An introduction to Agent-Oriented Software Engineering

Javier Vázquez-Salceda

KEMLg Seminar
April 25, 2012

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
http://www.kemlg.upc.edu

Introduction to Agent-Orientation

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
http://www.kemlg.upc.edu

Computing now-a-days

- Internet Technology
  - Internet 2.0, Broadband access, exploding usage…
- Mobile “Telephony” Technology
  - 3G, iMode, WAP, Wireless PDAs, Bluetooth…
- Software Technology
  - JavaBeans, Soap, UDDI, JINI…
- Web Technology
  - XML, RDF, JavaBeans, Services, “Semantic Web”, IoT
- AI
  - Reasoning, Knowledge Representation, Agents…

5 trends (1 of 2)

- Five ongoing trends have marked the recent history of computing [M. Wooldridge]:
  - *ubiquity*:

    HW costs reduction → processing power everywhere
  - *interconnection*:

    (large) computing nets → "computation as interaction"
  - *intelligence*:

    increased complexity of tasks that are automated in computers
  - *delegation*:

    giving control to computers, even in (safety) critical tasks
  - *human-orientation:*

    Programmers conceptualize SW in terms of higher-level – more human-oriented – abstractions

5 trends (2 of 2)

- Delegation and Intelligence imply the need to build computer systems that can act effectively on our behalf
- This implies:
  - The ability of computer systems to act *independently*
  - The ability of computer systems to act in a way that *represents our best interests* while interacting with other humans or systems
- Interconnection and Distribution have become core motifs in Computer Science
- But Interconnection and Distribution, coupled with the need for systems to represent our best interests, implies:
  - Systems that can *cooperate* and *reach agreements* (or even *compete*) with other systems that have different interests (much as we do with other people)

Computer Science progression

- These issues were not studied in Computer Science until recently
- All of these trends have led to the emergence of a new field in Computer Science: *multiagent systems*
- Wooldridge says that programming has progressed through:
  - machine code;
  - assembly language;
  - machine-independent programming languages;
  - sub-routines;
  - procedures & functions;
  - abstract data types;
  - objects;

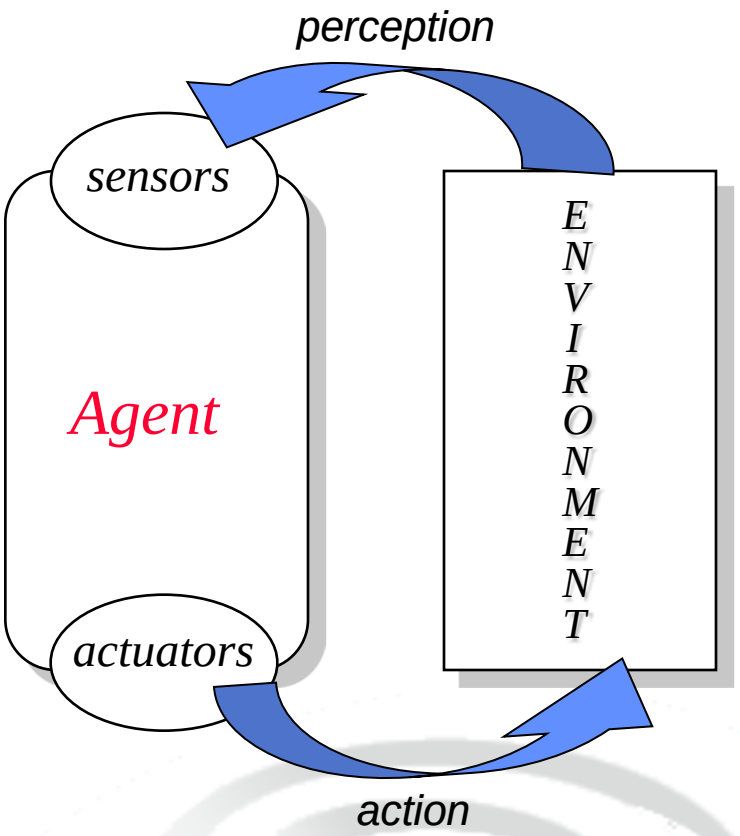
to *agents*.

Hot topic: Open Service Environments

- Explosion of Agent technology with new uses for Open Service Environments
- Automation of Services
  - Proactive, responsible, intelligent, peer to peer
- Dynamic Composition of Services
  - Automated discovery, automated coordination, "Just in Time" Enterprises, Virtual Companies
- Semantics
  - HTML won't do anymore
  - "Semantic Web"
  - Service-level semantics
  - Semantics for E-commerce
  - Service-Oriented Architectures' frameworks

Agents

- *An **agent** is a computer system capable of* autonomous action *in some environment* on behalf of its user or owner
  - figuring out what needs to be done to satisfy design objectives, rather than constantly being told



- Usually the environment is *complex* and *dynamic*, and agents should interact with it in real time.

o *Main property: Autonomous*

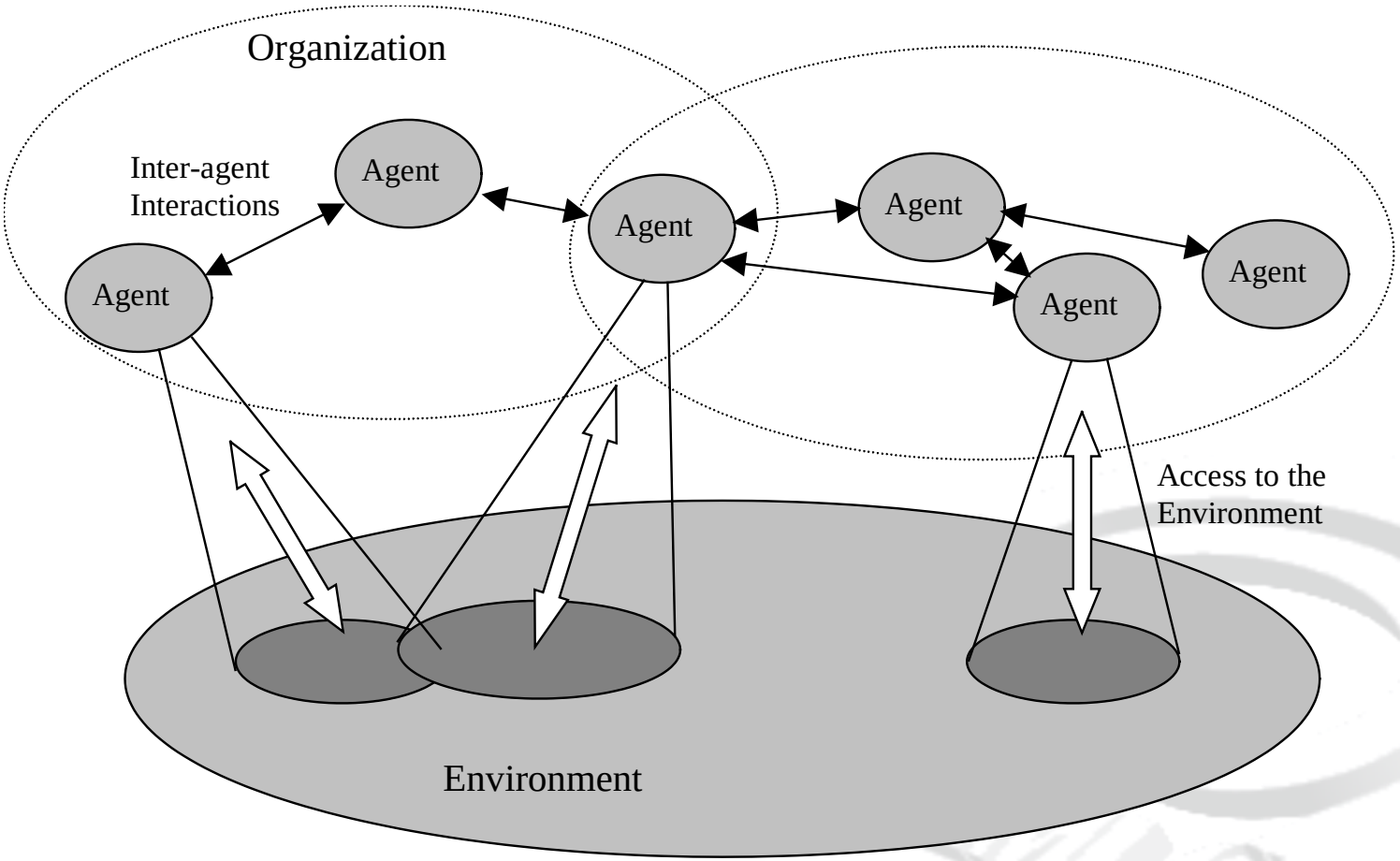
capable of acting independently, exhibiting control over their internal state

Agents and Multiagent Systems (1 of 2)

- An agent is a computer system that is capable of *independent* action on behalf of its user or owner
- A ***multiagent system (MAS)*** is one that consists of a number of agents, which *interact* with one-another in order to achieve more complex tasks.
- In the most general case, agents will be acting on behalf of users with different goals and motivations
- To successfully interact, they will require the ability to *cooperate*, *coordinate*, and *negotiate* with each other, much as people do

Agents and Multiagent Systems (2 of 2)

Characterisation of a MAS



Autonomy vs Dependency in Agents (1 of 2)

- We defined ***Autonomy*** as the main property in agents
  - "capable of acting independently, exhibiting control over their internal state"
- But… do agents loose autonomy when they enter in a MAS?
  - they pass to depend on others!!!

Autonomy vs Dependency in Agents (2 of 2)

