Software Architecture Patterns for Mobile Robots and Games

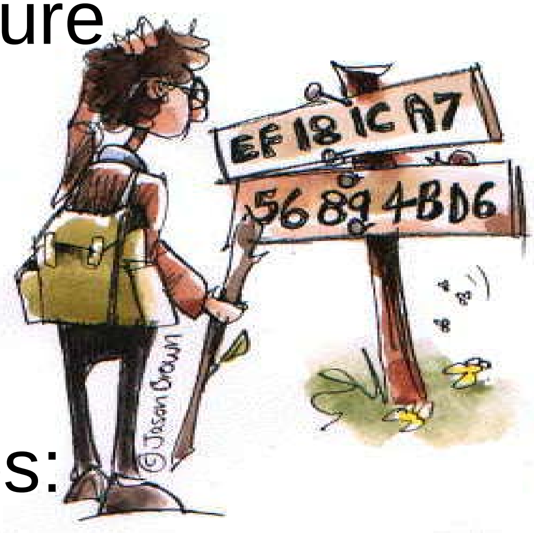
NTNU
Innovation and Creativity

Goal for the Lecture

- The students should after this lecture know:
  - Game architecture basics
  - Some general architecture patterns
  - Some reference architectures for mobile robot control
  - Some reference architectures for games

Agenda

- Basics in creating a game architecture
- General architecture patterns:
  - Pipe and Filter, Layered, Blackboard, and Task control
- Mobile robot/Game architecture patterns and reference architectures:
  - AI approach, Subsumption, Control loop, Elfes, CODGER, and NASREM.



Creating a Game Architecture

Based on:

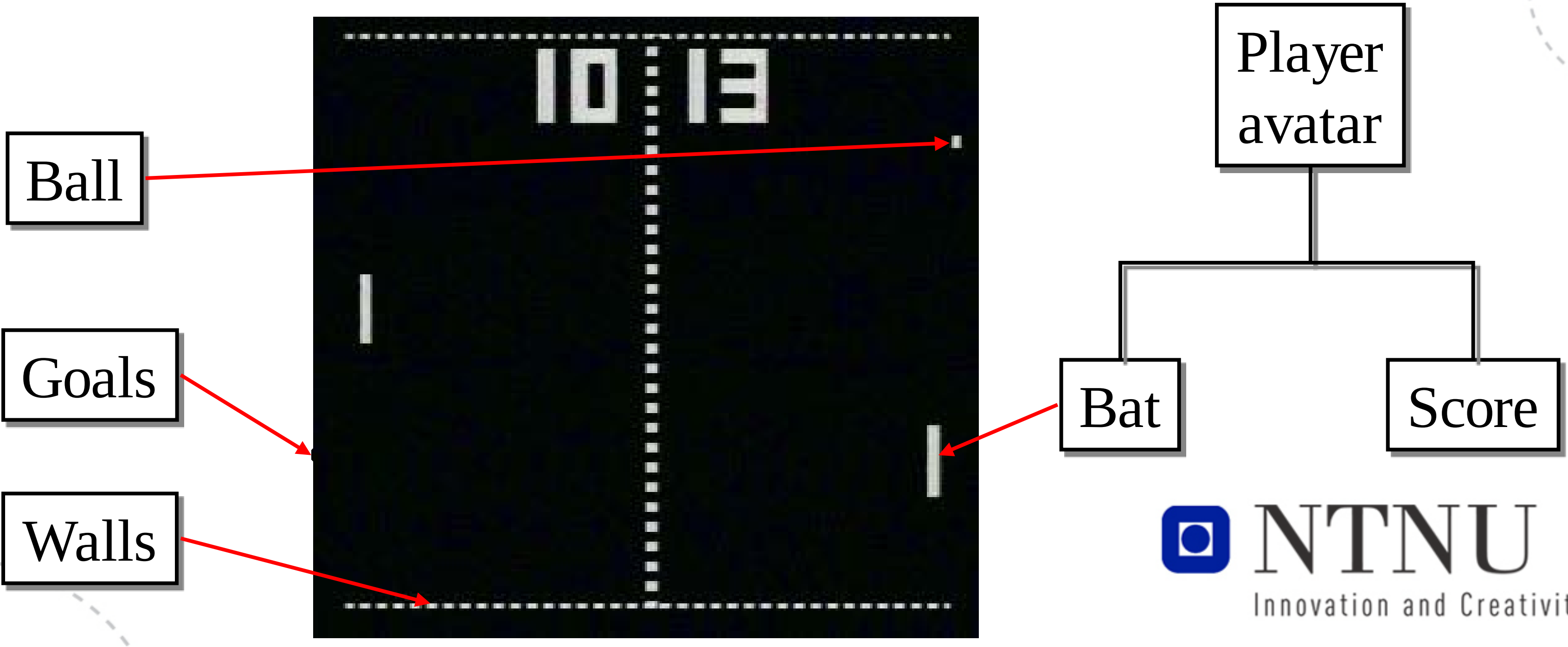
Andrew Rollings & Dave Morris

Game Architecture and Design - A New Edition, New Riders Publishing 2004

NTNU
Innovation and Creativity

Creating a game architecture
1. Find Tokens

- Tokens are objects related to the gameplay: Playable objects, non-playable character (NPC), game environment objects, environment, score, etc.
- Tokens in Pong:



