The Prometheus Methodology

Chapter · January 2004

DOI: 10.1007/1-4020-8058-1_14

CITATIONS
35

READS
685

2 authors:

Michael Winikoff
University of Otago
186 PUBLICATIONS **3,930** CITATIONS

SEE PROFILE

Lin Padgham
RMIT University
208 PUBLICATIONS **3,877** CITATIONS

SEE PROFILE

The Prometheus Methodology

Michael Winikoff
(winikoff@cs.rmit.edu.au)

433-682 Intelligent Software Agents

2nd April 2004

http://www.cs.rmit.edu.au/agents

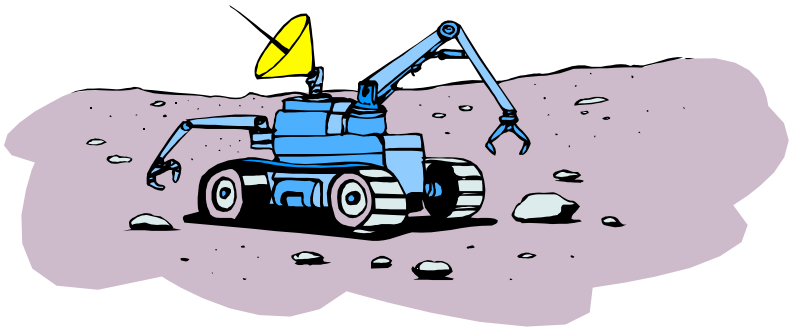
Contents of this talk

1. Brief introduction to agents (what, why)
   - because we may be assuming different definitions …
2. Prometheus: An agent-oriented software engineering methodology (why, what)
   1. Concepts
   2. Process
   3. Techniques
   4. Models
   5. Tools

"*I don't know. I'd never been that brief*." – Alan Kay
(when asked if he could give a two-hour lecture on the FLEX machine)

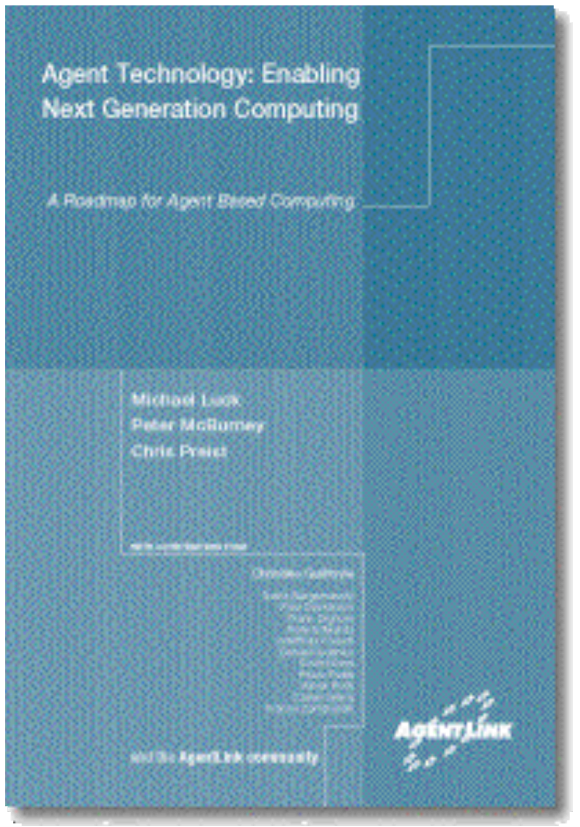
Agent?

- Agent: **autonomous**, **situated** in an environment
- An intelligent agent is **proactive**, **reactive**, **social**
- Also flexible, robust, rational
- Real world: dynamic, uncertain, non-deterministic …things *will* go wrong



Why?

"One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems." [Roadmap,page 11]



Michael Luck, Peter McBurney and Chris Preist, *Agent Technology: Enabling Next Generation Computing: A Roadmap for Agent-Based Computing*, AgentLink, 2002, available from *http://www.agentlink.org/roadmap*.

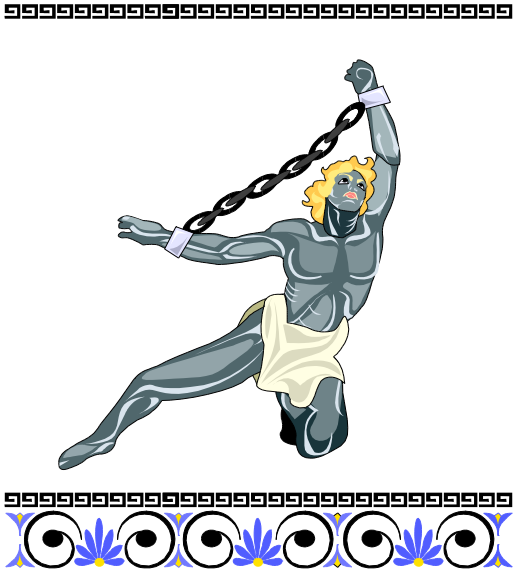
Why not Existing Methodologies?

- 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?

- Low level design differs since agent systems face uncertainty and failing actions
  - Need to have alternative plans - can't assume things will work

The Prometheus Methodology

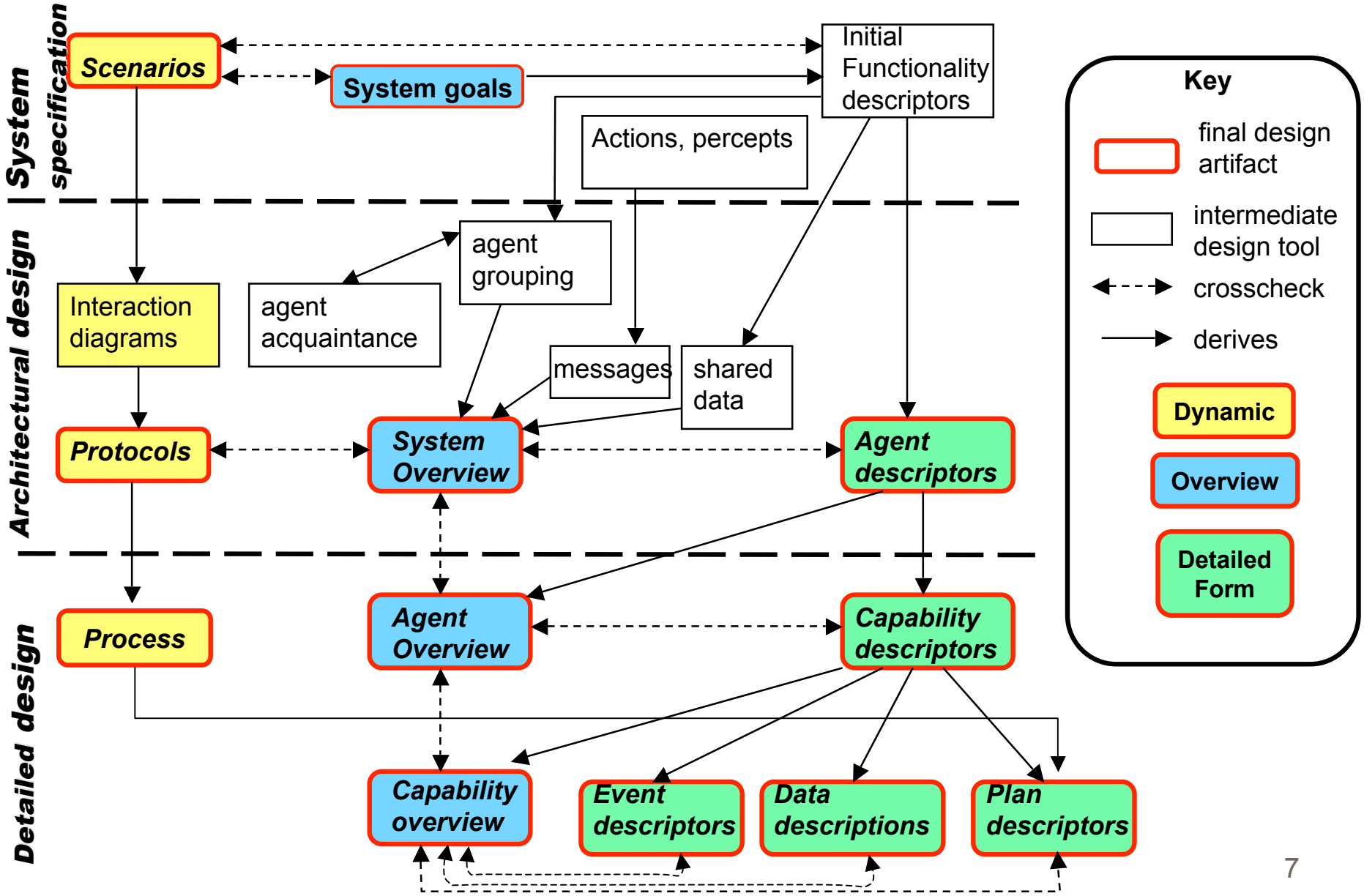
- Supports the development of *intelligent* agents
- “start-to-end” support
- Detailed process and products
- Evolved out of practical industrial and pedagogic experience
- Has been used
- Hierarchical structuring: scales to large designs
- Cross checking



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/*)

Methodology = Concepts + Process + Models/Notations + Techniques + Tool Support

Prometheus

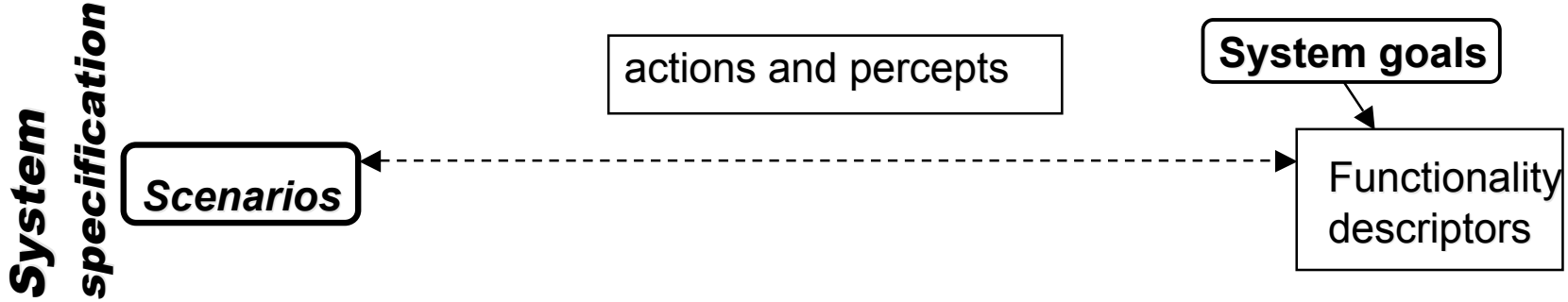

System specification
Architectural design
Detailed design
Scenarios
System goals
Initial Functionality descriptors
Actions, percepts
agent grouping
Interaction diagrams
agent acquaintance
messages
shared data
Protocols
System Overview
Agent descriptors
Process
Agent Overview
Capability descriptors
Capability overview
Event descriptors
Data descriptions
Plan descriptors
Key
final design artifact
intermediate design tool
crosscheck
derives
Dynamic
Overview
Detailed Form

Notes on Previous Slide

- Shows models and dependencies
- Doesn’t show process (but implies it)
- Doesn’t show implicit iteration
- Separates design phase into architectural and detailed design - only latter depends on the implementation platform.

System Specification Summary

- Identify **System Goals** (and subgoals)
  - Because proactiveness is a desired property
  - … but also turns out to be a good way of doing specification
- Situated → **Actions** & **Percepts**
  - Reactive → Events (derived from processing of percepts)
  - Also external data
- **Functionalities**: chunks
  - Linked to Percepts, Goals, etc.
- **Use Case Scenarios**
  - Based on OO, concrete examples, capture *dynamics*

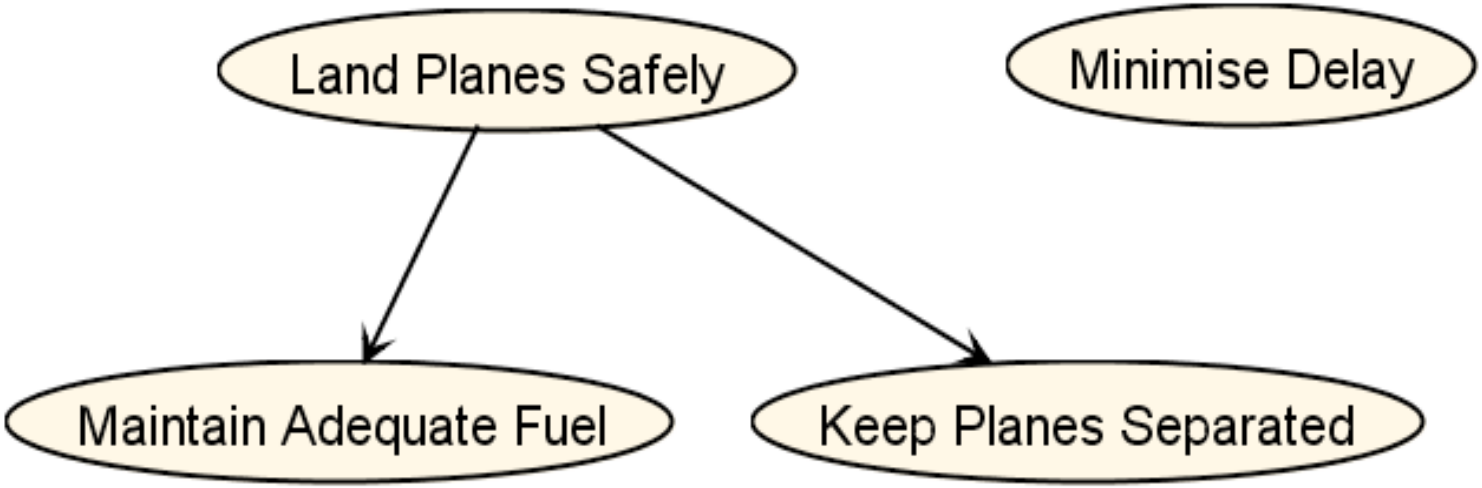


Goal identification

- What is the system being built for? The **primary goal(s)**
- What are the subsidiary goals that will assist in achieving these? What will make this system unique/successful?
- Described using goal diagram
- “How?” (subgoals)
- “Why?” (parent goals)

