Agent Oriented Software Engineering

Michael Winikoff and Lin Padgham

Chapter 15 of
Multiagent Systems
Edited by Gerhard Weiss

MIT Press, 2012

http://www.the-mas-book.info/

Introduction

- Agent Oriented Software Engineering (AOSE) concerned with ***engineering aspects of developing MAS***.
- Focus on methodologies and tools
- This chapter aims to:
  1. Give a sense for what an AOSE methodology looks like.
  2. Describe the current state of work in the area of AOSE.

Methodology

- Methodology includes:
  - overall **process**
  - which produces design artefacts ("**models**")
  - **notation** used to capture the models
  - **techniques** (i.e. how to do things – heuristics)
  - underlying **concepts**
  - **tool support** very valuable, but not focus of chapter
- Activities follow typical development life cycle: requirements, design, implementation, assurance, maintenance
  - … but typically done iteratively, not sequentially (i.e. not waterfall)

History of AOSE: Three Generations

Year	Methodologies
1995	DESIRE
1996	AAII, MAS-CommonKADS
1999	MaSE
2000	Gaia (v1), Tropos
2001	MESSAGE, Prometheus
2002	PASSI, INGENIAS
2003	Gaia (v2)
2005	ADEM
2007	O-MaSE

1. mid to late 90s
   - Examples: DESIRE, AAII, MAS-CommonKADS, Gaia
   - Generally briefly described
   - Lacking tool support
   - May not cover full life cycle
2. late 90s to early 00s
   - Examples: MaSE, Tropos, MESSAGE, Prometheus
   - More detailed descriptions
   - Tend to have tool support
   - Tend to cover Requirements to Implementation
3. mid to late 00s
   - Examples: PASSI, INGENIAS, ADEM
   - Increased focus on UML and Model-Driven Development
   - Tend to be more complex

Historical Observations

- Reduced focus on developing new methodologies
- Reduction over time in number of actively developed methodologies
- … increased focus on standardisation and consolidation?

Agent Concepts

- Agents defined as having certain properties
- Design agents with these **properties** using supporting **concepts**

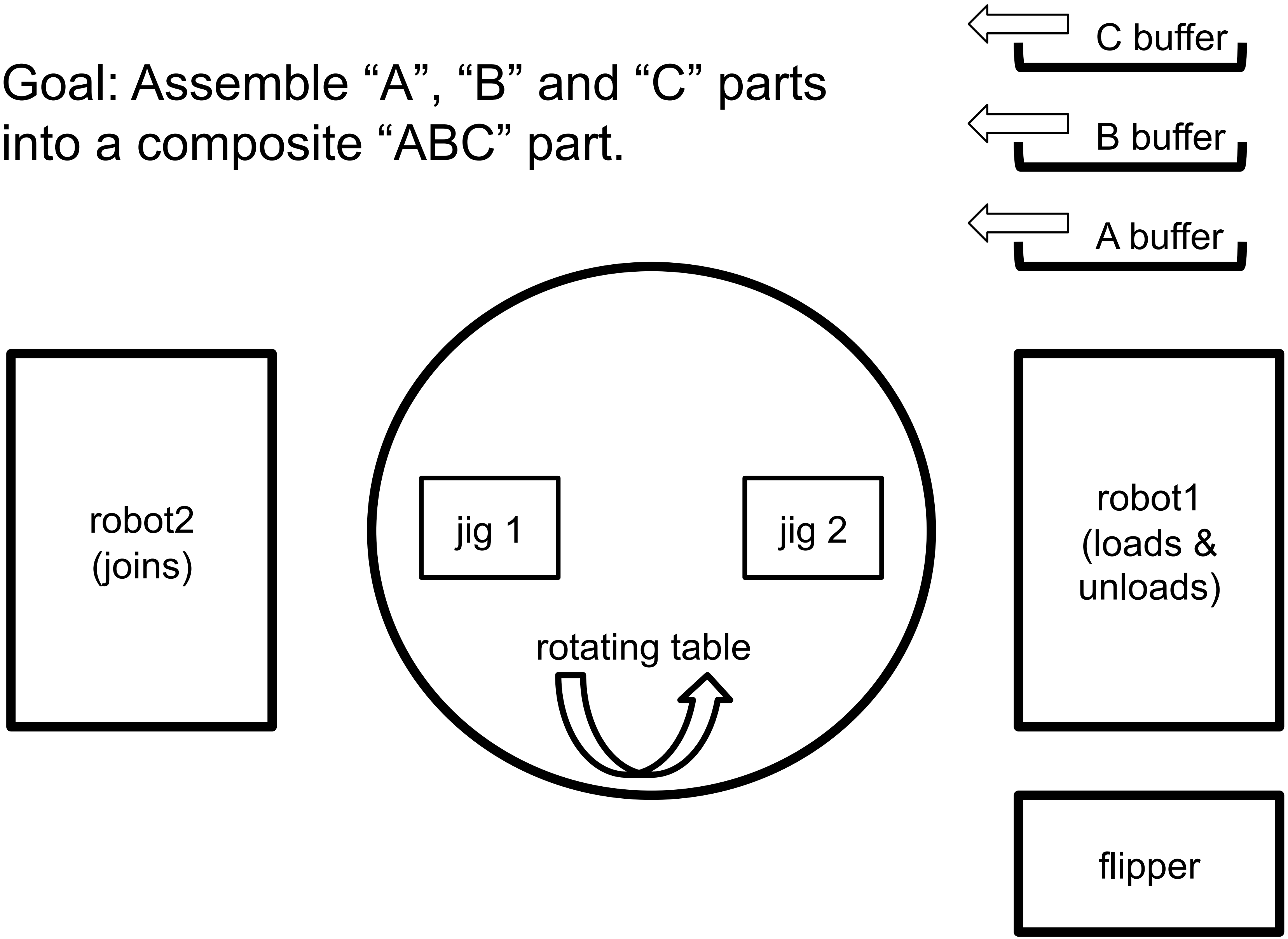
Property	**Supporting Concepts**
Situated	Action, Percept
Proactive & Autonomous	Goal
Reactive	Event
Social	Message, Protocol …

Agent Concepts

- Design **situated** agents by modeling interface with environment in terms of **actions** and **percepts**
- Design **proactive** and **autonomous** agents using **goals**
- Achieve **reactivity** using **events**
- Agents interact with each other (**social**) using **messages** and **protocols**

Example: Holonic Manufacturing

Goal: Assemble “A”, “B” and “C” parts into a composite “ABC” part.