Ball

Goals

Walls

Player avatar

Bat

Score

NTNU

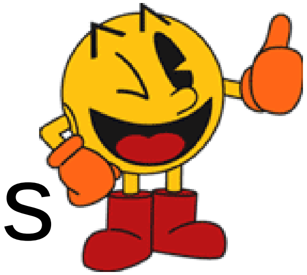
Innovation and Creativity

Creating a game architecture: 2.Analyse interaction and events

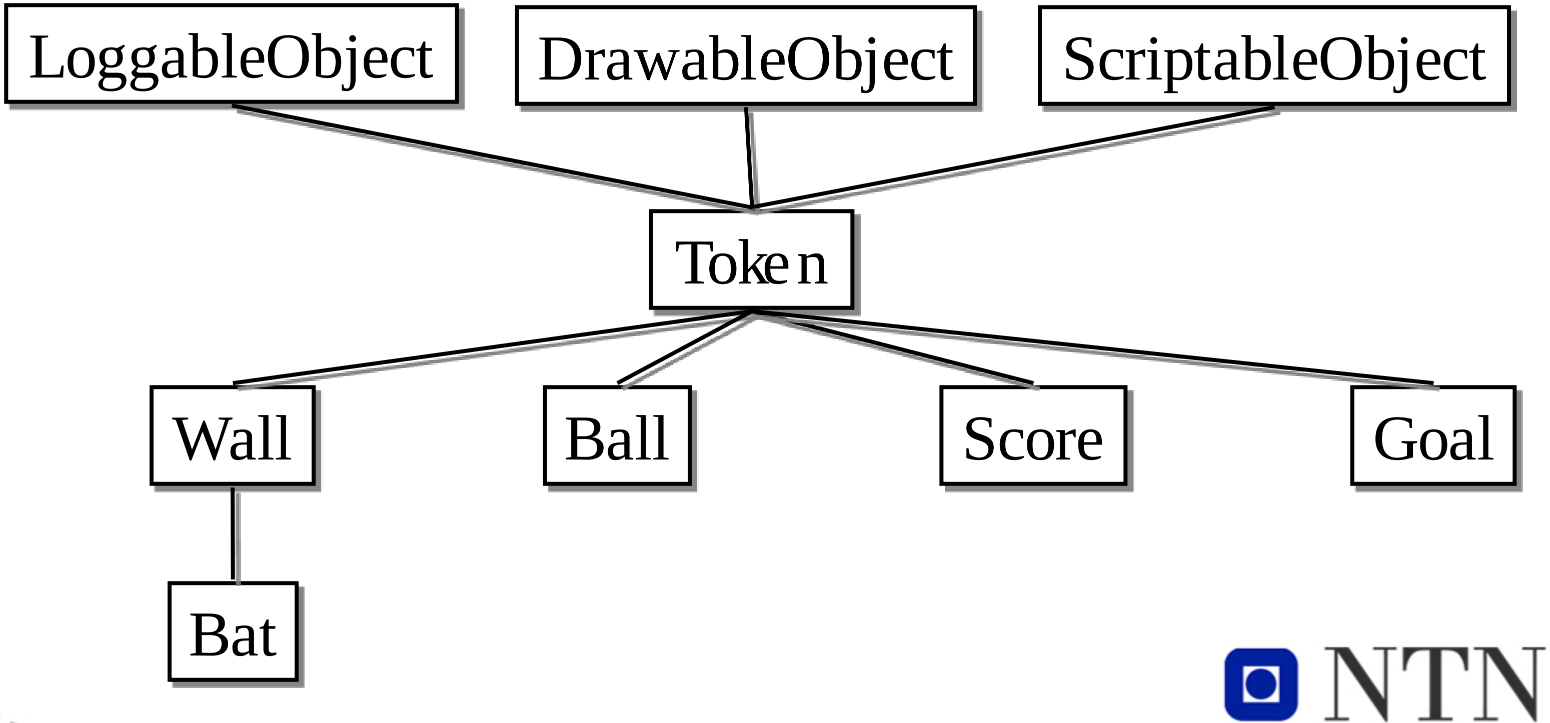
	Bat	Ball	Wall	Goal	Score
Bat	X				
Ball	Collision event: Deflection	X			
Wall	Collision event: Stop	Collision event: Deflection	X		
Goal	X	Collision event: Trigger Goal event	X	X	
Score	X	X	X	Goal event: Goal score	X

- Create Token Interaction Matrix
- Trace events in the game (event diagram)
- Create finite-state machine diagrams for NPCs, game world, etc.
- Starting point for *process view* as well as *logical view.*

Creating a game architecture: 3.Create logical view using tokens



- Tokens can be used to sketch game logical view based on token interaction matrix (according to behaviour).



NTNU
Innovation and Creativity

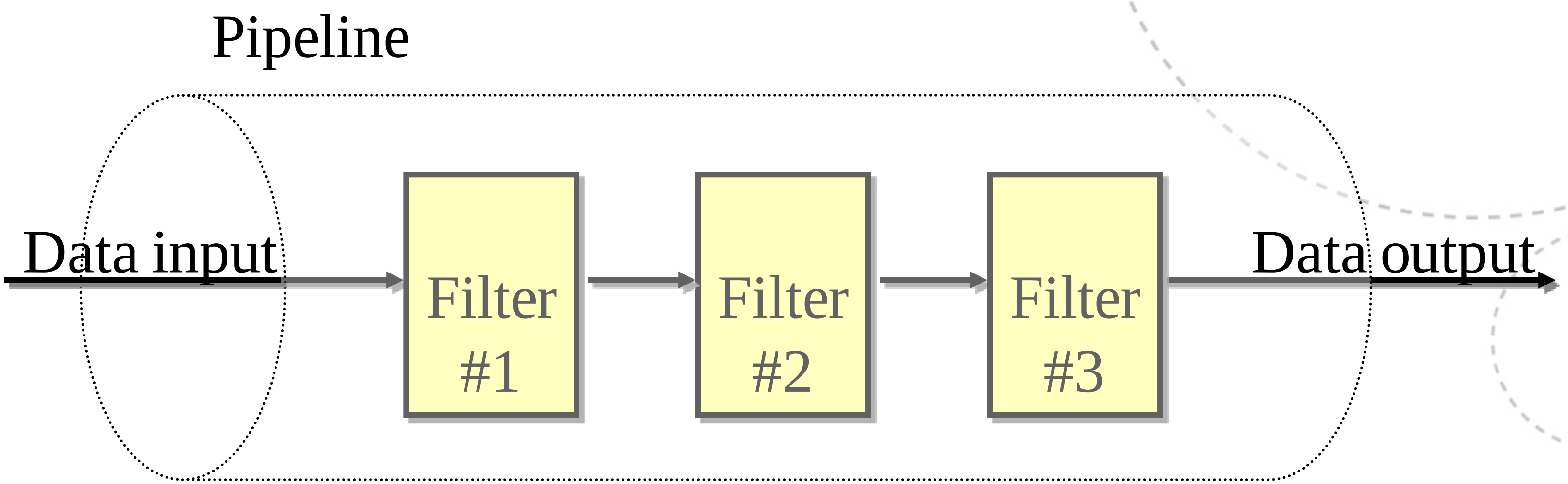
Some architectural patterns and reference architectures

Focus on

Mobile Robot controller &

Games

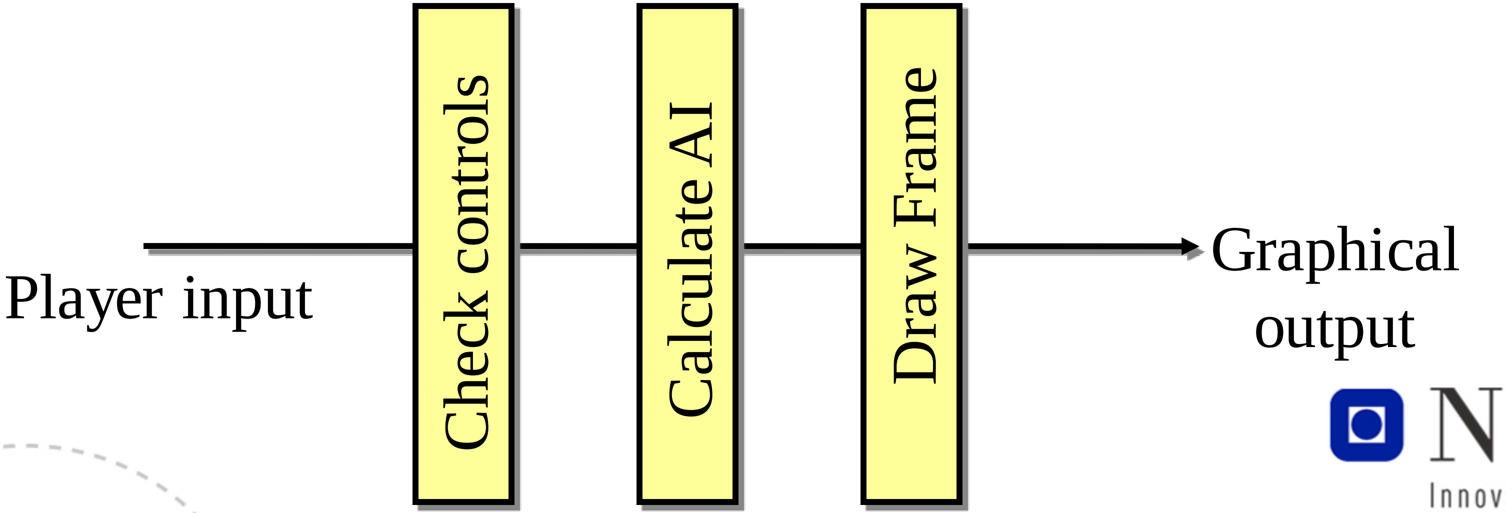
1. Pipe and Filter Architectural Pattern