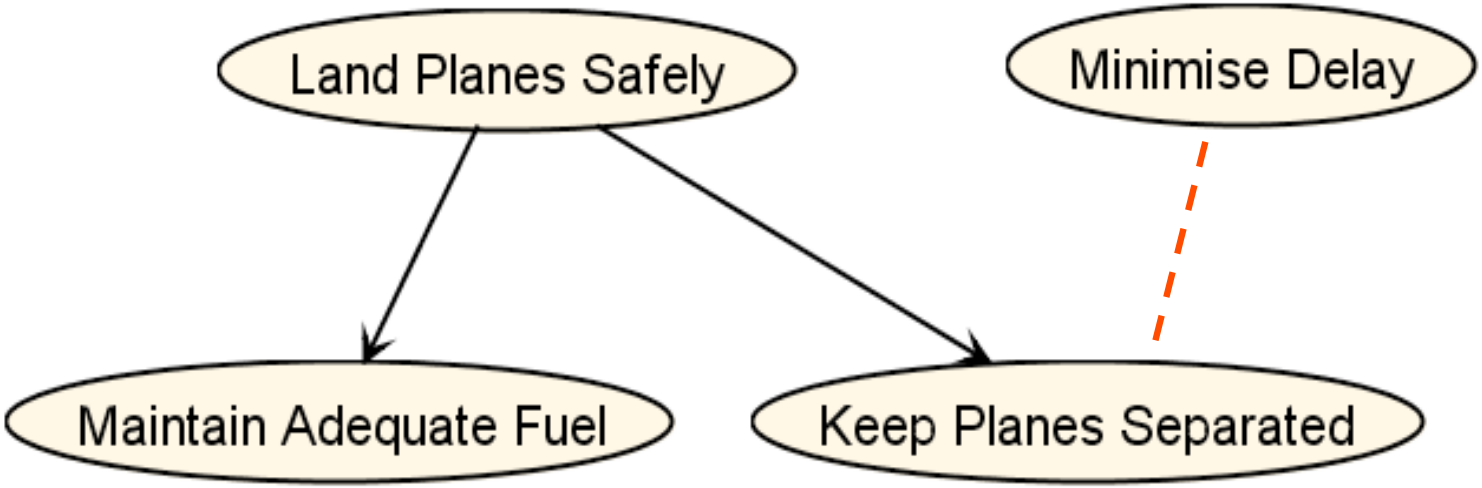
Example Goals

- Land Planes Safely
- Have as little delay as possible
- Keep planes separated
- Land with adequate fuel reserves



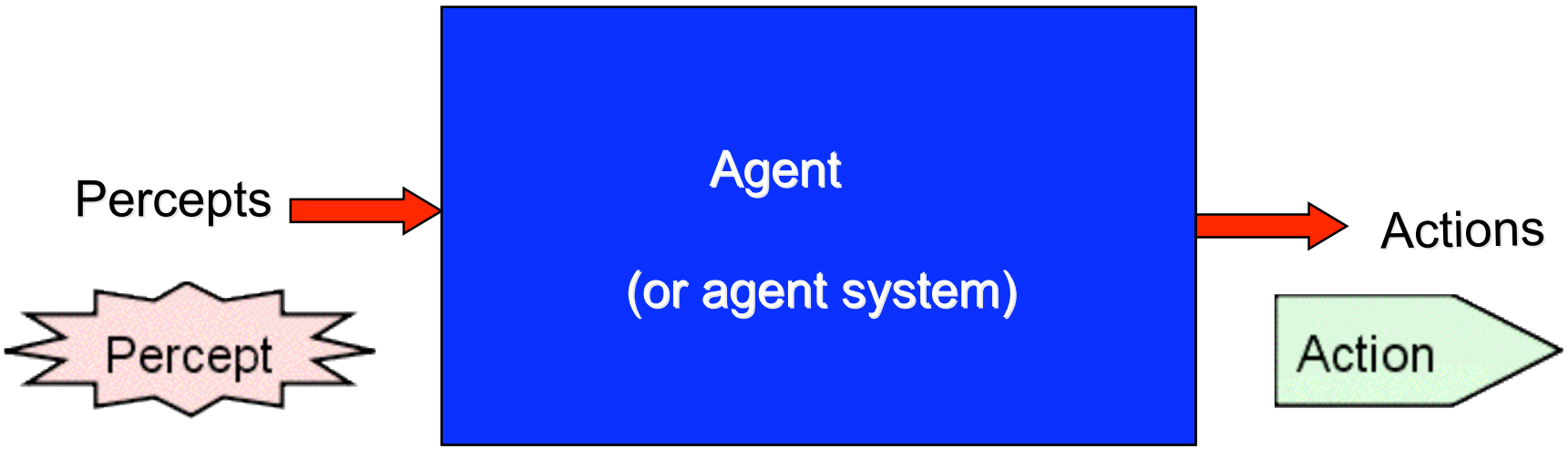
Example Goals

- Land Planes Safely
- Have as little delay as possible
- Keep planes separated
- Land with adequate fuel reserves



Agents are situated

… therefore need to define the interface(s) with the environment



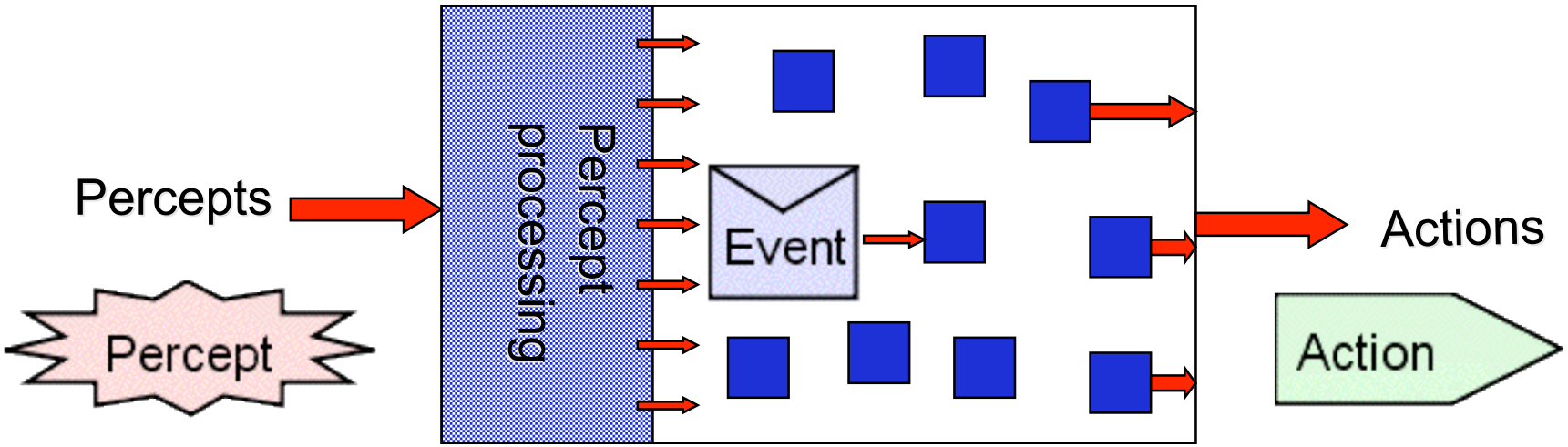
Percepts

- **Percepts** are the raw data arriving from the environment.

- Some things that happen in the environment are significant for the agent system - we call these **Events**.
- **Events** may be the same as the percepts, or they may be obtained from percepts following some processing …

Percept Processing

- We often look for changes between current and previous percepts. Or between believed state of world and percept.
- E.g. If a ball is in a different position, compared to last percept, we may want to generate a ball-moved event.

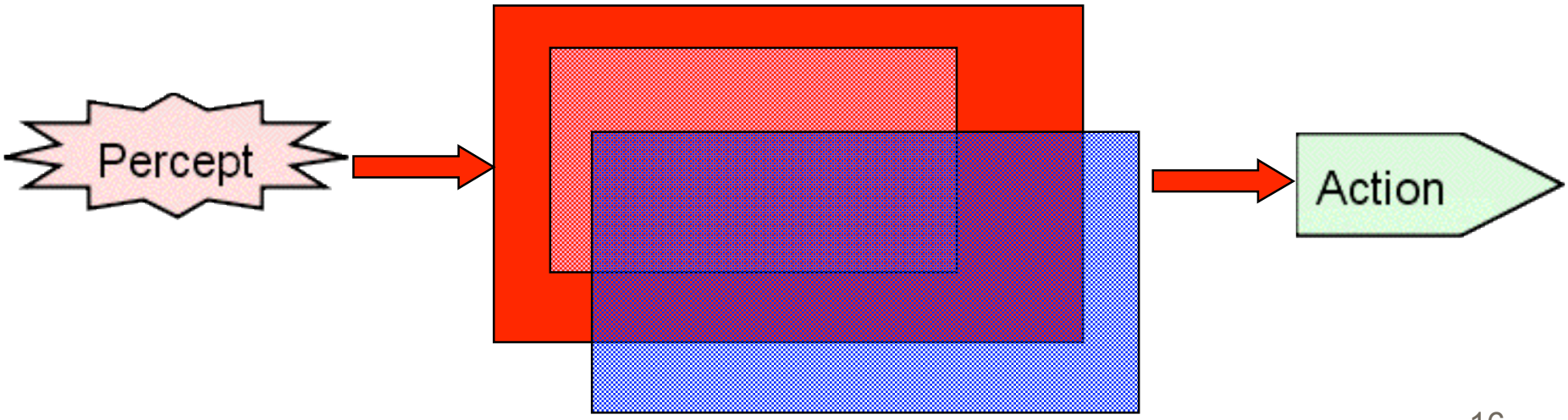


Actions

- The things which the agent does which affect the environment we call **actions**.
- Actions may be simple and direct, or they may be complex - particularly in physical systems actions may involve complex code interacting with mechanical devices.
- Is the action durational? Can it fail? Will you know if it failed? If it fails, will it have changed the environment?
- Perspective – e.g. deliver package is durational but requestDelivery+deliveryComplete (percept) is not

Where is the boundary?

- A matter of design choice (sometimes)
- Dictated by existing software (sometimes)
- Dictated by other constraints such as hardware, rules etc. (sometimes)



Goals, Functionalities & Scenarios

- Identify system goals - main and subsidiary
- Identify functionalities that will achieve these goals
- Prepare Functionality Descriptors
- Define Scenarios (and variations)
- Various sanity cross checks such as ensuring that all goals covered by scenarios

Goals, Functionalities & Scenarios

- Identify system goals - main and subsidiary
- Identify functionalities that will achieve these goals
- Prepare Functionality Descriptors
- Define Scenarios (and variations)
- Various sanity cross checks such as ensuring that all goals covered by scenarios

Iterative process!

Functionality Identification

- Describe functionalities that you believe the system should have
- Functionalities should be narrow and definable in 1-2 sentences (highly cohesive!)

- Link these back to system goals
  - There may well be more than one functionality per goal
  - Functionalities may also address more than one goal

Example

Goals:

1) Land planes safely
2) Have as little delay as possible
3) Keep planes separated
4) Land with adequate fuel reserves

Functionalities

schedule manager, (2)

separation manager, (3)

fuel watcher, (4)

weather observer, (1)

runway watcher, (3, 2, 1)

…

Functionality Descriptors

Detailed description of each functionality:

NAME: short descriptor

DESCRIPTION: natural language description, reasonable detail.
If more than 3 sentences, consider breaking down.

PERCEPTS/EVENTS: what activates this functionality.

ACTIONS: actions on the external environment this functionality is responsible for.

MESSAGES: messages to/from other functionalities (often triggers)

INFORMATION USED: what data does this functionality need

INFORMATION PRODUCED: what data is produced

INTERACTIONS: what other functionalities does this functionality interact with (can be derived)

GOALS: what goals does this functionality achieve?

Scenarios

- Scenarios show a particular instance of system execution - no branching, etc.
- Alternatives are described by separate scenarios, or by variations (annotation describing the variation)
- Scenarios consist of a sequence of steps
  - Scenarios may contain steps which are themselves scenarios
  - Other step types: action, percept, goal, scenario, other.
- Each step linked to a functionality and to data used/produced
- Each system goal should be represented in at least one scenario
- Scenarios typically involve multiple functionalities

Example Scenario

New plane enters control area: scenario 23

Description: New plane enters control area and is given instructions

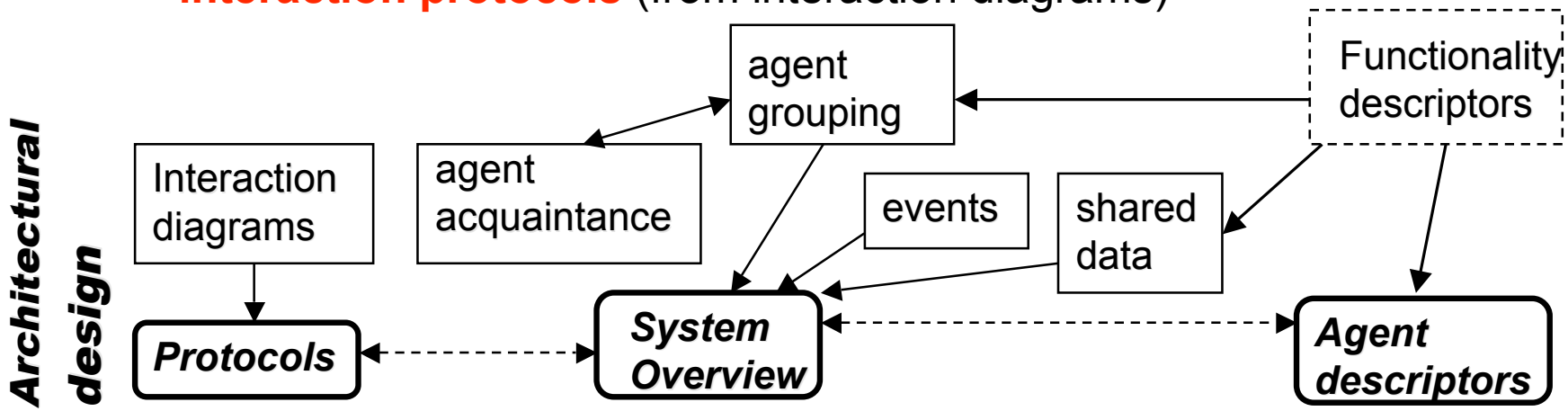
Start condition: plane sends message soon entering area

Steps	Functionality	Data
1. Watch radar to identify plane	radar observation	
2. Check expected arrival time	airport information	Arrivals DB
3. Place plane in schedule	scheduler	Schedule
4. Generate landing instructions	runway manager	Schedule Runways DB
5. Communicate landing instructions	communications	Comms DB
6. Monitor plane	monitoring	
7. Plane landing observed	communications	

Variations: …

Summary of high level design

- Identify **Agents** by *grouping Functionalities*
  - Data Coupling Diagram
  - Agent Acquaintance Diagram
- Describe entities with descriptors: agents, actions, percepts, events/messages
  - Agent life-cycle
- Top level (system) view
  - **system overview** - agents, protocols, events, actions, shared data
  - **interaction diagrams** (from scenarios)
  - **interaction protocols** (from interaction diagrams)