Process for making an “ABC” part

1. robot1 loads an A part into one of the jigs on the rotating table
2. robot1 loads a B part on top of it
3. the table rotates so the A and B parts are at robot2
4. robot2 joins the parts together, yielding an “AB” part
5. the table rotates back to robot1
6. if an AB part is required, robot1 unloads the part, else continue
7. robot1 moves the AB part to the flipper
8. the flipper flips the part over (“BA”) at the same time as robot1 loads a C part into the jig
9. robot1 loads the BA part on top of the C part
10. the table rotates
11. robot2 joins the C and BA parts, yielding a complete ABC part
12. the table is rotated, and
13. robot1 then unloads the finished part.

Actions and Percepts in the Holonic Manufacturing Example

Robot1:

- percept: manufacture (composite)
- load(part) into jig
- unload()
- moveToFlipper()
- moveFromFlipper()

Robot2:

- join(jig): join the bottom part to the top part

Flipper:

- flip() the item in the flipper

Table:

- rotateTe(jig, position)

REQUIREMENTS

Requirements

- Requirements concerned with ***defining the required functionality of the system-to-be***.
- Commonly used activities:
  - specifying instances of desired behaviour using ***scenarios***
  - capturing system ***goals*** and their relationships
  - defining the ***interface*** between the system-to-be and its environment
- These activities are typically done in parallel in an iterative manner
- Some methodologies define *roles* …

Roles

- Coherent grouping of related goals, percepts, actions
- Example:
  - **manager**: this role is responsible for overall management of the manufacturing process. It does not perform any actions.
  - **pickAndPlacer**: this role is responsible for moving parts in and out of the jig when it is located on the East side of the table. Associated actions are: load, moveToFlipper, moveFromFlipper, unload
  - **fastener**: this role is responsible for joining parts together. Associated action: join.
  - **transporter**: this role is responsible for transporting items by rotating the table. Associated action: rotateTo.
  - **flipper**: this role is responsible for flipping parts using the “flip” action.

Requirements: Scenarios

- Similar to OO use cases, but more details in some methodologies
- Structure and format varies
- Example (right) shows goals and actions, along with the roles

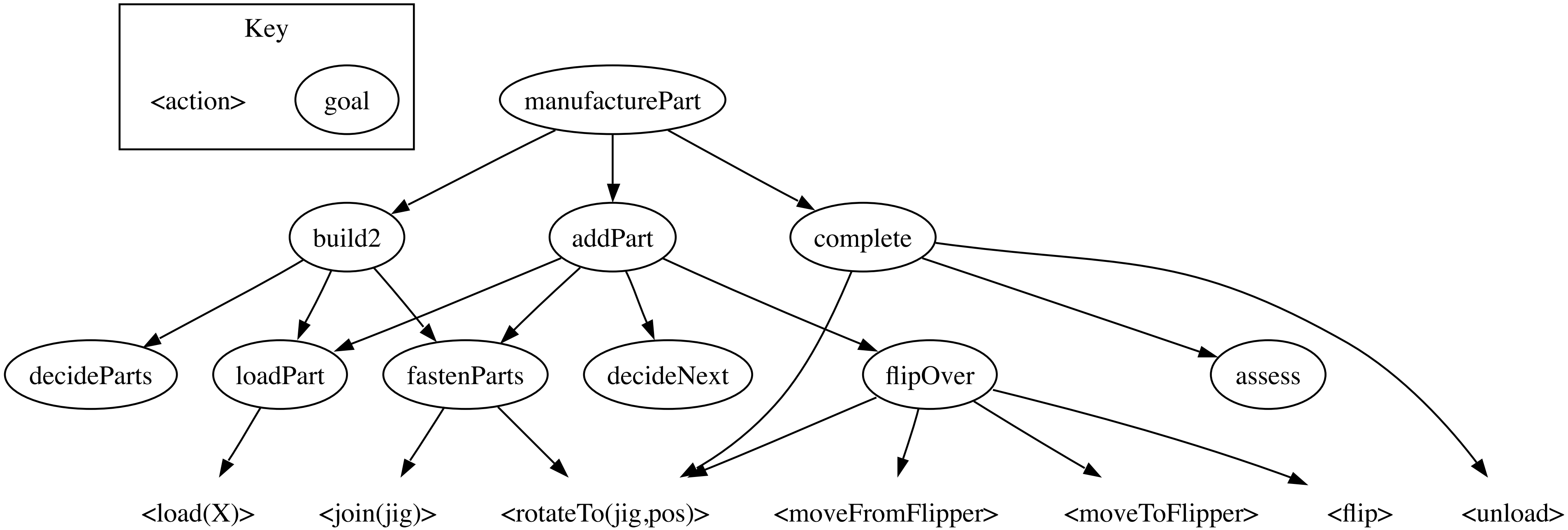
Scenario: manufacturePart(ABC)

Type			**Name**	**Roles**
G			build2	manager, pickAndPlacer, fastener
	G		decideParts	manager
	G		loadPart	pickAndPlacer
		A	load(A)	pickAndPlacer
	G		loadPart	pickAndPlacer
		A	load(B)	pickAndPlacer
	G		fastenParts	fastener, transporter
		A	rotateTo(1,W)	transporter
		A	join(1)	fastener
G			addPart	manager, pickAndPlacer, fastener
	G		decideNext	manager
	G		flipOver	manager
		A	rotateTo(1,E)	transporter
		A	moveToFlipper()	pickAndPlacer
		A	flip()	flipper
	G		loadPart	pickAndPlacer
		A	load(C)	pickAndPlacer [in parallel with flip]
		A	moveFromFlipper()	pickAndPlacer
	G		fastenParts	fastener, transporter
		A	rotateTo(1,W)	transporter
		A	join(1)	fastener
G			complete	manager
	G		assess	manager
	A		rotateTo(1,E)	transporter
	A		unload()	pickAndPlacer

Requirements: Goals

- Capture goals of system
- Is complementary to scenario
  - not specific to a given trace, but doesn’t capture sequencing.
- Extract initial goals from the scenario
- Refine by asking “why?” (gives parent goal) and “how?” (gives child goals)
- Results in goal model, e.g. goal tree
  - Some methodologies have richer notations
  - Example (next slide) also shows actions

Example Goal Model



Requirements: Environment

- Specify interface to environment in terms of actions and percepts
- May be pre-determined by problem
  - e.g. robot capabilities in Holonic Manufacturing

- Overlap exists between the three models (scenarios, goals, environment interface).
- Hence each model influences the others …

Variations on requirements

- Some methodologies have an *early requirements* phase that captures the context of the system-to-be in terms of stakeholders, and their goals and dependencies.
- Capturing domain concepts ("ontology") is important – can use UML class diagrams, Protégé …
- Some work has proposed richer environmental models (e.g. chapters 2 and 13)

DESIGN

Design

- Design aims to define the overall structure of the system by answering:
  - What agent types exist, and what (roles and) goals do they incorporate?
  - What are the communication pathways between agents?
  - How do agents interact to achieve the system's goals?

