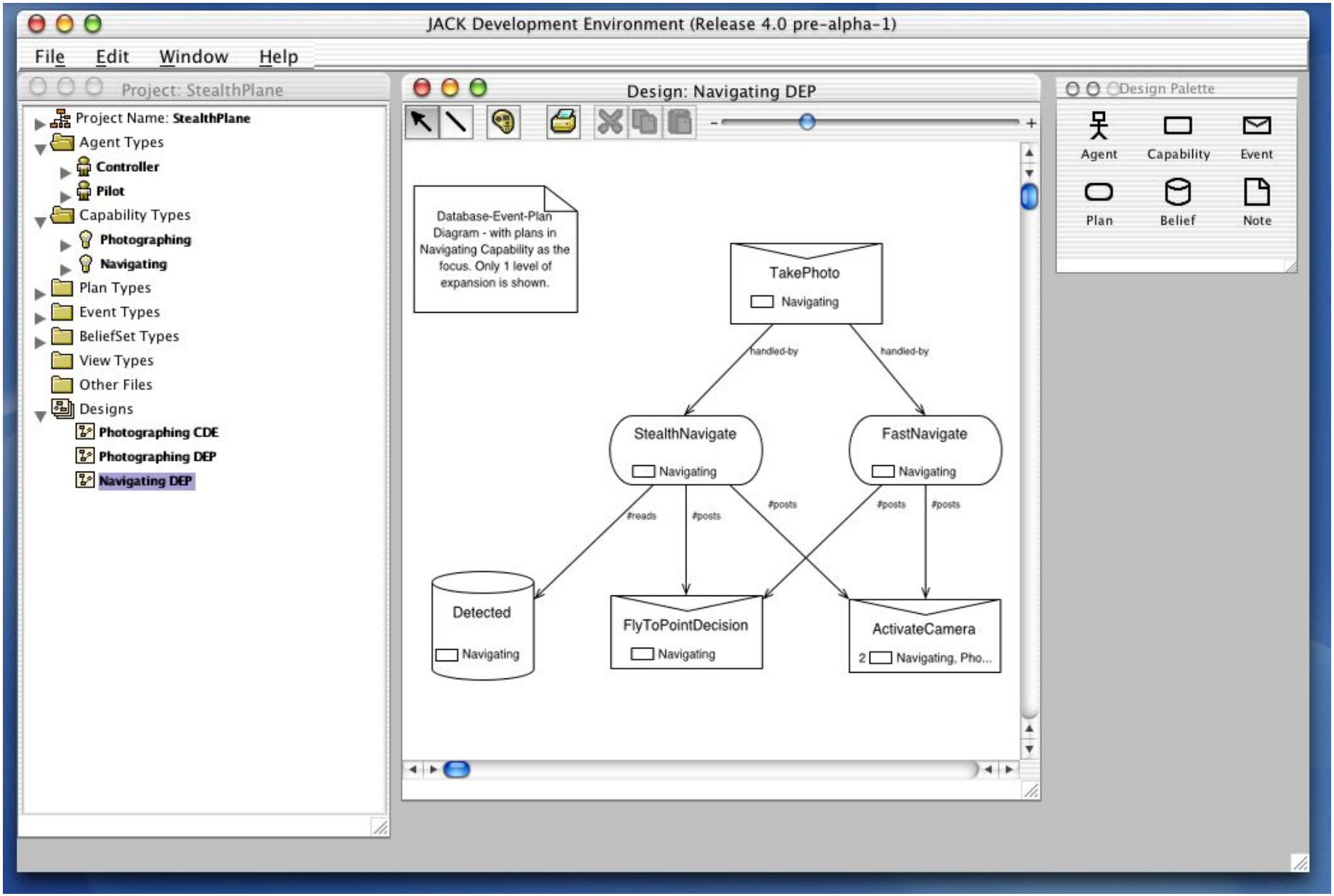
Focuses on developing the internal structure of each agent and how it will achieve its task within the system