Grouping of Functionalities into Agents

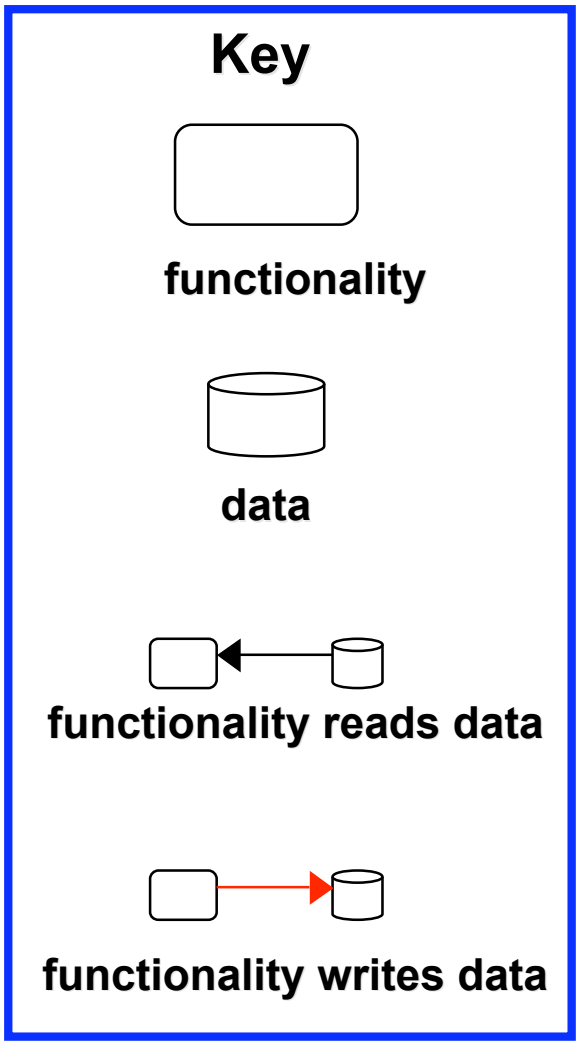
Reasons for:

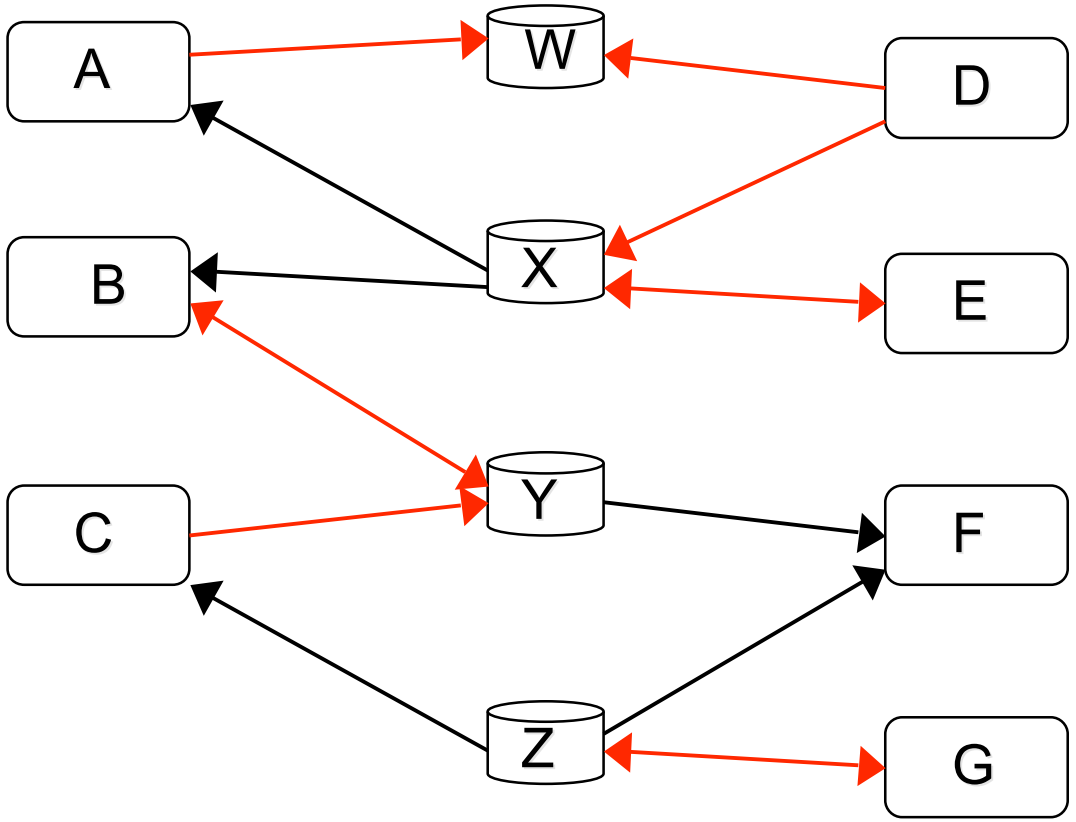
- functionalities seem related - naturally belong together (cohesion), e.g. delivery organisation & transport info.
- functionalities use the same data (coupling)
  - especially if both write to the data!
- functionalities interact a lot (coupling, weaker)

Reasons against:

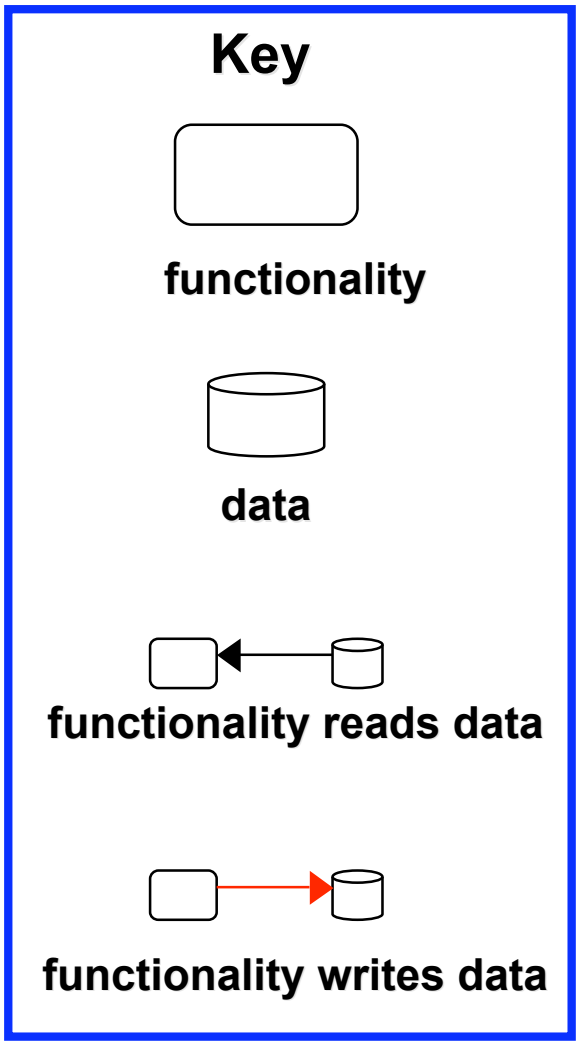
- Functionalities unrelated - a non-cohesive agent
- Functionalities need to reside on different hardware platforms
- Differing cardinality - e.g. 1/system vs 1/customer