Design

- Two key models:
  - a **static** view of the system's structure, and
  - a model that captures the **dynamic** behaviour of the system.
- Also capture shared data.

“What agent types exist?”

- Common technique is to consider grouping of smaller “chunks” (e.g. roles), taking into account:
  - the degree of **coupling** between agents,
  - the **cohesiveness** of agent types, and
  - any other reasons for keeping “chunks” separate (e.g. deployment hardware, security, privacy)
- No single “right” answer – technique is about identifying tradeoff points.

"What agent types exist?"

In the Holonic Manufacturing example:

- Natural to have each robot be a separate agent …
- … but assign pickAndPlacer and manager roles to Robot1

Role		Agent Type	Goals and Actions[3]
pickAndPlacer	$\rightarrow$	Robot1	loadPart, *load, unload, moveToFlipper, moveFromFlipper*
manager	$\rightarrow$	Robot1	decideParts, decideNext, flipOver, assess
transporter	$\rightarrow$	Table	*rotateTo*
fastener	$\rightarrow$	Robot2	fastenParts, *join*
flipper	$\rightarrow$	FlipperRobot	*flip*

italics = actions

System (static) structure

- Specified using System Overview Diagram
- First, derive goals, action and percept assignment.
  - Derived from the role-agent assignment: a role’s goals, actions and percepts are assigned to the agent that plays that role.
- Then consider communication …

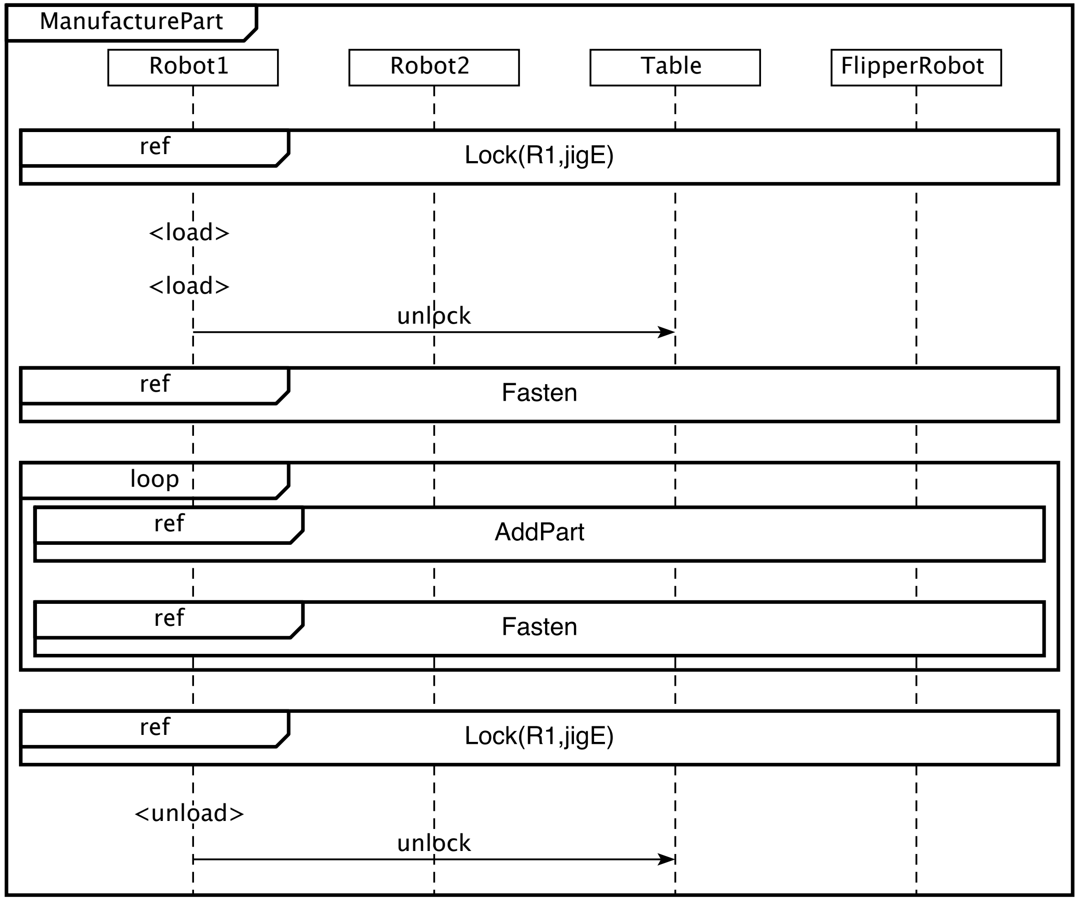
System Dynamics

- Captured using interaction protocols
- Process:
  - Begin with scenarios
  - Insert messages where communication is needed (i.e. when step N by agent A is followed by step N+1 by a different agent)
  - Generalise: “what else could happen here?”, “what could go wrong?”
- Agent UML (AUML) sequence diagrams often used for depicting interaction protocols.