- The developer must define capabilities, internal events, plans and detailed data structures
- **Capability descriptor:** contains inputs and produced events, a description of functionality, a name, interactions with other capabilities, inclusions and a reference to read and write data
- **Agent overview diagram:** shows capabilities within an agent
- ***Capability overview diagram:*** takes a single capability and describes its features
- The final design artifacts are the **individual plan, even and data descriptors**
- ***The Plan descriptor*** provides an identifier, the triggering event type, the plan steps, a context and a list of data read and written

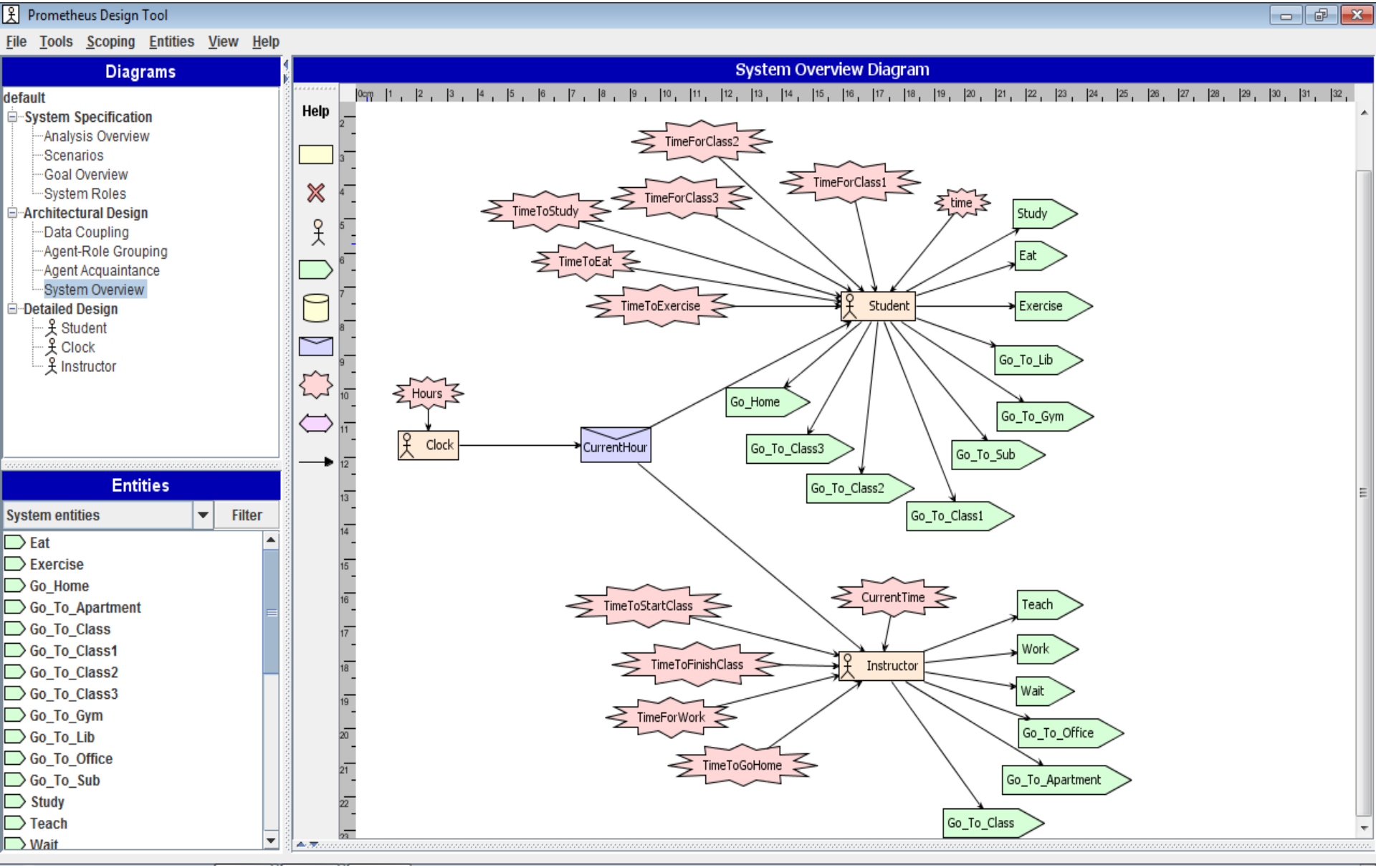
- Agent Detail Diagram
  - Capabilities
  - For single agent