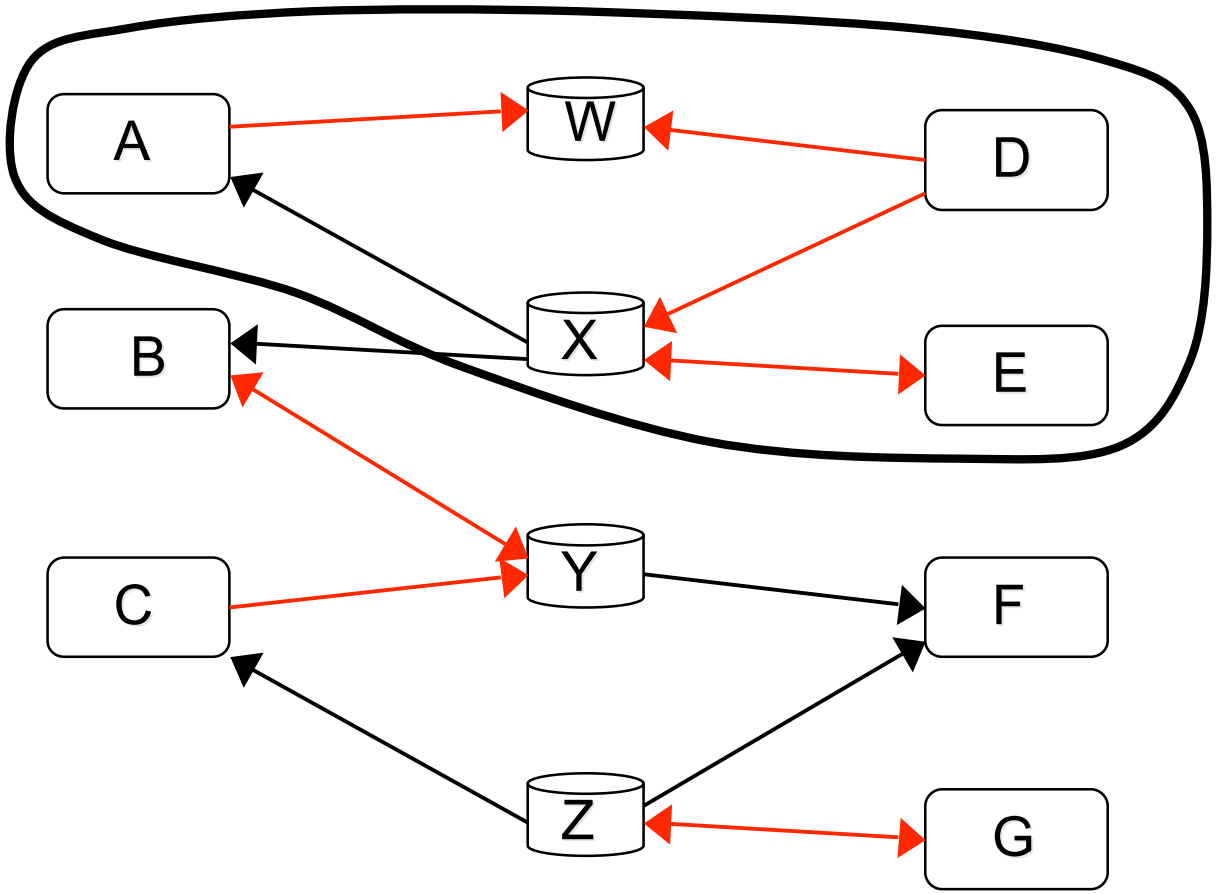
Data Coupling Diagrams



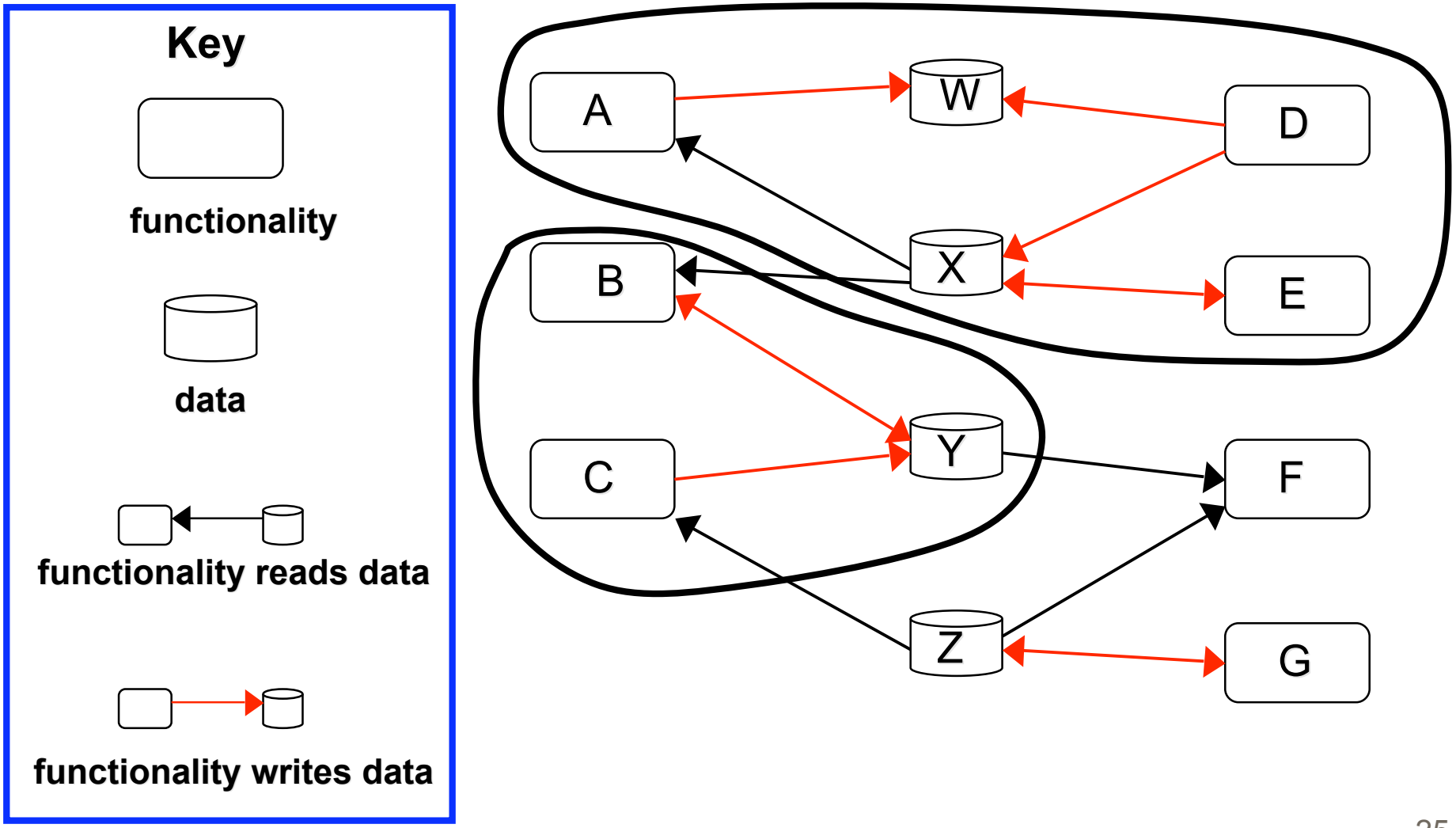


Data Coupling Diagrams

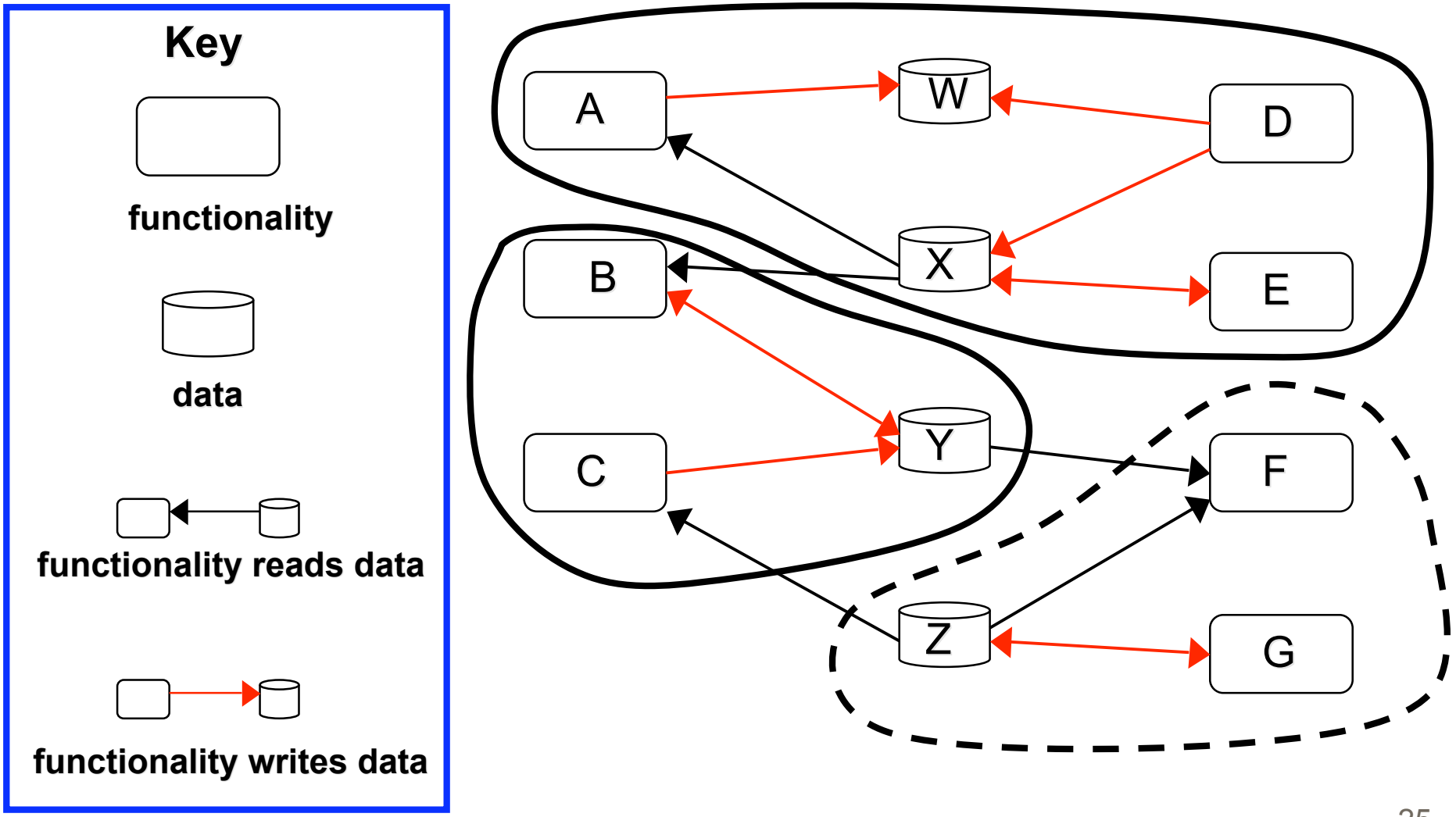




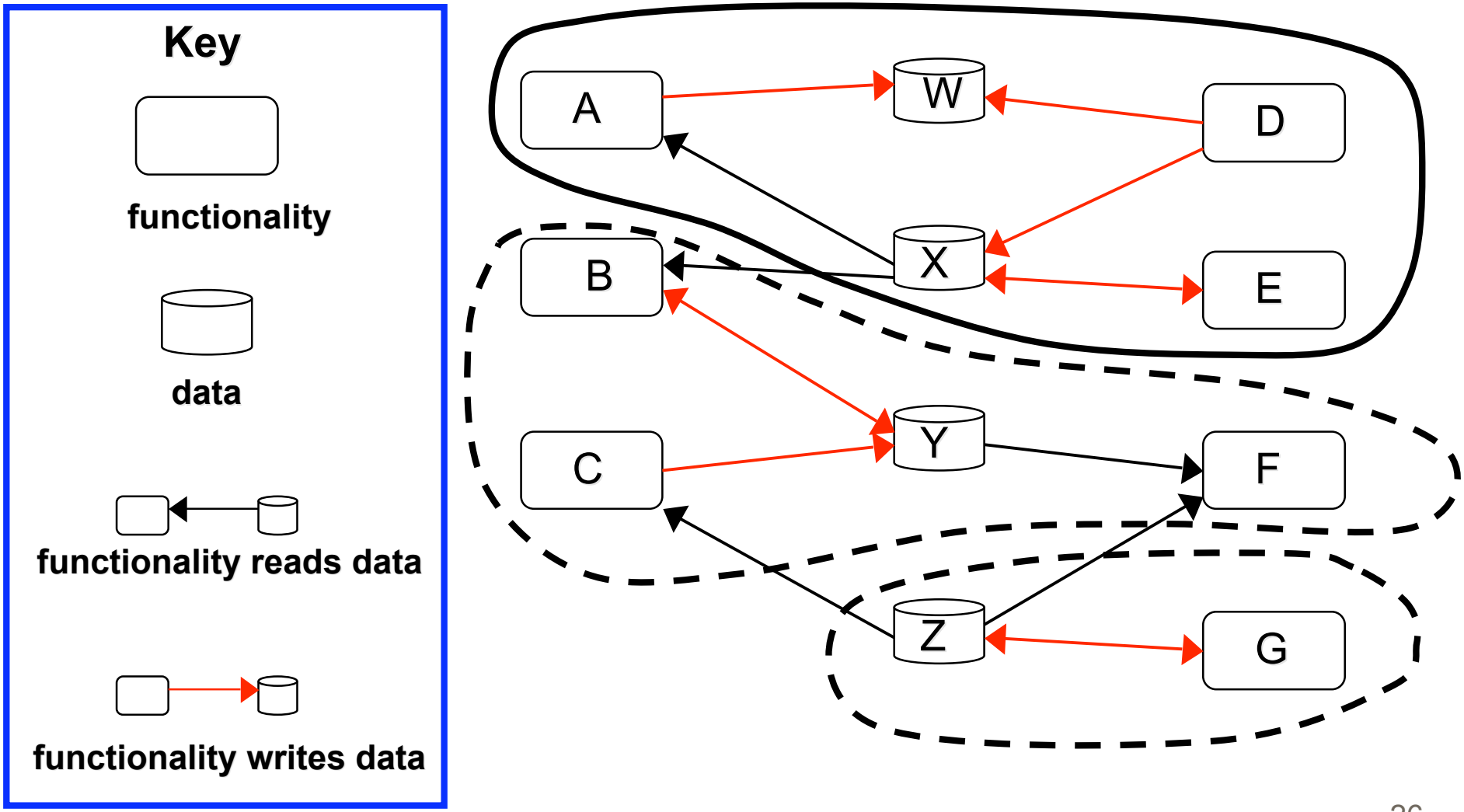
Data Coupling Diagrams



Data Coupling Diagrams

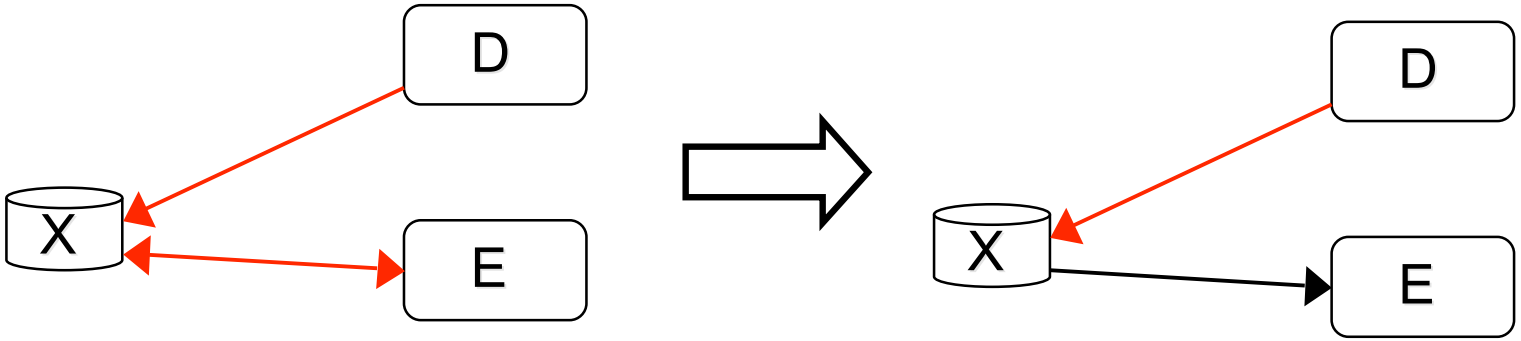


Data Coupling Diagrams



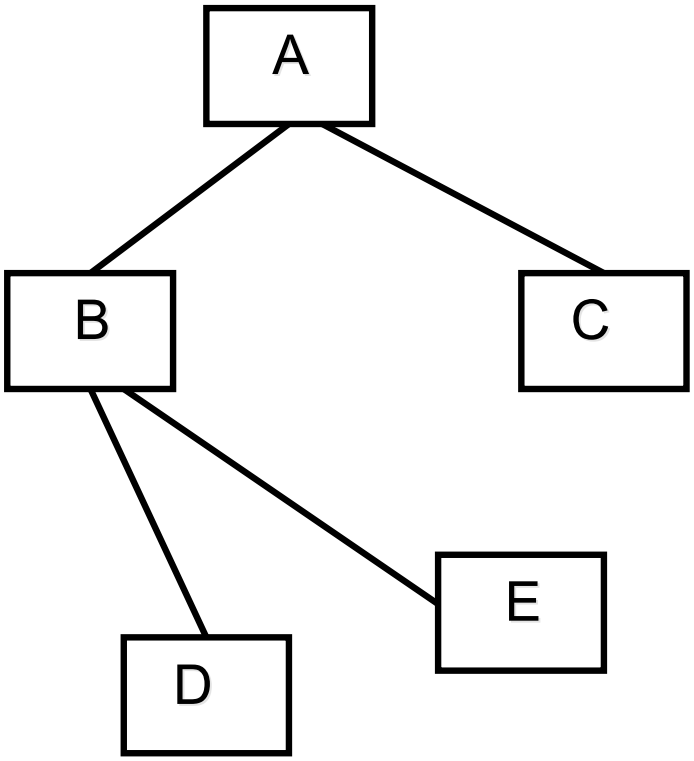
Revising the Design

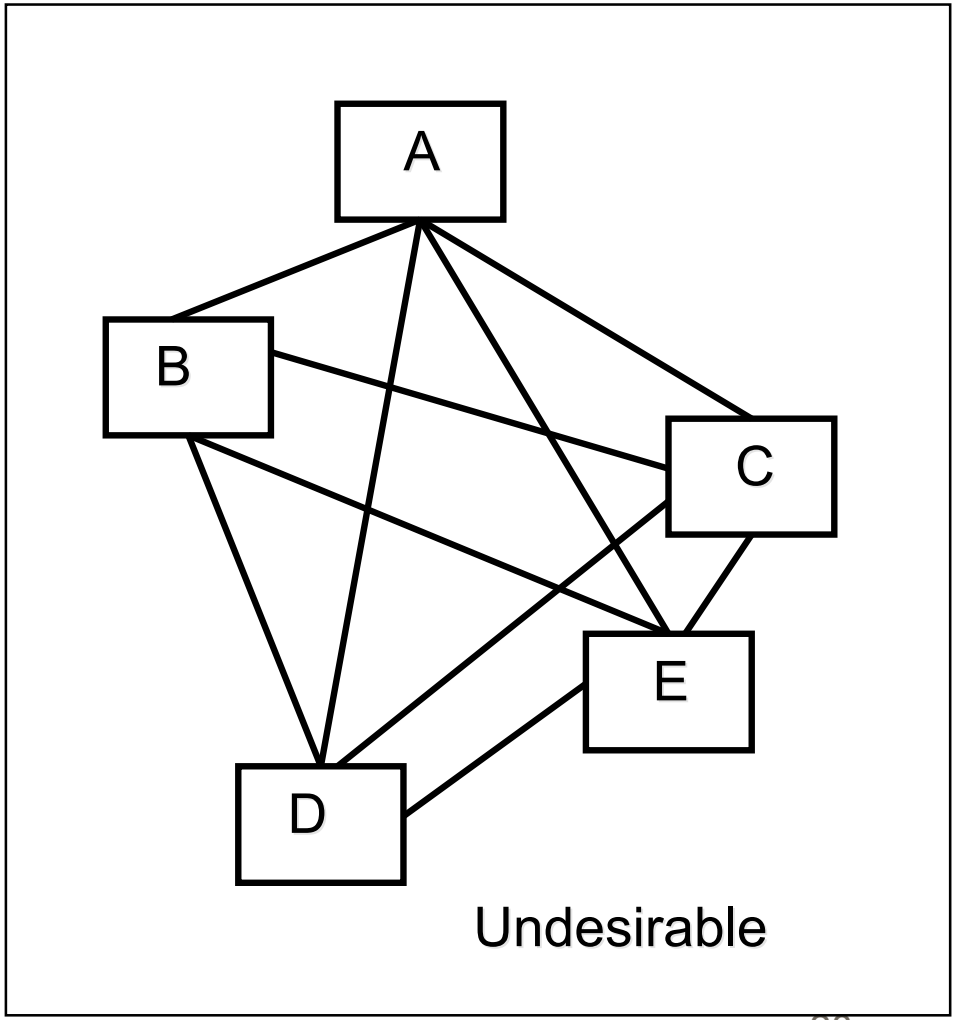
- In addition to creating groupings, also sometimes modify existing design by delegating writes:



