Software in Robot : Software Engineering Perspectives

Sooyong Park
Computer Science and Engineering
Sogang University
sypark@sogang.ac.kr

Background

- The research field of Intelligent Service Robots …
  - Has become more and more popular over the last years.
  - Covers a wide range of applications from cleaning robots to robotic assistance for disabled or elderly people.
- Public Service Robot (PSR) systems have been developed for indoor service tasks at Korea Institute of Science and Technology (KIST)
  - *PSR-1,*
  - *PSR-2 and*
  - *Jinny*

Background (cont.)

- The worldwide population of elderly people is rapidly growing and is set to become a major problem in the future.
  - This can lead to a huge market for assistive robots.

- In this context, the intelligent service robot for the elderly, called ***T-Rot*** is under-developed at Center for Intelligent Robotics (CIR).

Background (cont.)

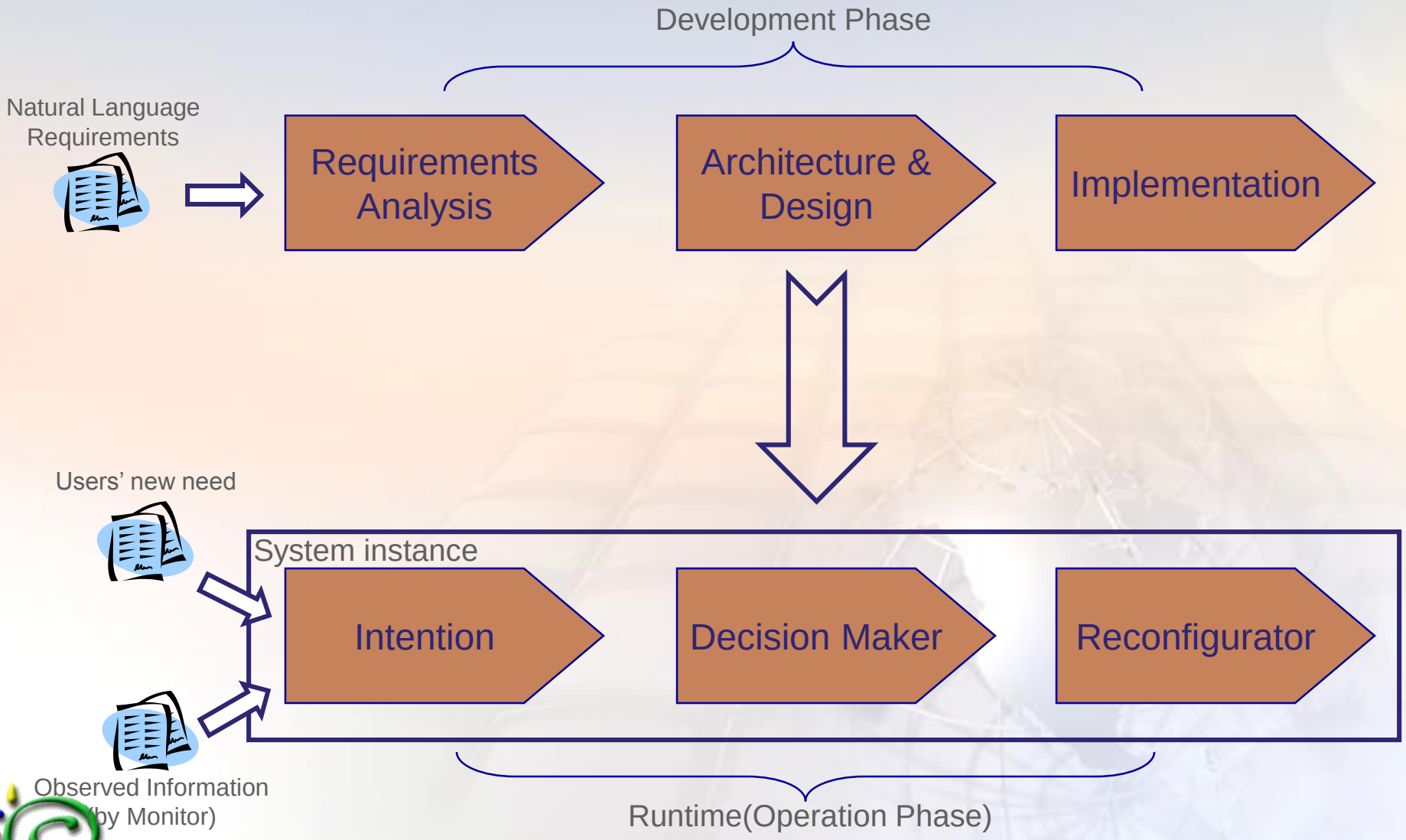
Issues

- Complexity of Software in Robot
  - More 10 groups consisting of more than 150 researchers and engineers from academia and industry.
  - 9 years project that is divided into three stages.
- Dynamics in Environments and user's needs during run-time
  - Changes in user needs
  - Various home environments/resources
  - Unexpected faults

Challenges

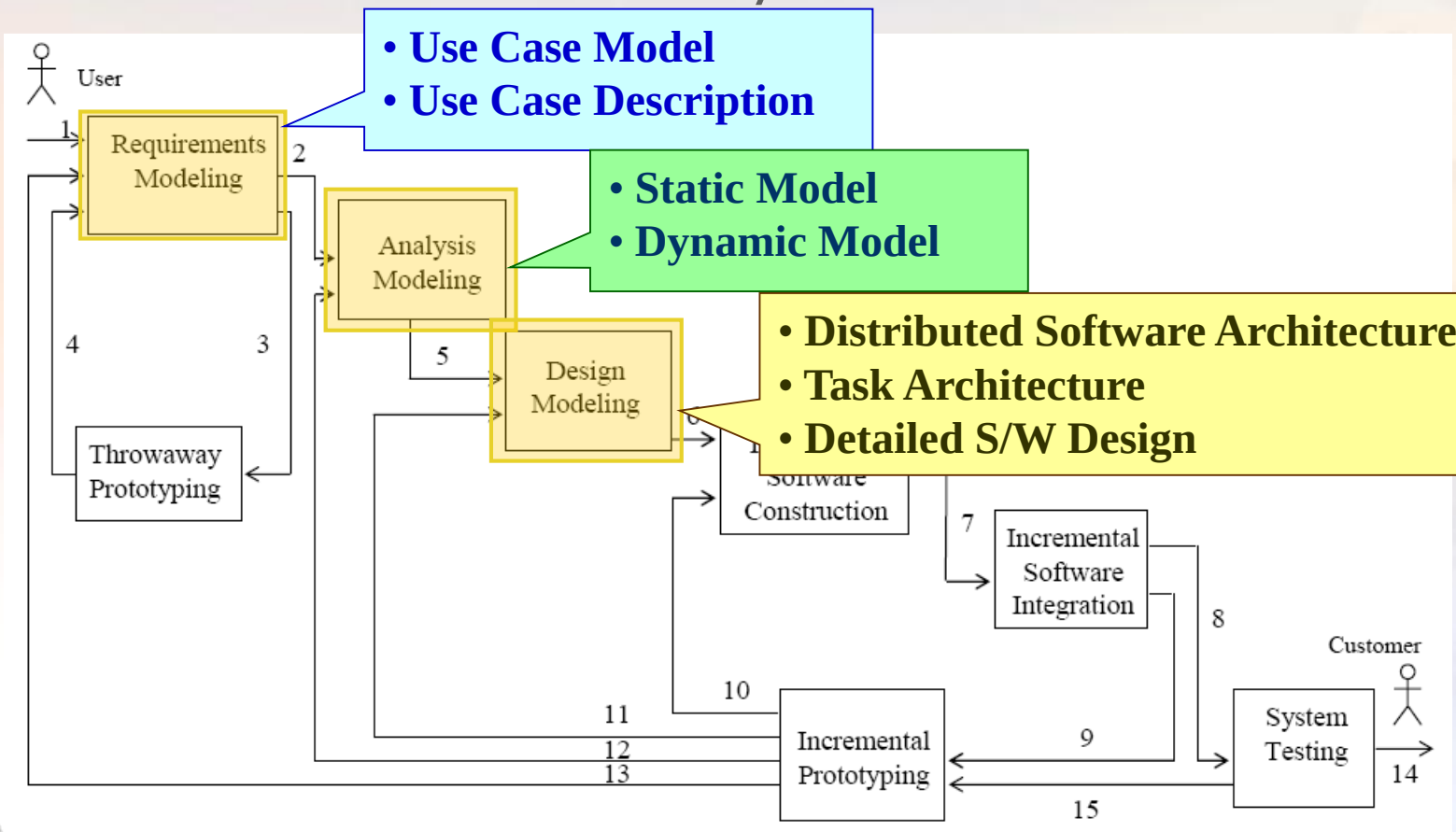
- Software Engineering for Development time
- Software Engineering for run-time