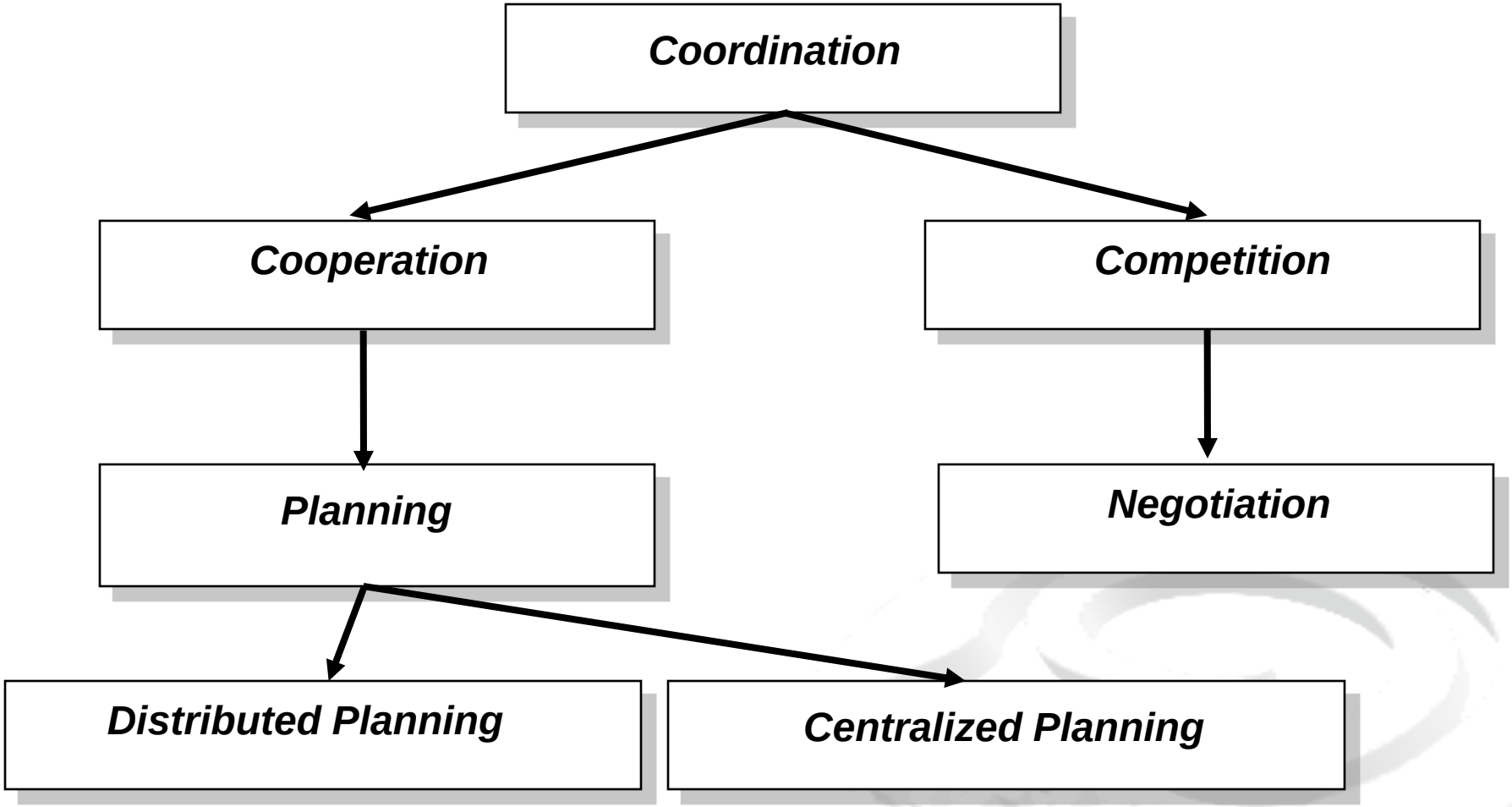
Agent Coordination

- It is the other way arround:
  - Agents enter into a MAS when they cannot achieve their goals by themselves and depend on others
  - The balance between agent autonomy and dependence is managed via coordination mechanisms

- ***Coordination*** can be defined as the process of managing dependencies between activities. By such process an agent reasons about its local actions and the foreseen actions that other agents may perform, with the aim to make the community to behave in a coherent manner.

Coordination

Types of coordination



Coordination

Another Classification

- **Coordination** can also be divided along another dimension:
  - **Explicit Coordination**: agents communicate goals, plans, actions, state of the world with the explicit goal of acting coherently.
  - **Implicit Coordination**: no communication – the environment acts as the interaction mechanism

Designing Agents and Multiagent Systems

- Building Agents, we address questions such as:
  - How do you state your preferences to your agent?
  - How can your agent compare different deals from different vendors? What if there are many different parameters?
  - What algorithms can your agent use to negotiate with other agents (to make sure you get a good deal)?

- In Multiagent Systems, we address questions such as:
  - How can cooperation emerge in societies of self-interested agents?
  - What kinds of languages can agents use to communicate?
  - How can self-interested agents recognize conflict, and how can they (nevertheless) reach agreement?
  - How can autonomous agents coordinate their activities so as to cooperatively achieve goals?

Agent Design, Society Design

- Two key problems:
  - How do we build agents capable of independent, autonomous action, so that they can successfully carry out tasks we delegate to them?
  - How do we build agents that are capable of interacting (cooperating, coordinating, negotiating) with other agents in order to successfully carry out those delegated tasks, especially when the other agents cannot be assumed to share the same interests/goals?
    - The first problem is *agent design* [that is, the internal design and implementation of the agent, which is not covered in this talk].
    - The second is *society design* (micro/macro) [this is covered in this talk].

2 Views of the Field

- *Agents as a paradigm for software engineering*: Software engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that *interaction* is probably the most important single characteristic of complex software

- Over the last two decades, a major Computer Science research topic has been the development of tools and techniques to model, understand, and implement systems in which interaction is the norm

2 Views of the Field

- *Agents as a tool for understanding human societies*: Multiagent systems provide a novel new tool for simulating societies, which may help shed some light on various kinds of social processes.

- This has analogies with the interest in “theories of the mind” explored by some artificial intelligence researchers

Introduction (to Agent Methodologies)

- *Software Engineering*
- *Agent-Oriented Software Engineering*
- *Software Methodologies*
- *Agent-Oriented Methodologies*

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
http://www.kemlg.upc.edu

Software Engineering

Status of Software Engineering in the New Millennium

- Current tendency to make *software functionalities* and *business cases* coincide - stimulated by the Internet era and reinforced by the DOTCOM economy
  - Leads to *linking software construction* and *business dynamics* more closely than ever
- In industry there is a need for swiftly-developed, complex software projects that are both *research-like* and *mission-critical*
  - Software development must no longer be thought of as *oriented toward a product* ***BUT*** it is an ongoing process *which continually delivers value* (continuous evolution)
- Software crisis
  - Hardware costs were decreasing while software costs were increasing.

Software Engineering
Abstractions

- Software deals with *"abstract" entities*, having a real-world counterpart
  - Numbers, dates, names, persons, documents, ...
- In what term shall we model them in software?
  - Data, functions, objects, agents, …
  - I.e., what are the abstractions that we have to use to model software?
- May depend on available technologies

Software Engineering

Towards Agent-Oriented Software Engineering

"Objects are far from perfect, but are the only game in town"

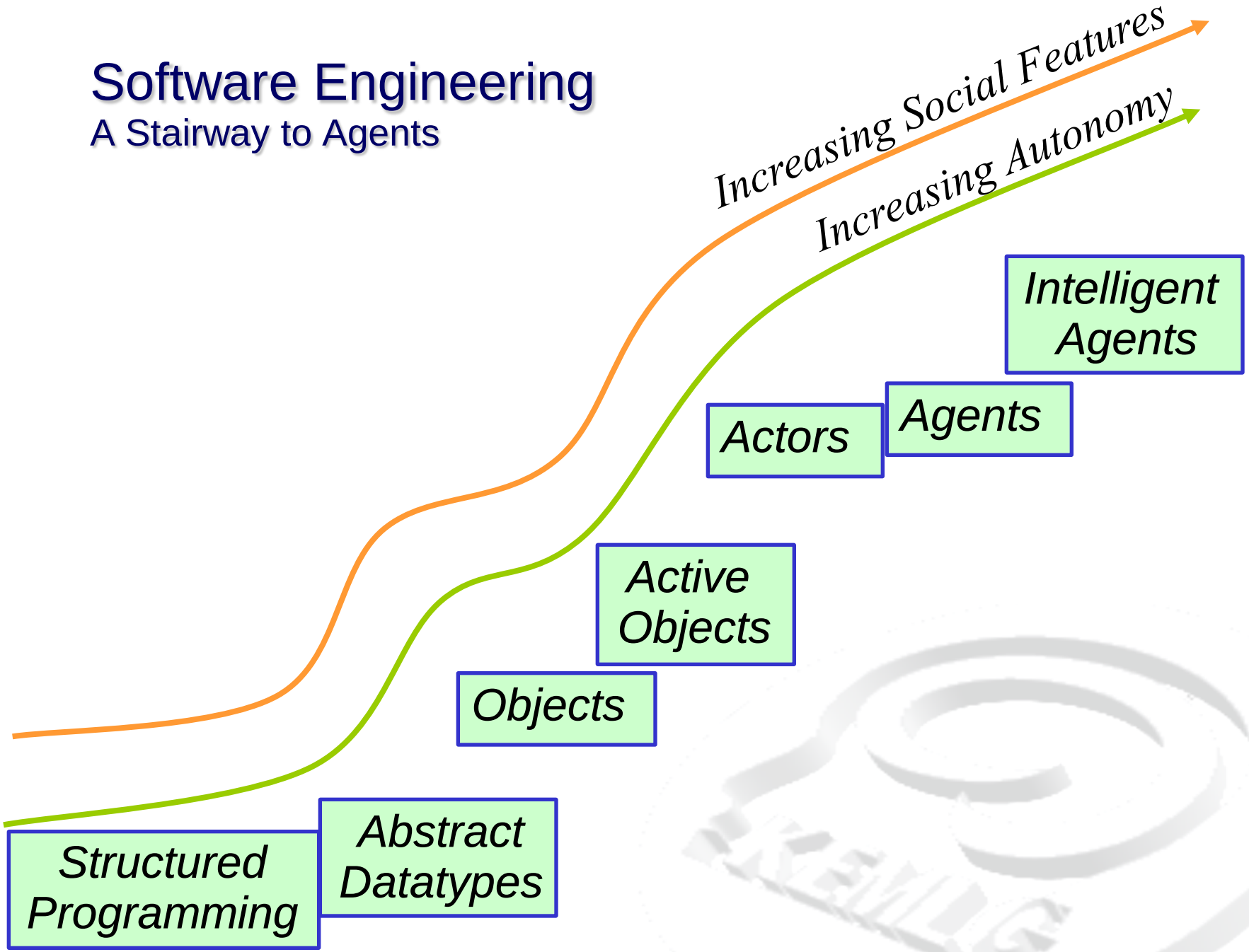
-- Grady Booch

- Maybe the agent community would like to reply...

- A lot of research work has been done to define what an agent and a MAS are, how they compare to object-oriented concepts and which their distinguishing features are
- Agent-Oriented (AO) paradigm *subsumes* the concepts supported by the previous programming paradigms, and in particular by the object-oriented programming
  - Tries to *raise the abstraction level*
  - Software agents are undoubtedly more than a promising approach to *complex software development*

Software Engineering

A Stairway to Agents



Agent-Oriented Software Engineering
Abstractions

- The development of a multiagent system should fruitfully exploit higher level abstractions
  - Agents, autonomous entities, independent loci of control, situated in an environment, interacting with each others
  - Environment, the world of entities and resources agents perceive, control, consume or exploit.
  - Roles and interactions: identify functionalities, activities, responsibilities and interaction patterns.
  - Organizational Rules, which can be constraints on roles and interactions, or relations between roles, between protocols, and between roles and protocols (open/close systems)
  - Organizational Structures and Patterns: Identify the topology of interaction patterns and the control regime of activities (efficiency, robustness, degree of openness)

Agent-Oriented Software Engineering

Agent-Oriented Computing

- There has been some debate
  - On what an agent is, and what could be appropriately called an agent
- Two main viewpoints in agent development
  - The (strong) artificial intelligence viewpoint
    - A multi-agent system is a society of individual (AI software agents) that interact by exchanging knowledge and by negotiating with each other to achieve either their own interest or some global goal
  - The (weak) software engineering viewpoint
    - A multi-agent system is a software system made up of multiple independent and encapsulated loci of control (i.e., the agents) interacting with each other in the context of a specific application

Agent-Oriented Software Engineering

Software Engineering Viewpoint on AO Computing

- The second view is useful because:
  - It focuses on the characteristics of agents that have impact on software development
    - Concurrency, interaction, multiple loci of control
    - Intelligence can be seen as a peculiar form of control independence; conversations as a peculiar form of interaction
  - It is more general:
    - Several software systems, even if never conceived as agents-based one, can be indeed characterized in terms of weak multi-agent systems

Agent-Oriented Software Engineering

Key Characteristics of Agents

- Basic characteristics (SE Viewpoint)
  - Autonomy & Proactivity (*delegation of responsibility)*
  - Situatedness
  - Interactivity (*communication, c*ollaborative or competitive interactions)
- Additional characteristics (SE Viewpoint)
  - Openness (need of standards; need of proper infrastructures supporting the interoperations)
  - Learning & Adaptative Capabilities (Improving the effectiveness of its actions; adapting their behaviour to changing situations)