- Often select a single agent type to be responsible for updating a data store

Agent Acquaintance Diagrams: evaluating groupings





Grouping Agents

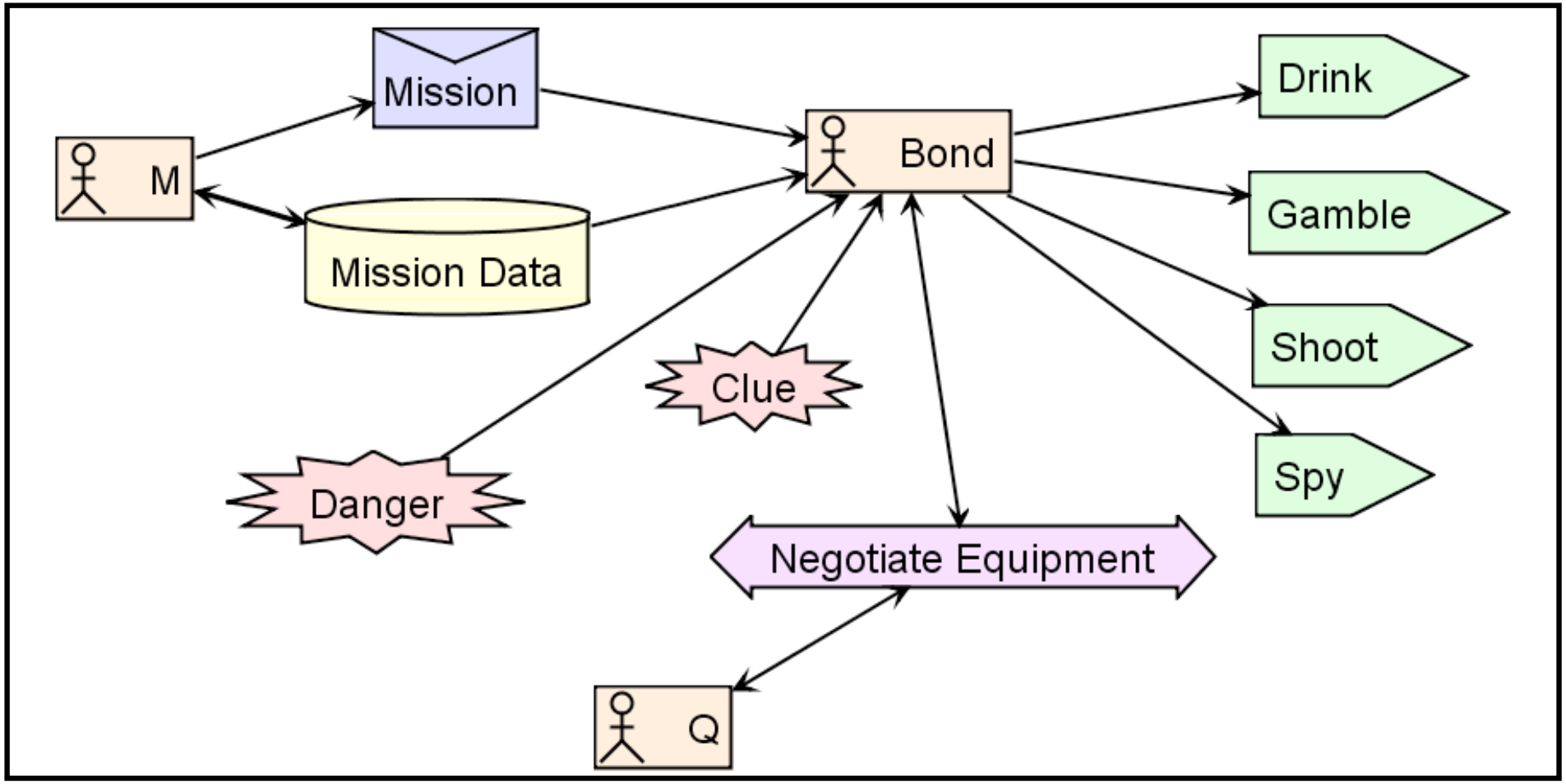
- Use Data Coupling diagram to suggest possible groupings that might make sense
- Use Agent Acquaintance diagram to evaluate proposed groupings
- Once have an agent grouping, work out how agents fit into system by considering information flow …

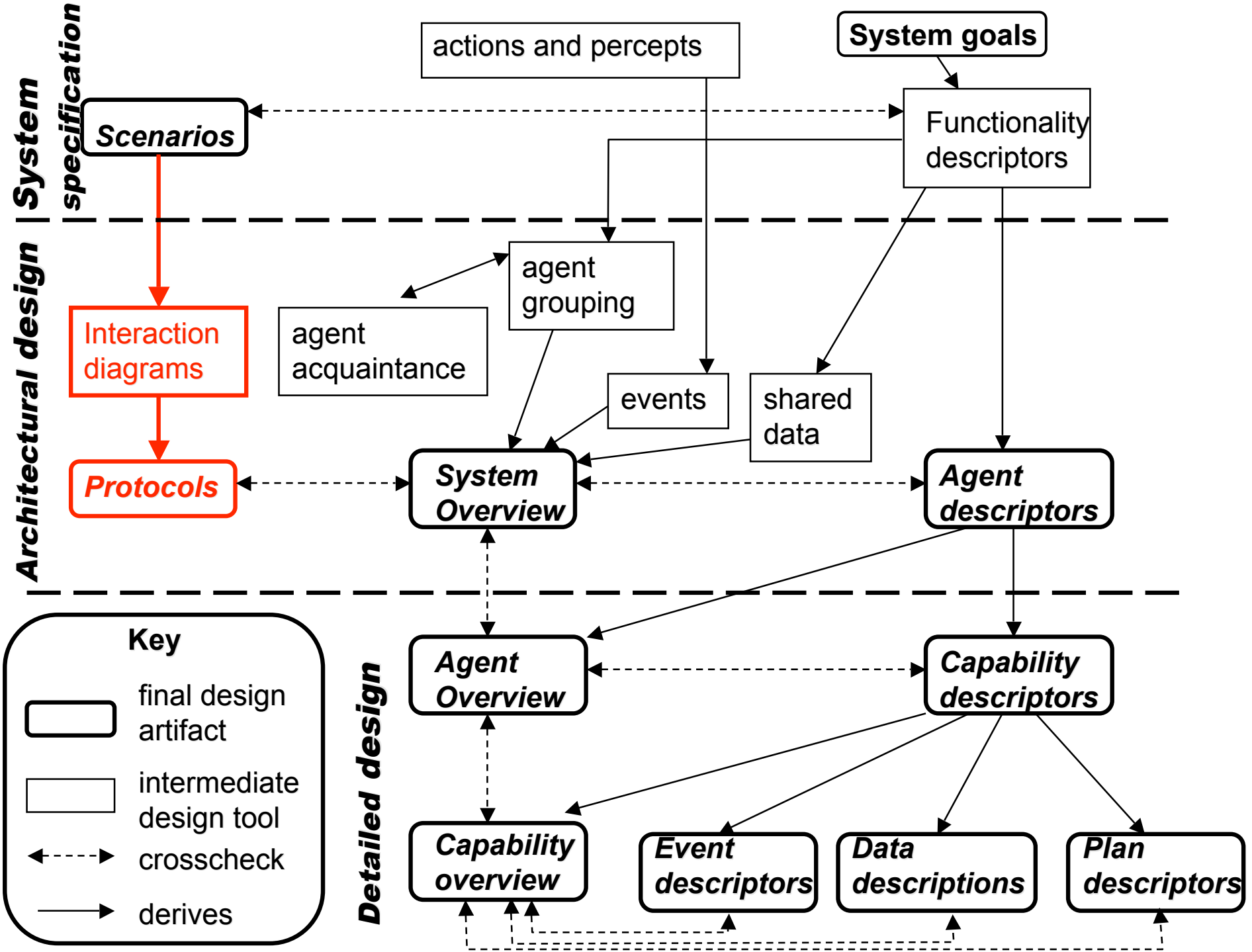
Agent Descriptors

- Name
- Description
- Percepts/Events responded to
- Actions executed
- Messages sent - to which agent, what content
- Messages received - from which agent, what content
- internal messages
- Data used - external or shared internal
- Data produced - external or shared internal
- Goals
- Functionalities included
- Creation/Destruction
- Cardinality (how many?)
- (plans and capabilities - added later)

A lot of information …
… but much of it can be determined from included functionalities
… much information also depicted in more easily understandable form in **System Overview Diagram** …

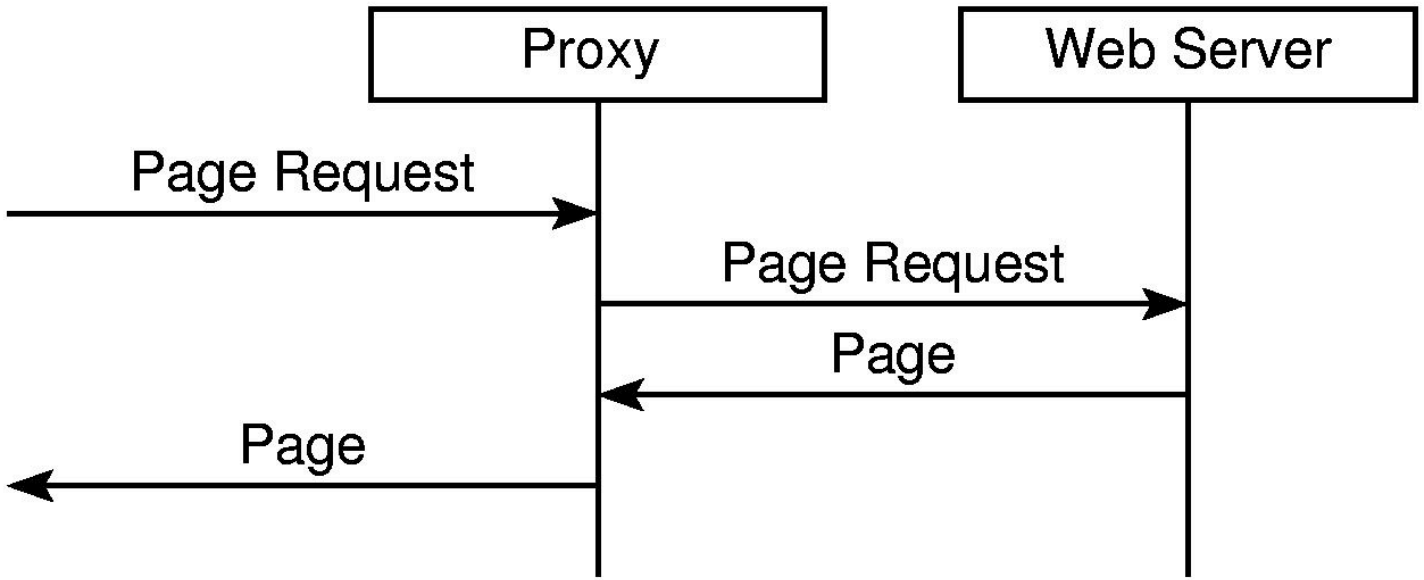
System Overview Diagram




System specification
Architectural design
Detailed design
actions and percepts
System goals
Scenarios
Functionality descriptors
agent grouping
agent acquaintance
Interaction diagrams
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

Interaction Diagram

- Purpose: design multi-agent co-operation activity
- Based on use cases (and have same limitations)
- Standard UML-like notation



Deriving Interaction Diagrams

- Consider Use Case Scenarios
- Roughly speaking, where step N involving functionality F is followed by step N+1 involving functionality G, and F and G are in different agents, there would be a message.

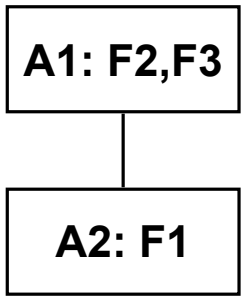
1. Action A by functionality F1

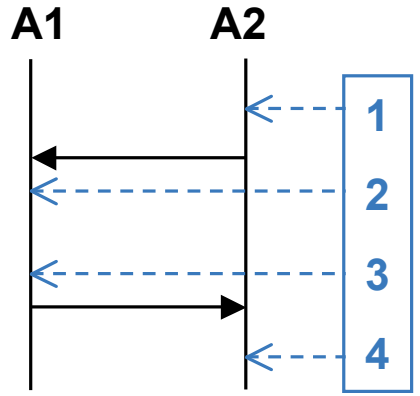
2. Goal G, functionality F2

3. Action A' by functionality F3

4. … by functionality F1

…





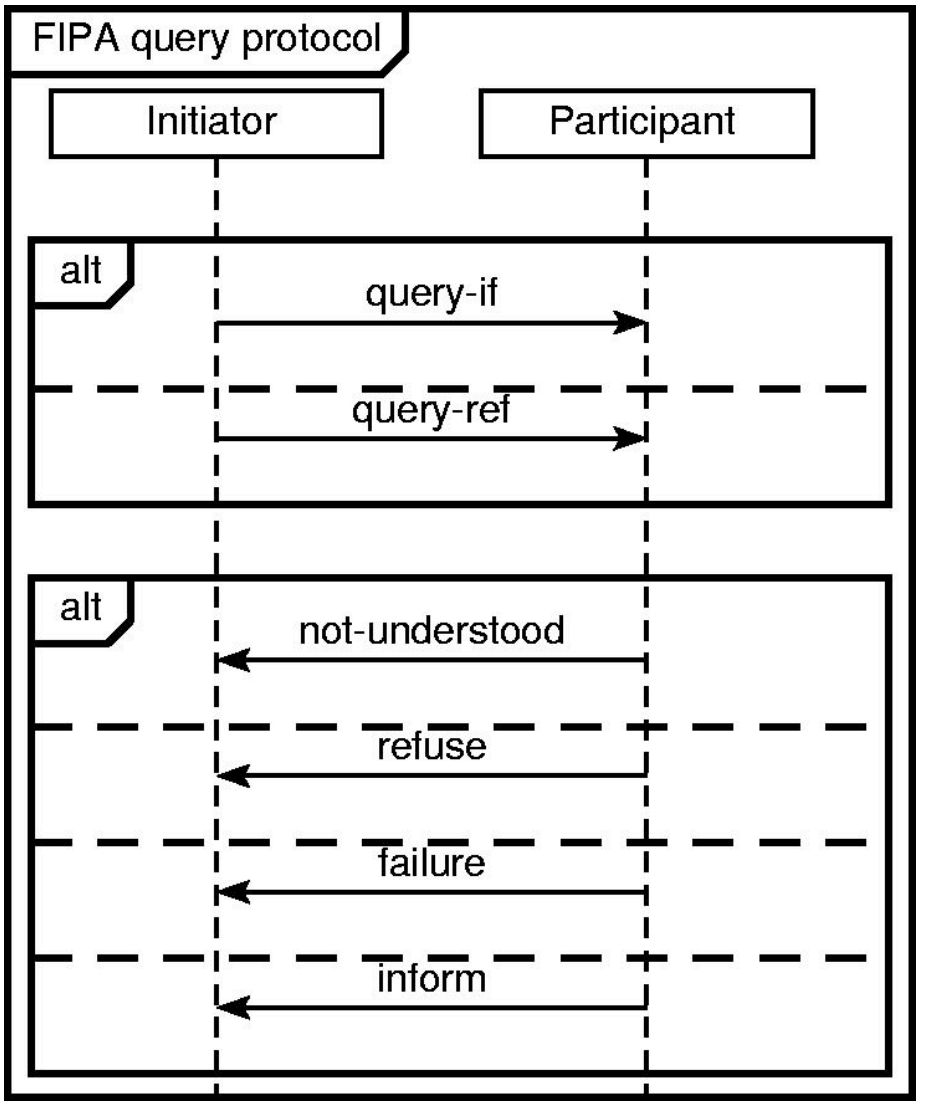
Generalizing to Protocols

- **Process:** Consider alternatives at each point
- **Notation:**
  - AUML (Agent UML)
  - AUML-2
  - UML activity diagrams
  - …

AUML-2 (DRAFT!)