New Software Engineering Approach

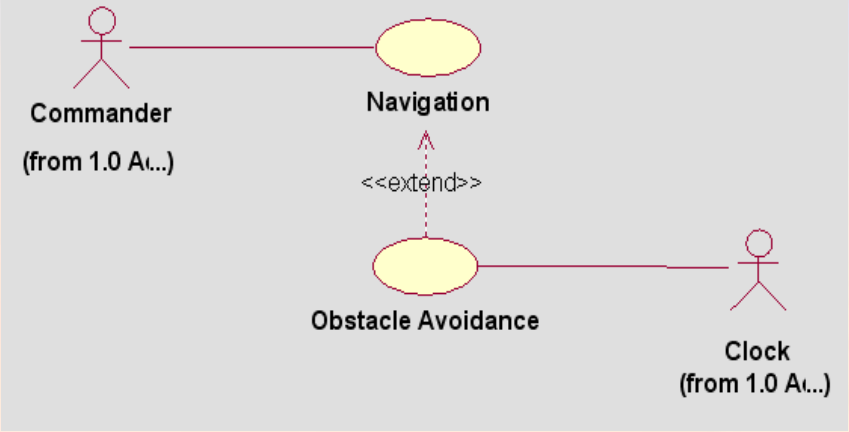


SE for Development Time : Applying the COMET/UML

- COMET Software Life Cycle Model



Requirements Modeling

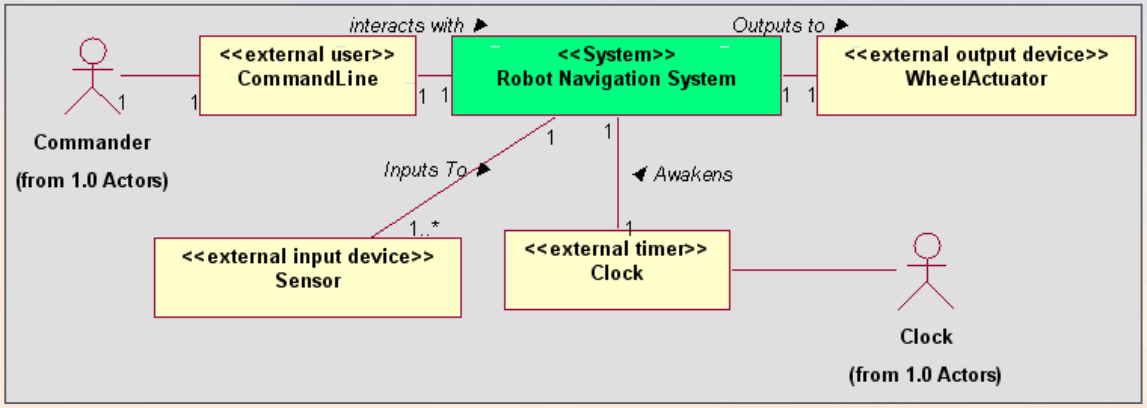


Use Case Diagram

Summary	The Commander enters a destination and the robot system moves to the destination.
Actor	Commander
Precondition	The robot system has the grid map and the current position is known
Description	1. The use case begins when the commander enters a destination. 2. The system calculates an optimal path to the destination. 3. The system commands the wheel actuator to start moving to the destination. 4. The wheel actuator notifies the system that it has started moving. 5. The system periodically reads sensor data and calculates the current position. 6. The system determines that it arrives at the destination and commands the wheel actuator to stop. 7. The wheel actuator notifies the system that it has stopped.