Agent-Oriented Software Engineering

There is more to Agent-Oriented Software Engineering

- AOSE is <u>not only</u> for "agent systems."
  - Most of today's software systems have characteristics that are very similar to those of agent and multiagent systems
- AOSE is suitable for a wide class of scenarios and applications

Agent-based computing, and the abstractions it uses, represent a new and general-purpose software engineering paradigm

Software Methodologies

- A methodology for software development…
  - is intended to *discipline the development*
  - defines the abstractions to use to model software
    - Data-oriented, flow-oriented, object-oriented, …
    - Defines the mindset of the methodology
  - disciplines the software process
    - What to produce and when
    - Which artefacts to produce

Agent-Oriented Methodologies

- There is need for agent-oriented methodologies
  - Centred around specific *agent-oriented abstractions*
  - The adoption of OO methodologies would produce mismatches
    - Classes, objects, client-servers: little to do with agents

- Each methodology may introduce further abstractions
  - Around which to model software and to organize the software process
    - E.g., roles, organizations, responsibilities, beliefs, desires and intentions, …
  - Not directly translating into concrete entities of the software system
    - E.g., the concept of role is an aspect of an agent, not an agent

Agent-Oriented Methodologies

'Classical' Agent-Oriented Methodologies

- Several methodologies and approaches for designing MASs exist in literature. In general they tackle different aspects of the MAS and in some cases they are quite complementary:
  - **GAIA**
    - Encourages a developer to think of building agent-based systems as a process of *organisational design.*
  - **TROPOS**
    - It is founded on the concepts of *goal-based requirements* adopted from the i* and GRL (Goal-oriented Requirements Language). Its distinguishing feature is the emphasis on *requirements analysis*
  - **Prometeus**
    - It focuses mainly on the *internal agent architecture*; it is basically a methodology for designing BDI agent systems
  - **ADELFE**
    - It is a methodology for the development of *adaptive* multiagent systems
  - **MESSAGE**
    - It covers most of the fundamental aspects of the MAS development, focusing mainly on analysis and high-level design. The main objective was to combine the best features of the pre-existing approaches, but … the result was a complex and farraginous methodology.
  - **PASSI**
    - It is a step-by-step requirement-to-code methodology. Integrates design models and concepts from both object oriented software engineering and artificial intelligence approaches

Agent-Oriented Software Engineering
Phases

- **Analysis** aims to understand, at least
  - What are the main actors interacting with the system
  - How the system interacts with these actors
  - What the system is supposed to do
- The system is a closed entity and we do not look into it to avoid anticipating design issues and decisions
- *In AO, we associate agents with the entities of the scenarios we are analyzing*
- Then, we associate accordingly
  - Roles, responsibilities and capabilities
  - Interaction patterns between agents
- This provides a neutral view of the problem.
- Methodologies such as Tropos and GAIA, do not use the word agent to identify analysis-phase entities

Agent-Based Software Engineering
Phases

- **Design** aims to engineer, at least
  - What are the main components interacting within the system
  - What are the responsibilities and the capabilities of each component in the system
  - How the components interact to implement the system, i.e., the architecture of the system
- *In AO, we associate agents with the components we use to build the system*
- Then, we refine accordingly
  - Roles, responsibilities and capabilities
  - Interaction patterns between agents
- Differently from analysis: we need to choose on which agents to use and how they interact

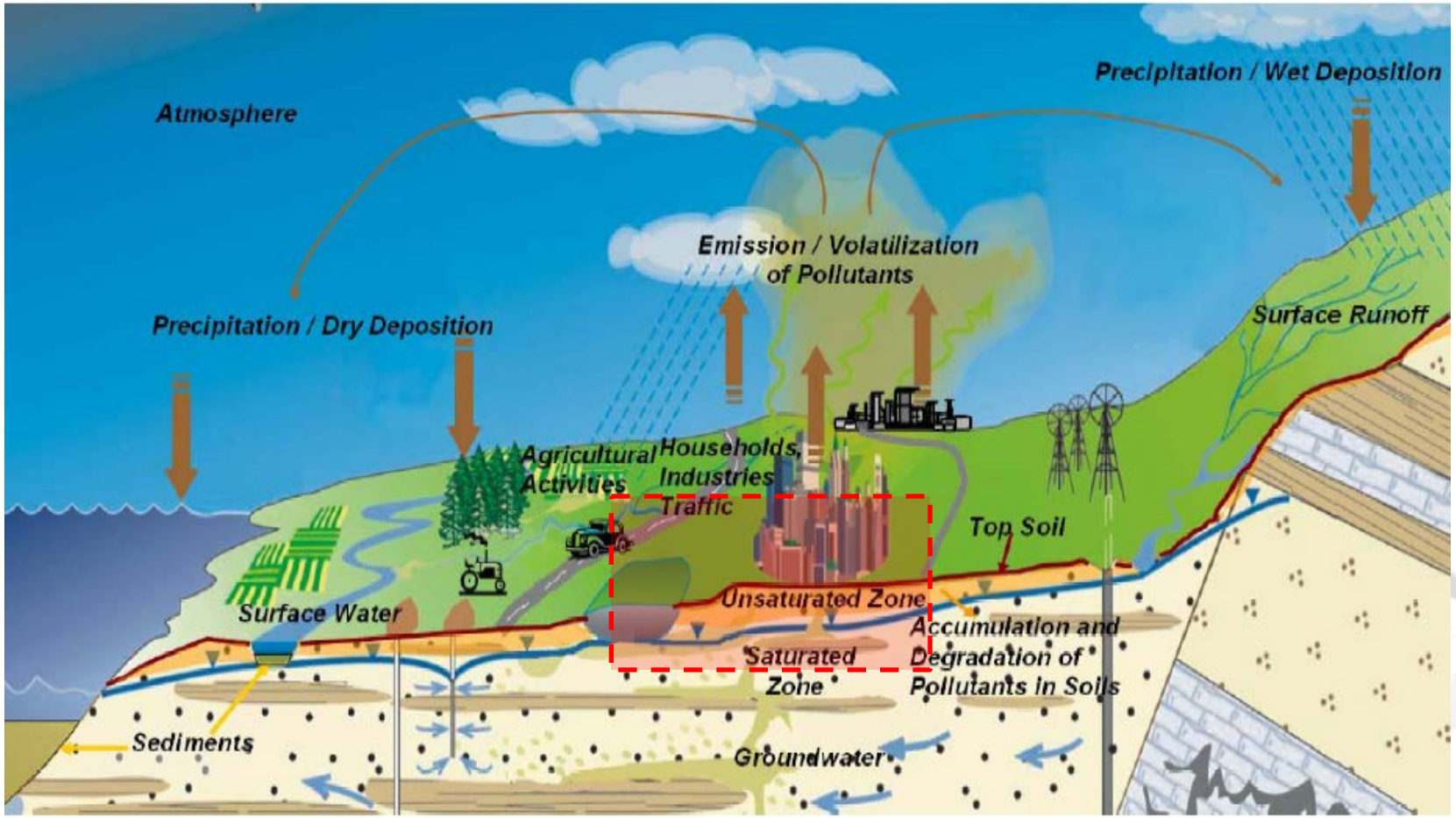
Methodological Extensions to Object-Oriented Approaches

- A means for agent technologies to gain traction within industrial settings may be by being introduced through well-established technologies
- The Unified Modeling Language (UML) is gaining wide acceptance for the representation of engineering artifacts using the object-oriented paradigm
- There are several attempts to extend UML so as to encompass agent concepts
- In general, building methods and tools for agent-oriented software development on top of their object-oriented counterparts seems appropriate
  - It lends itself to smoother migration between these different technology generations
  - It improves accessibility of agent-based methods and tools to the object-oriented developer community which, as of today, prevails in industry.

Designing a Multiagent System

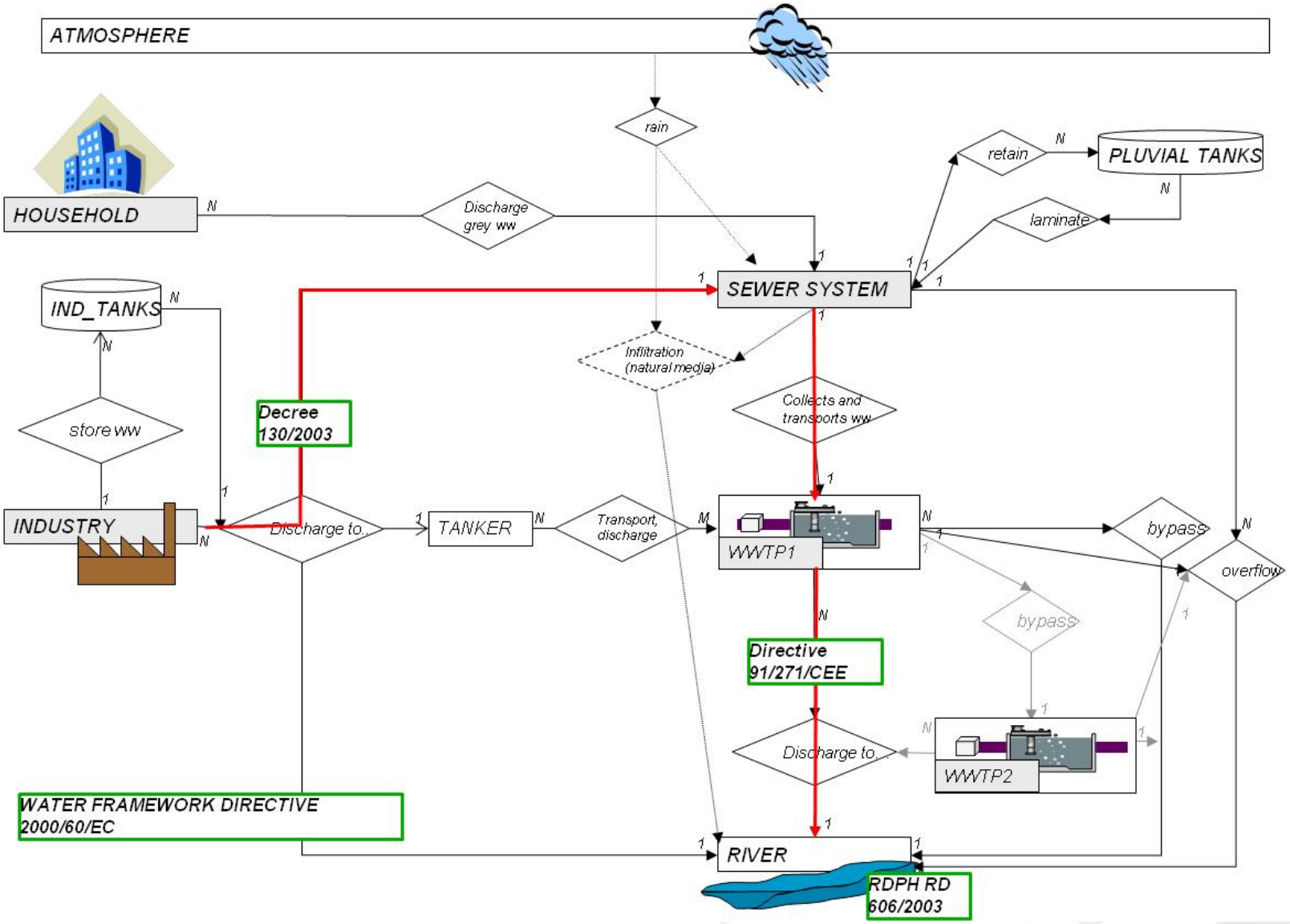
Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
http://www.kemlg.upc.edu