Holonic Manufacturing Protocols

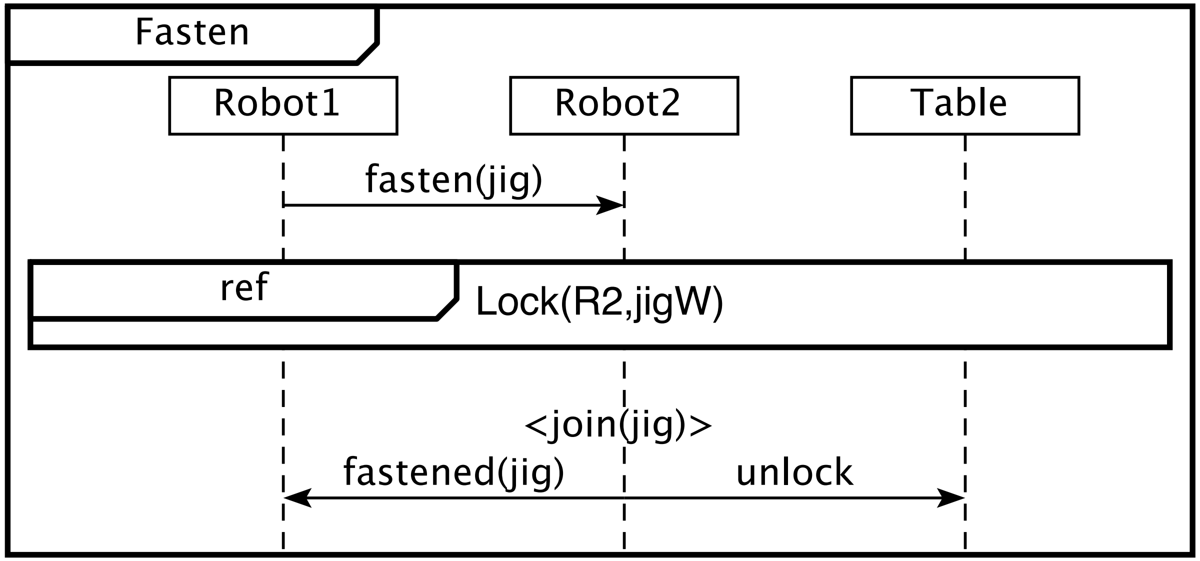
Top level protocol shows manufacturing process:

- initial loading of two parts and joining them
- then repeatedly adding a part and fastening it
- finally, unload the result



Fasten Protocol

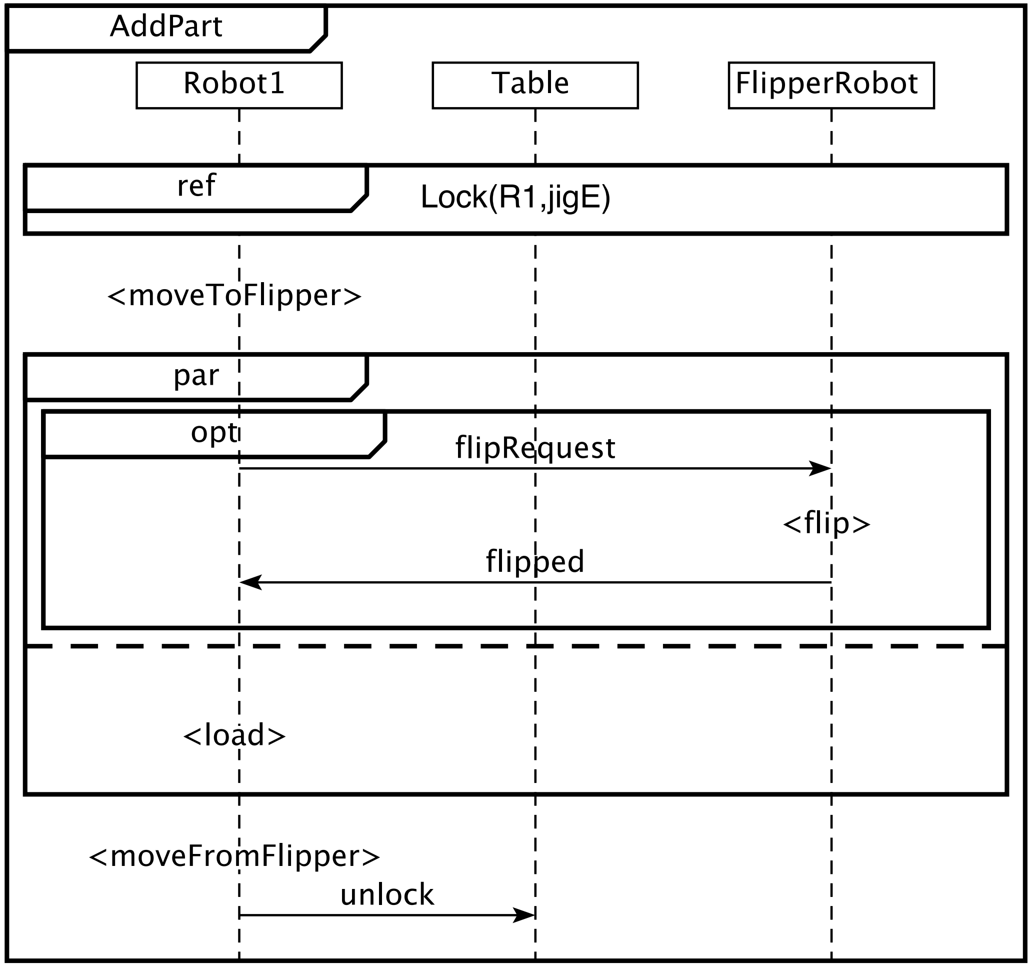
- Fasten protocol simply involves a request (from Robot1 to Robot2) to fasten.
- Robot2 then locks the table, performs the *join* action, and informs Robot1



AddPart Protocol

AddPart protocol shows:

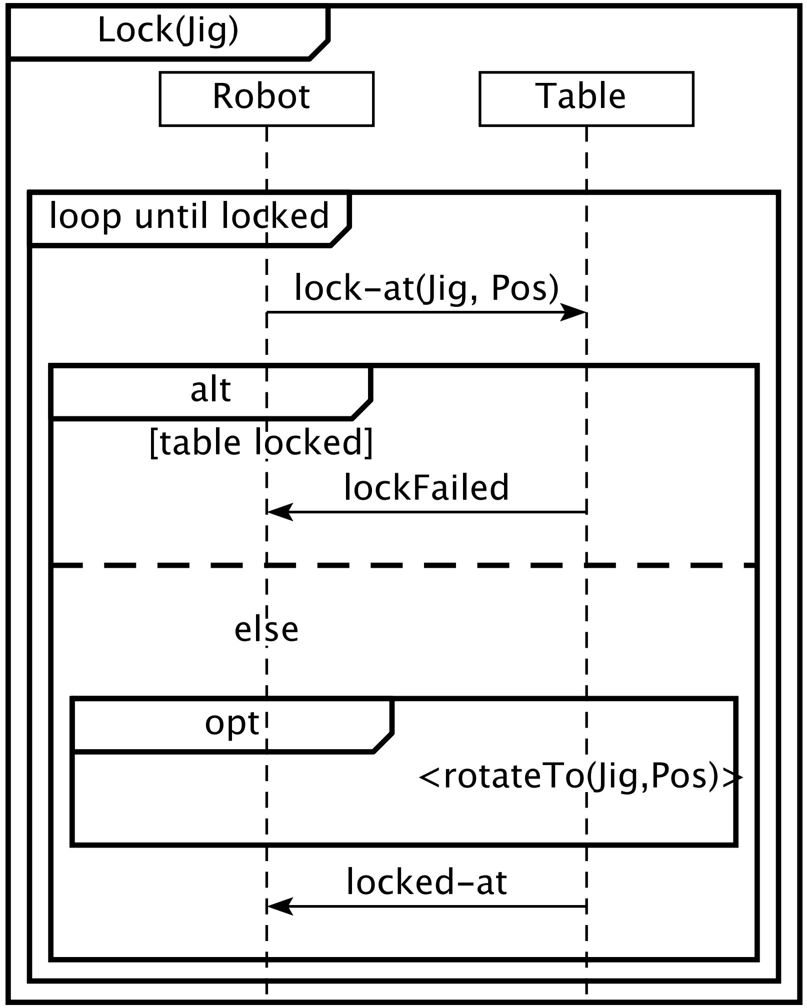
- the table being locked
- then the existing part is moved to the flipper
- then the new part is loaded (and, optionally, the old part is flipped at the same time (“par”))
- and then the old part is moved back