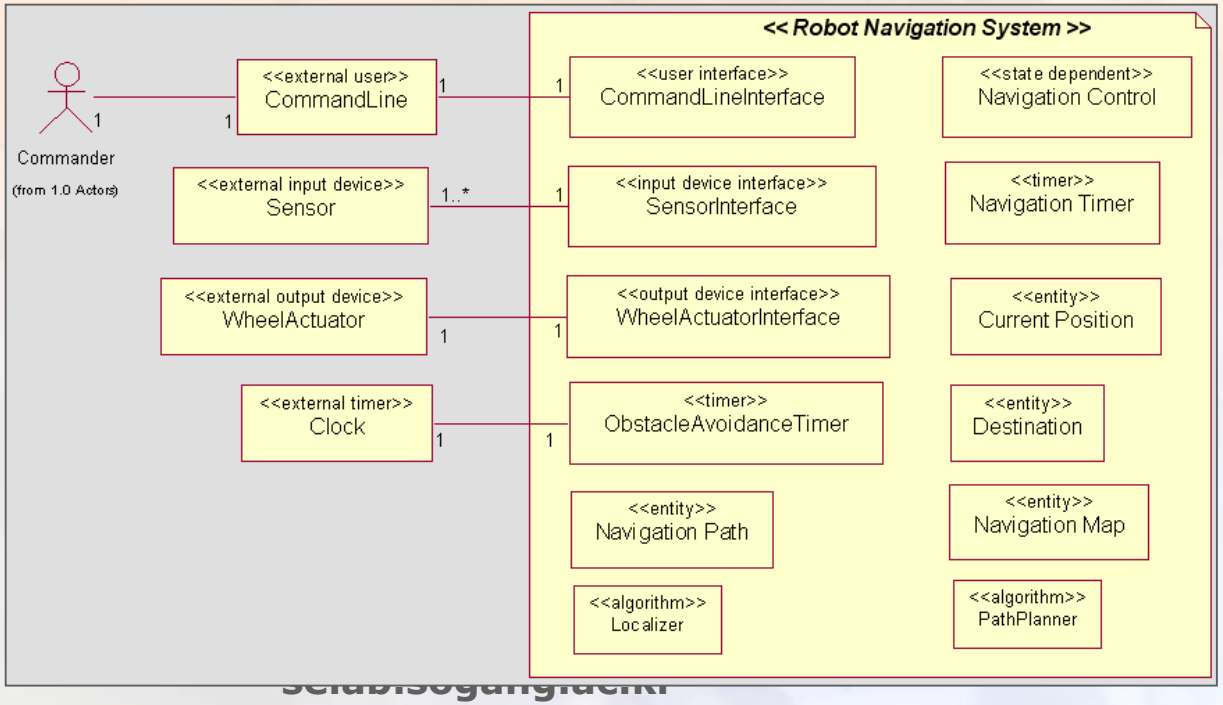
Use Case Description for Navigation

Analysis Modeling - Static Modeling

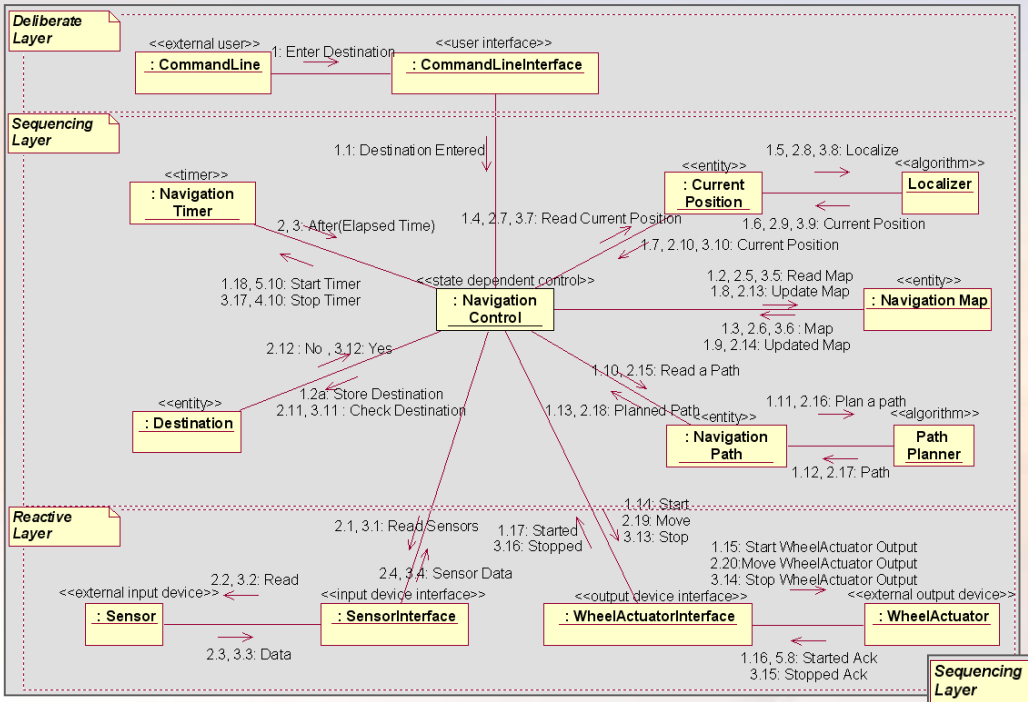


Robot Navigation System context class diagram

Object structuring class diagram for Robot Navigation System

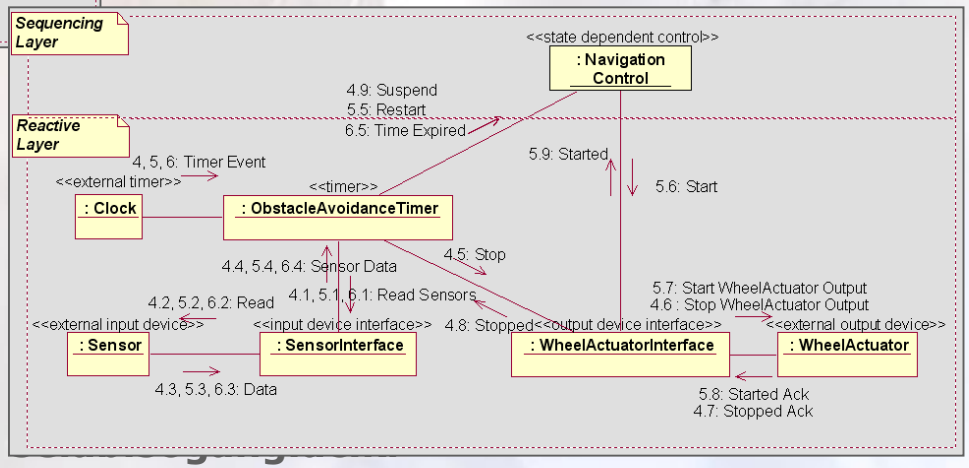


Analysis Modeling - Dynamic Modeling

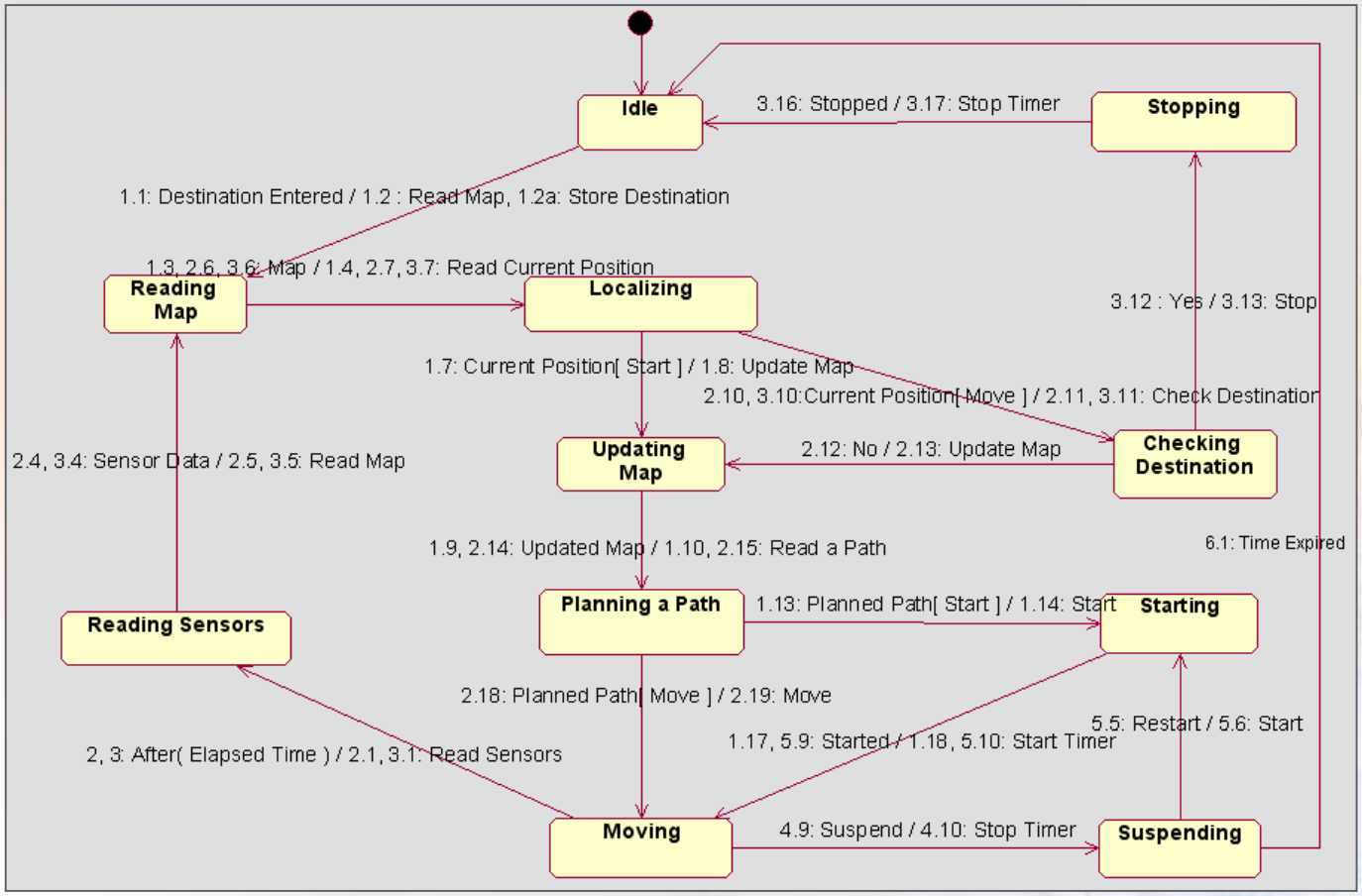


Collaboration diagram for Navigation use case

Collaboration diagram for Obstacle Avoidance use case

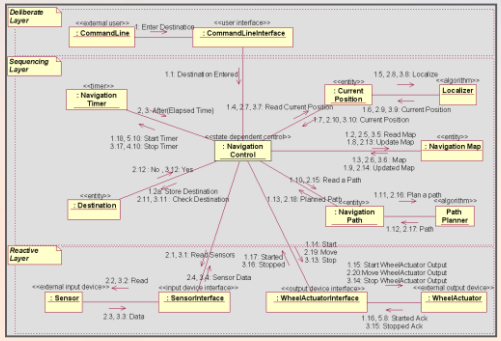


Analysis Modeling
- Dynamic Modeling (cont.)