- Used to manipulate a data stream
- Typical usage:
  - 3D graphics engine
  - Data conversion
  - Compiler
  - Workflow systems

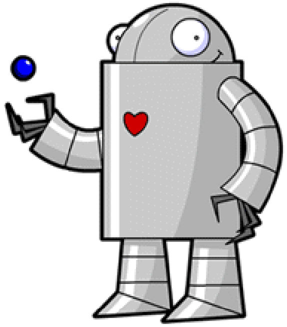
Reference architecture: Game loop pipe and filter

- Characteristics:
  - Simple control flow
  - Easy concept
  - Unstable framerate (depending on Adaptive Intelligence)
  - Can refine (decompose) existing filters or add new ones

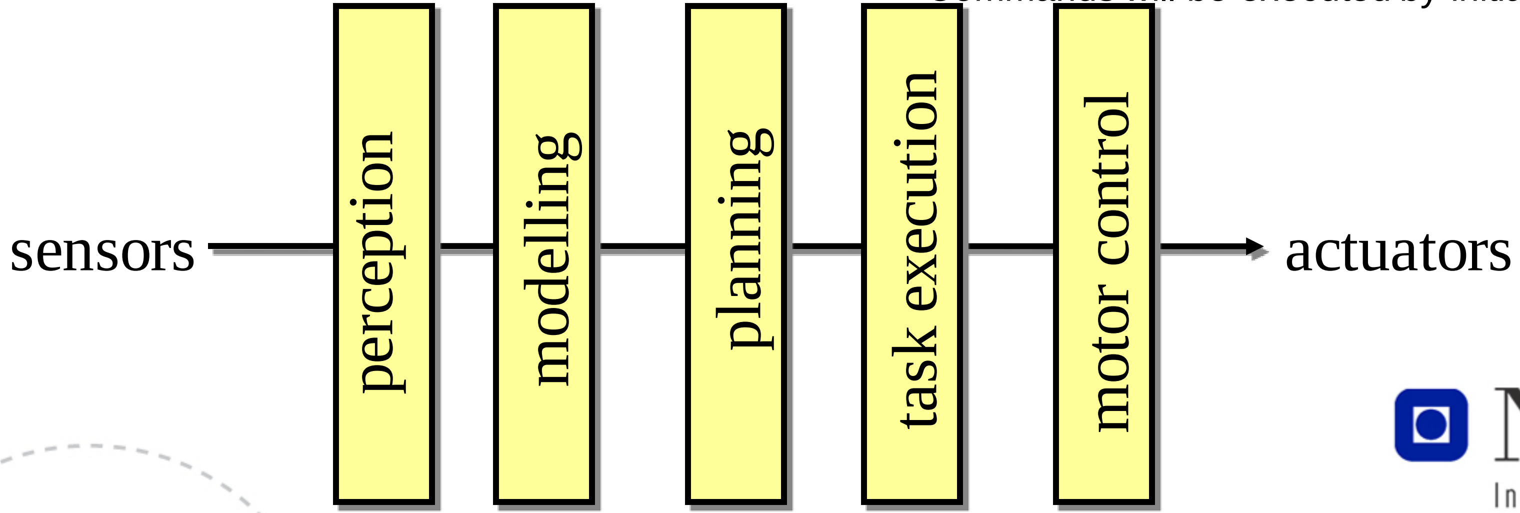


NTNU
Innovation and Creativity

Reference Architecture: Adaptive Intelligent Approach and Pipe and Filter



- **Perception**
  - Handling/management of sensors
- **World modelling**
  - Converts sensor input into a description of where the robot is in the surroundings
- **Planning**
  - Work out how it will achieve its goals given the current world state
- **Task execution**
  - Breaks down the plan into detailed motion commands
- **Motor control**
  - Commands will be executed by initiating motors



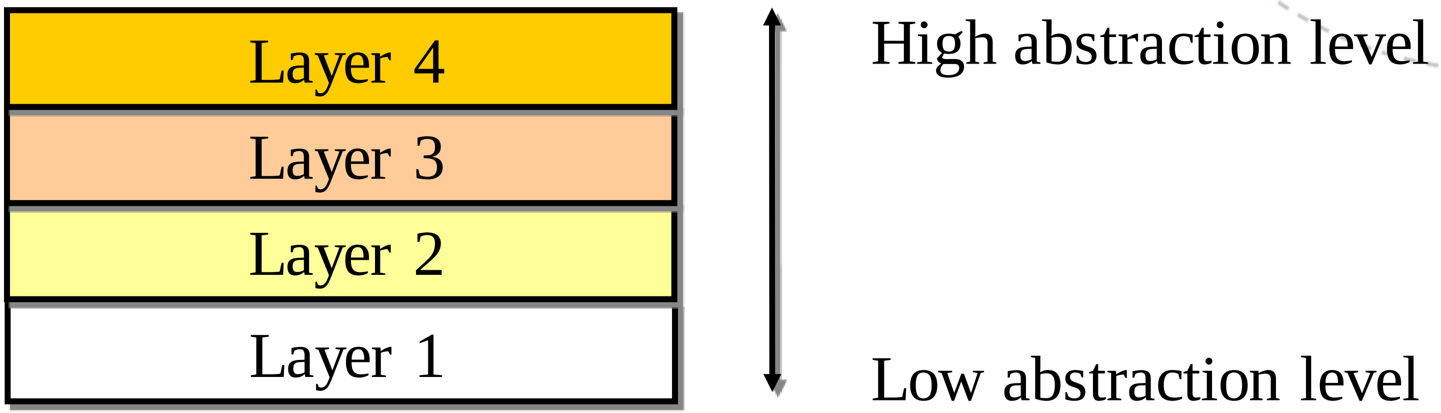
NTNU

Innovation and Creativity

Pipe and Filter Consequence

- Benefits
  - Flexibility (filter exchange)
  - Reuse of filters
  - Rapid prototyping of pipelines
  - Efficiency by parallel processing
- Liabilities
  - Inefficient if state sharing required, or data structure complex
  - Error handling constrained to reporting