Note that showing actions in the protocol (e.g. “<load>”) is needed to show clearly what’s going on.

Lock Protocol

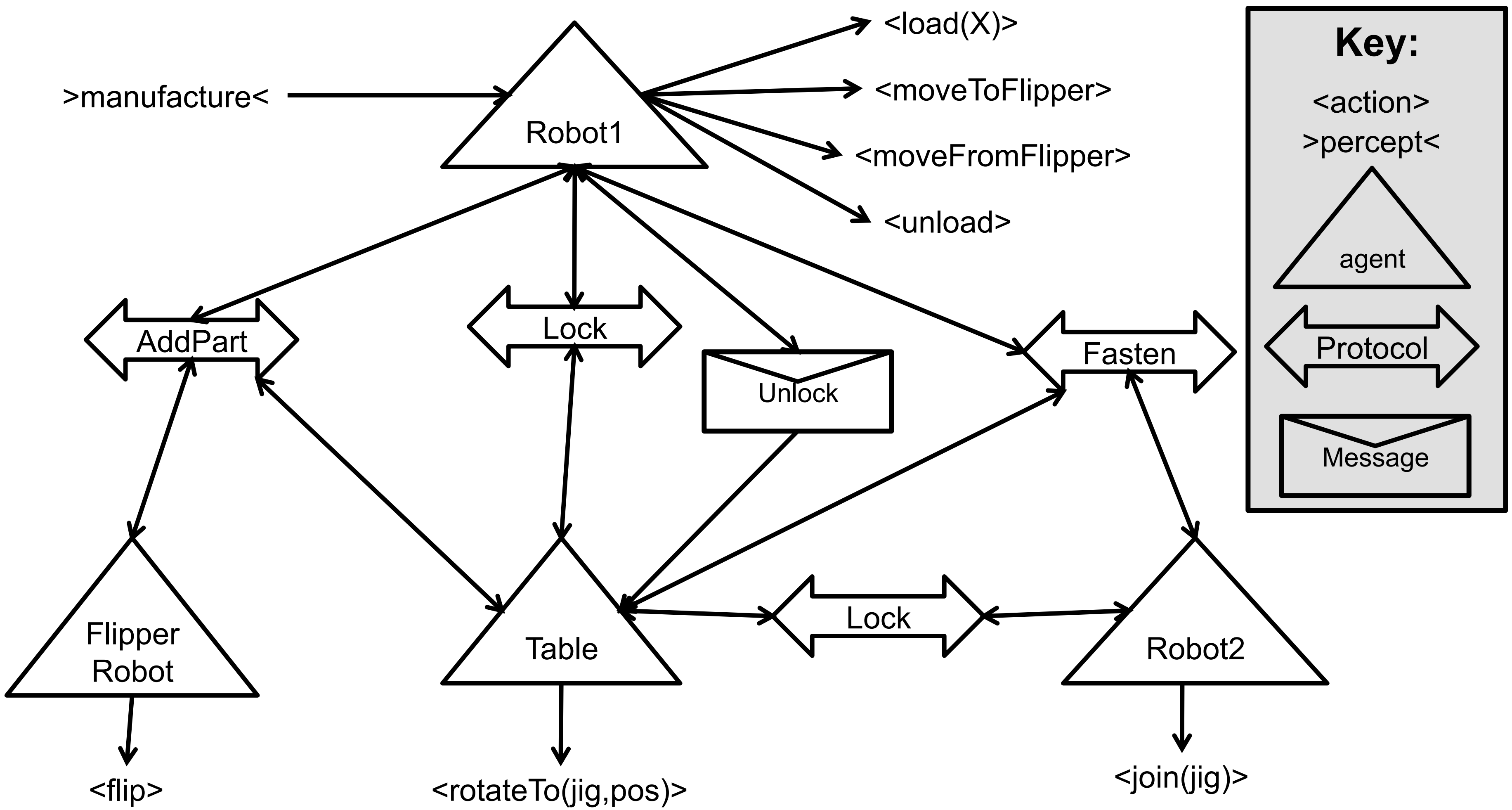
- The Lock protocol is a simple request-response (“please lock”, “ok”) …
- … extended to deal with failure (by retrying),
- and with an optional rotation to the desired position



System Overview Diagram

- Having developed the protocols, we can now capture the system’s (static) structure using a System Overview Diagram (next slide)
- May also need to define shared data at this point.

System Overview Diagram



DETAILED DESIGN

Detailed Design

- Detailed design aims to **specify the internal structure of each agent**, so that implementation can be done.
- Do this by starting with each agent's interface (messages sent/received, actions, percepts, goals) and defining its internals.
- To do this, need to know the target implementation platform type
- We consider two examples:
  - A Belief-Desire-Intention (BDI) platform
  - A design using a Finite State Machine (FSM) targeting JADE