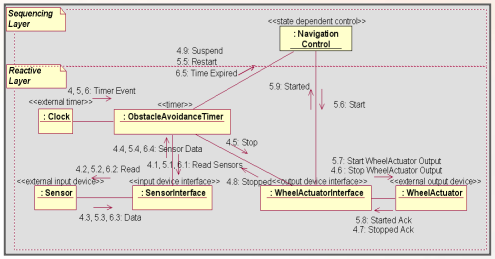
Statechart for Navigation Control

Design Modeling

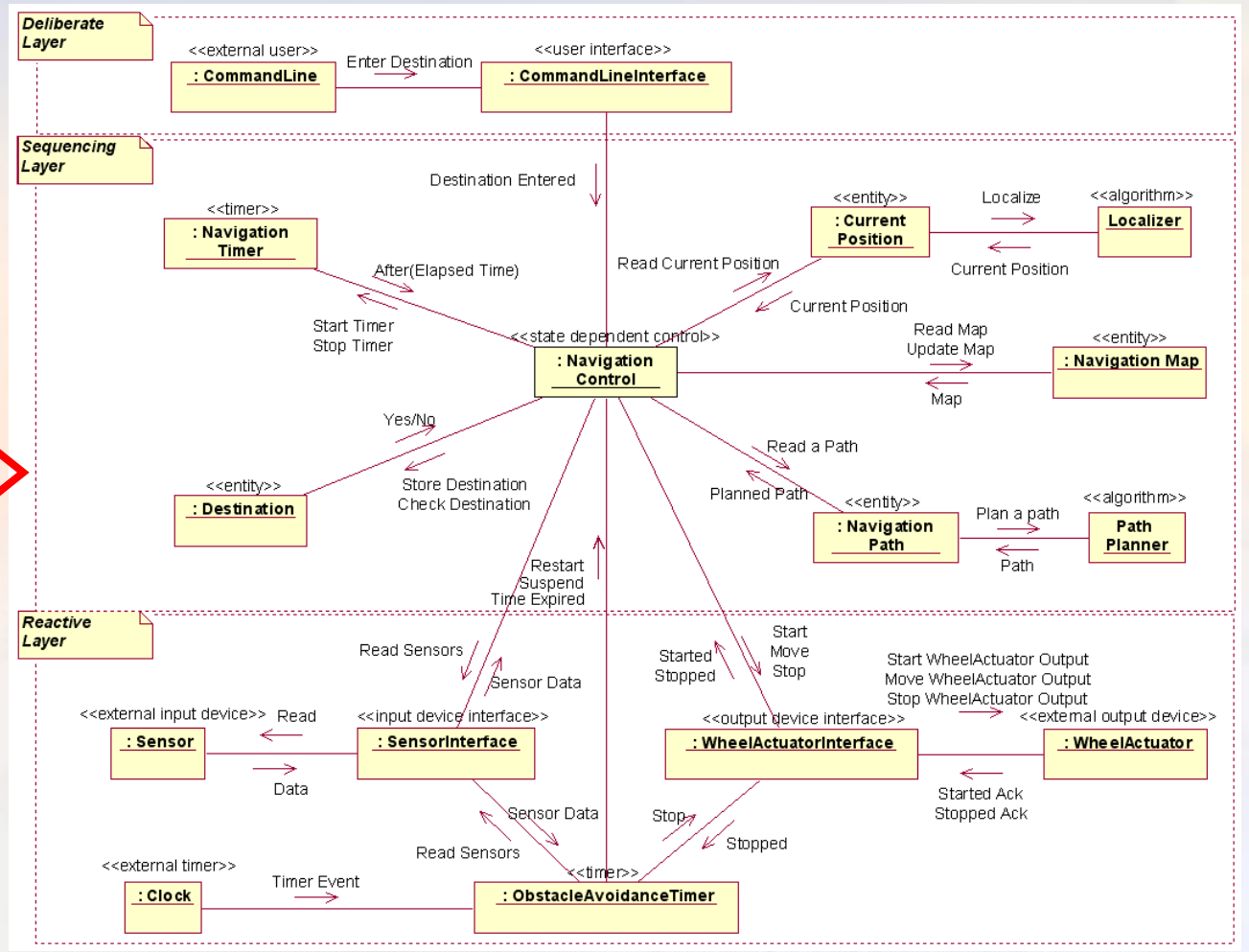


Collaboration diagram for Navigation use case

Merged

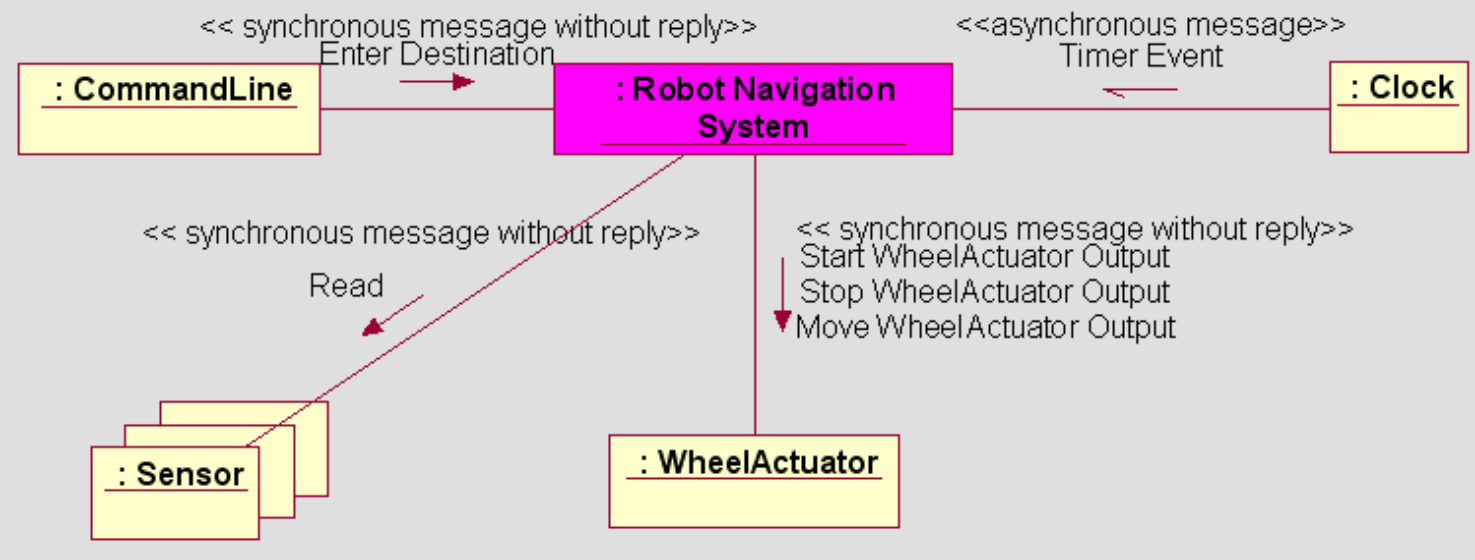


Collaboration diagram for Obstacle Avoidance use case



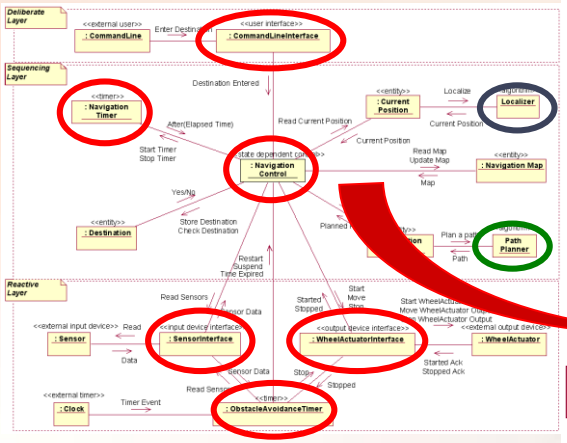
Consolidated collaboration diagram for Navigation System

Design Modeling (cont.)