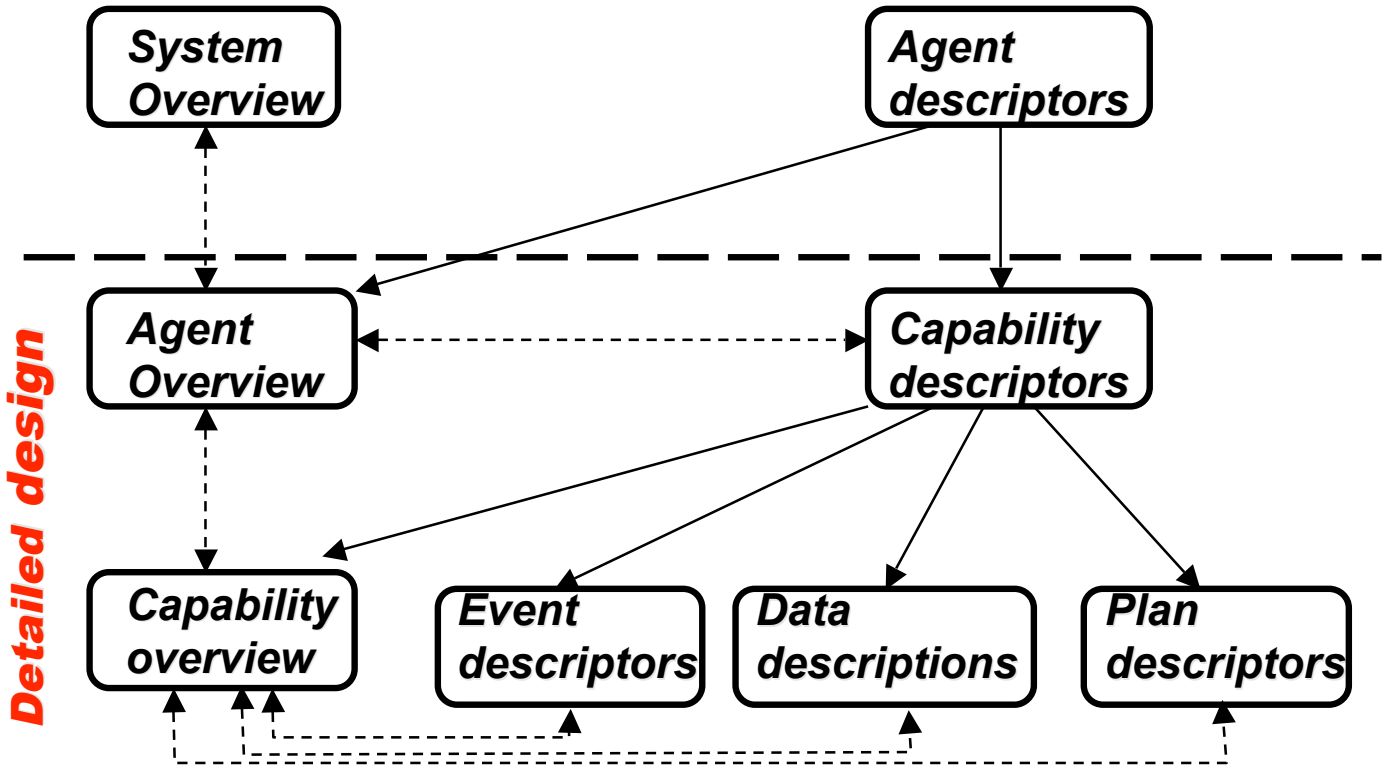
- Based on UML2
- More structured (graphical) notation
- Still in draft form!

Papers with further information on possible syntax etc. available at www.auml.org, under "*working documents*"



Detailed Design Summary

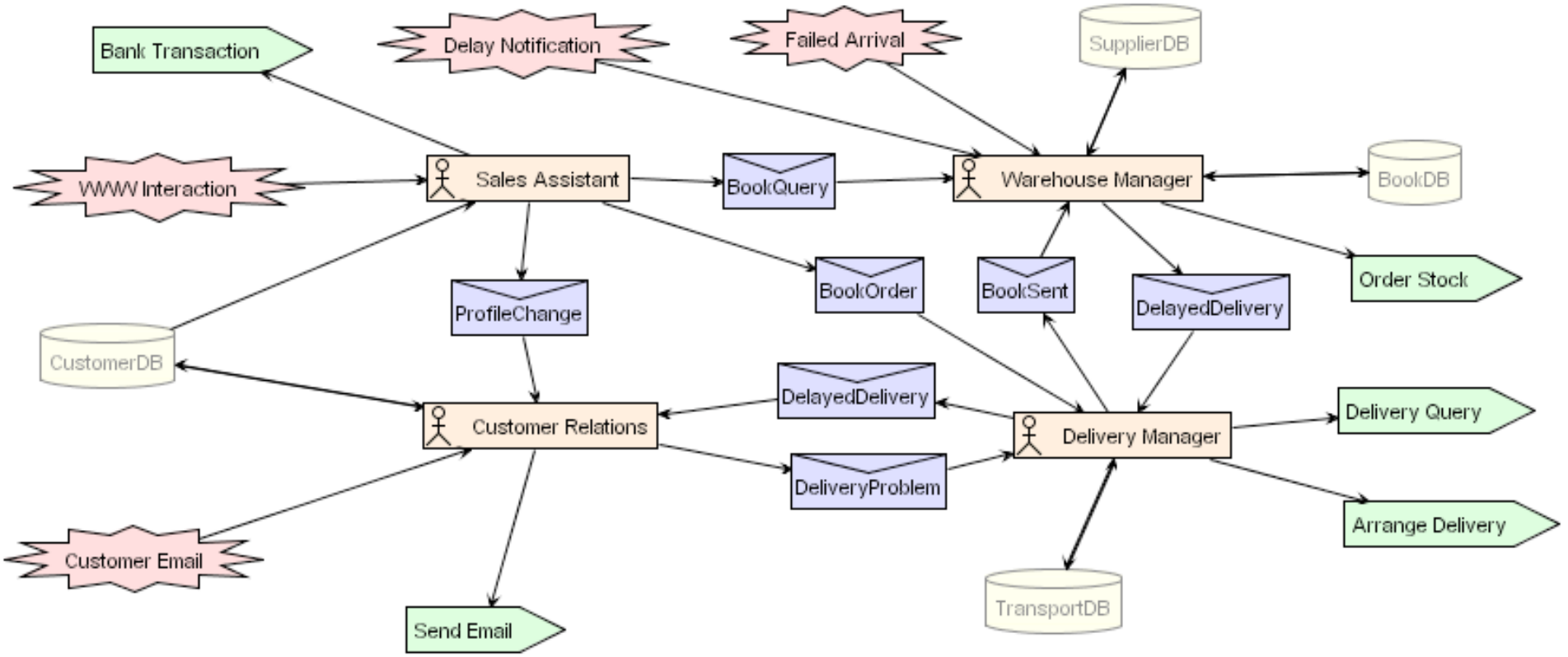
- Progressive refinement - zoom in
  - Agents in terms of capabilities (and plans)
  - Capabilities in terms of capabilities and/or plans
  - Also events and data; as well as process diagrams



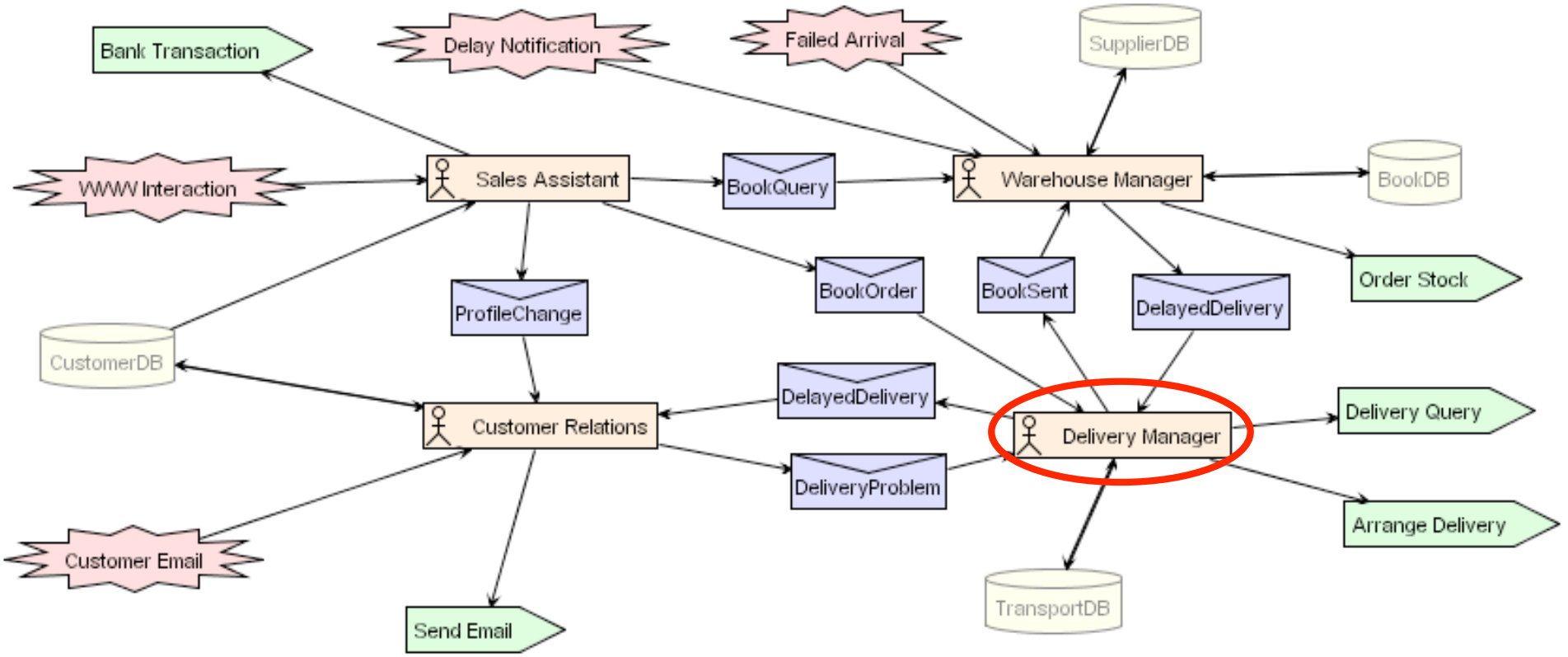
Defining capabilities

- Think in terms of “modules” - a limited package of functionality that does a particular thing.
- Functionalities from phase one are a starting point - they may need to be further broken down and nested.
- Low level functionality needed by many agents can also provide basis for capabilities - e.g. communications capability, route finding capability.
- Supports code re-use - can reuse in multiple agents.