Use case: Distributed River Basin Management



Use case: Distributed River Basin Management

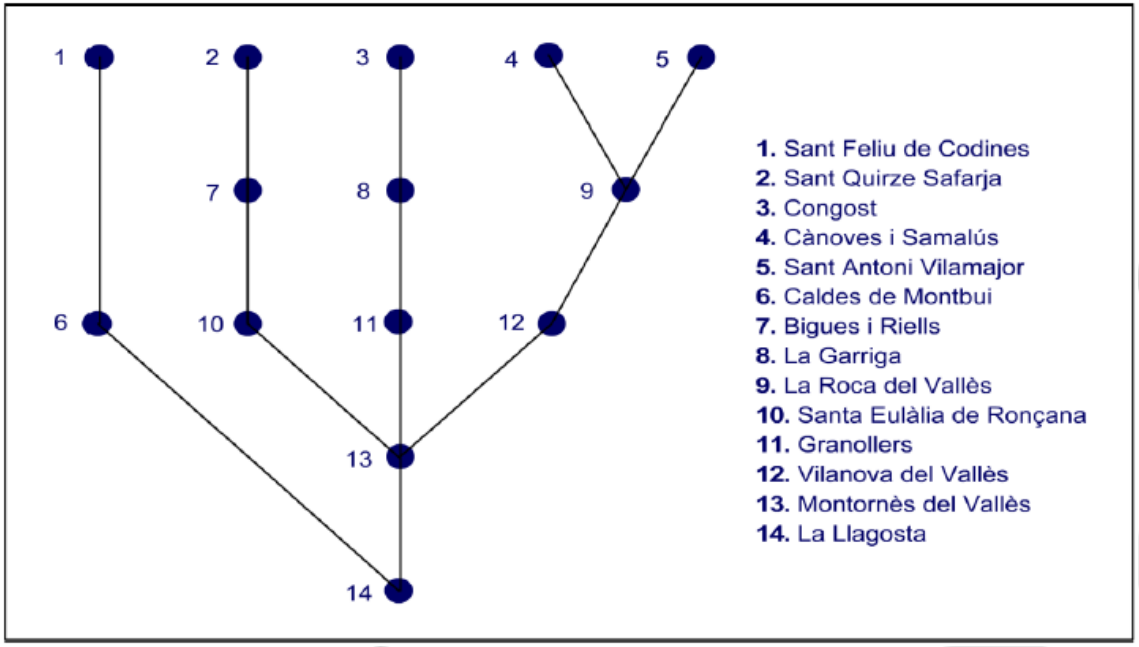
Process description and regulations



Use case: Distributed River Basin Management

Distributed nature of the problem

- Idea: to build a MAS to coordinate the operation of the 14 Waste Water Treatment Plants (WWTP) located in the Besos River



Chosen AO Methodology: Prometheus

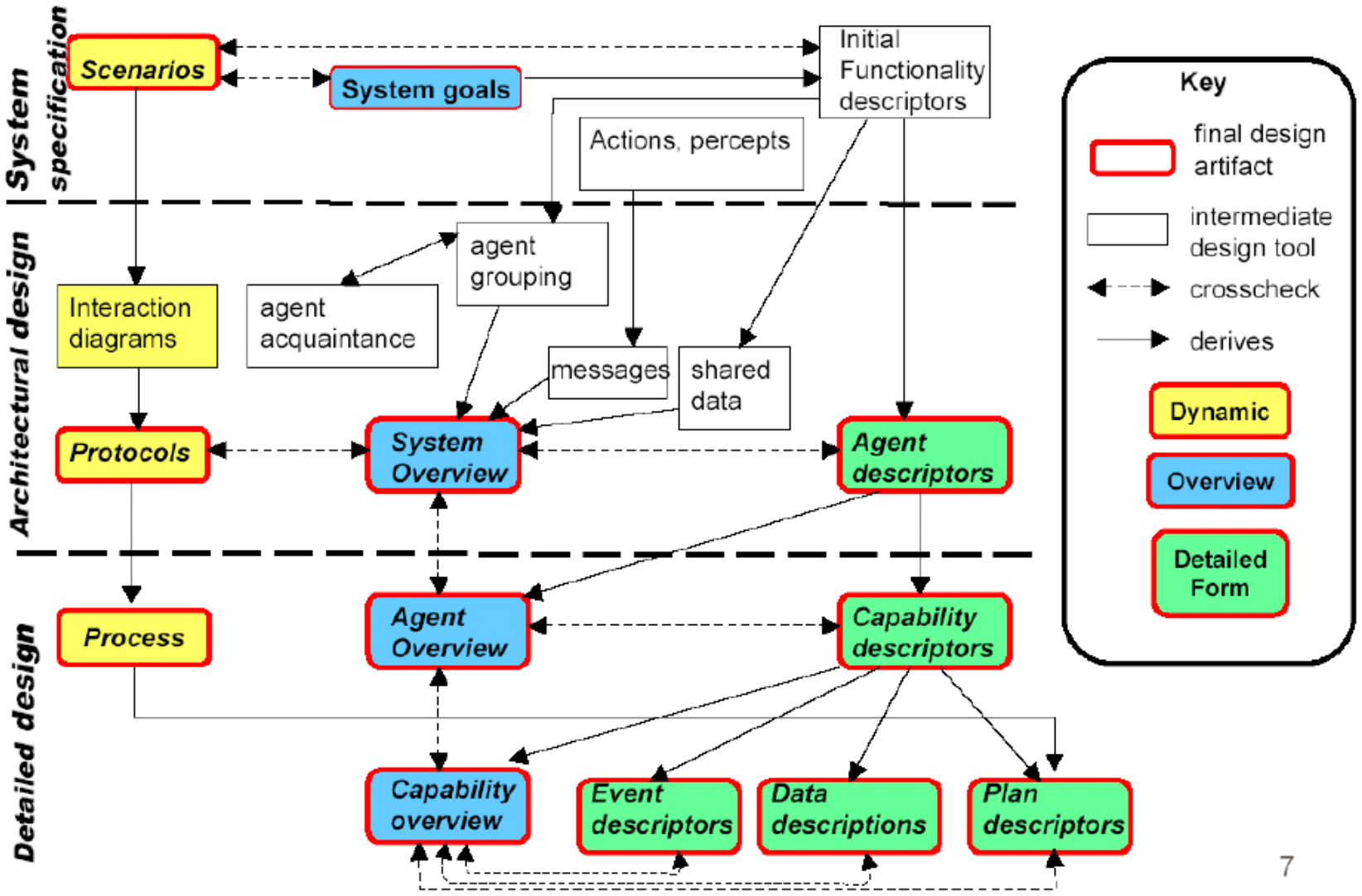
- Prometheus is an iterative methodology covering the complete software engineering process
  - Analysis, Design, Detailed design, Implementation
- Aims at the development of intelligent agents (in particular BDI agents)
  - Uses goals, beliefs, plans, and events.
- The resulting specification can be implemented in any agent implementation software that covers such abstractions
  - Specially aimed for implementation with JACK
- It is evolved out of practical experiences
- It is aimed at industrial software development, not researchers

Prometheus
Phases

- The Prometheus methodology covers three phases
  - The **system specification** focuses on identifying the basic functions of the system, along with inputs (percepts), outputs (actions) and their processing (for example, how percepts are to be handled and any important shared data sources to model the system's interaction with respect to its changing and dynamic environment)
  - The **architectural design phase** subsequent to system specification determines which agents the system will contain and how they will interact
  - The **detailed design phase** describes the internals of each agent and the way in which it will achieve its tasks within the overall system. The focus is on defining capabilities (modules within the agent), internal events, plans and detailed data structures.

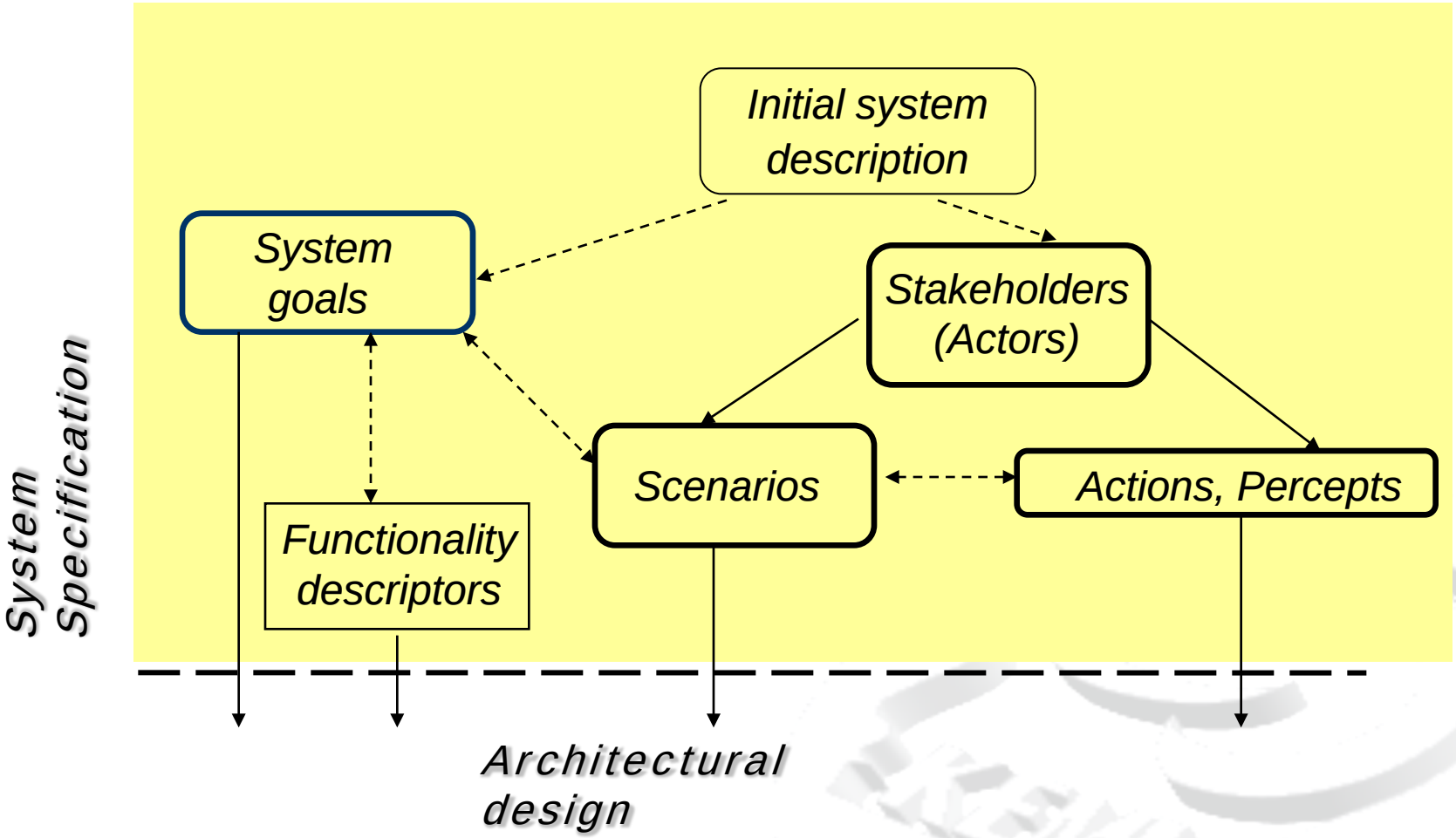
Prometheus

Process Overview



Prometheus

System Specification Phase



Prometheus

System Specification phase

- System defined by
  - Stakeholders: actors
  - Goals: *goal diagram*
  - Scenarios: *user case scenarios*
  - Functionalities: *functionality descriptors*
- System interface with environment described in terms of
  - actions,
  - percepts
  - external data