Tool support for Methodology



Prometheus –> Brahms



Tropos – Top-down STS Modelling for Software Engineering

Tropos

- Tropos is an Agent-Oriented Software Engineering (AOSE) Methodology
  - Bresciani, Giorgini, et al, 2002. (Trento/Toronto, et al.)
  - Based on Eric Yu's i* framework.

- Used extensively in Trento for:
  - Security (Si_star)
  - Goal Oriented Requirements Engineering
  - Risk management

Tropos (Cont'd)

- Designed for guiding software development through 4 key phases:
  - Early requirements (stakeholder dependencies)
  - Late requirements (system dependencies)
  - Architectural design
  - Detailed design
- Based on Agent-based concepts (BDI, Goals, etc) throughout all phases. AOP-centric.
- Implementation phases uses these design artifacts

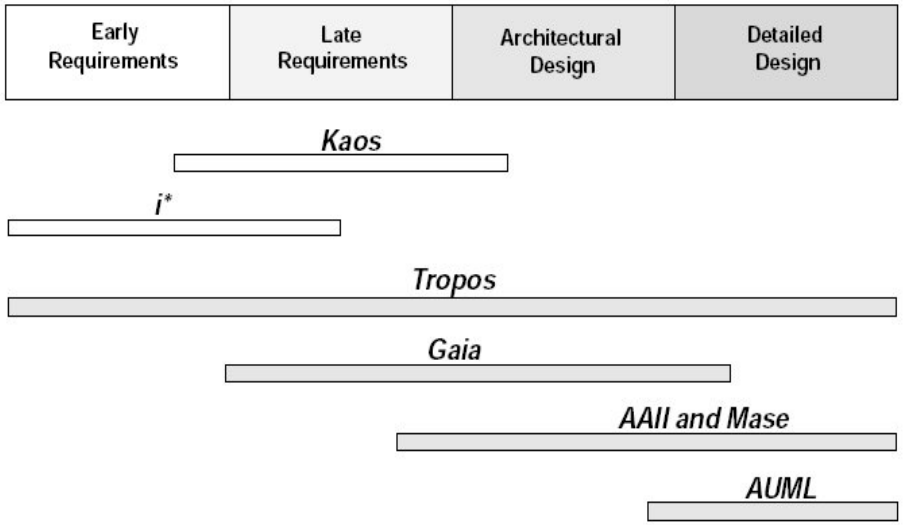


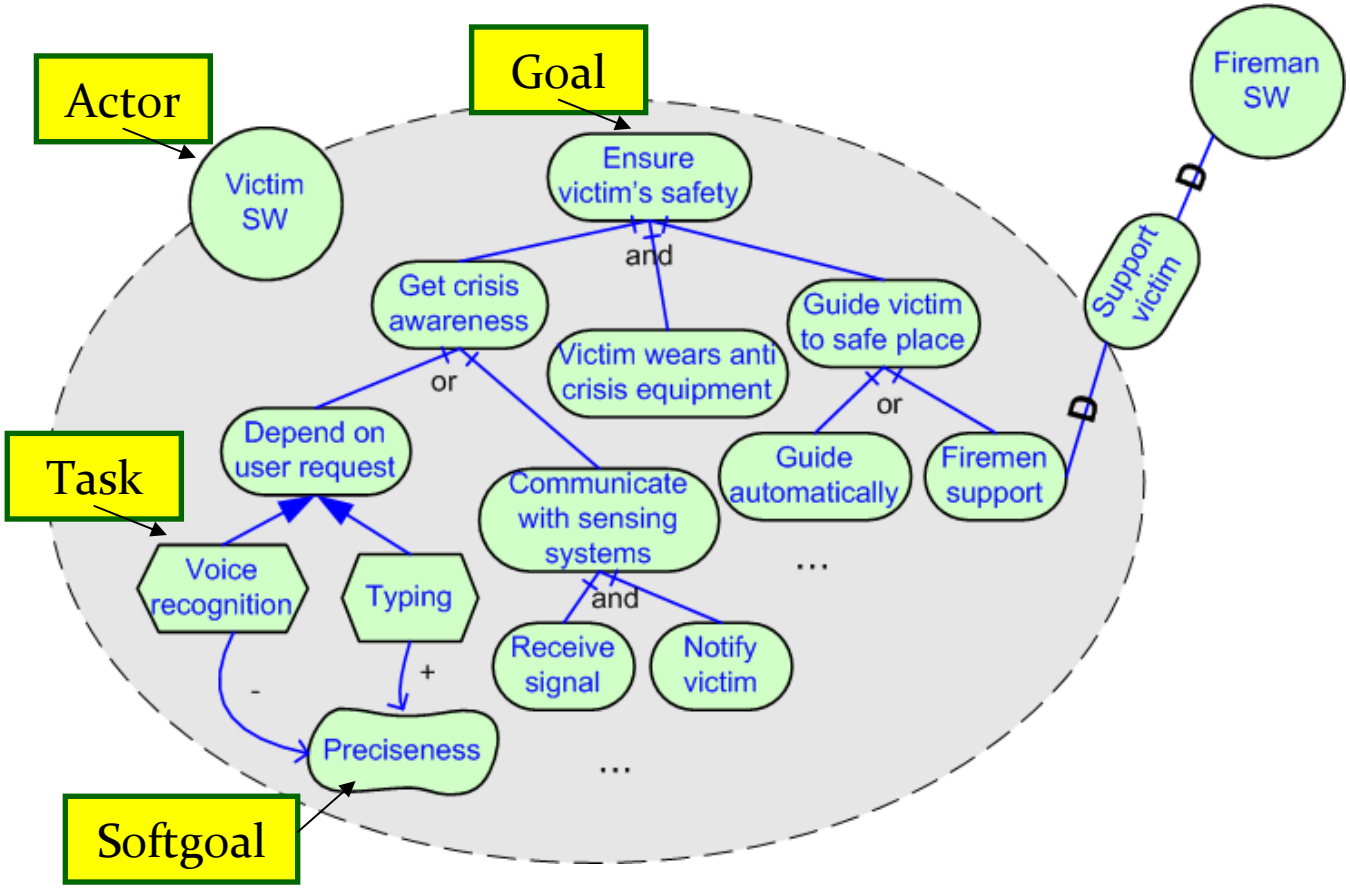
Figure 16. Comparison of Tropos with other software development methodologies.