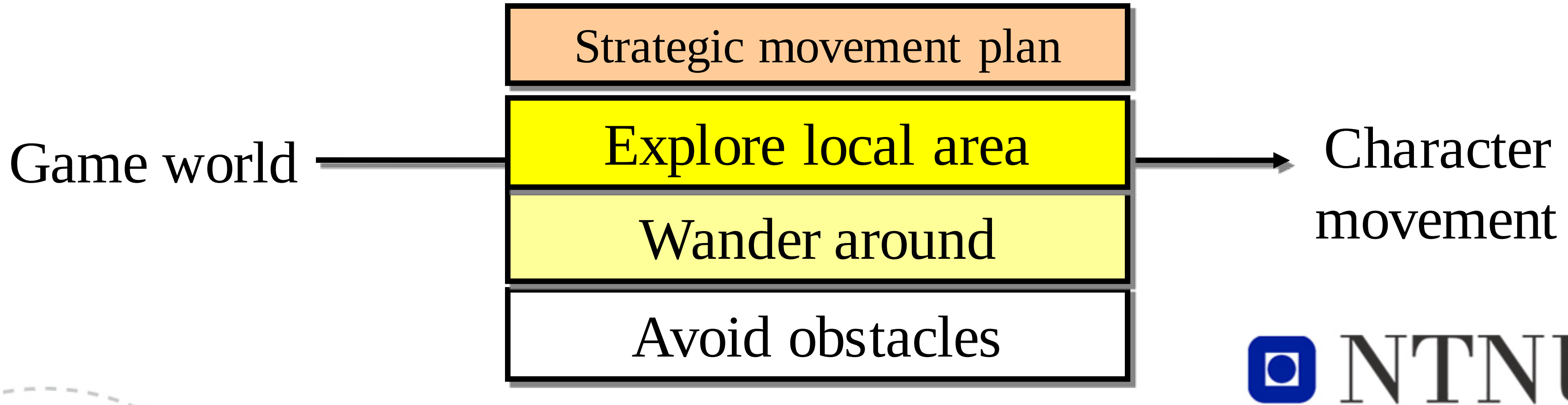
2. Layered Architecture Pattern



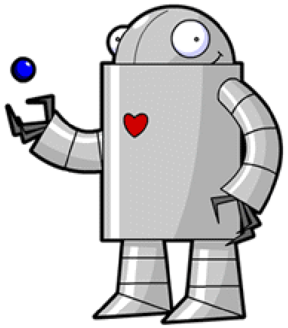
This architecture pattern divide the different parts of the system into different abstraction levels.

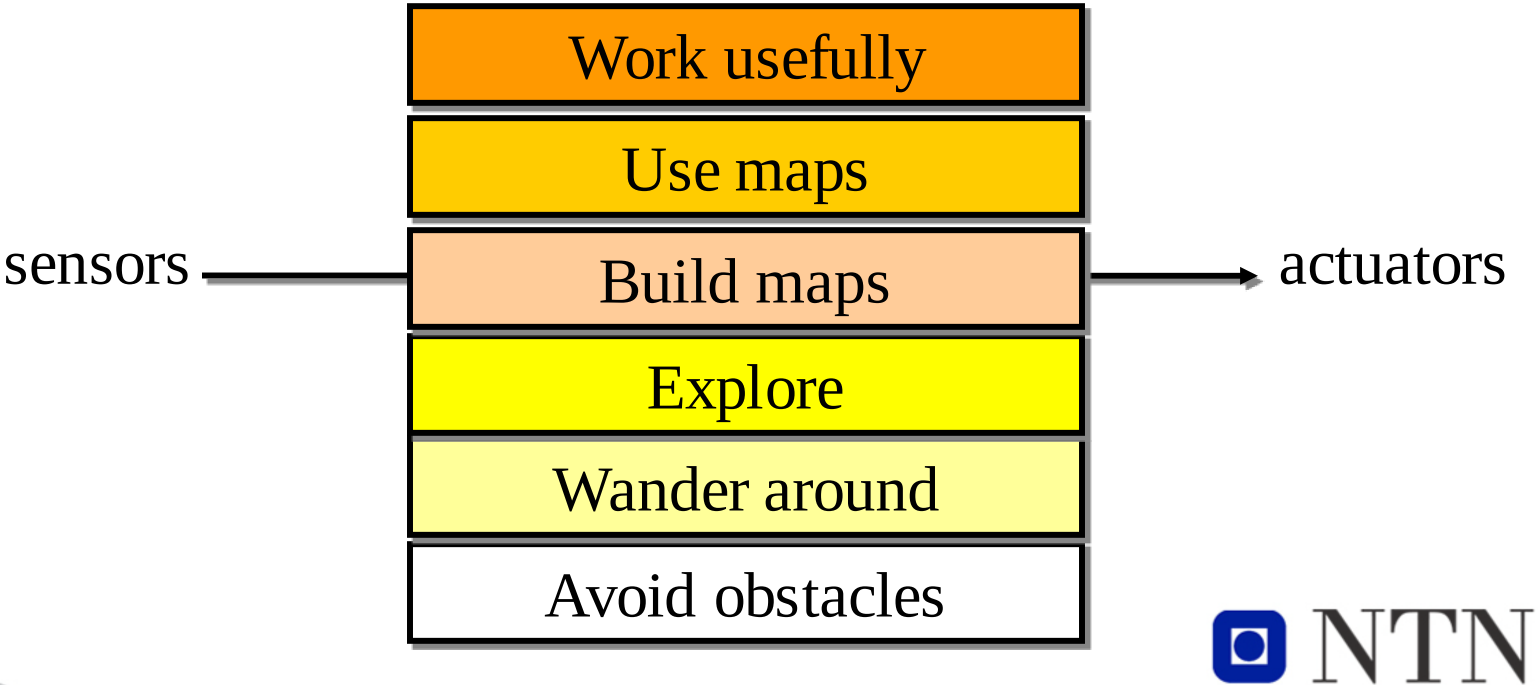
Proposed layered architecture Approach for non-playable character (NPC) movement

- Characteristics:
  - Hierarchical Adaptive Intelligence
  - Simplify Adaptive intelligence by decomposing into several layers
  - Same pattern can be used for fighting and other Adaptive Intelligence behaviour