Note: Most of the MAS design showed in the following slides was made by CAROLINA RUBIO, ATIA CORTÉS and FRANCESC TRAVESA

STAKEHOLDER	PERFORMANCE	ENVIRONMENT	PERCEPTS	ACTIONS
Sewer System	- Drainage and transport of rainfall water, industrial and household ww[2] to the receiving media or to a WWTP - Flood protection	- Sewage transportation - Rain overflows - Pollution episodes	- Detect inflow of rainfall water - Detect inflow of household water	- Temporal storage of rainfall water - Check availability of the WWTPs - Collect and transport water to the pluvial tanks or to the WWTPs
WWTP	- Recycle sludge - Treat ww to be returned to the river according to the Catalan Sanitation Plan	- Urban ww treatment	-Detect quantity of produced ww - Percept the chemical/toxic components of the water	- Collect and transport ww to others WWTPs - Collect and transport water to the river - Check if water treatment is feasible - Apply water treatment - Bypassing water between WWTPs
Industry	- Obey the Catalan Sanitation Plan - Collect polluting wastewater for its processing	- Pollutants transference	- Detect quantity of produced ww	- Collect and transport ww to the sewer system, industrial tanks, TANKER or river - Inform about toxic effluent dumped
River	- Maintenance of an acceptable quantity and quality of the water	- Receiving water	- Percept the chemical/toxic components of the water	- Process quality and quantity data to warn about fraudulent dumps into the river

System Specification phase

Scenarios (1 of 2)

SCENARIO 1	Treat industry's uncontrolled toxic effluent
OVERVIEW	An industry has an uncontrolled toxic effluent and warns authorities
CONTEXT	The toxic concentration in the industrial wastewater is unexpectedly high
STEPS	1. *(Percept)* Detect quantity of produced wastewater 2. *(Action)* Collect and transport ww into industrials tanks 3. *(Action)* Inform about toxic effluent dumped 4. *(Action)* Collect and transport ww into the TANKER 5. *(Action)* Check availability of the WWTPs 6. *(Action)* Collect and transport ww into the suitable WWTP 7. *(Action)* Apply WW treatment
VARIATIONS	*Step 4*. Collect and transport WW to the sewer system

SCENARIO 2	Process a non-notified toxic effluent
OVERVIEW	A wastewater treatment plant detects a non-notified toxic effluent and needs help to process it
CONTEXT	It is not mandatory for the UWS to have a treatment for the toxic effluent
STEPS	1. *(Percept)* Percept the chemical/toxic components of the water 2. *(Action)* Check if water treatment is feasible 3. *(Action)* Bypassing water between WWTPs 4. *(Action)* Apply WW treatment
VARIATIONS	*Step 3*. Apply the primary treatment to the wastewater

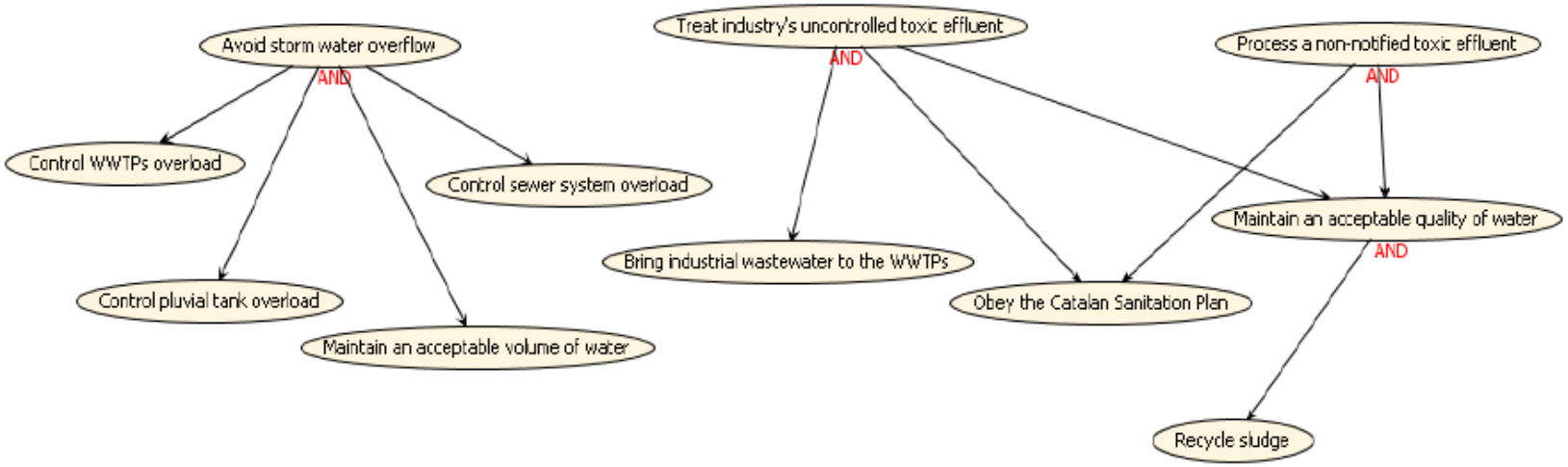
System Specification phase

Scenarios (2 of 2)

SCENARIO 3	Avoid storm water overflow
OVERVIEW	There is a mild thunderstorm and a waste water treatment plant cannot process all the in-flow alone, and needs to coordinate with others
CONTEXT	The sewer system has only one pipe that collects and transports together the storm water and the different types of wastewater
STEPS	1. *(Percept)* Detect inflow of rainfall water 2. *(Action)* Collect and transport water to the pluvial tanks 3. *(Activity)* Temporal storage of rainfall water 4. *(Action)* Check availability of the WWTPs 5. *(Action)* Bypassing water between WWTPs 6. *(Action)* Collect and transport water to the river
VARIATIONS	*Step 5*. Collect and transport water to the river

System Specification phase

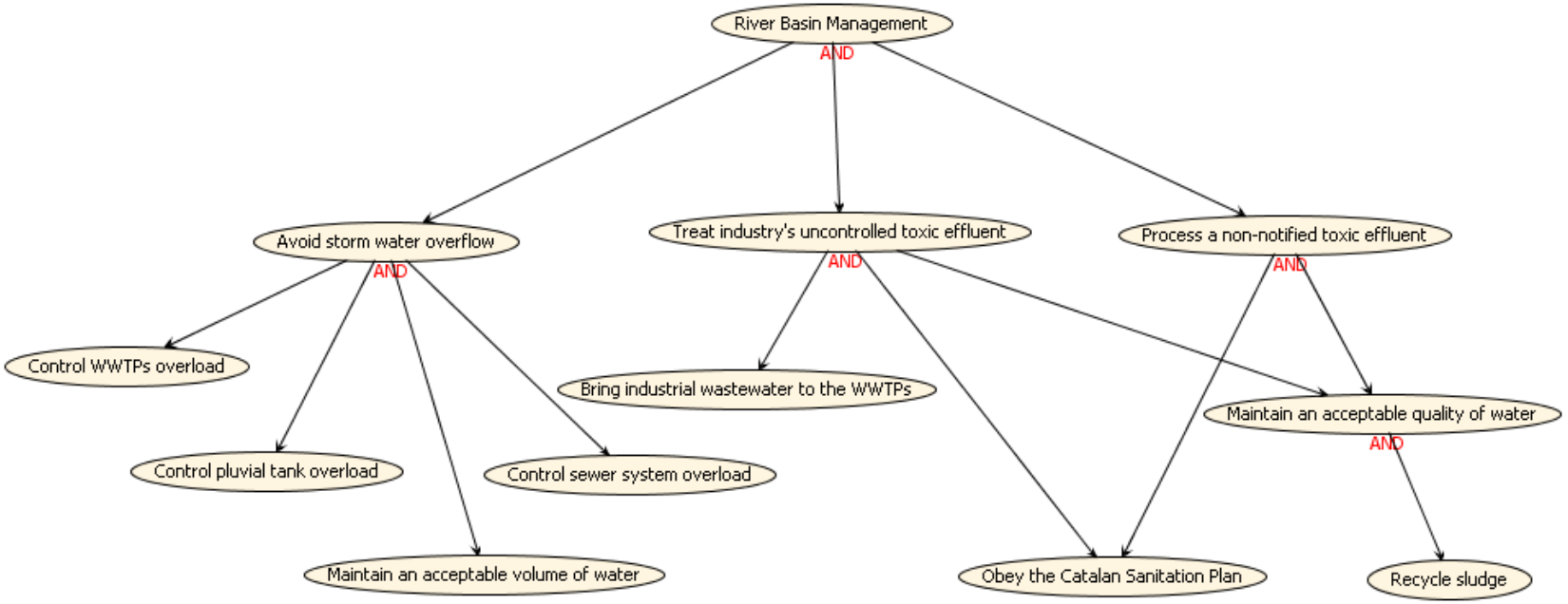
Goal Overview Diagram: first attempt

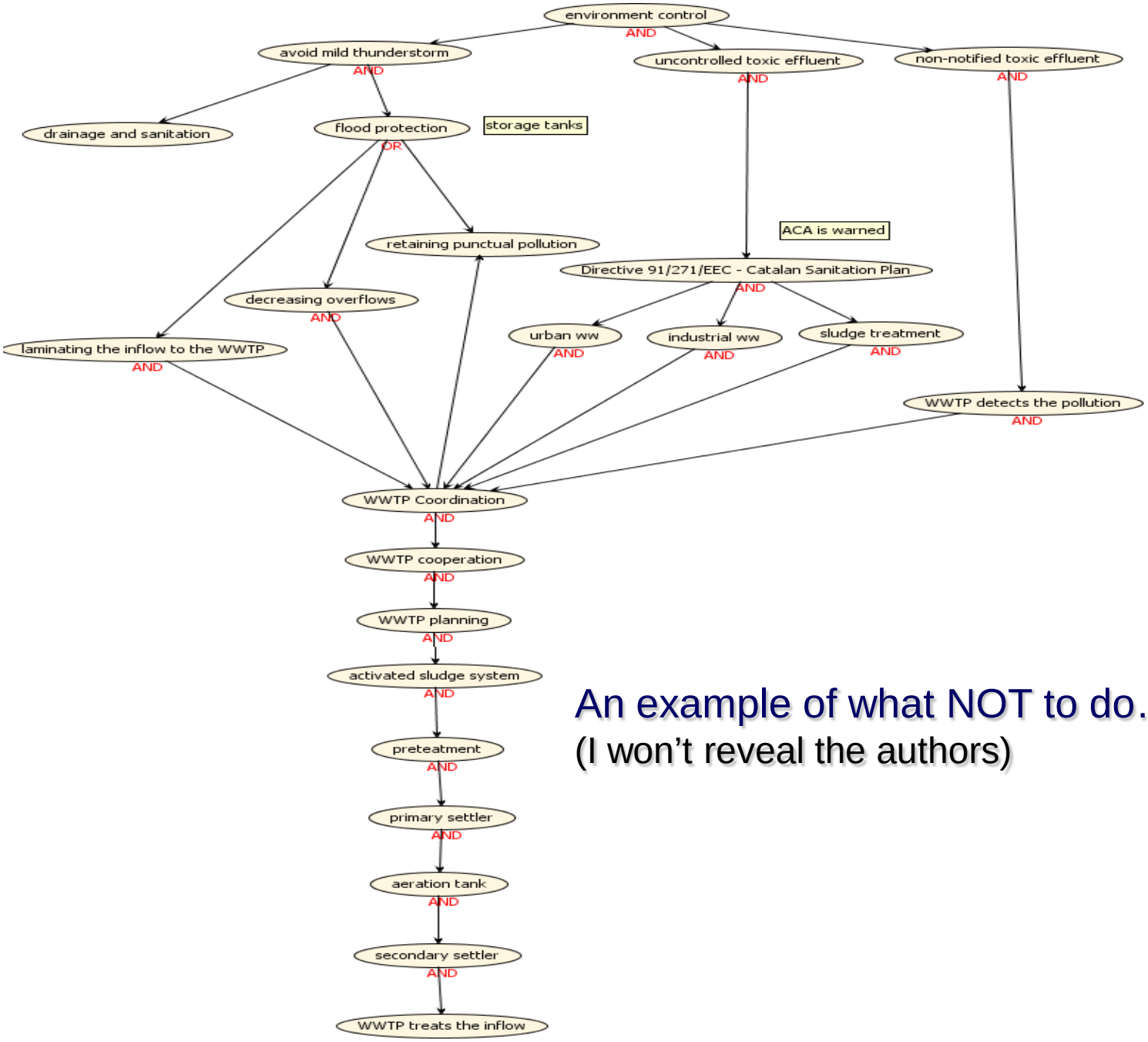


- good practices:
  Except in extreme situations, the goal diagram...
  - should be a fully-connected graph
  - The abstraction level should be balanced in the different branches.
  - All (sub)goals should be linked to scenarios