Example: Robot1

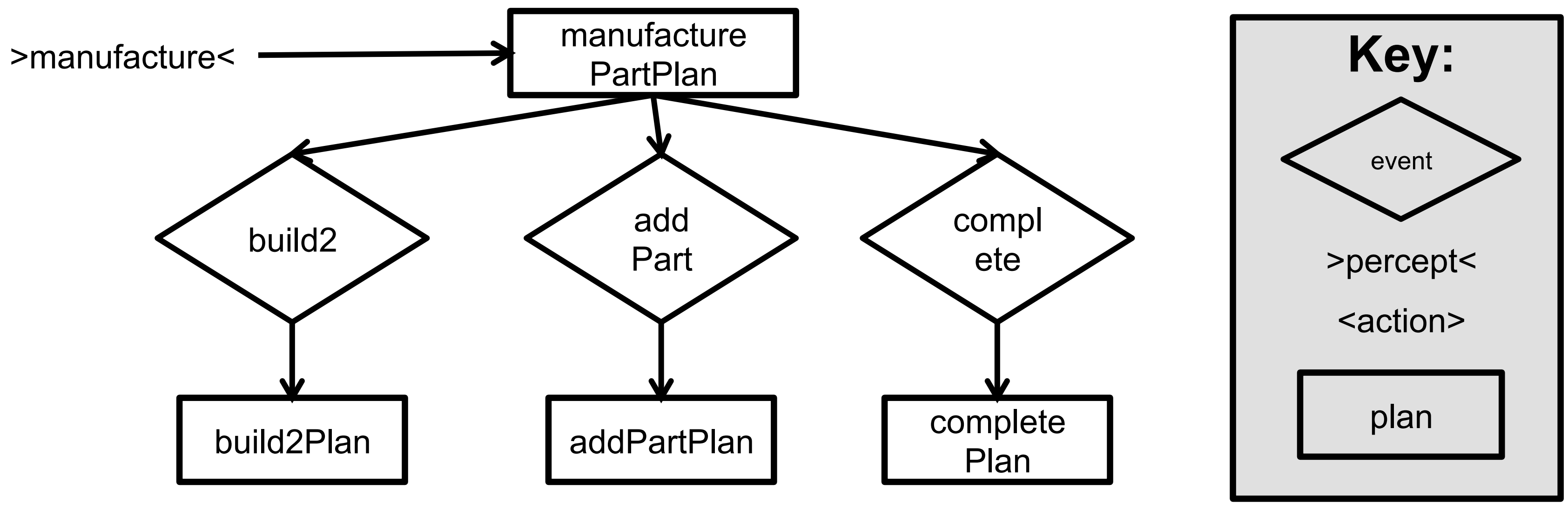
- We know that Robot1:
  - participates in the protocols: AddPart, Lock, Fasten
  - has actions: load, unload, moveToFlipper, moveFromFlipper
  - receives percept manufacture
  - has goals: loadPart, decideParts, decideNext, flipOver
- What plans and internal events does Robot1 need to play its part?

BDI Platform Design

- BDI platforms define an agent in terms of a collection of **plans** that are triggered by **events** (or messages).
  - Each event may trigger more than one plan – which plan to use is determined by the plan's *context condition*
- To capture detailed design use an **Agent Overview Diagram** for each agent type
  - This shows plans, events, messages, percepts and actions; and the relationships between them

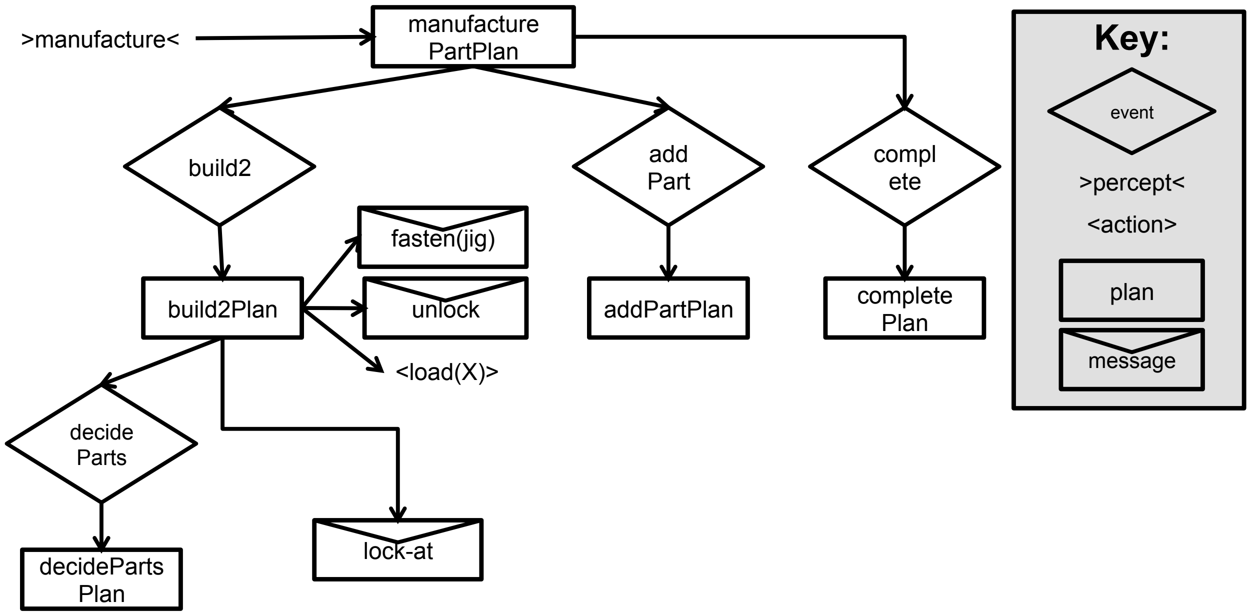
Example: Initial Structure

- Start by creating a plan to handle the percept
- This plan then posts events corresponding to the subgoals
- Each of these events needs a plan to handle it