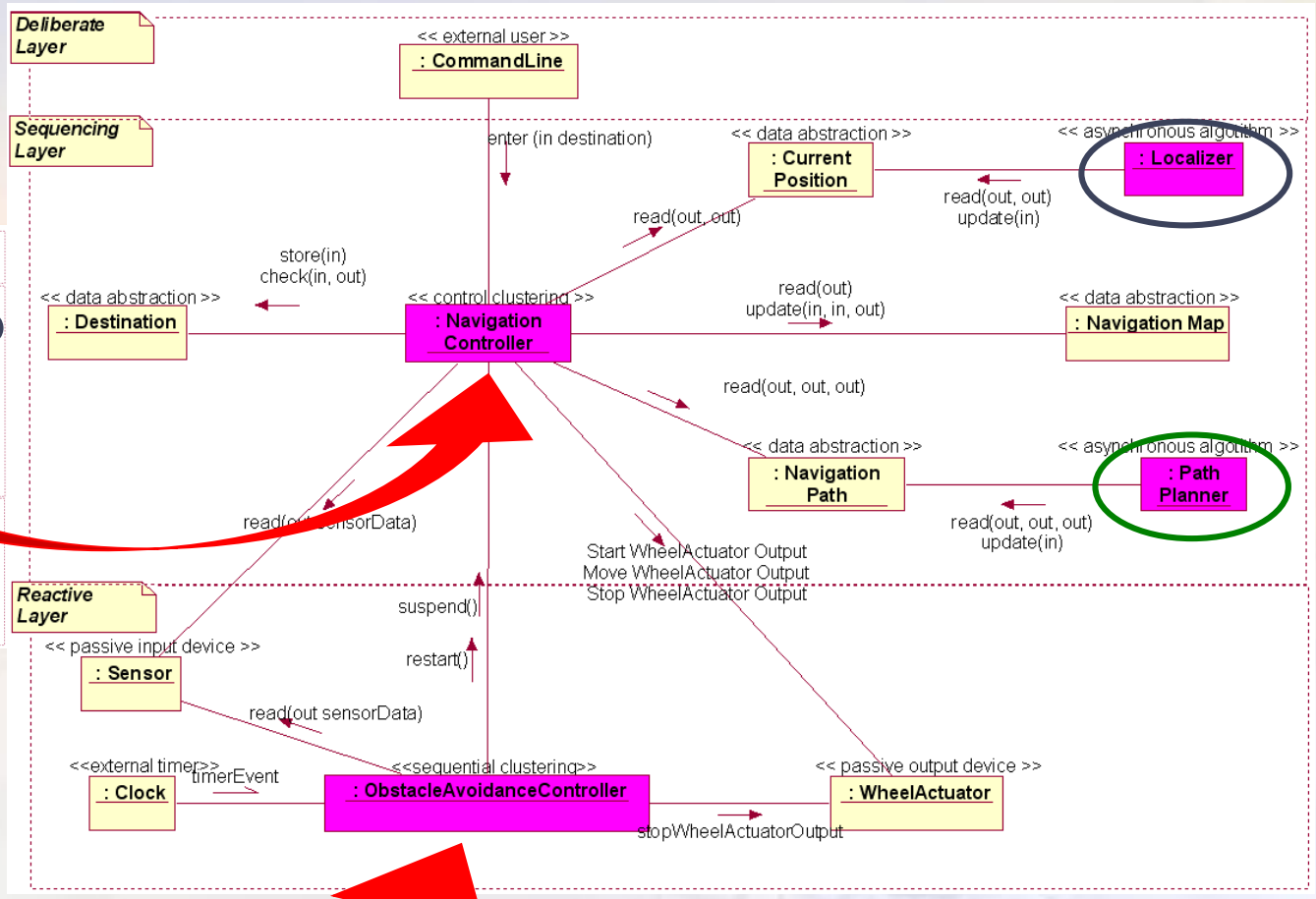


Distributed Software Architecture

Design Modeling - Task Structuring

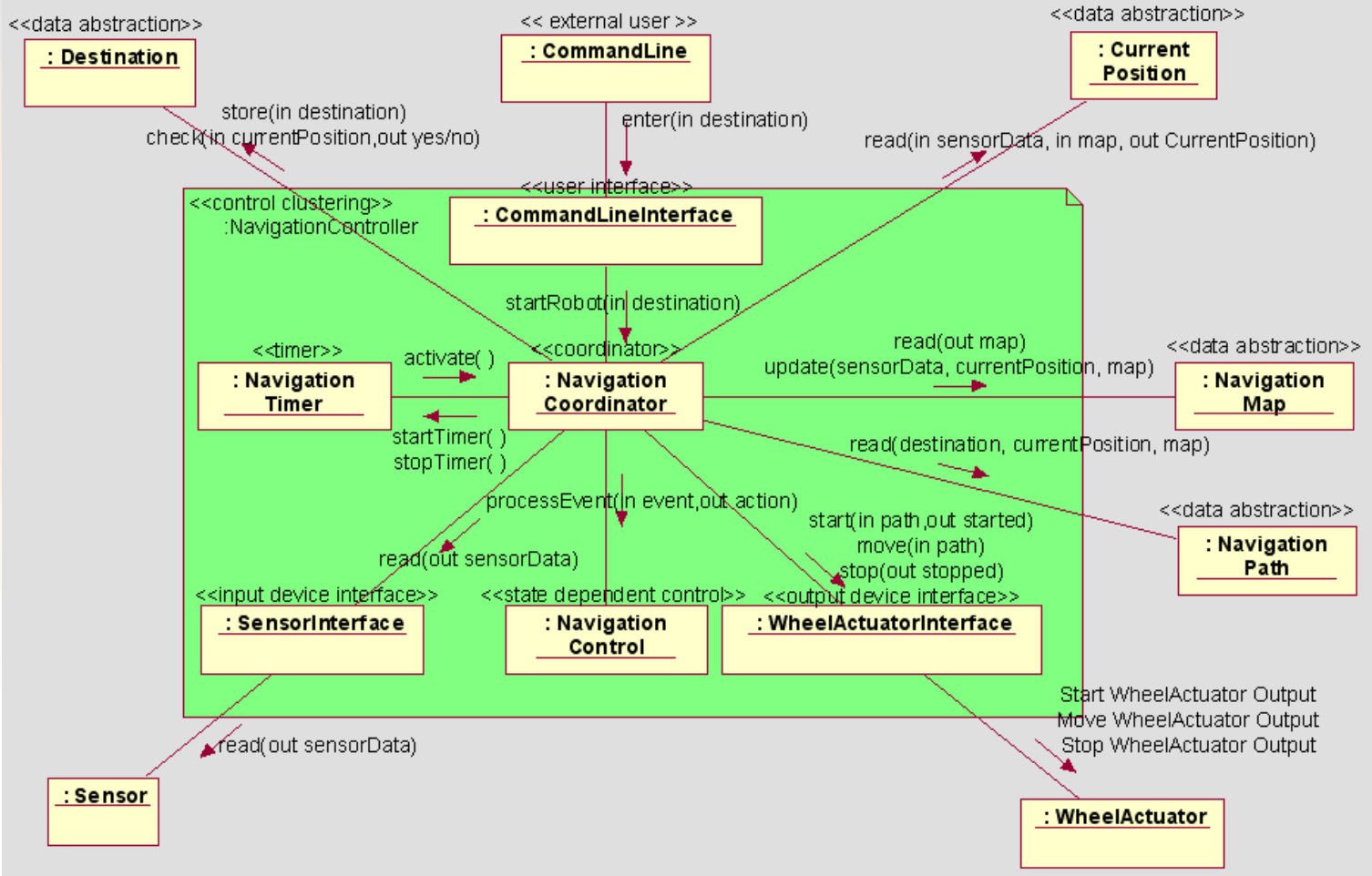


Consolidated collaboration diagram for Navigation System



Task Architecture for Navigation System

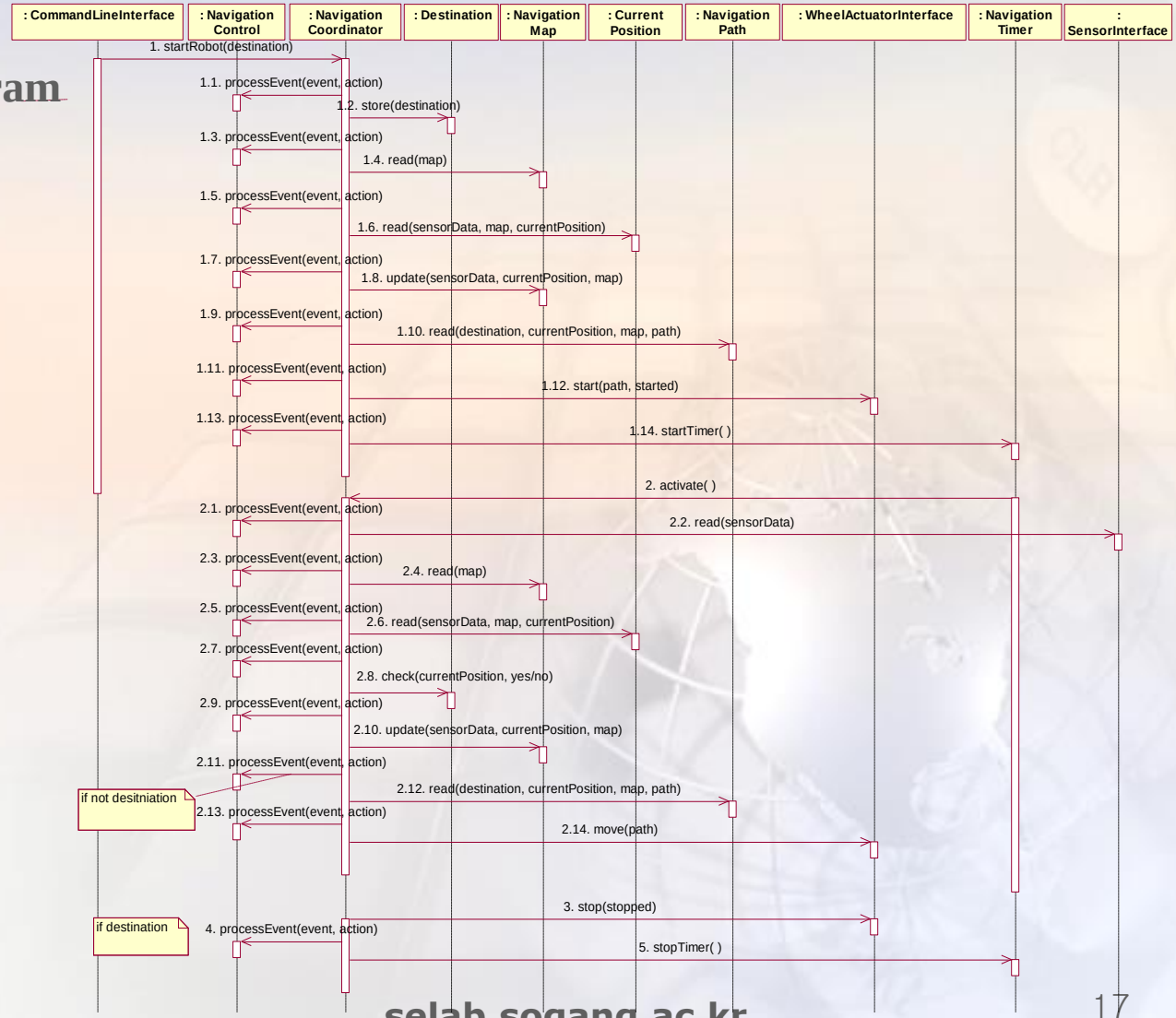
Design Modeling
- Detailed Software Design



Detailed software design for Navigation Controller

Design Modeling
- Detailed Software Design (cont.)

The task event diagram for Navigation Controller



Lessons Learned

- ***UML for service robot domain***
  - UML was very useful for analyzing, designing and modeling the service robot system
  - Different research groups and development teams can communicate among themselves and with others to develop and integrate specific components by UML.

- ***Importance of systematic process/method for service robot domain***
  - It is not possible to resolve the issues in integrating and developing the robots without systematic software development methods, particularly for service robots.
  - Applying the COMET/UML method led to developing an effective service robot architecture, implementing technical components based on the architecture, and integrating these components systematically.

Lessons Learned (cont.)

- ***Human communication***
  - Human communication to understand and develop what is desired of the service robot is likely to be more difficult than expected.
  - Several things can be done to improve the situation.
    - It is very important that all engineers and developers from different groups and teams interact directly.
    - A common medium or language such as UML is critical.
    - Guidelines about what notation to use, when to use it, and how to use the notation systematically are required.
    - One day or half-day technical workshop is needed when there is little domain knowledge and experience.

Lessons Learned (cont.)

- ***Customizing the COMET method for service robot domain***
  - The layered strategy of the prior PSR systems has been applied for designing and modeling the T-Rot and was helpful in arranging various hardware and software modules.
  - The task event diagrams were used for the event sequencing logic instead of pseudo code to improve understanding and readability in the detailed software design.

Lessons Learned (cont.)

- ***Necessity of multi-aspect integration method for service robot domain***
  - We focused on designing and modeling the robot's behavioral aspect.
  - Planning and learning abilities have to also be considered when designing and developing the intelligent service robots.
  - Task Manager has been in charge of these robotic abilities.
  - Different analysis and design methods are needed for the task manager.
    - To integrate these methods with COMET into a multi-aspect integration method is required for developing intelligent service robot software.

Software engineering for run-time

- We understood that SE for development time is not enough to handle run-time changes
- SW systems must become very flexible enough to handle these requirements => really "soft" software is needed

Why "Soft" Software is difficult ?

- Complexity
  - Software is limited by the skill of the human and not limited by the strength of the raw materials
- Invisibility
  - Hard to understand – Progress, changes and its impacts

Making software really soft – 1st Generation

- Just do it !
- Programming focused development
- Development of computer program not software

Making software really soft – 2nd Generation

- Reduction of complexity - Modularization
  - Decomposition
- Software visualization
  - Visualized software model
- Visualization of software development activities
  - Visible software process
  - Software measurement
- Structured Method, OO method etc

Making software really soft – 3rd Generation

- 소프트웨어를 아키텍쳐 기반으로 구성 (build by composition)
- 소프트웨어 구성요소들의 교체가 가능 (Interface)
- 소프트웨어 구성요소들의 재 구성이 가능 (Connector)
- 소프트웨어 구성요소들 내부의 변화가 외부에 영향을 미치지 않음(Component)
- CBD, Software Product line

Making software really soft – 4th Generation

- Run Time Softness
- 소프트웨어 스스로 내/외부 변화를 인식
- 소프트웨어 스스로 변화에 대한 대처 방안을 결정
- 소프트웨어 스스로 구성요소들의 교체 및 재구성 가능
- 소프트웨어 스스로 자신의 행위를 검증 가능
- Self-Managed Software