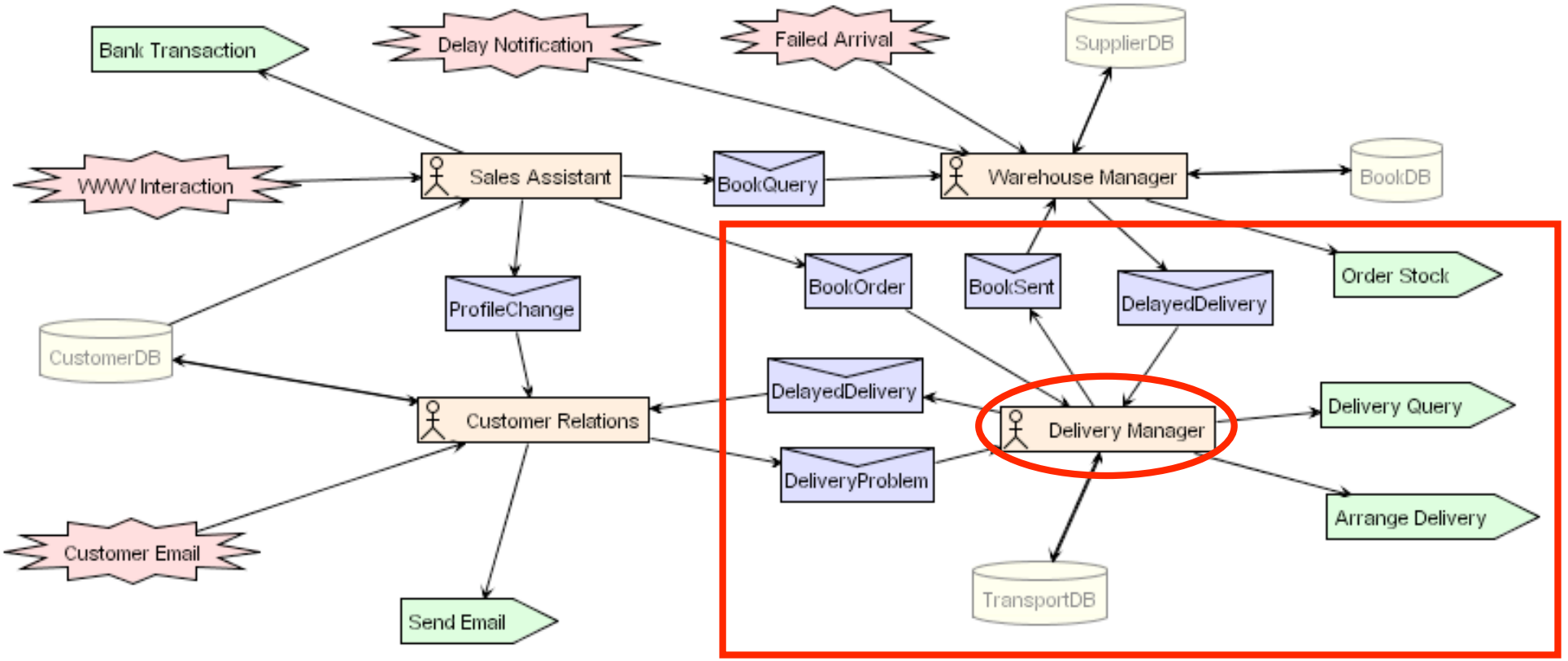
Recall System Overview Diagram



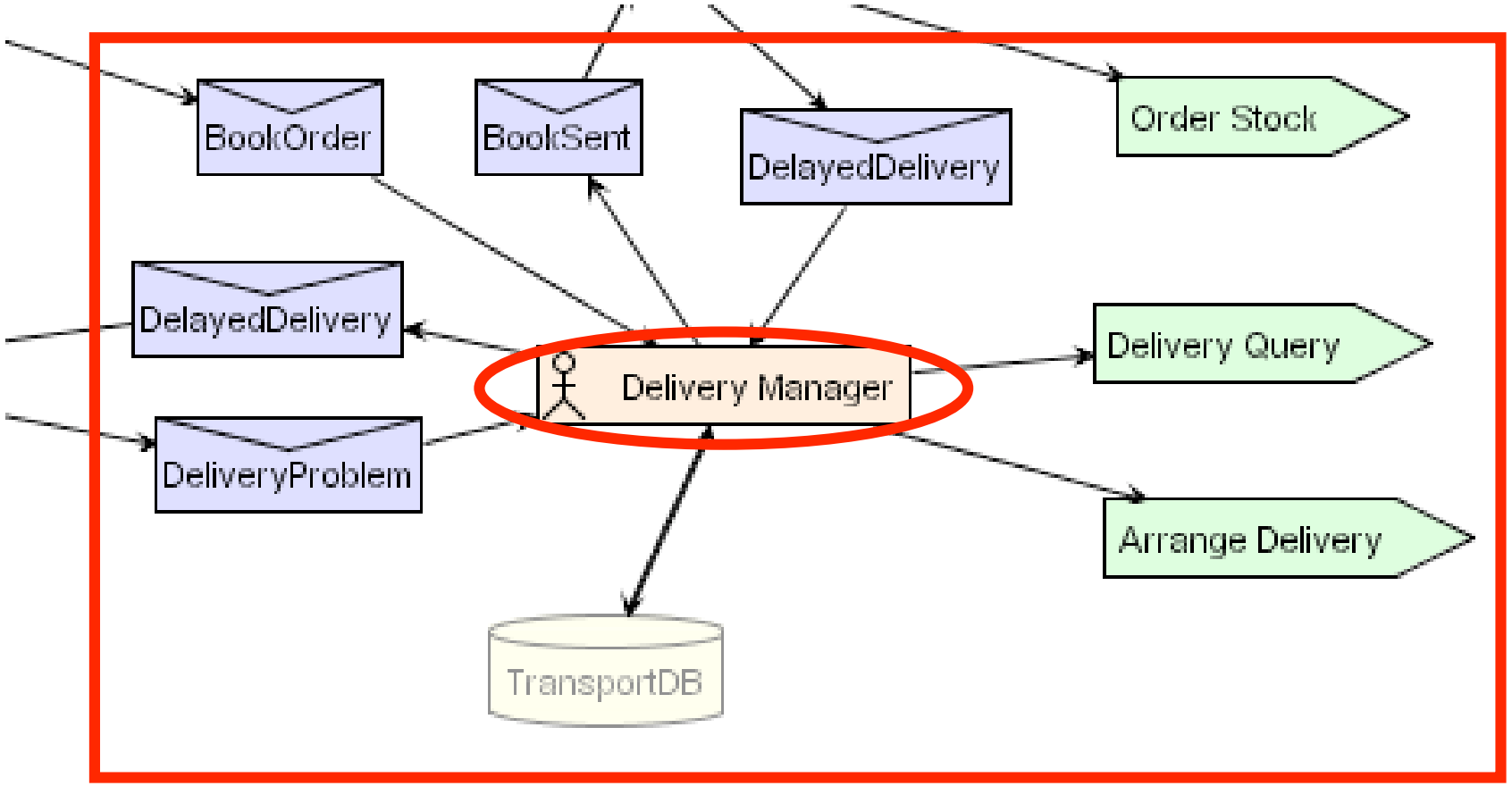
Recall System Overview Diagram



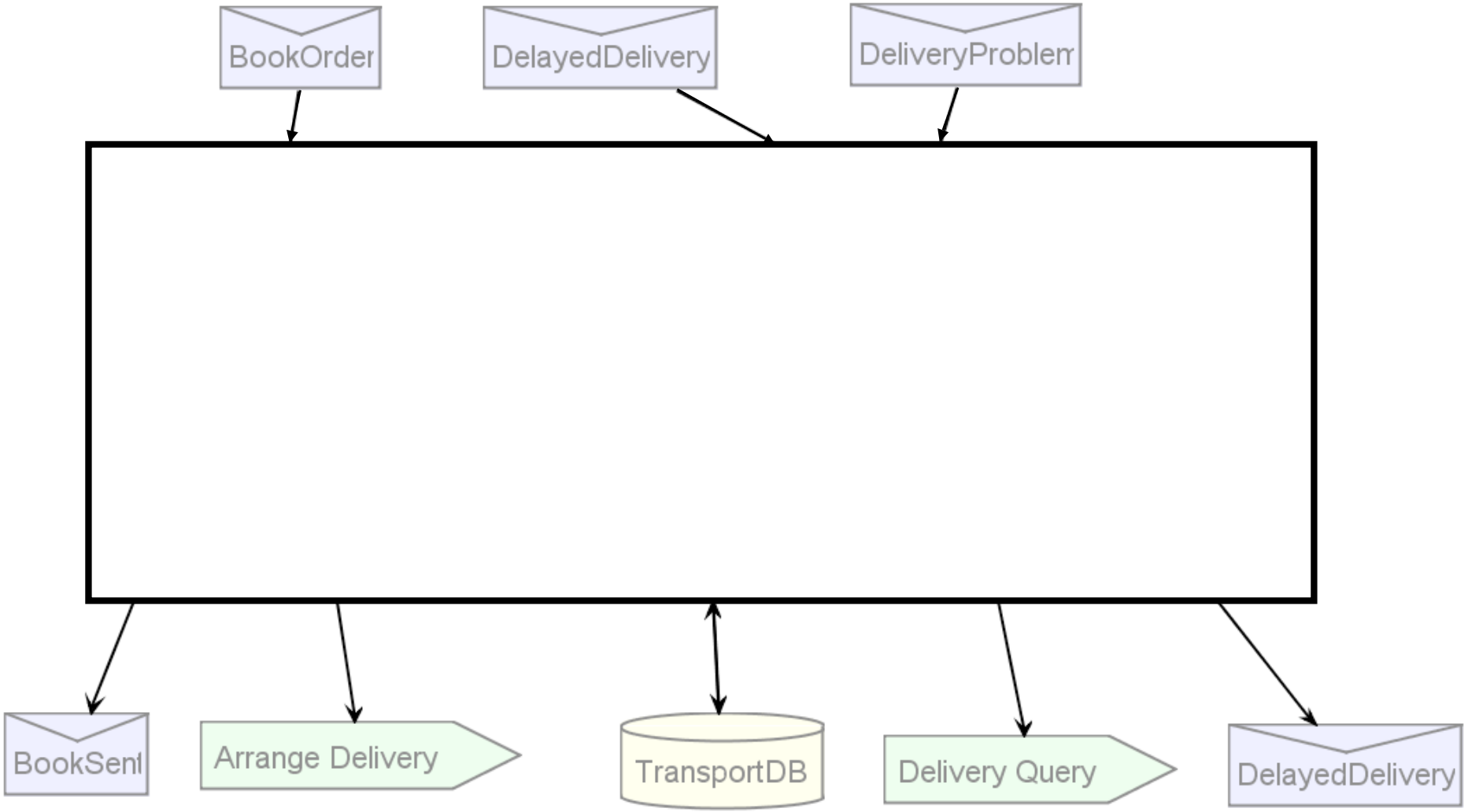
Recall System Overview Diagram



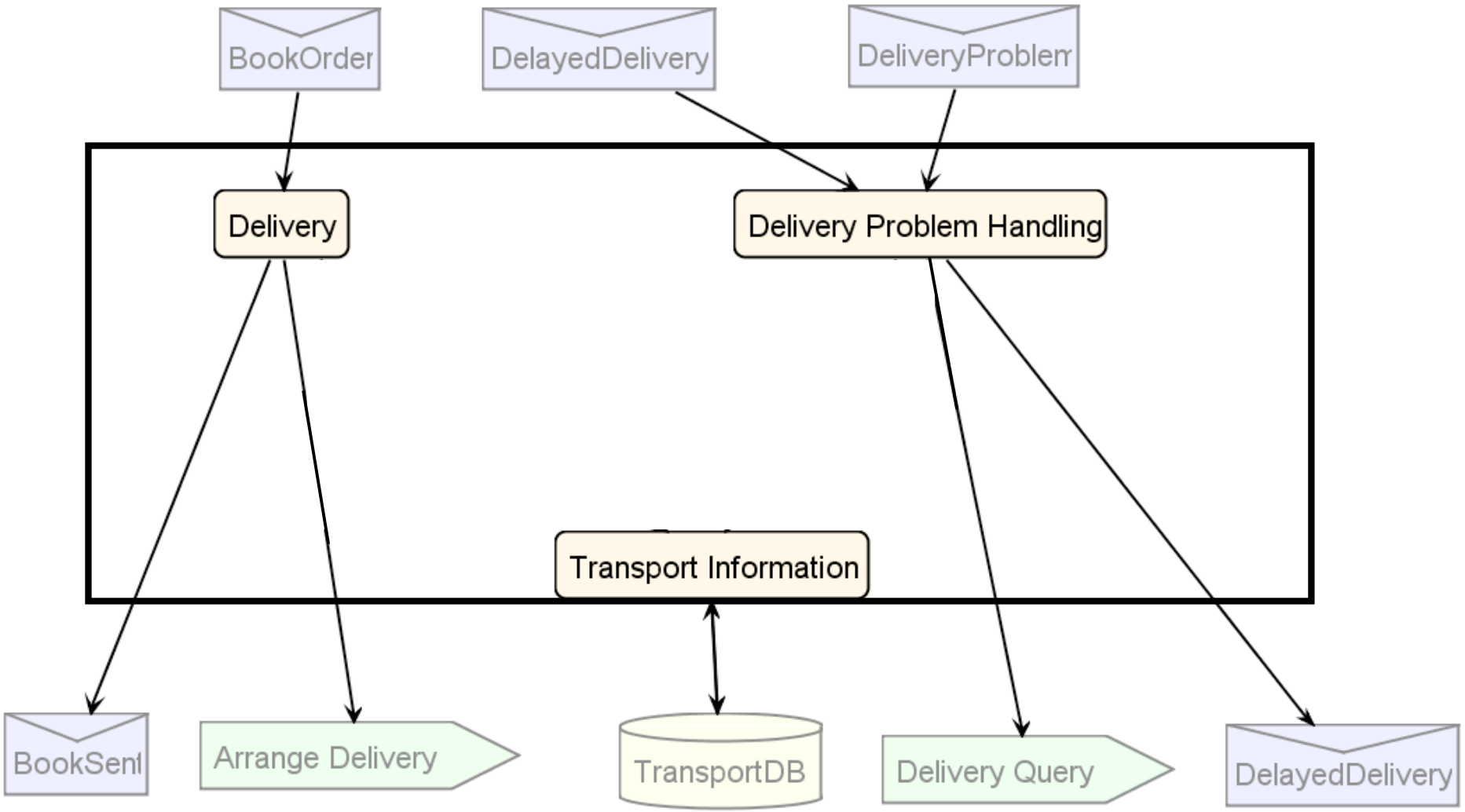
Recall System Overview Diagram



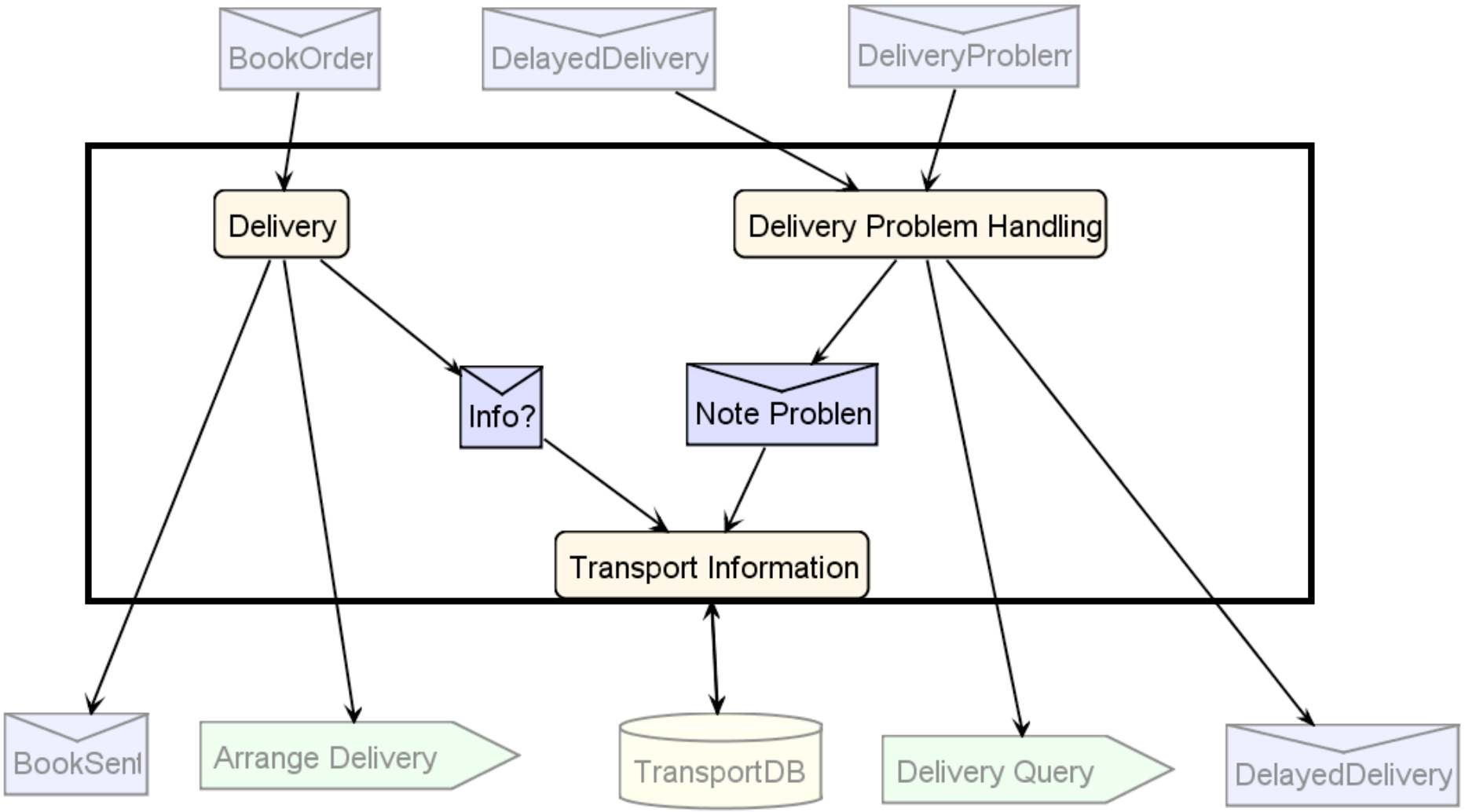
Agent Overview Diagram



Agent Overview Diagram

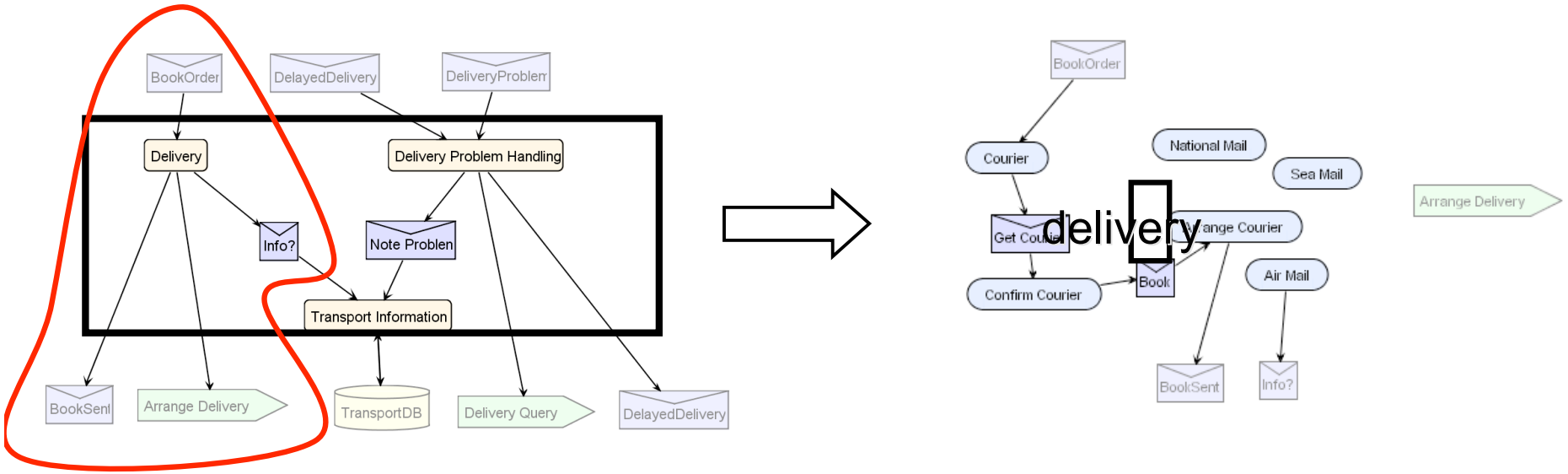


Agent Overview Diagram



Capability Overview

- Same notation as agent overview
- Same process - zoom in on capability and its interface



Events

- Events are the “triggers” or “messages” between capabilities within the agent and between plans within a capability.
- Events can be used to start a subtask (or generate a subgoal) or to start a new task (in a separate “thread”).
- JACK uses the same data type “event” for percepts, messages, and events within an agent.