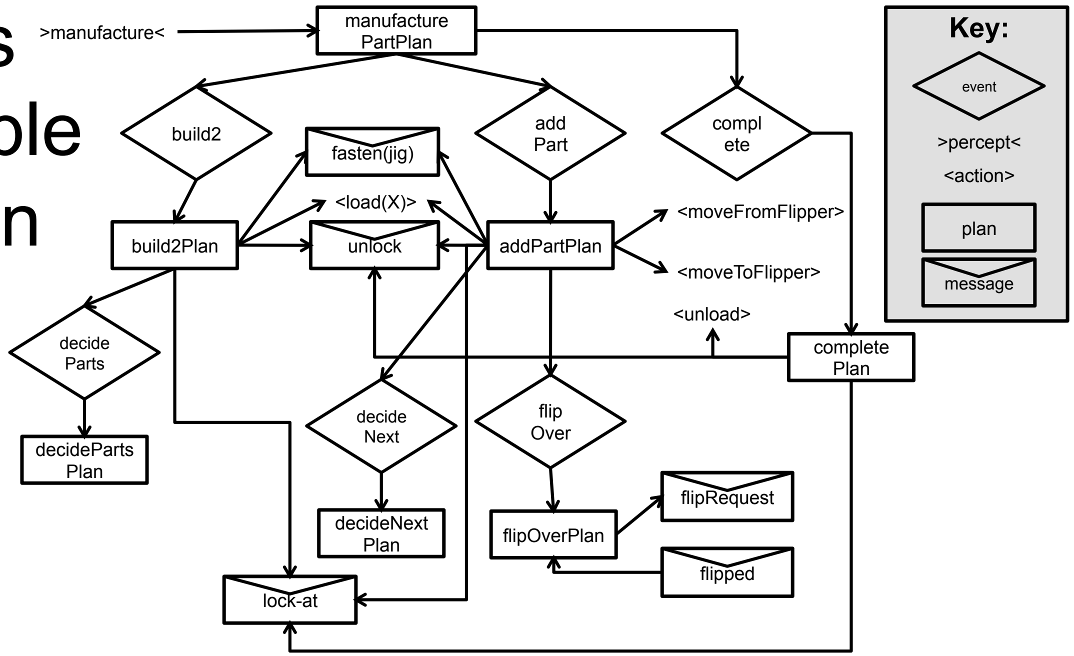
Example: developing build2

- The build2Plan posts events corresponding to its subgoals – loadPart simply becomes the action load
- Since the fastenParts subgoal is performed by Robot2, instead of posting an internal event, send a message (to Robot2)
- Also need to lock and unlock, so add these messages



Example: addPart and complete

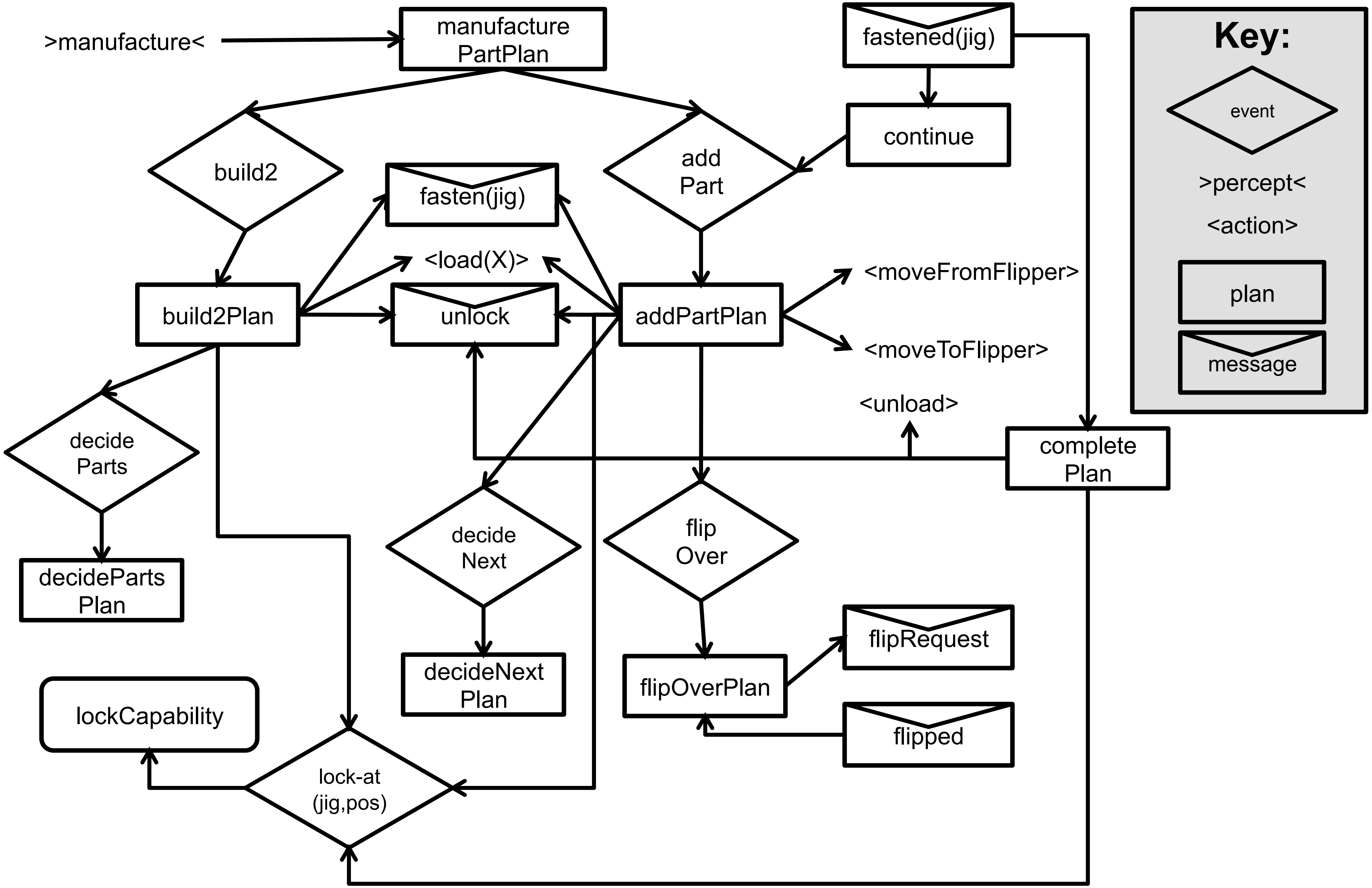
- addPart has subgoals loadPart, fastenParts, decideNext and flipOver – add them.
- add messages in line with the interaction protocols
- subgoal “assess” is handled by a suitable context condition on the completePlan



Example: final BDI design

- Check that all messages that Robot1 should be able to send or receive are in the detailed design
  - sent: lock-at, fasten, flipRequest, unlock - all present in design
  - received messages missing: lockFailed, locked-at, fastened, flipped – add, and ensure there is a plan that deals with each incoming message.
- Use *capability* to encapsulate lock management