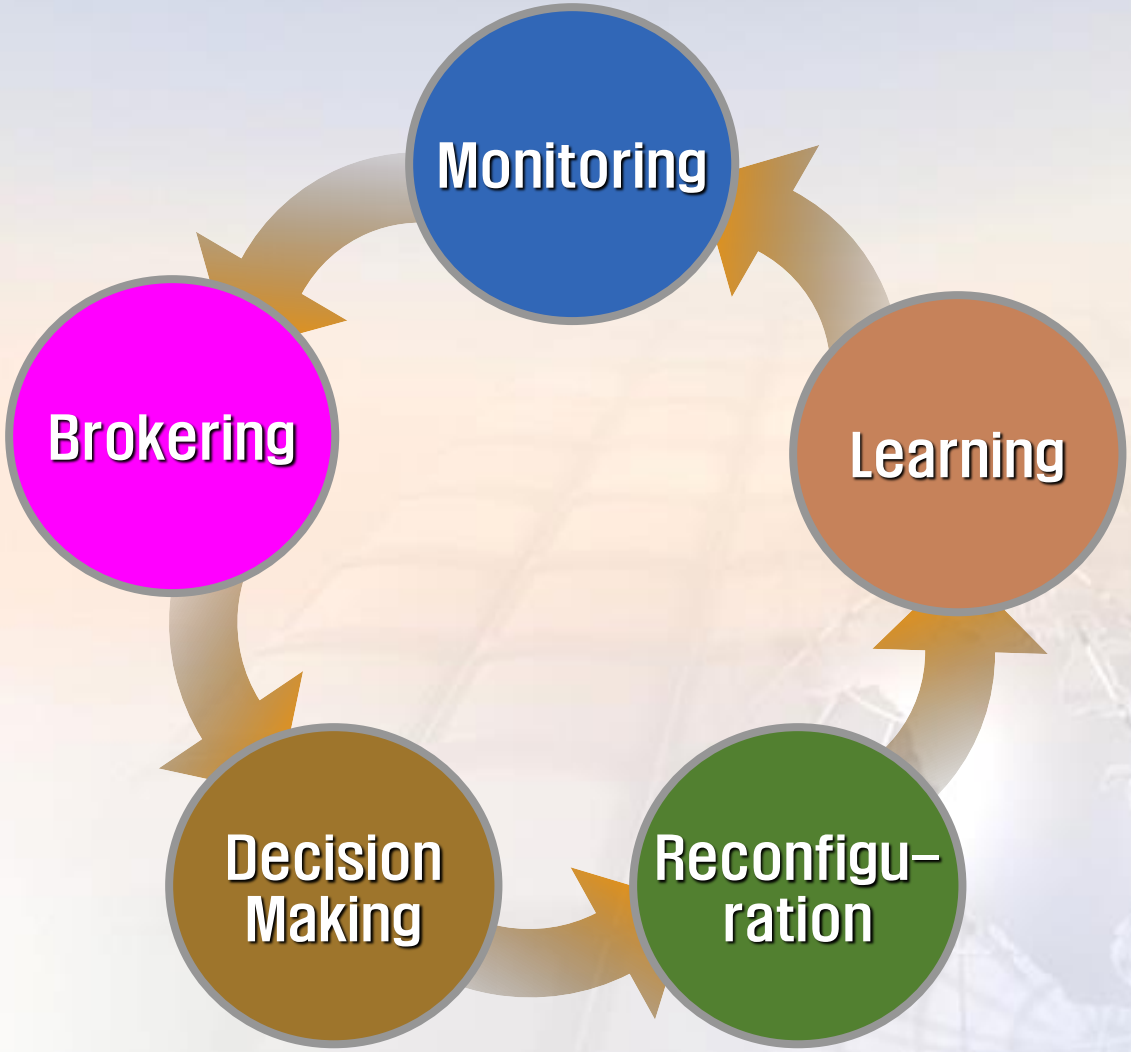
Self-Managed Software

- *"Self-managed software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible." (WOSS 2004)*

Run-Time Software Engineering Activities

- Run-Time Requirements Analysis
  - Observe the running system and abstract observed behavior
  - Analyze new environments or situation based on original requirements
- Run-Time Design
  - Determine the cause of constraint violation and choose a repair strategy in terms of SW architecture
- Run-Time re-implementation
  - Adapt new SW components or change the structure of SW without violating run-time environment
- Run-Time Testing
  - Continuously check design constraints via explicit run-time models

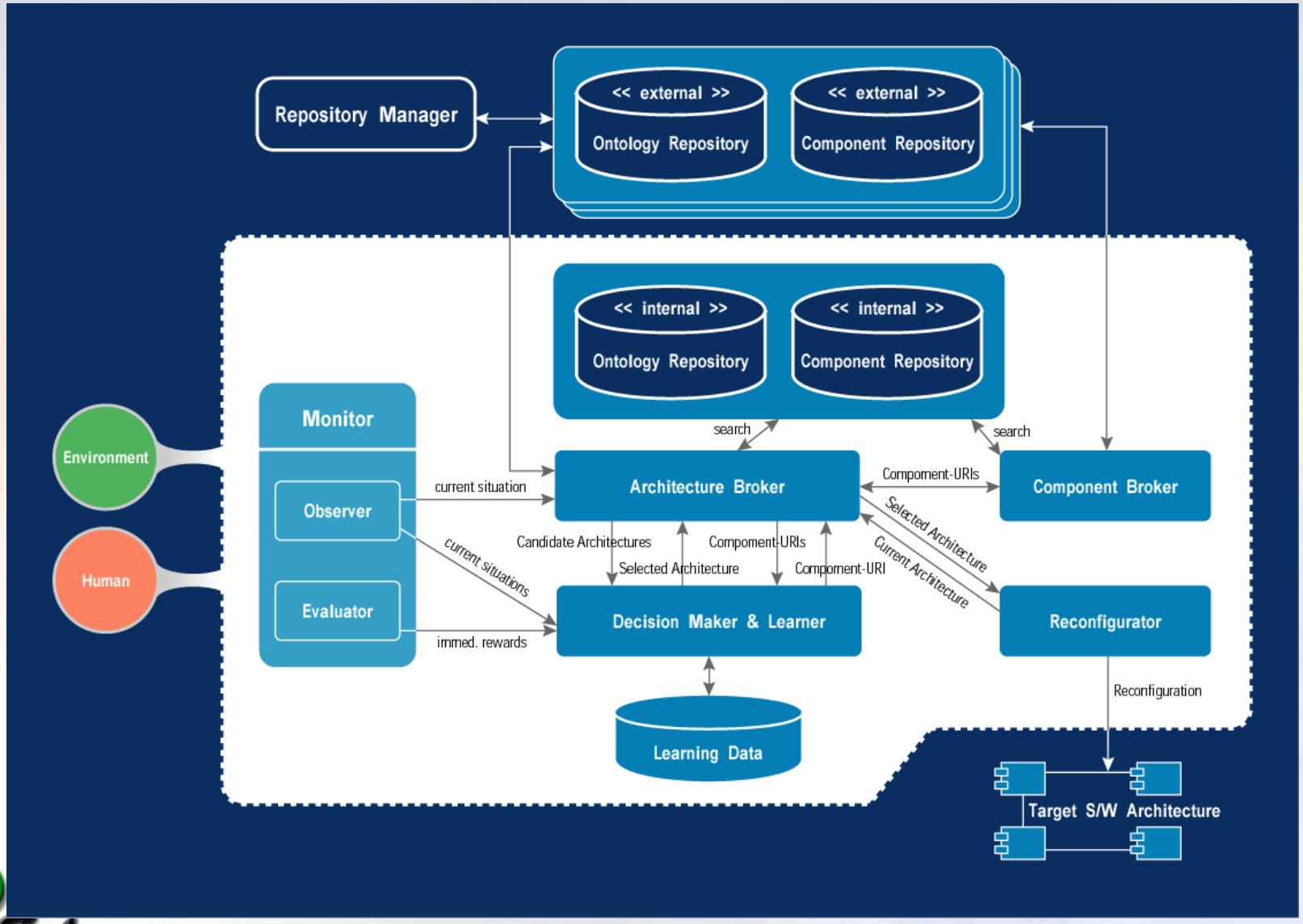
Our Approach



SHAGE Framework

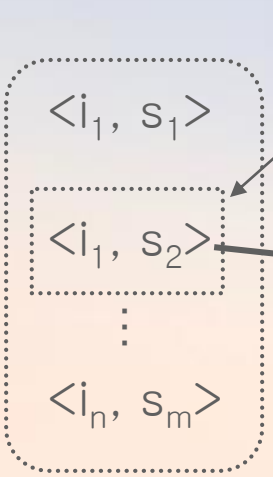
- SHAGE(Self-Healing, Adaptive, and Growing SoftwarE) Framework integrates following technologies
  - Monitoring
  - Brokering: Ontology(authoring relations between environmental information and architectural information)
  - Decision & Learning: Case-Based Decision Theory
  - Reconfiguration: Slot-based architectural style