System Specification phase

Goal Overview Diagram: improved



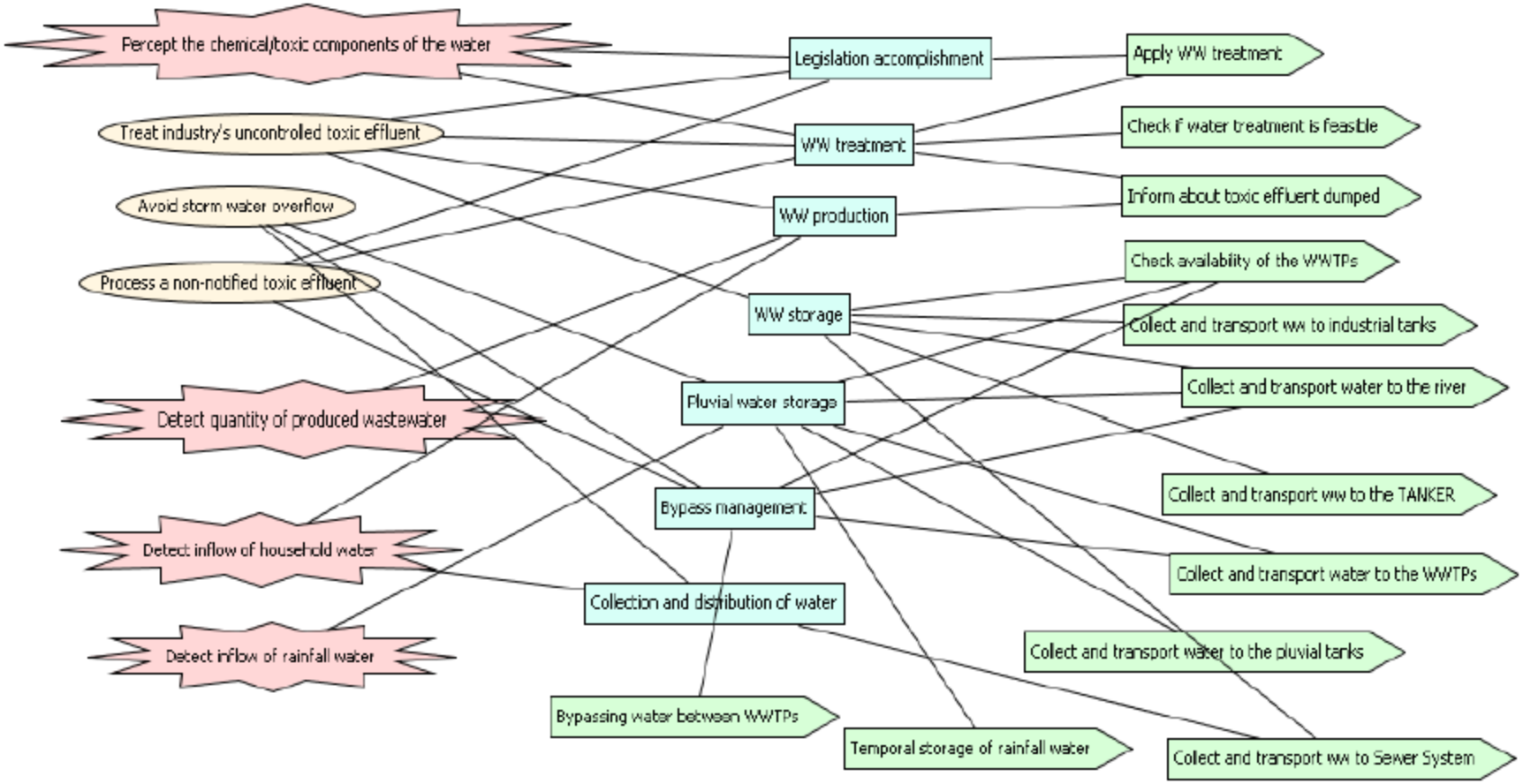


An example of what NOT to do…

(I won't reveal the authors)

System Specification phase

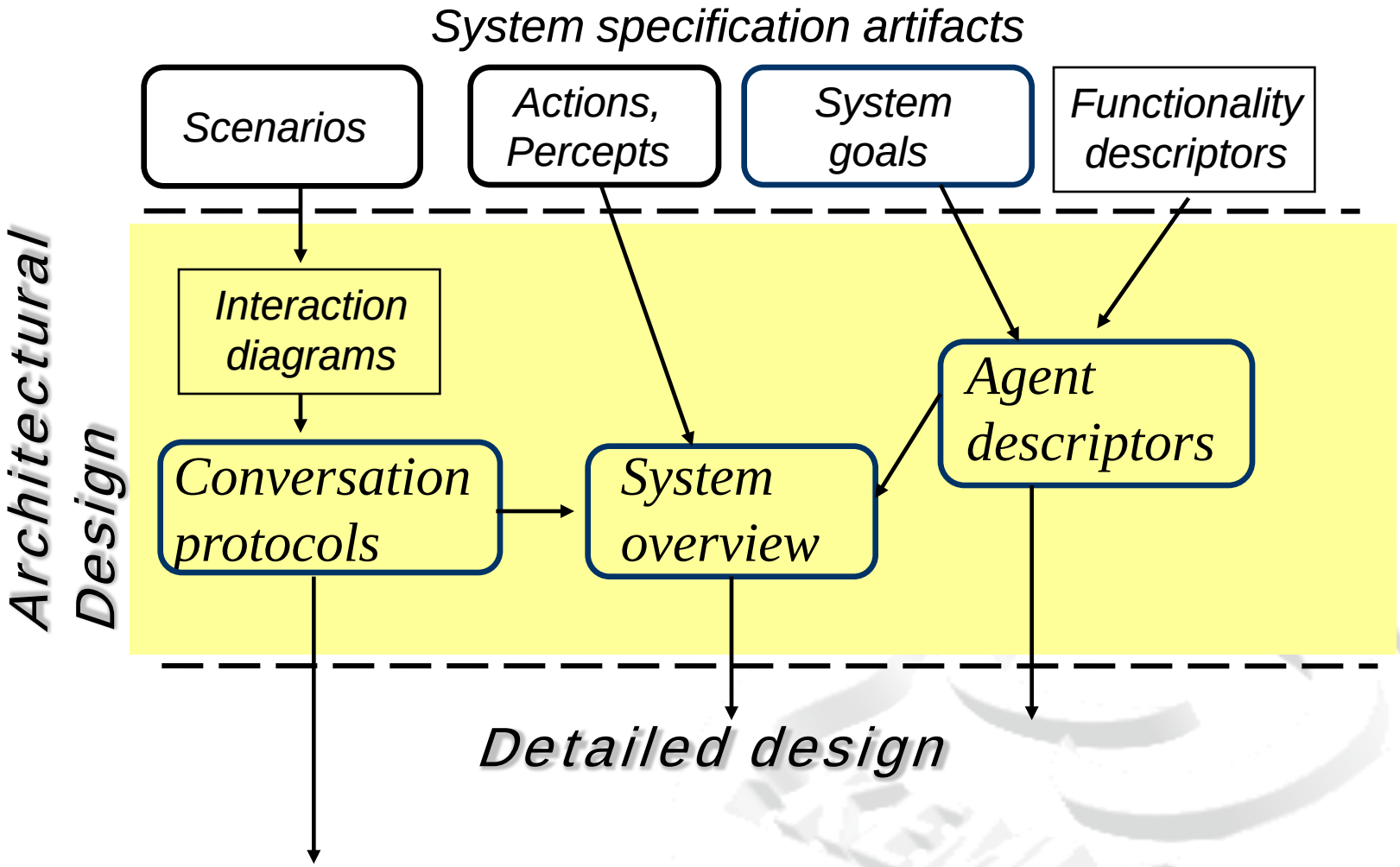
System Roles Diagram



- A good practice:
  Keep roles/activities small and specific, so later during design phase you have more flexibility to group them into agents

Prometheus

Architectural Design Phase



Prometheus

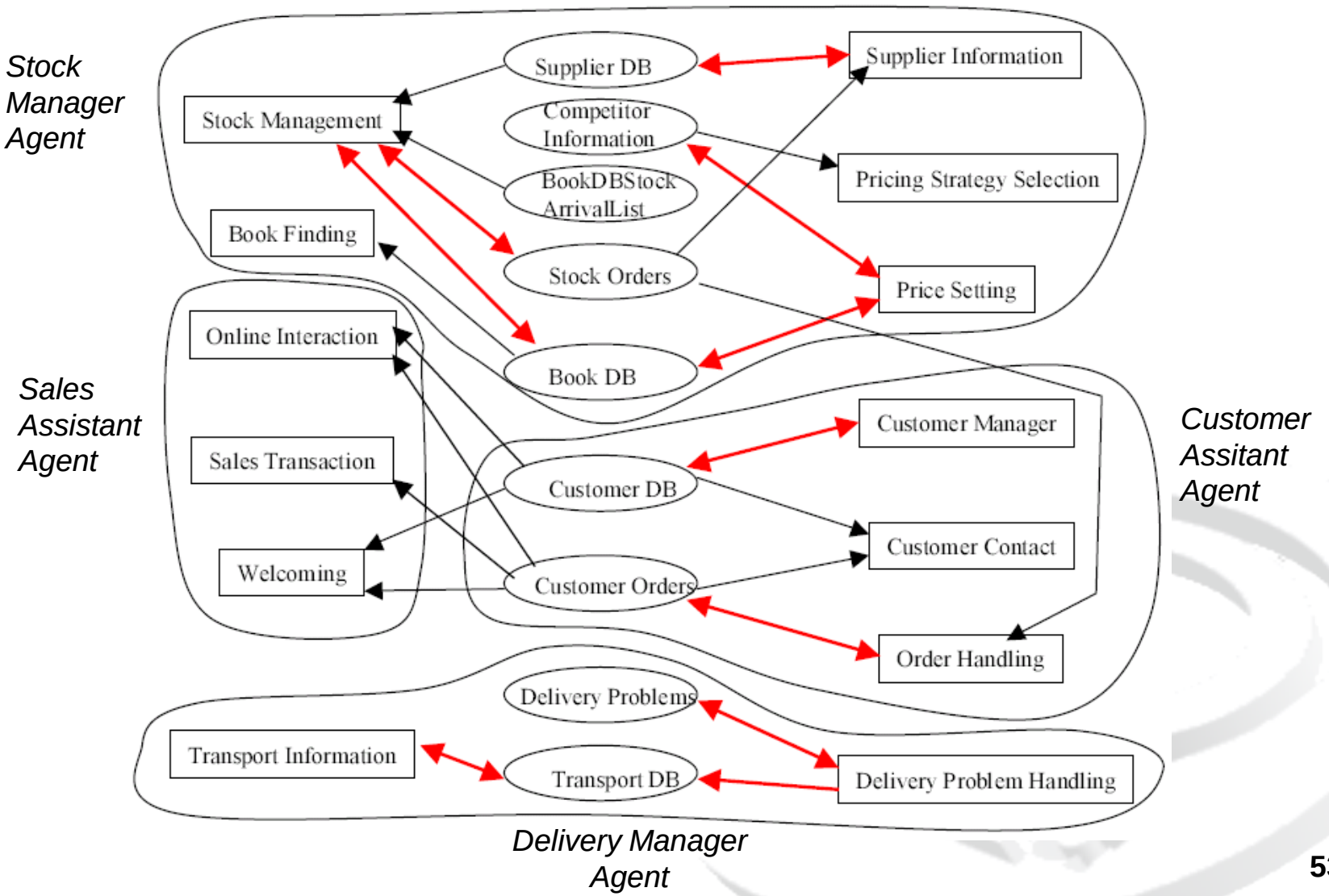
Architectural Design Phase: Identifying Agent types