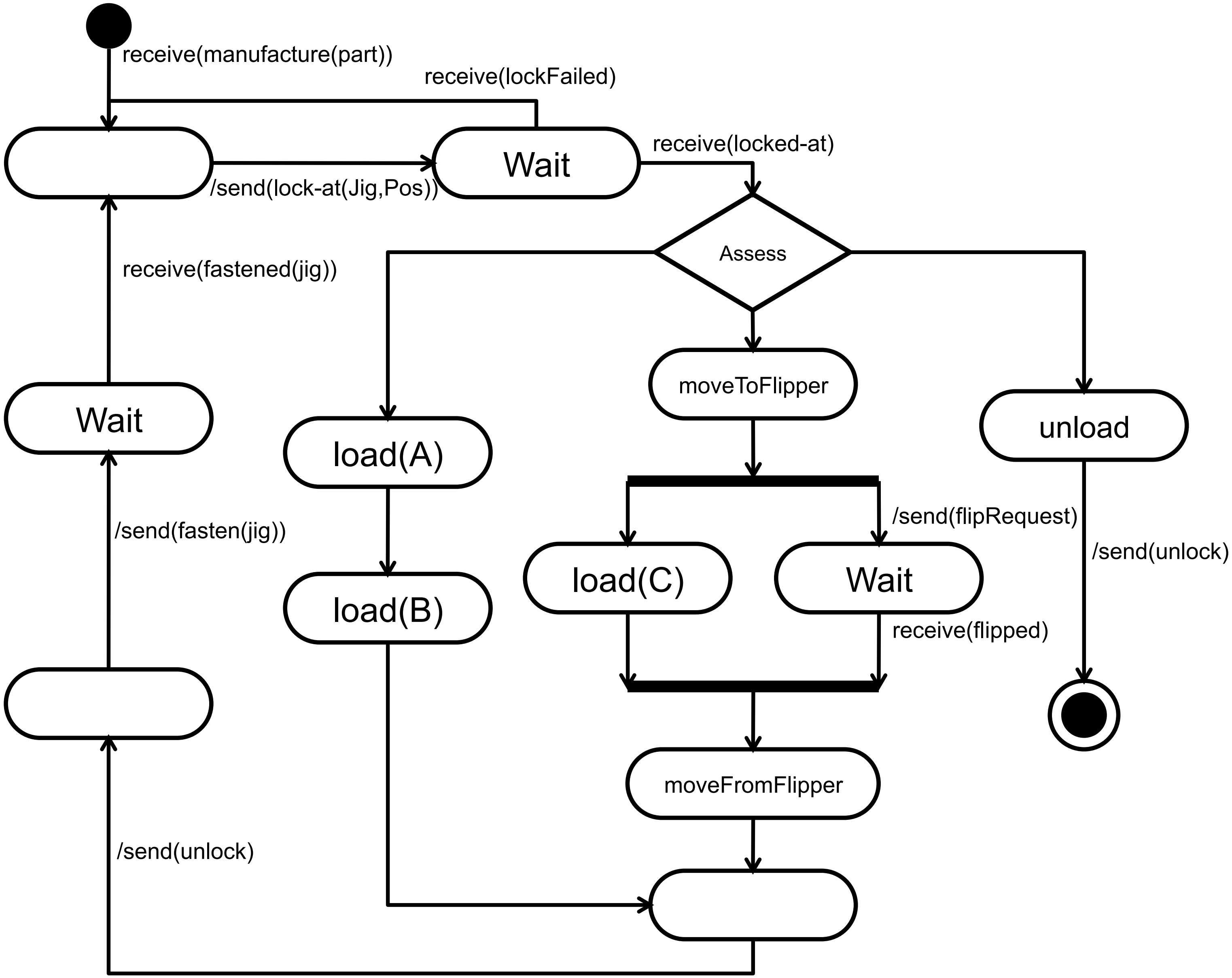
Final BDI Design for Robot1



Approach II: FSM

- Derive internal process for Robot1 by identifying states of interaction (gaps between messages in the protocols)
  - messages are transitions between states
  - compress interactions that don't involve Robot1 (e.g. Robot2 and Table locking the table in the Fasten protocol)

Final FSM Design for Robot1



Implementation

- Mapping detailed design to implementation generally done manually
- Some design tools can generate skeleton code in an agent-oriented programming language
- Some work on round-trip engineering exists
- Some work has been done on model driven development of agent systems
  - implementation generated from design
  - … but design expands to include additional information to make this possible …

Assurance

- Support for this is less well developed than support for “core” activities (requirements, design, detailed design).
- Much of the work in testing and debugging uses information created during design
  - e.g. using interaction protocols to monitor system execution

Testing and Debugging

- Testing agents is hard: concurrent systems, with goal-directed flexible behaviour …
- Testing takes places at different levels: units, modules, integration, system, and acceptance
- Testing has different aspects: test case design, execution, and checking of test results.

Testing and Debugging (2)

- Most well developed contemporary methodologies provide some support for automated execution of tests, and checking test results.
- … but test case *generation* is usually manual

Testing and Debugging (3)

- Tropos has a tool (eCAT) that provides support for test case generation
  - This uses ontologies to generate message content
- Prometheus has work on test case generation

- *But is a given set of tests adequate?*

Testing Adequacy

- Given the complexity of agent systems, a set of tests may not be adequate
- There has been some work on adapting existing notion of code coverage to agents
- But this work is not yet used in agent testing tools

Formal Methods

- The difficulty of testing agent systems has motivated the development of formal methods
- Formal methods use mathematical techniques to *prove* that a system is correct (with respect to its formalised specification)
- Much of the work uses *model checking*, where an (often abstract) model of the system is systematically checked against a specification
- But current state-of-the-art is still limited to very small programs (e.g. six line contract net with three agents)

Software Maintenance

- Once software has been deployed, it is subject to further change, such as:
  - adapting to changes in its environment
  - adding new functionality
- Only one piece of work that has looked at maintenance of agent systems
- Dam *et al*. focused on change propagation in design models: given a change to a design model, what other changes are needed to restore consistency of the model?

Comparing Methodologies

- In the early days of the field there were many methodologies
- This prompted work (around 2001-2003) on *comparing* methodologies
- Typical approach was feature based:
  - Define a list of features of interest
  - Assess each methodology against each feature, resulting in a large table
- Unfortunately this approach suffers from subjectivity
  - … in some cases even the authors of a methodology didn’t agree on how to rate their methodology on given criteria!

Conclusions

- Areas for further research:
  - Understanding the benefits of the agent paradigm
  - Designing flexible interactions
  - Extending methodologies to deal with systems
    - … that have complex and dynamic organisational structure
    - … that have many simple agents with emergent behaviour
    - … that are an *open* society of agents
  - Techniques for assurance of agent systems

Conclusions

- Areas for further work (not research):
  - Standardisation of methodologies
    - Reduce unnecessary differences between methodologies
    - Enable reuse of tool development, rather than duplicated effort
    - One approach that has been proposed is *method engineering*
  - Integration of agent practices, standards and tools with mainstream