SHAGE Overall Architecture



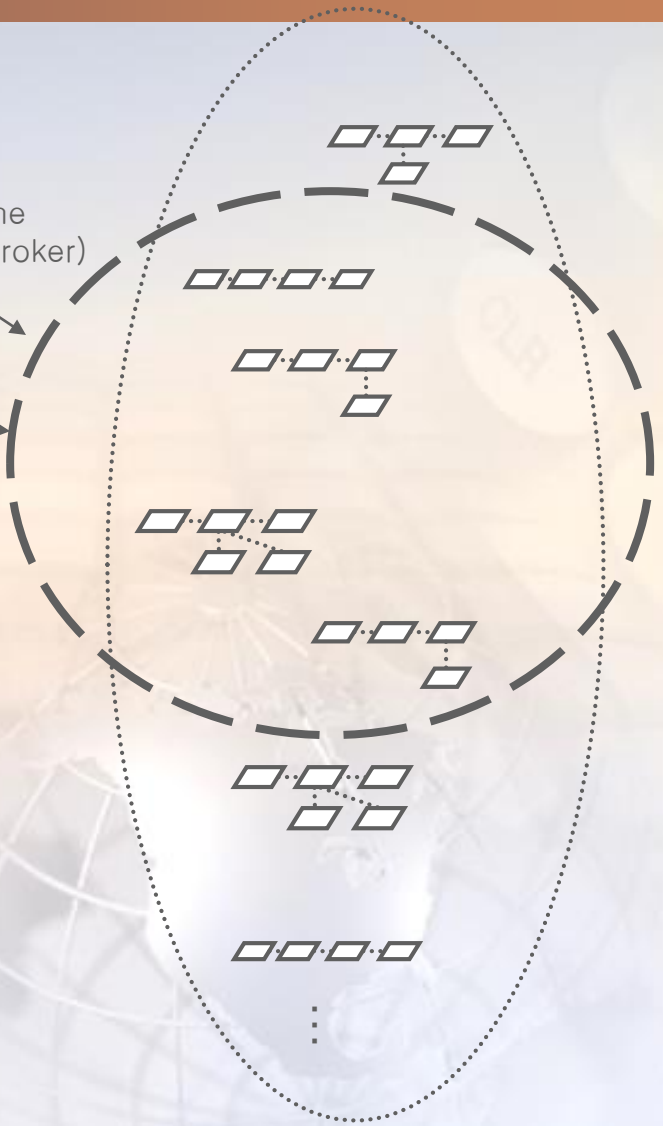
Architecture/Component Broker

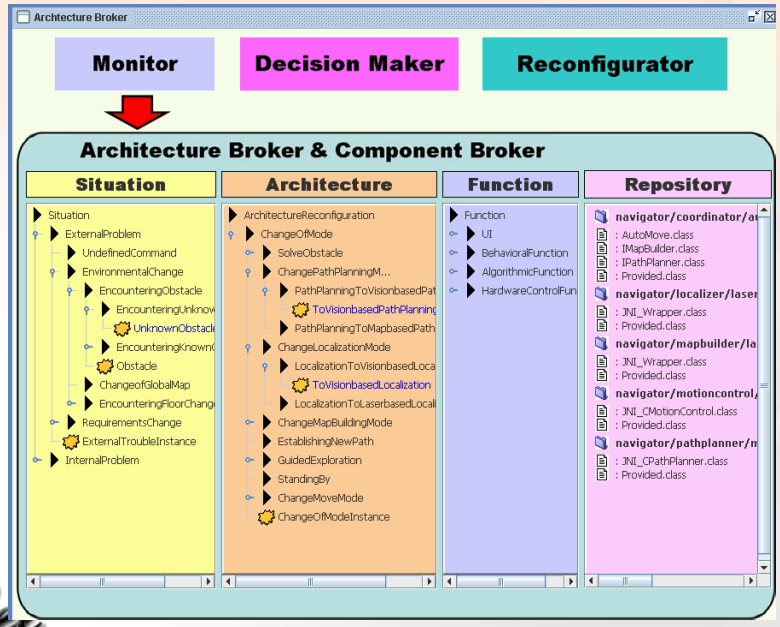
A set of Inclination-Situation Pairs



Current Situation
(given by the
Task Manager)

Candidate Set
(reduced by the
Architecture Broker)

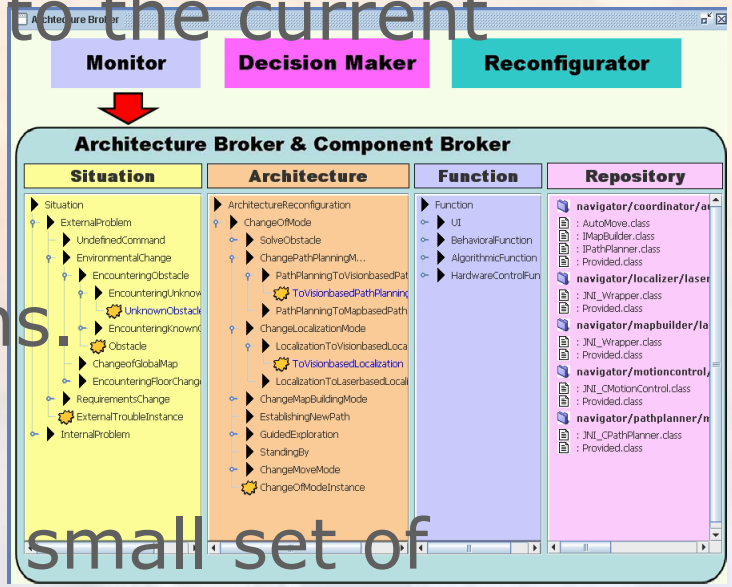




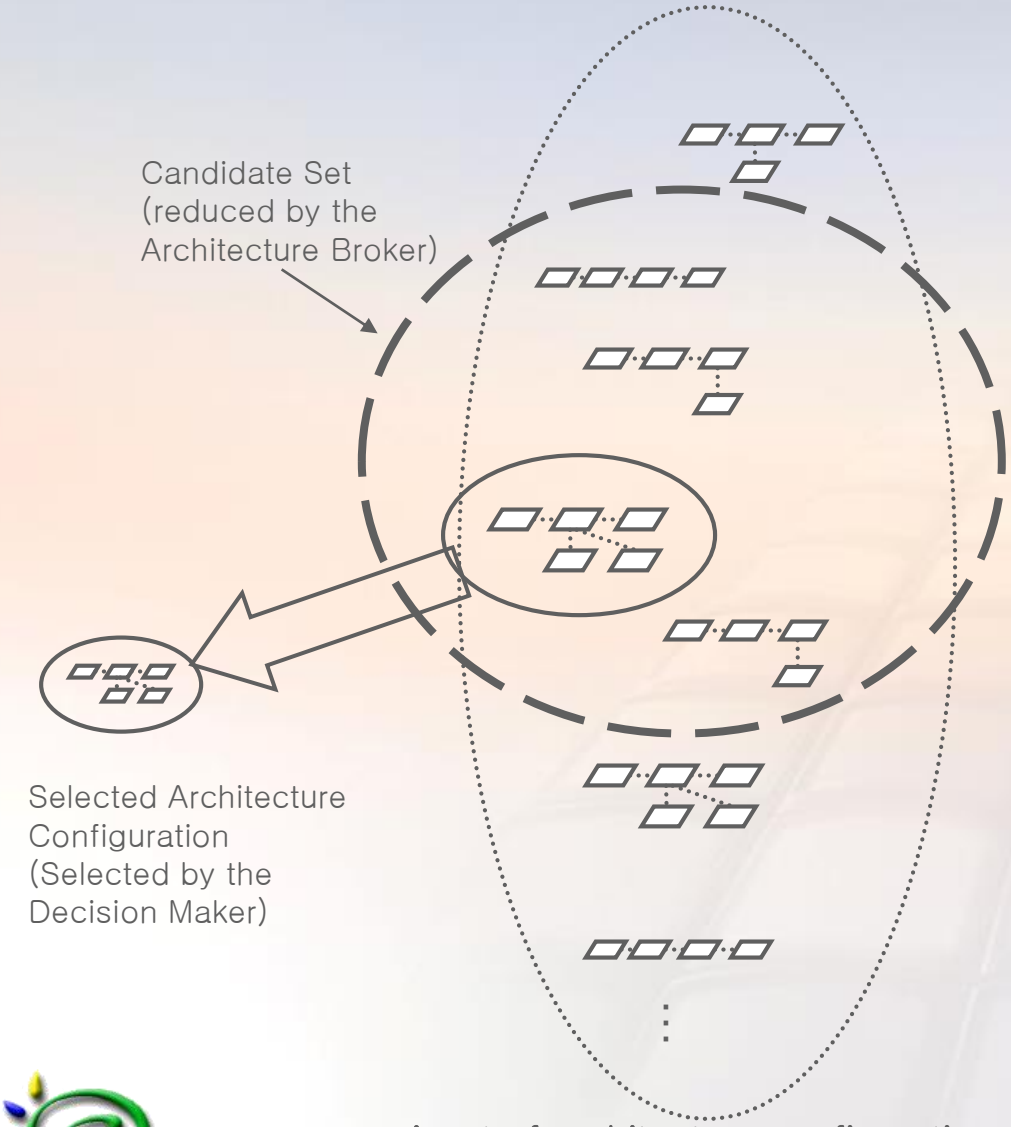
A set of architecture configurations
(Abstract Level Architectures)

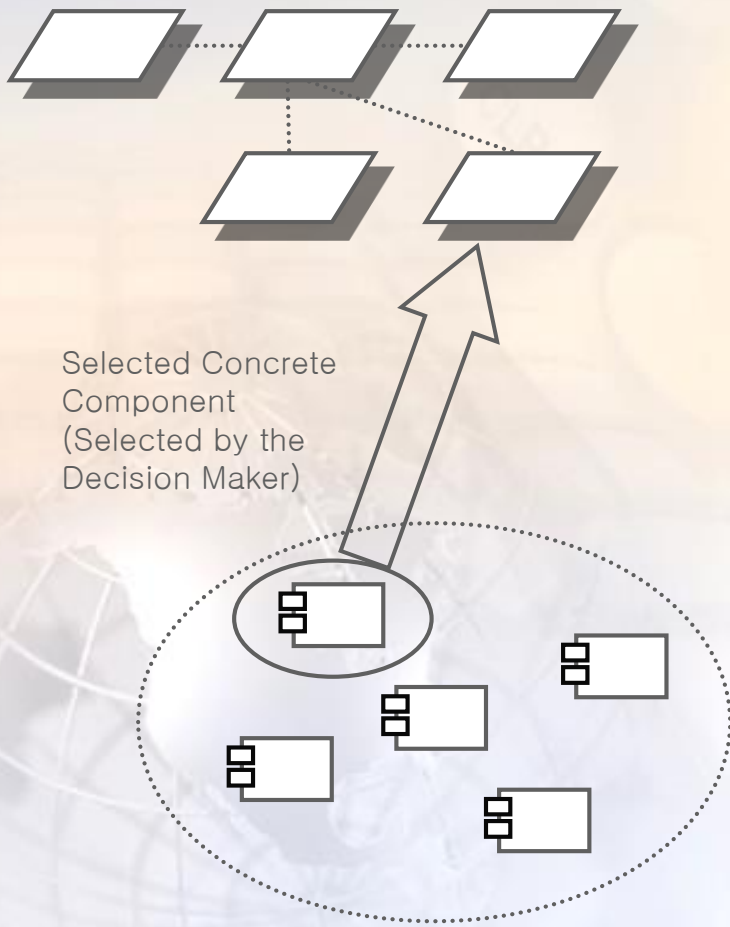
Architecture/Component Broker

- Role
  - Searching abstract-level architecture configurations related to the current situation.
- Technology
  - Ontological descriptions.
- Current Status
  - It can only search in a small set of configurations related to the navigation subsystem.
  - Rule-based search; it cannot relax rules.