Subsumption Reference Architecture (layered)





sensors

actuators

Work usefully

Use maps

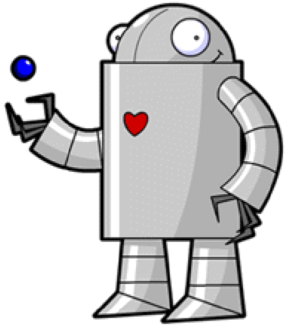
Build maps

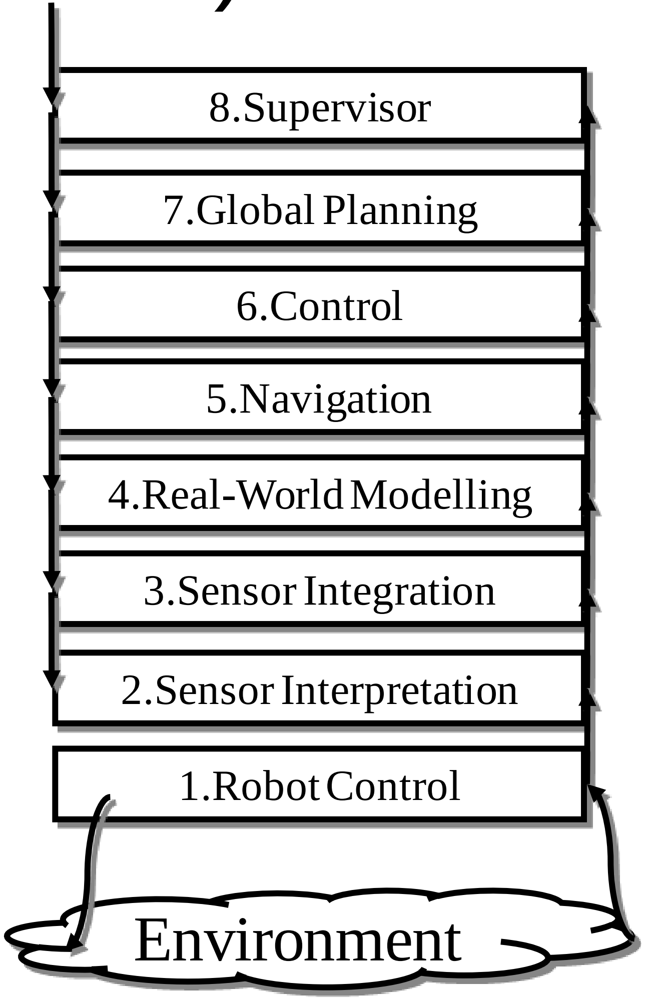
Explore

Wander around

Avoid obstacles

Elfes Reference Architecture (layered)





8: UI and supervision functions

7: High level scheduling and planning

6: Low level scheduling and planning

5: Managing navigation

4: Maintaining world model

3: Combined analysis of sensors

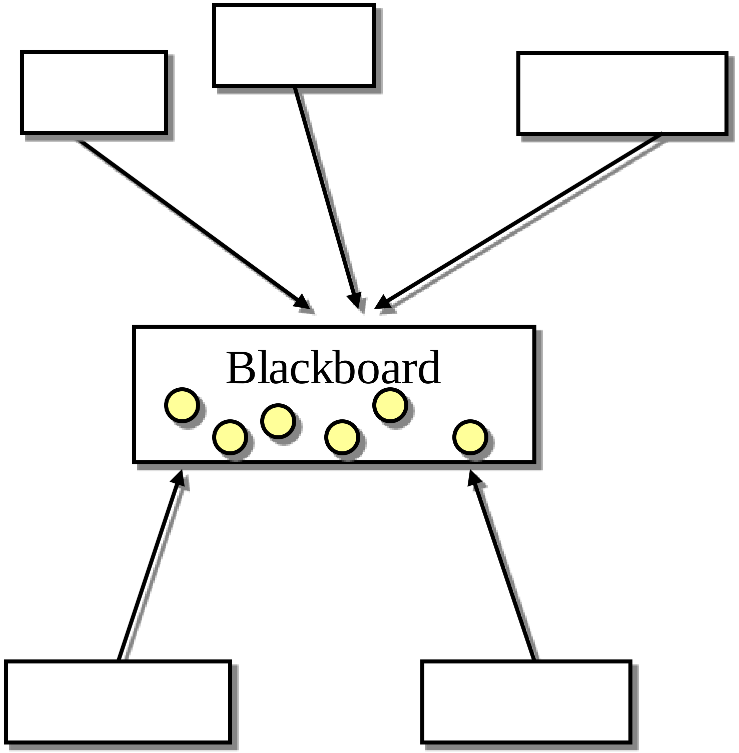
2: Individual sensor analysis

1: Provide robot control routines (motors etc.)

Layered architecture consequences

- Benefits
  - Reuse of layers
  - Support for standardization
  - Code changes are isolated to indvidiual layers
- Liabilities
  - Problem can occurr when the behavior of a layer changes
  - Lower efficiency
  - Difficult to establish correct granularity of layers