From Bresciani et al, 2002

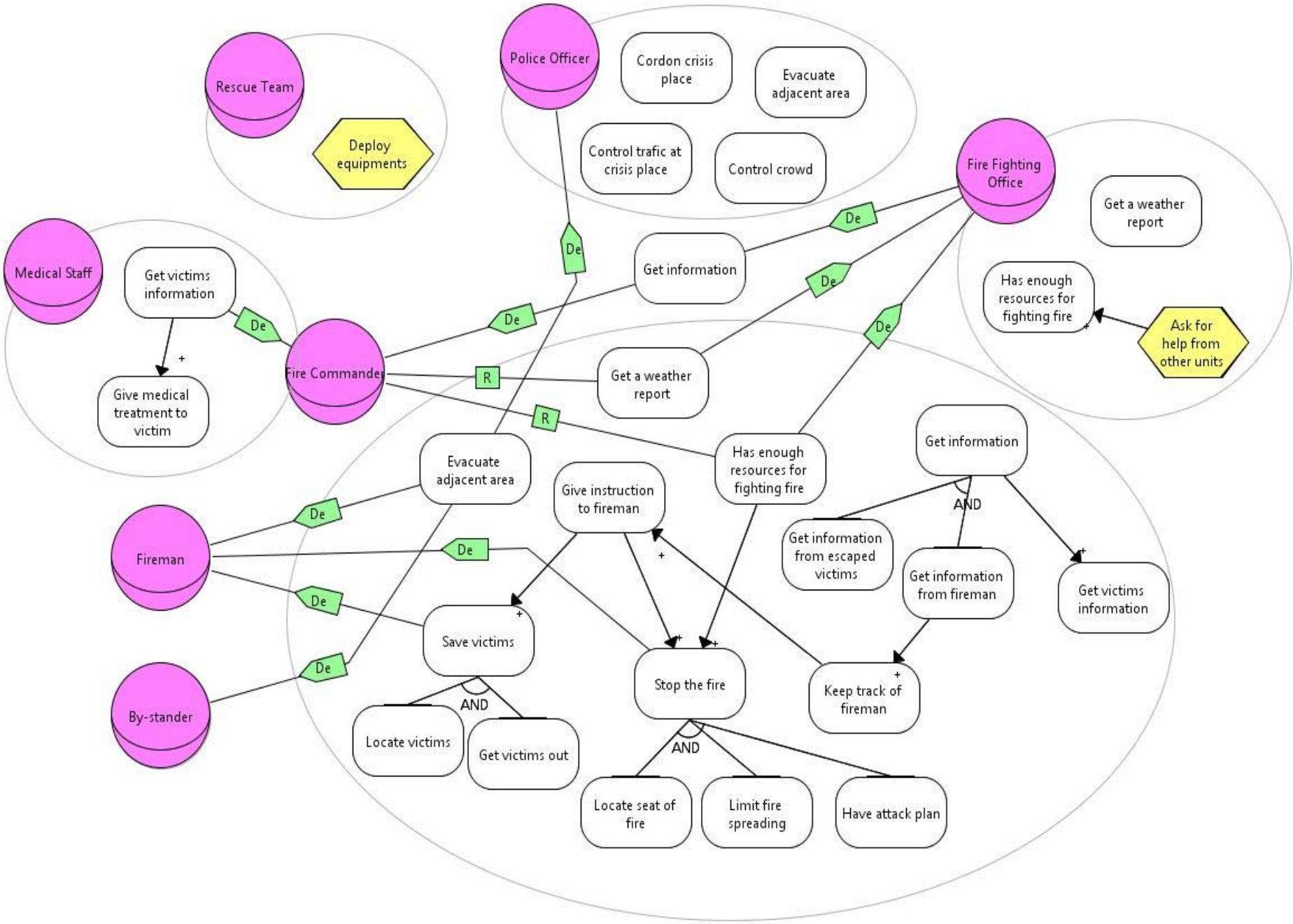
Tropos (Cont'd)

- Key Components:
  - Actor
  - Goal
  - Plan
  - Resource
  - Dependency
  - Capability
  - Belief
  - Other extensions
    - eg.
- Key Models:
  - Actor model
  - Dependency model
  - Goal model
  - Plan model
  - Capability model

Tropos (Components)



From Dalpiaz, et al, 2008



From JACriM project, 2009

Final Thoughts

- Comparing Prometheus and Tropos Methodologies
  - Similarities:
    - Both provide an end-to-end method for system development
    - Both proceed from Early requirements to Detailed design.
    - Both can handle goal models
  - Differences:
    - Prometheus is more holistic
      - Goal based and non goal based agents can be designed
      - Has a nice design tool
      - Encourages use of iterative AOSE development
      - Approach is readily understandable by undergrad students
      - Approach is already tested, and personal favorite.
    - Tropos is better suited for GORE approaches
      - Models convertible to PDDL (XML language)
      - Representation of plans
      - Formal specification
      - But, Largely untested in industry scenarios
      - More intermediate steps before full system functionality