Decision Maker & Learner

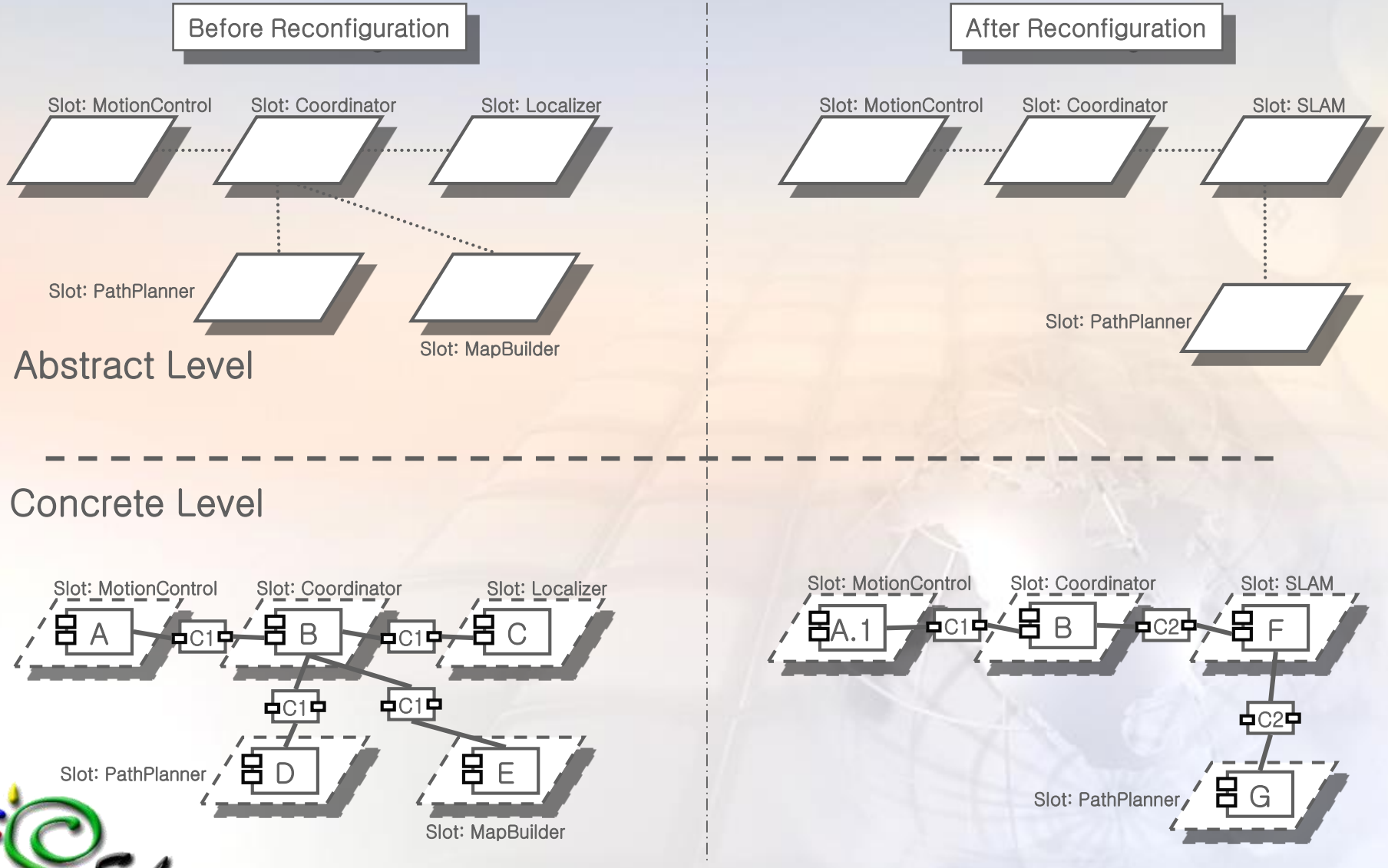




Decision Maker & Learner

- Role
  - Select exactly one configuration and one component for each slot from the candidate set retrieved by the architecture broker.
- Technology
  - Case-Based Decision Theory
- Current Status
  - It only carries out in limited scope.
  - Limited search space: only in the navigation subsystem.
  - Limited learning time: few scenarios.

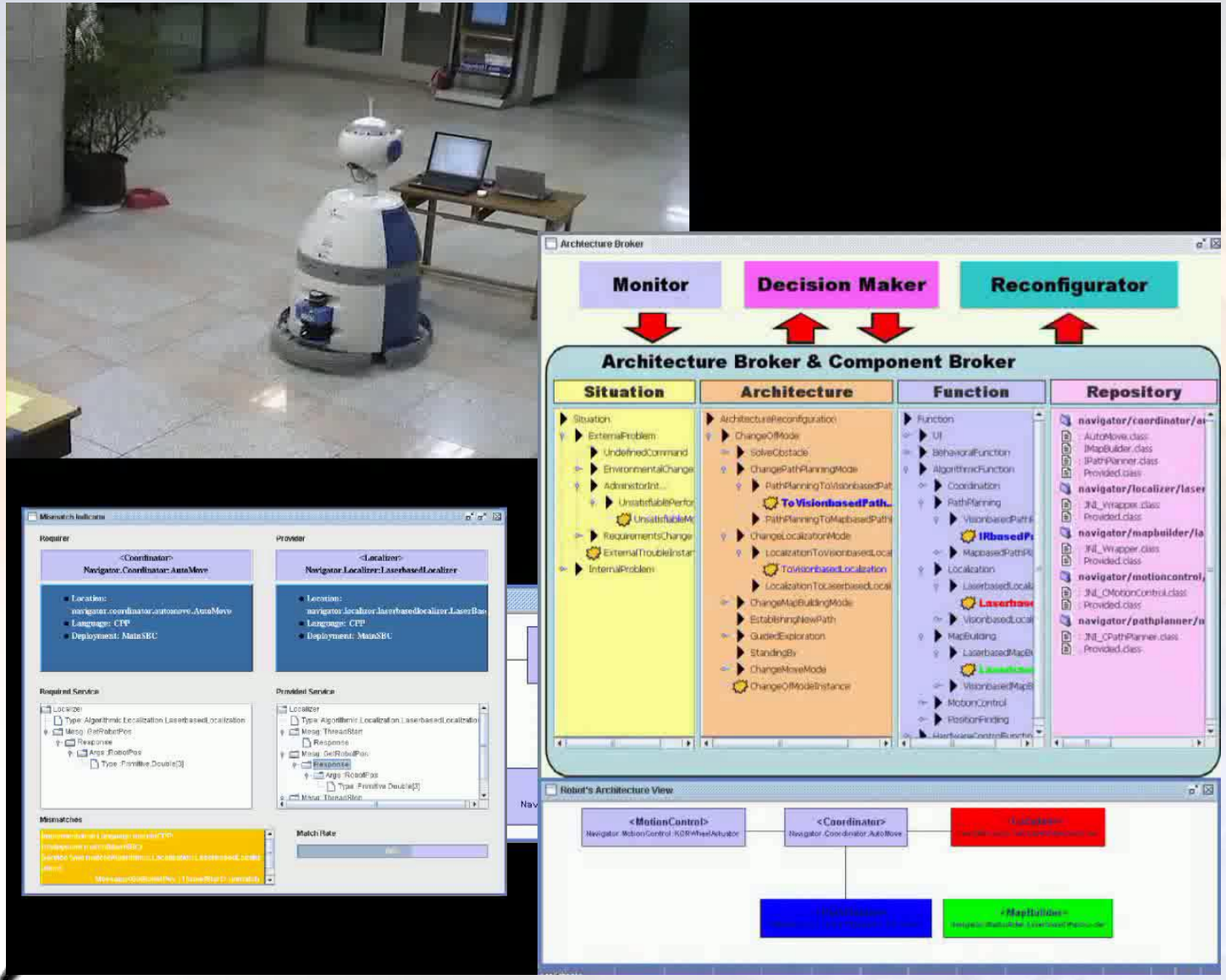
Reconfigurator



Reconfigurator

- Role
  - Reconfiguring the current software architecture dynamically.
- Technology
  - Slot-based two-level software architectural style.
- Current Status
  - It has reconfigured only the navigation subsystem.
    - All configurations for the subsystem were verified in the demonstration.
  - It can manage components distributed in SBCs(Single Board Computers) by RMI.
  - It supports components implemented in Java and C++(through JNI).

Demonstration



Research Issues

- Internal monitoring
- Ontology construction
- Learning speed
- Run-time measurement and validation
- Componentization
- Domain Knowledge

Conclusions

- Software in Robot is getting more important
- Software Engineering need to be applied not only for development but run-time softness
- SHAGE Framework has been developed to provide 'self-managing capabilities' to robot software.
- The framework integrated ontology, decision theory, and dynamic architecture and comprises
  - Monitor
  - Architecture/Component Broker
  - Decision Maker & Learner
  - Reconfigurator