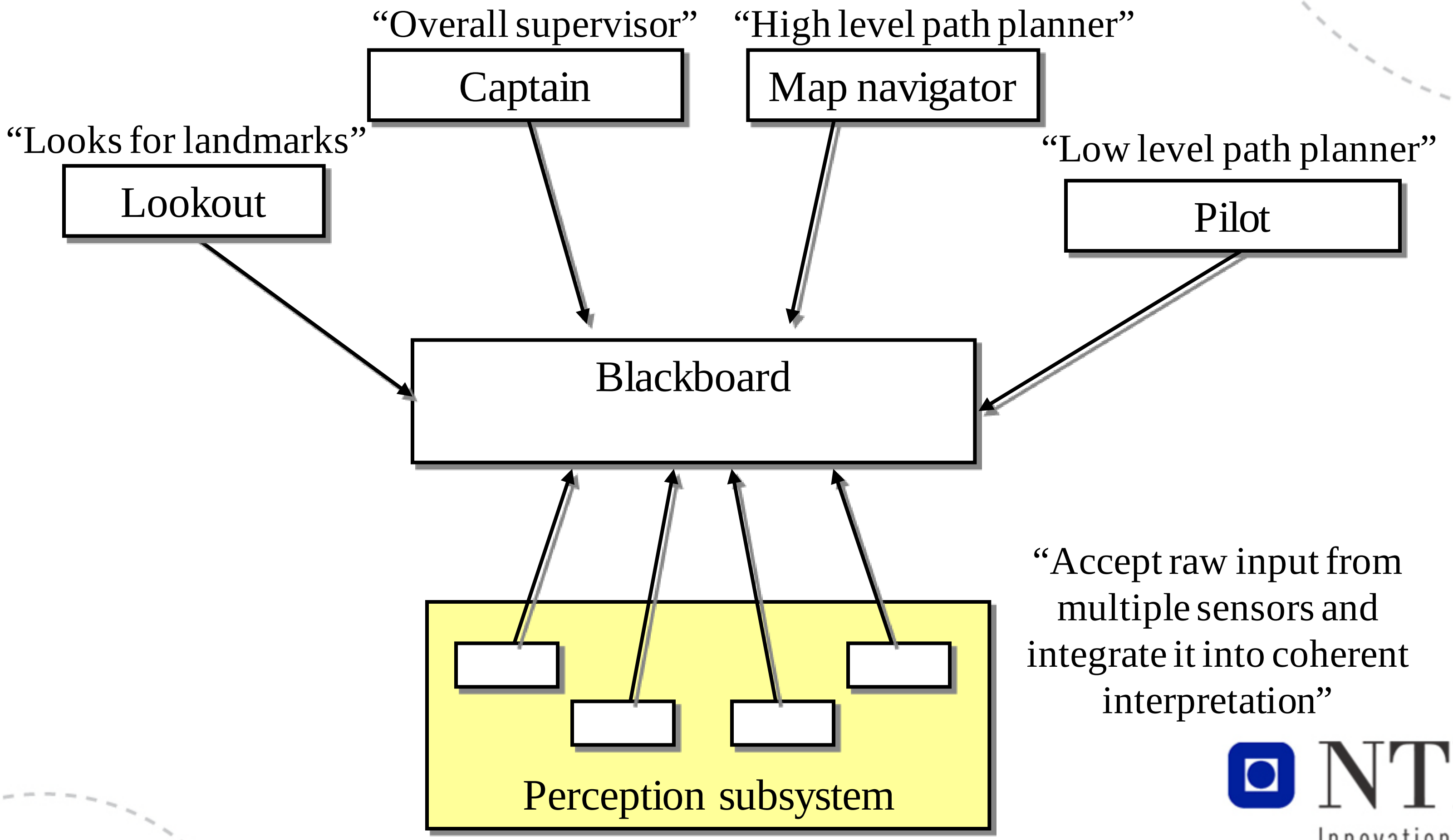
3. Blackboard Architecture Pattern



- Blackboard is a central database where all components can publish and subscribe info objects.
- Components can place observers that looks for certain characteristics.
- Often transaction management of info objects.
- Info objects can be inserted, duplicated, read, and removed.

NTNU
Innovation and Creativity

CODGER Reference Architecture (Blackboard)

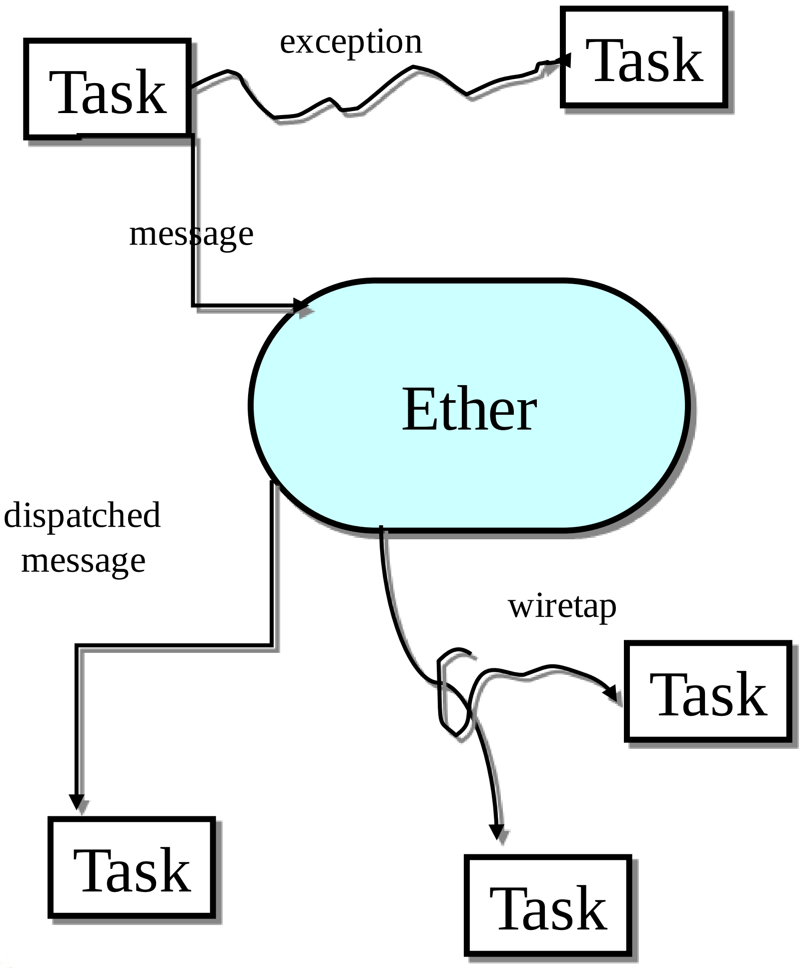


“Accept raw input from multiple sensors and integrate it into coherent interpretation”

Blackboard Architecture Pattern

- Benefits
  - Efficiency by parallel processing
  - Flexibility by recombination
  - Makes exchanging product families easy
- Liabilities
  - Can be expensive on system resources if large volume of data error handling

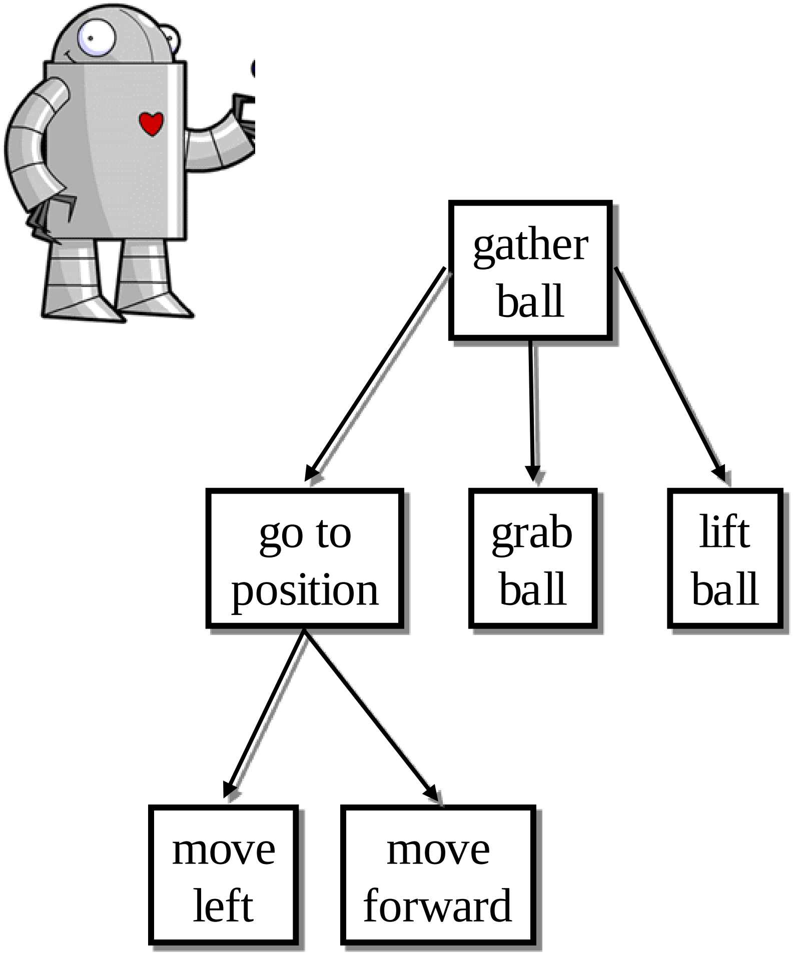
4. Task Control Architecture Pattern



- Tasks communicates by sending messages to central server.
- Server redirects messages to tasks that have registered to handle them.
- The sender does not need to know the receiver.
- Exceptions: Override current executing task.
- Wiretapping: Messages can be intercepted (safety check).
- Monitors: Read information and execute action if the data fulfil a certain criterion.