Coverage

- For this event (message / percept) will there always be an appropriate response - i.e. some plan defined which has a context condition which matches the situation?
- Or may there be some situations where there is no plan with a matching context condition - i.e. there will be no processing of the event?

Overlap

- Are there situations where there may be multiple responses to this event - i.e. the situation can be matched by more than one context condition?
- If so what is the prioritisation between the different plan types?
- May there be multiple matching plans (of the same type) due to variables in the context condition that may have multiple bindings?

Coverage & Overlap Example

```
Plan1:
#handles goto_rmit
Context() {not raining;}
Body() {walk(rmit);}
```

```
Plan2:
#handles goto_rmit
Context() {raining;}
Body() {@post(catch(train));}
```

```
Plan3:
#handles goto_rmit
Context() {true;}
Body() {@post(catch(tram));}
```

```
Plan4:
#handles catch
Context() {true;}
Body() {
  walk(station);
  check_timetable;
  fail_if_long_wait;
  …
}
```

goto_rmit:

Coverage? Overlap?

Defining the plans

- When should the plan be used?
  - what event does it handle?
  - what goal does it achieve?
  - in what situation is it applicable?
- What does the plan do?
- Under what conditions should it be aborted?
- What should be done if it fails?
  - two kinds of “failure”: plan failure, and failure to achieve intended outcome (goal failure). Latter must be explicitly tested for.

Defining the plan body

- Plan body consists of
  - subgoals
  - internal plan methods (reasoning methods)
  - other code
- use subgoals wherever possible
- different nuances of subgoals - subtask, insist, achieve, test, …
- think whether you need to check success / failure via percepts?

… towards Implementation

Plan, Agent, Event, Data, etc. are all JACK concepts

```
agent MyAgent extends Agent {
 #posts event AnEvent;
 #handles event AnotherEvent;
 #has capability MyCapability;
 #uses plan ACunningPlan;
 #private database MyDB db;
}
```

Experiences with Prometheus

- Successfully taught to undergraduate students (CS526)
- Successfully used by summer students
- Evolved over a number of years
- Makes a difference!

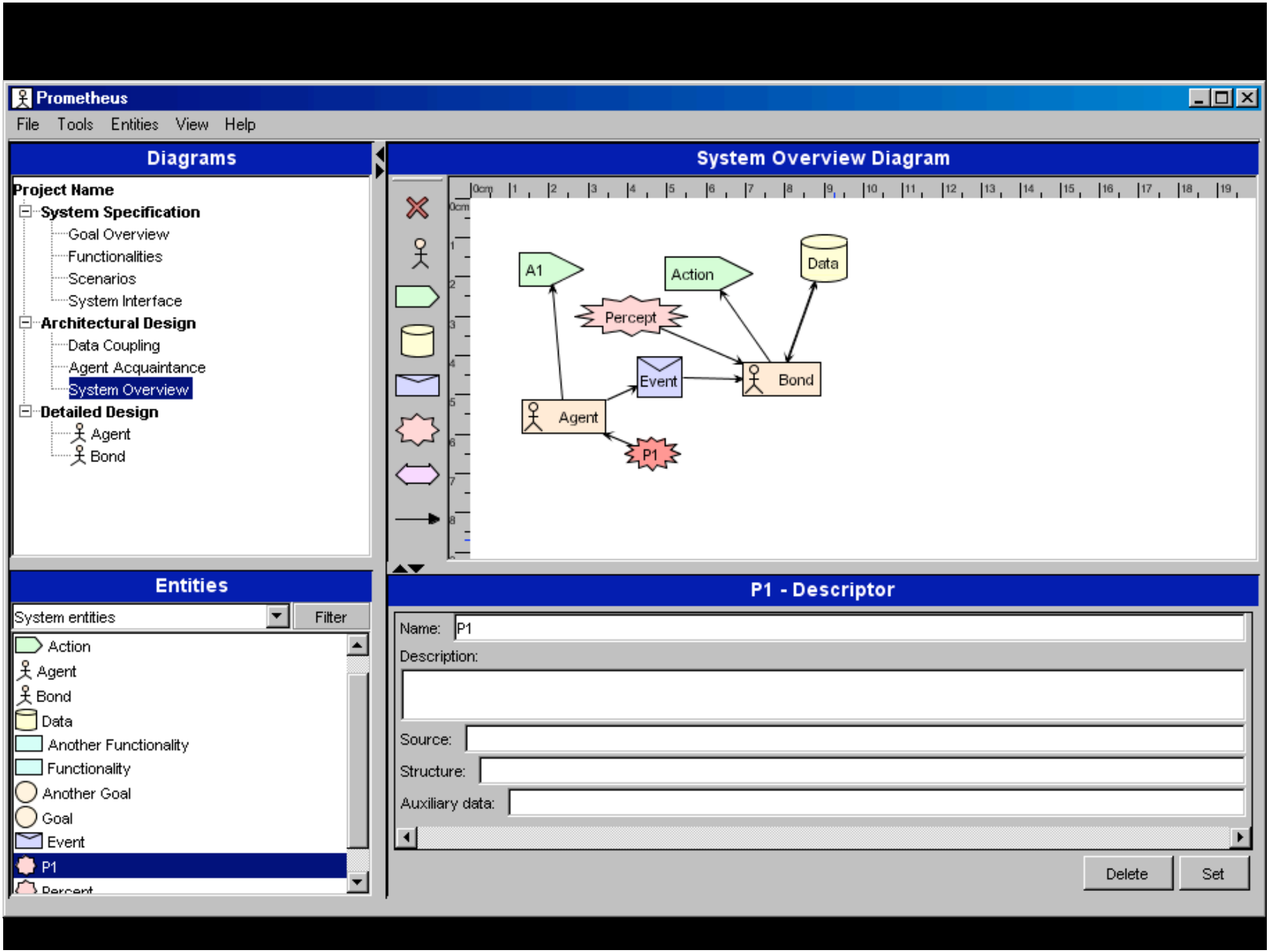
Tool Support for Methodology

- Prometheus Design Tool (PDT)
  Developed first half of 2003 by Anna Edberg & Christian Andersson with further development by Claire Hennekam and Jason Khallouf at RMIT
- Also (not covered today)
  - JDE (Jack Development Environment, by AOS)
  - PDT (earlier prototype by Michael Winikoff)
  - AUML tool (prototype by Michael Winikoff)

Prometheus Design Tool

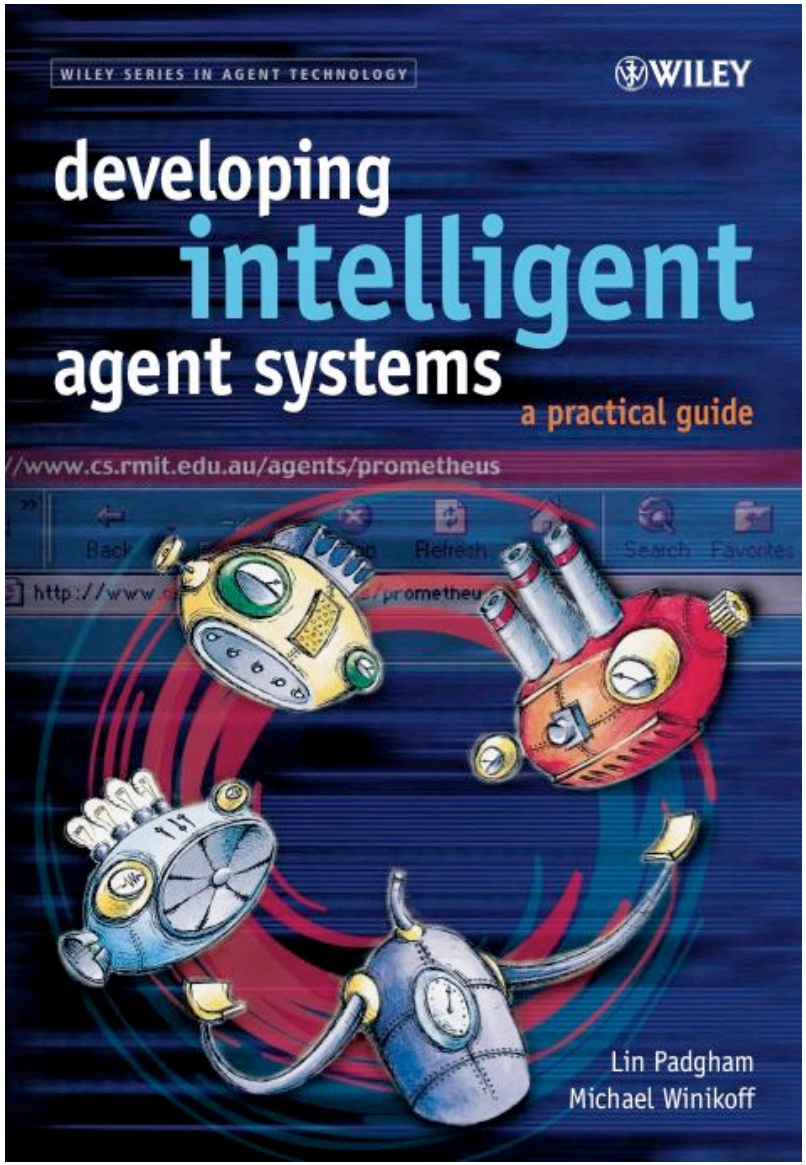
- Design tool supporting the Prometheus methodology developed at RMIT
- Written in Java, freely available on web

  www.cs.rmit.edu.au/agents/pdt
- Beta – feedback welcome!
- Features:
  - Enter and edit designs (descriptor forms, diagrams)
  - Cross checking:
    - undefined and unused entities
    - interface consistency
    - …
  - Generate design report (HTML) + images (PNG)


Prometheus
File Tools Entities View Help
Diagrams
Project Name
System Specification
Goal Overview
Functionalities
Scenarios
System Interface
Architectural Design
Data Coupling
Agent Acquaintance
System Overview
Detailed Design
Agent
Bond
System Overview Diagram
A1
Action
Data
Percept
Event
Bond
Agent
P1
Entities
System entities
Filter
Action
Agent
Bond
Data
Another Functionality
Functionality
Another Goal
Goal
Event
P1
P1 - Descriptor
Name: P1
Description:
Source:
Structure:
Auxiliary data:
Delete
Set

No suitable text books?

www.cs.rmit.edu.au/agents/prometheus



1. Agents and Multi-Agent Systems
2. Concepts for Building Agents
3. Overview of the Prometheus Methodology
4. System Specification
5. Architectural Design: Specifying the Agent Types
6. Architectural Design: Specifying the Interactions
7. Finalising the Architectural Design
8. Detailed Design: Agents, Capabilities and Processes
9. Detailed Design: Capabilities, Plans & Events
10. Implementing Agent Systems

A. Electronic Bookstore
B. Descriptor Forms
C. The AUML Notation

Conclusion

- Alternative Concepts to BDI
  - Easier to understand & use
  - (not covered in detail today)
- Methodology
  - Usable by undergraduates
  - Useful
- Tool Support
  - Design: cross checking important!
  - Debugging
- Book (coming soon!)

Why are Agents hard?

- BDI difficult to unde
  - BDI
  - Terminology, viewpoints
- Lack of Methodology

"One of ... fundamental obstacle... scale take-up of agent tec... is the lack of mature software development methodologies for agent-based systems."

- Imm... tools
- No suitable text book
- New - no consensus!

59

Conclusion

- Alternative Concepts to BDI
  - Easier to understand & use
  - (not covered in detail today)
- Methodology
  - Usable by undergraduates
  - Useful
- Tool Support
  - Design: cross checking important!
  - Debugging
- Book (coming soon!) real

Why are Agents hard?

- BDI difficult to unde
  - BDI
  - Terminology, viewpoints
- Lack of Methodology

"One of ... fundamental obstacles ... to large scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems."

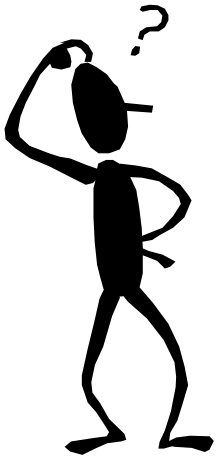
- Immature tools
- No suitable text book
- New - no consensus!

59

Acknowledgements

- Lin Padgham, James Harland
- Anna Edberg & Christian Andersson
- CS526 (Agent Oriented Programming and Design) students
- AOS (and especially Paul Maisano, Jamie Curmi, Ralph Rönnquist, Andrew Hodgson, Andrew Lucas)
- Anyone I've forgotten (with apologies!)

Questions?



http://www.cs.rmit.edu.au/agents/SAC

➔ http://www.cs.rmit.edu.au/agents ➔

http://www.cs.rmit.edu.au/~winikoff

winikoff@cs.rmit.edu.au