- option 1: There exists no definition in the environment → we have to identify them
  - Group functionalities to agent types based on cohesion and coupling
  - Group functionalities that are
    - related based on common sense
    - group functionalities that require a lot of the same information:
      - *Data Coupling Diagram*
  - Do not group functionalities that are
    - clearly unrelated
    - exist on different hardware platform
    - security and privacy
    - Modifiable by different people
  - Evaluate grouping:
    - Simple descriptive names (heuristic)
    - Generate agent acquaintance diagram
- option 2: The domain already defines the agent types

Identifying agent types

Option 1: The domain does not define stakeholder types/roles

Example from the Prometheus creators (On-line book store)



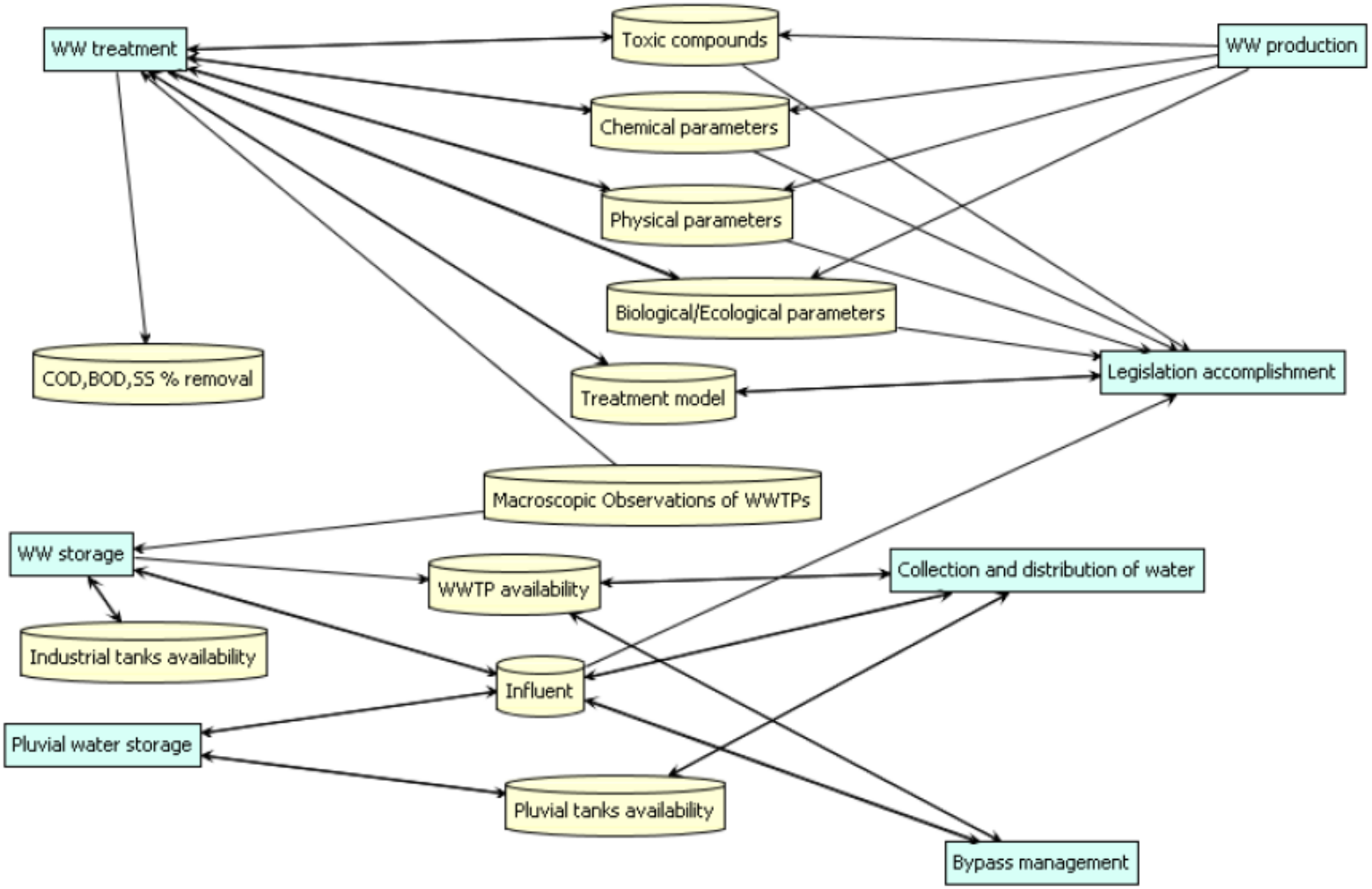
Identifying agent types

Option 1: Example of Agent Descriptor for the on-line Book Store

Name: Sales Assistant agent
Description: greets customer, follows through site, assists with finding books
Cardinality: one/customer.
Lifetime: Instantiated on customer arrival at site. Demise when customer logs out or after inactivity period.
Initialisation: Obtains cookie. Reads Customer DB.
Demise: Closes open DB connections.
Functionalities included: Online Interaction, Sales Transaction, Welcomer, Book Finder.
Uses data: Customer DB, Customer Orders, Book DB.
Produces data: Customer preferences, orders, queries
Goals: Welcome customer; Update customer details; Respond to queries; Facilitate purchases;
Events responded to: new arrival; customer query; customer purchase; credit check response customer response;
Actions: Display information to customer (greetings, book info, info requests, Display customised WWW page, RequestCreditCheck messages
Interacts with: Warehouse Manager (book request protocol), Delivery Manager (order protocol, order query protocol), Customer Manager (customer information query protocol, customer information update protocol)

Identifying agent types

Option 2: These are predefined in the domain



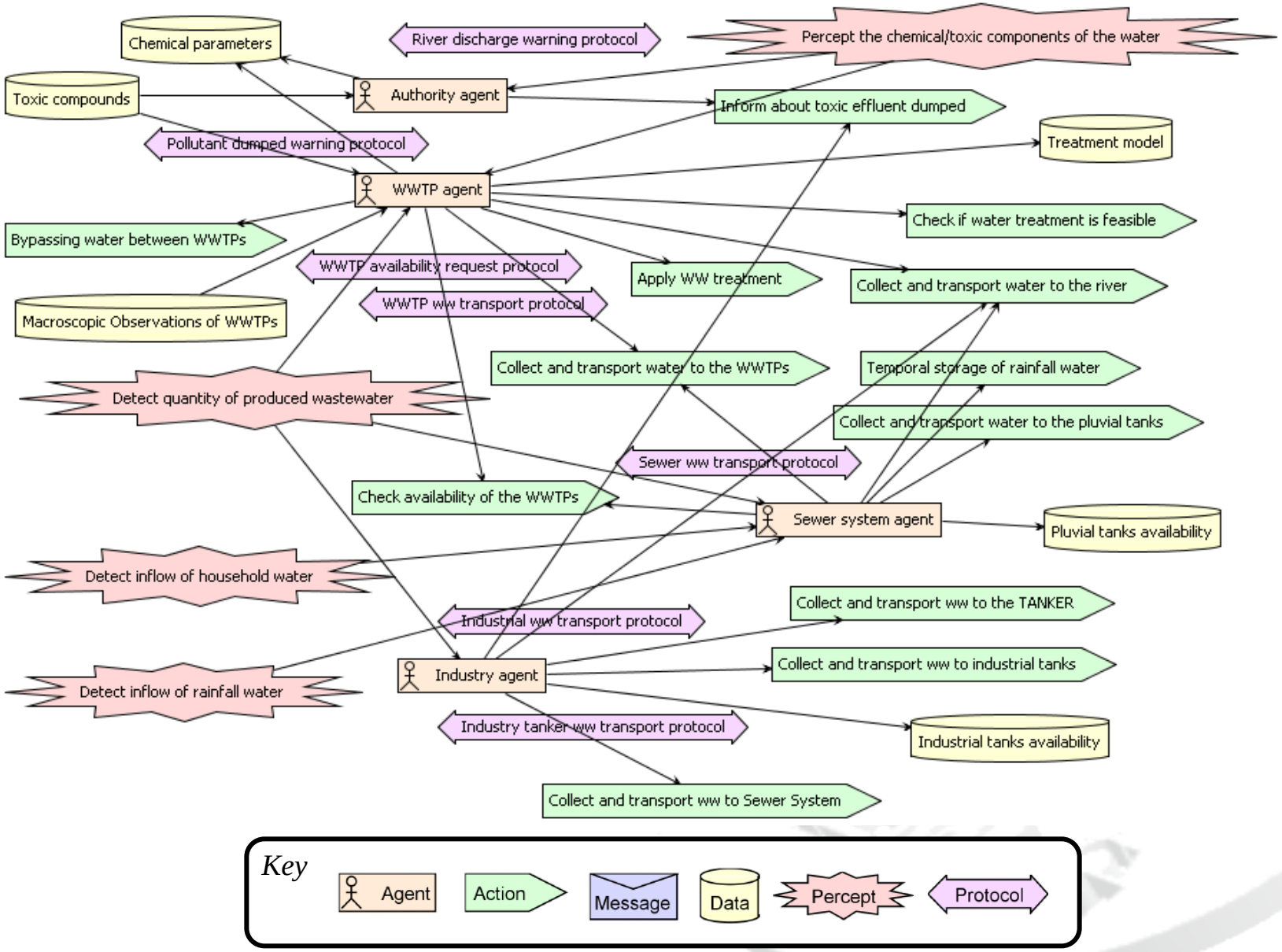
Identifying agent types

Option 2: Example of Agent Descriptor for the Besos River scenario

AGENT	WWTP AGENT
DESCRIPTION	Collection of treatment plants where wastewaters are treated chemically for its sanitation and discharged into the river
CARDINALITY	N (as many treatment plants as the UWS considers)
LIFETIME	Instantiated when in-flow of urban wastewater needs to be discharged for treatment
FUNCTIONALITIES INCLUDED	WW production, Collection and distribution of water, WW treatment, Bypass management
USES DATA	Influent, WWTP availability, Toxic compounds, Chemical parameters, Physical parameters, Biological/Ecological parameters, Treatment model, Macroscopic observation of WWTPs
PRODUCES DATA	Toxic compounds, Chemical parameters, Physical parameters, Biological/Ecological parameters, Influent, WWTP availability, Treatment model, COD BOD SS % removal
GOALS	Maintain an acceptable volume of water, Control WWTPs overload, Obey the Catalan Sanitation Plan, Recycle sludge, Maintain an acceptable quality of water
EVENTS RESPONDED TO	Detect quantity of produced wastewater, Percept the chemical/toxic components of the water
ACTIONS	Apply WW treatment, Check if water treatment is feasible, Check availability of the WWTPs, Collect and transport water to the WWTPs, Collect and transport water to the river, Bypass water between WWTPs
INTERACTS WITH	WWTP Agent (*WWTP availability request protocol, WWTP ww transport protocol*), Industry Agent (*Industry tanker ww transport protocol*), Sewer System Agent (*Sewer ww transport protocol*)

Prometheus

Architectural Design Phase: System Overview Diagram



Design Tip: When agent communication?