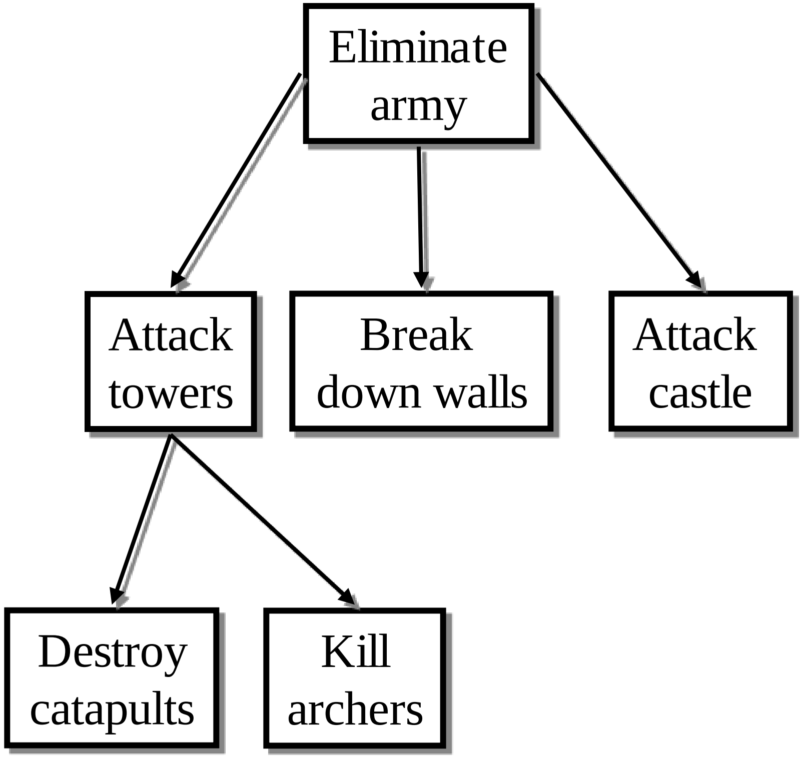
NTNU
Innovation and Creativity

Task Control Architecture Pattern #2



- The pattern uses hierarchical task trees:
  - Parent task initiate child tasks.
  - Task trees can be dynamically reconfigured at run-time:
    - Add task
    - Remove task
    - Abort task
    - Retry task
    - Etc..
  - Traverse the tree from left to right, from top to bottom.

Hierarchical task trees in games



- Hierarchical task trees is useful for modelling NPC/AI behaviour.
- Advanced Adaptive Intelligence can be decomposed into simpler tasks.
- Task trees can be dynamically changed during game play.

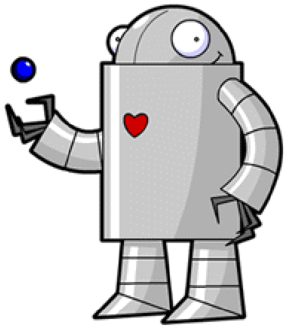
NTNU
Innovation and Creativity

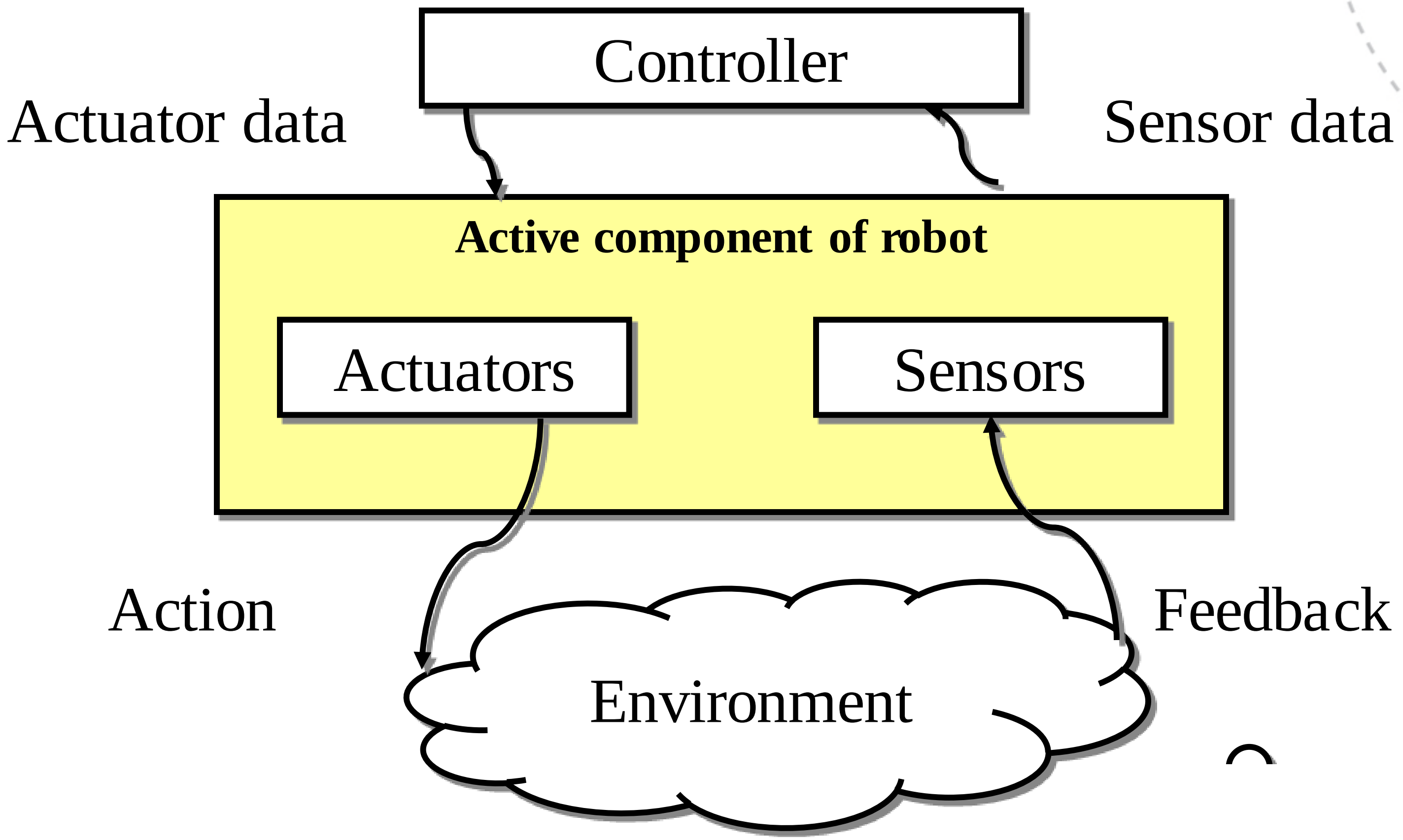
Task Control Architecture Consequence

- Benefits
  - Clear-cut separation of action (normal) and reaction (exception, monitor).
  - Incorporate independent concurrent agents (multiple tasks at the same time).
- Liabilities
  - Reusablility is low
  - Centralized approach that could be a bottleneck.

