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

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

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/)
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.
• 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
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)

...
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
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

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

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

Key

- functionality
- data
- functionality reads data
- functionality writes data

Diagram:

- A → W
- B → X
- C → Y
- Z

Connections:
- A → W → D
- B → X → E
- C → Y → F
- Z → G
Data Coupling Diagrams

Key

- **functionality**
- **data**
- **functionality reads data**
- **functionality writes data**

```plaintext
A ——— W ——— D
    |       |       |
    |       |       |
    B ——— X ——— E
    |       |       |
    |       |       |
    C ——— Y ——— F
    |       |       |
    |       |       |
    Z ——— G
```
### Data Coupling Diagrams

**Key**
- **functionality**
- **data**
- **functionality reads data**
- **functionality writes data**

![Diagram](https://via.placeholder.com/150)

- **A**
- **B**
- **C**
- **W**
- **X**
- **Y**
- **Z**
- **D**
- **E**
- **F**
- **G**

- Red arrows indicate data flow:
  - **functionality reads data**
  - **functionality writes data**
Data Coupling Diagrams

Key

functionality

data

functionality reads data

functionality writes data

Diagram:

- A
- B
- C
- W
- X
- Y
- Z
- D
- E
- F
- G

Arrows indicate data flow:
- A to W
- A to D
- B to X
- B to E
- C to Y
- C to F
- Z to G
Data Coupling Diagrams

Key

- functionality
- data
- functionality reads data
- functionality writes data

A → W
B → X
C → Y
D
E
F
G

X → Y

Z
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

Undesirable
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
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:**
  – A UML (Agent UML)
  – A UML-2
  – UML activity diagrams
  – ...
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
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?
• 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?
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;
}
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)