- Any protocol interaction should come from some agent communication needs.
- Goals for Agent Communication:
  - Agents able to request (to other ags.) actions or services that they cannot perform by themselves
  - Agents able to ask for information (to other ags.)
  - Agents able to share their beliefs with other ags.
  - Agents able to coordinate with other ags. To solve complex tasks.

- Design Tip:
  - In Prometheus any protocol interaction should be connected to a (sub)goal.

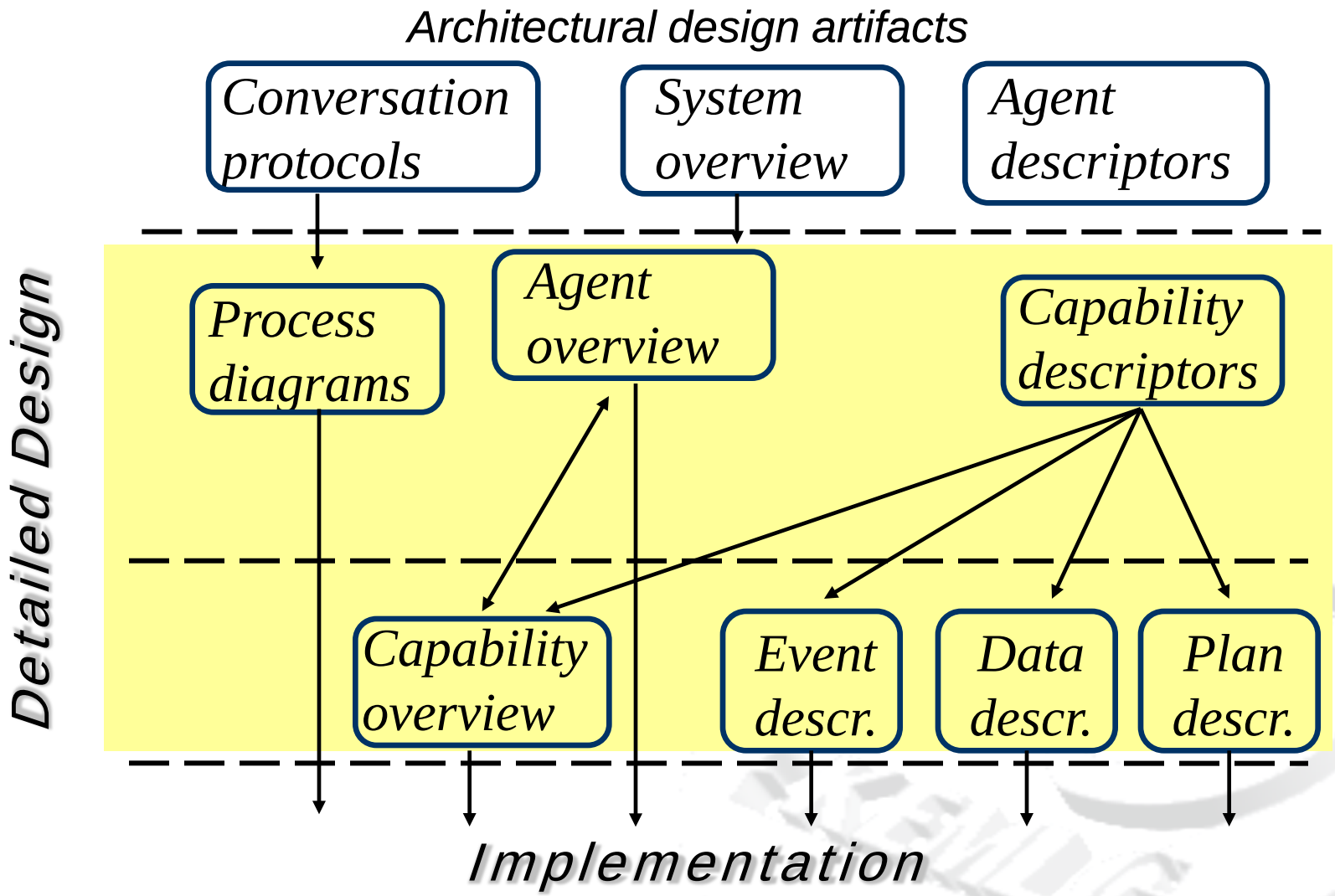
Prometheus

Architectural Design Phase: Protocol description

PROTOCOL	DESCRIPTION	AGENTS INVOLVED
Sewer ww transport	Transports ww from the Sewer system to the nearest WWTP	Sewer System -> WWTP
River discharge warning	Notifies the Authority that the Sewer System discharges ww to the river	Sewer System -> Authority
WWTP ww transport	Sends ww from a WWTP to another	WWTP -> WWTP
WWTP availability request	Requests an available WWTP to treat a toxic effluent	WWTP <-> WWTP Industry -> WWTP Sewer System -> WWTP
Pollutant dumped warning	Industry notifies the discharge of ww	Industry <-> Authority
Industry tanker ww transport	Non-expected ww stored in the industrial tanker transported into the WWTP	Industry -> WWTP
Industrial ww transport	ww coming from industry that pass by the Sewer System before going to the WWTP	Industry -> Sewer System

Prometheus

Detailed Design Phase



Prometheus

Detailed Design Phase

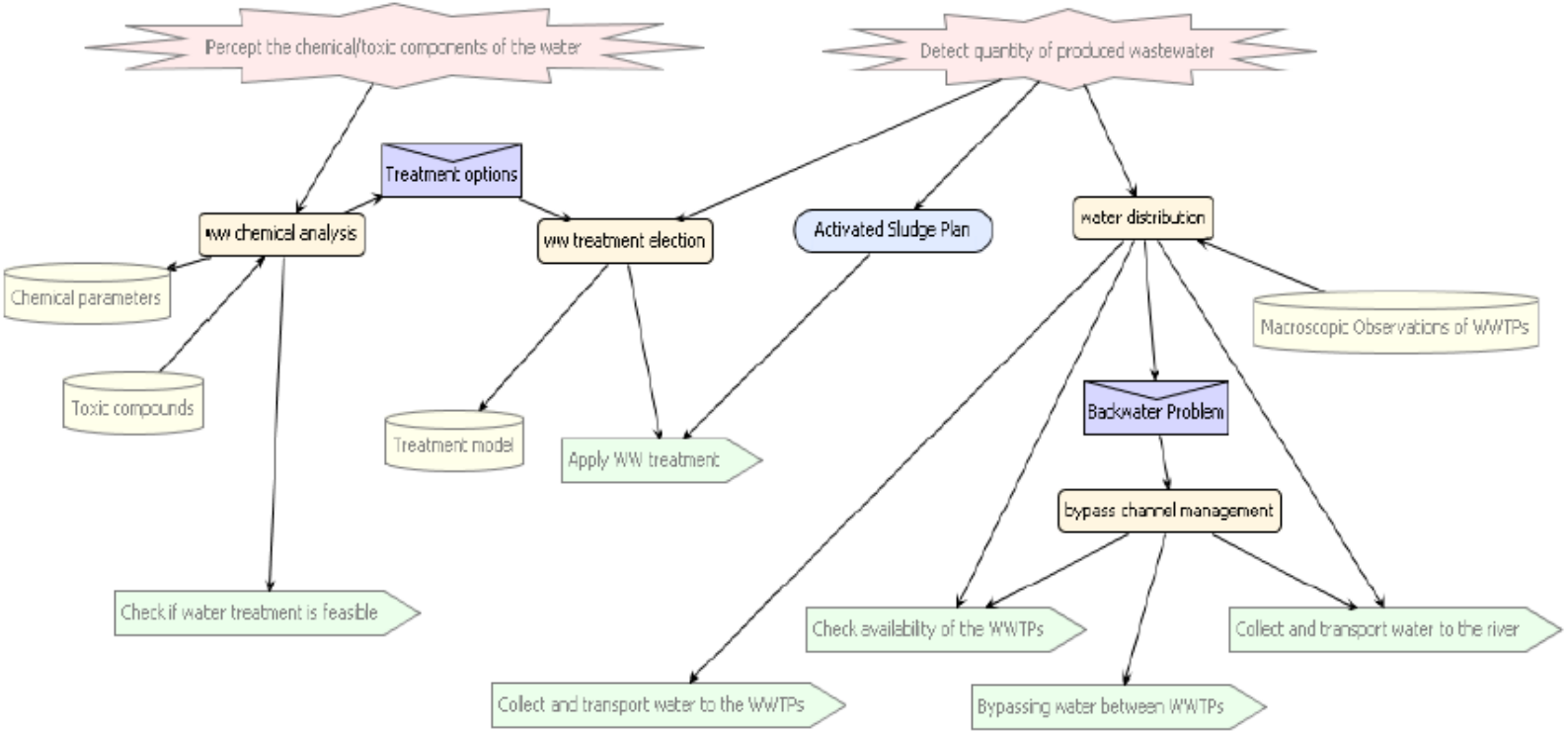
- The details of the agent internals are developed
  - Defined in terms of capabilities, data, events and plans
  - Process diagrams are used as stepping stone between interaction protocols and plans
- Steps (I)
  - Develop the internal structure of individual agents
  - Identify the capability of each agent (start with functionalities)
  - Generate *capability descriptors*

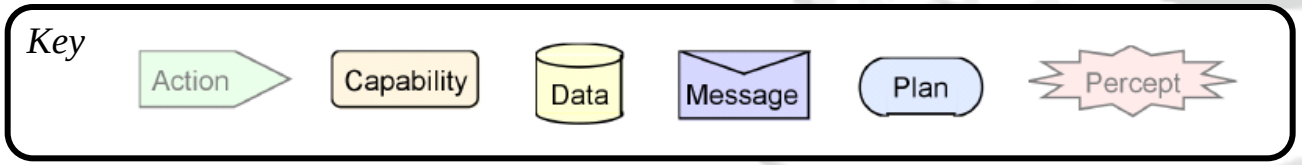
Name: Bypass channel management
External interface to the capability: events used/produced
Natural language description: Respond if books are not in stock
Interaction with other capabilities: Blackwater problem
Data used/produced by the capability: Note problem to transport capability
Inclusion of other capabilities: None

  - Generate *agent overview diagrams*

Prometheus

Detailed Design Phase: Agent Overview Diagrams – WWTP Agent





Prometheus

Detailed Design Phase: Event, Data & Plan Descriptions

- Steps (II)
  - *Plan descriptions*

Name: Activated Sludge Plan
Natural language description: Process WW with slidges in tank
Triggering event type: Detected quality of incoming water
Plan steps: Apply WW treatment
Context of performing the plan: normal functioning
Data used/produced: none

  - *Event descriptions*
    - Identify the purpose of events and the data carried by it
  - *Data descriptions*
    - Identify the data structure and operations on the data

http://www.kemlg.upc.edu/

For more material on Agent-Oriented Software Engineering you can visit my MAS course website:

http://www.lsi.upc.edu/~jvazquez/teaching/sma-upc/