NTNU
Innovation and Creativity

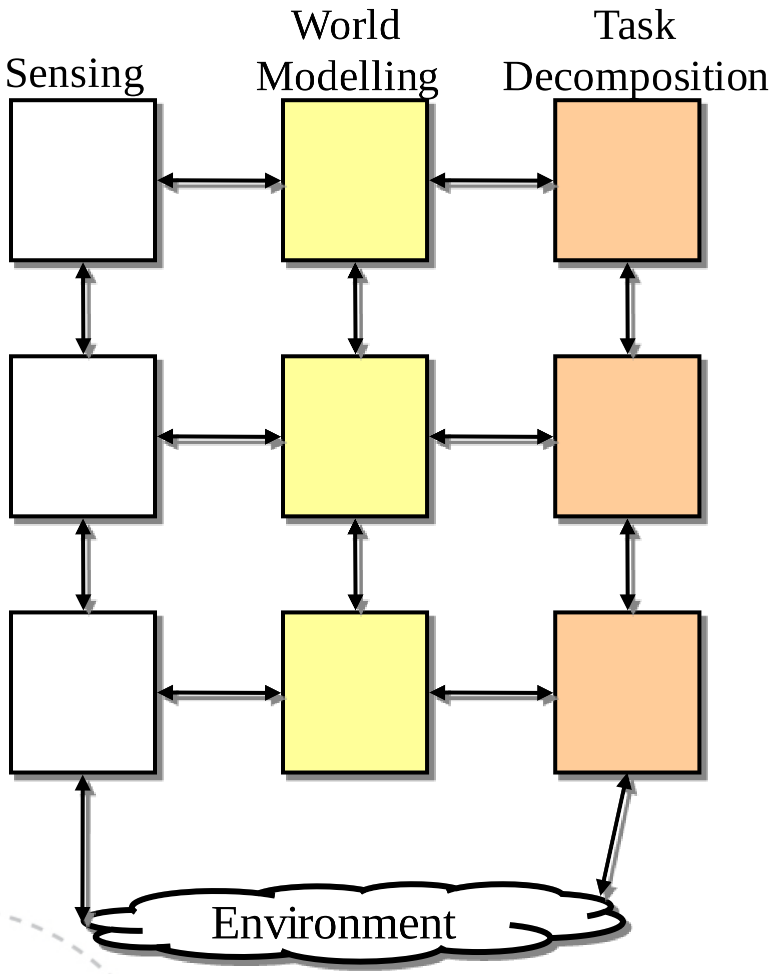
Control Loop Reference Architecture



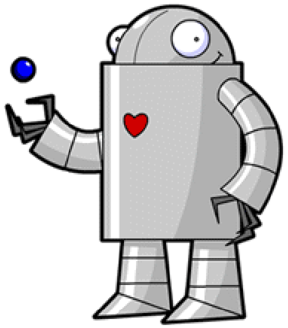


NTNU
Innovation and Creativity

The NASREM Reference Architecture



- Combination of control loop and layered architecture.
- Layers left to right represents functional abstractions.
- Layers describes execution sequence from top to bottom.
- Hierarchical control loops with increasingly tighter response time constraints.



NTNU
Innovation and Creativity

Summary

- When creating an software architecture:
  - Determine the impact of the quality attributes of the final system.
  - Look for software architecture patterns or reference architectures that fit the quality attributes.
  - Tailor the architecture patterns or reference architecture.

Bibliography (check It’s Learning)

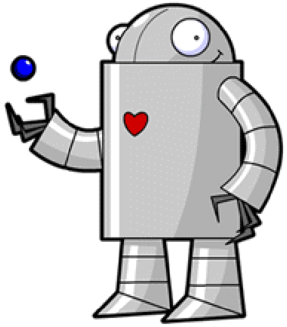
- **Game architecture:** Andrew Rollings & Dave Morris, Game Architecture and Design - A New Edition, New Riders Publishing 2004.
- **Control loop**: Tomás Lozano-Pérez. Preface to Autonomous Robot Vehicles. L. J. Cox and G.T. Wilfong, eds. Springer Verlag, New York, NY, 1990.
- **Elfes**:Alberto Elfes. Sonar-Based Real-World Mapping and Navigation. IEEE Journal of Robotics and Automation, no.3, 1987, pp. 249-265.
- **Task Control**: Reid Simmons. Concurrent Planning and Execution for Autonomous Robots. IEEE Control Systems, no. 1, 1992, pp.46-50.
- **CODGER**: Steven A. Shafer, Anthony Stentz, and Charles E. Thorpe. An Architecture for Sensor Fusion in a Mobile Robot. Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 7-10, 1986, pp. 2002-2011.
- **Subsumption:** Toal D., Flanagan C., Jones C., Strunz B., Subsumption architecture for the control of robots, IMC-13, Limerick 1996, pp. 703-711.
- **NASREM:** R. Lumia, J. Fiala, and A. Wavering. The NASREM Robot Control System and Testbed. International Journal of Robotics and Automation, no.5, 1990, pp. 20-26.

Bonus slides

NTNU

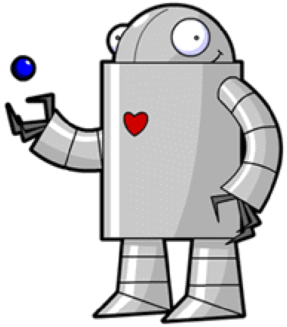
Innovation and Creativity

Elfes Reference Architecture (layered) Characteristics



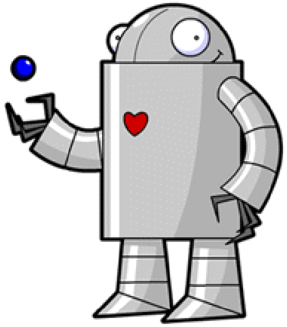
- Points out concerns of an autonomous robot.
- Defines abstraction levels to guide the design.
- Layered architecture does not fit actual data and control flow patterns.
- The model does not separate two abstraction hierarchies: Data (1-4) and control hierarchy (1,5-8).
- Abstract layers addresses management of uncertainty.
- Fault tolerance and safety by multi-level data and control analysis.
- Poor performance may require shortcuts.
- Hard to replace components because of all dependencies.

CODGER Reference Architecture Characteristics



- An architecture for sensor fusion in a mobile robot
- Components communicate via central database:
  - Components indicate their interest in certain type of info.
  - Database return info immediately or when some other module inserts info into the database.
- All control flow must go via blackboard, even if direct interaction is more natural.
- Uncertainties can be solved by allow the modules responsible for uncertainties register for needed data.
- Exceptions, wiretapping and monitors can be implemented as modules that watch the database.
- Supports concurrency and decouple senders from receivers gaining flexibility for maintenance.

Control Loop Reference Architecture Characteristics



- Simple: Captures the basic interaction between robot and the outside.
- Difficult in unpredictable environments.
- Assume linear environments and reactions.
- Model does not say how events are managed.
- No decomposition into cooperating software components (various concerns).
- Typical process: Trial-and-error where possibilities are eliminated.
- Fault tolerance and safety are supported by its simplicity makes duplication easy and reduces errors because of complex structure.
- The major components of the architecture is separated and can be replaced independently (sensors, controller, actuators).