NTNU
Norwegian University of Science and Technology

Faculty of Informatics, Mathematics and Electronics

Department of Computer and Information Sciences

ENGLISH

Examination results will be announced: 2. July

Exam in the subject
TDT4240 Software Architecture

Saturday 9. June 2012
9:00 am – 1:00 pm

Aids code C:

Simple calculator allowed.
The following specified printed documents are allowed:

- IEEE (2000), "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems", Software Engineering Standards Committee of the IEEE Computer Society.
- Kruchten, P. (1995), "The 4+1 View Model of Architecture", IEEE Software, 12(6).
- English-Norwegian dictionary (or to your native language if your not Norwegian) and/or an English thesaurus (English-English).

Contact person during the exam:

Alf Inge Wang, phone 7359 4485, mobile phone: 9228 9577

The points show how much each problem is worth in this exam. For each problem, each question has the same weight unless otherwise stated. The exam has 5 problems giving a total of 70 points. The remaining 30 points are credits awarded from the software architecture project.

Good Luck!

Meng Zhu and Tosin Daniel Oyetoyan

Controlled 25th of May 2012

Problem 1: Various questions (20 points)

Answer these questions *shortly*:

1.1	What is Bass, Clements and Kazman's definition of Software Architecture (the definition in the textbook)?
1.2	Why is graphics hardware abstraction important in game architectures according to Rollings and Morris?
1.3	What other hardware abstractions than graphics is useful in game architectures according to Rollings and Morris?
1.4	What is *fps* in a game and what parts of the game software affects the fps?
1.5	Draw a sketch of and explain the principle of decoupling game loop.
1.6	What is a token in the analysis of a game architecture?
1.7	What is the purpose of a token interaction matrix in a game architecture analysis?
1.8	What are the three main parts of Perry and Wolf's model of software architecture?
1.9	Why is software architecture important according to the textbook?
1.10	Write the three main areas of tactics described in the textbook related to performance.
1.11	What is the main advantage of using the *Composite* design pattern in a software architecture?
1.12	Give examples of typical response measures for the quality attribute usability.
1.13	What is architectural drift?
1.14	Draw a sketch of the architecture business cycle according to the textbook.
1.15	What is an architectural pattern?
1.16	How are wrappers used in development with Off-The-Shelves components according to the textbook?
1.17	Give five examples of important architectural drivers for a massive-multiplayer game such as World of Warcraft.
1.18	Describe the Attribute-Driven Design Process as described in the textbook.
1.19	What is the purpose of the IEEE1471 standard?
1.20	What is a design pattern?

References:

- Bass, Clements & Kazman: "Software Architecture in Practice"
- Rollings and Morris: "Game Architecture and Design – A New Edition"
- Perry & Wolf: "Foundation for the Study of Software Architecture"
- Coplien: "Software Design Patterns: Common Questions and Answers"
- IEEE: "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems"

Solution Problem 1:

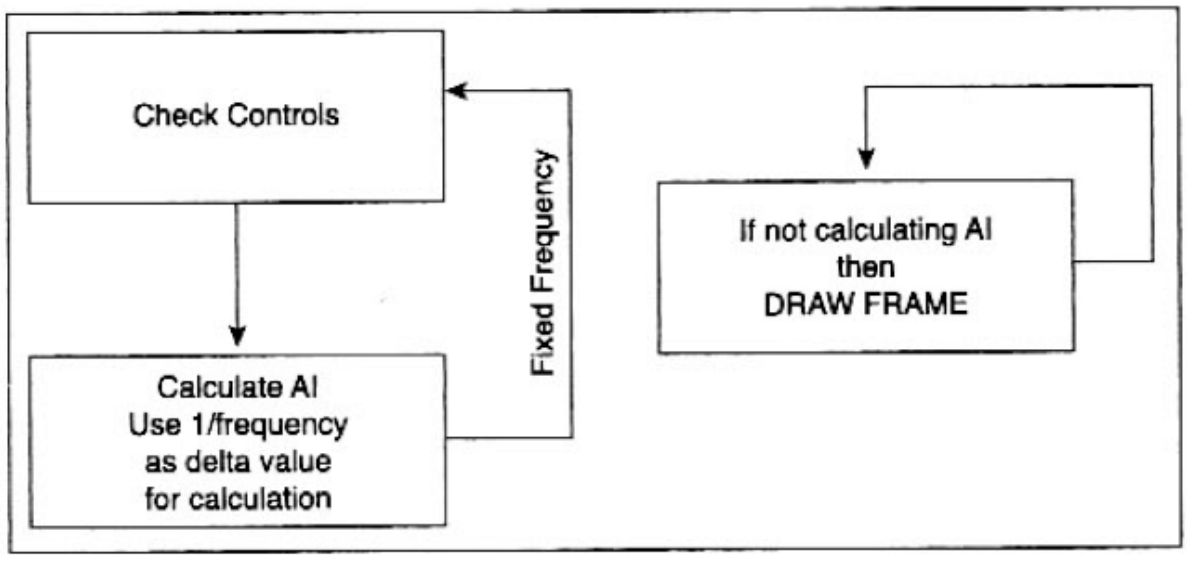
1.1 Software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

1.2 Hardware abstractions make it possible to create pretty much the same software to run on various hardware independent of graphics resolution, number of colors, use of sprites etc. The main benefit is reuse!

1.3 Sound hardware, input devices (keyboard, mouse, joystick, joypad etc.), and network.

1.4 FPS is Frames per second – the number of screens a program can produce (render) in a second. Any game logic affects FPS, e.g. CPU heavy computations, AI computation, collision detection, graphical computations, sound computations etc. As long as it takes time to compute, it will affect the FPS.

1.5 The game loop checks the controls and calculates AI at a fixed frequency. If not the CPU is not occupied with calculating AI, then it draws a frame (drawing frame whenever have time).



1.6 Tokens are discrete elements that are directly or indirectly manipulated by the player in a game.

1.7 The purpose of the token interaction matrix is to discover how the different game elements interact in order to find the various events in a game.

1.8 Elements, Form and Rationale

1.9 Communication among stakeholders, Early design decisions, Transferable abstraction of a system, Insurance on doing things right, Ensuring quality, Easier to manage change, Easier to reuse, Enables more accurate cost and schedule estimates, Enable building large systems, etc.

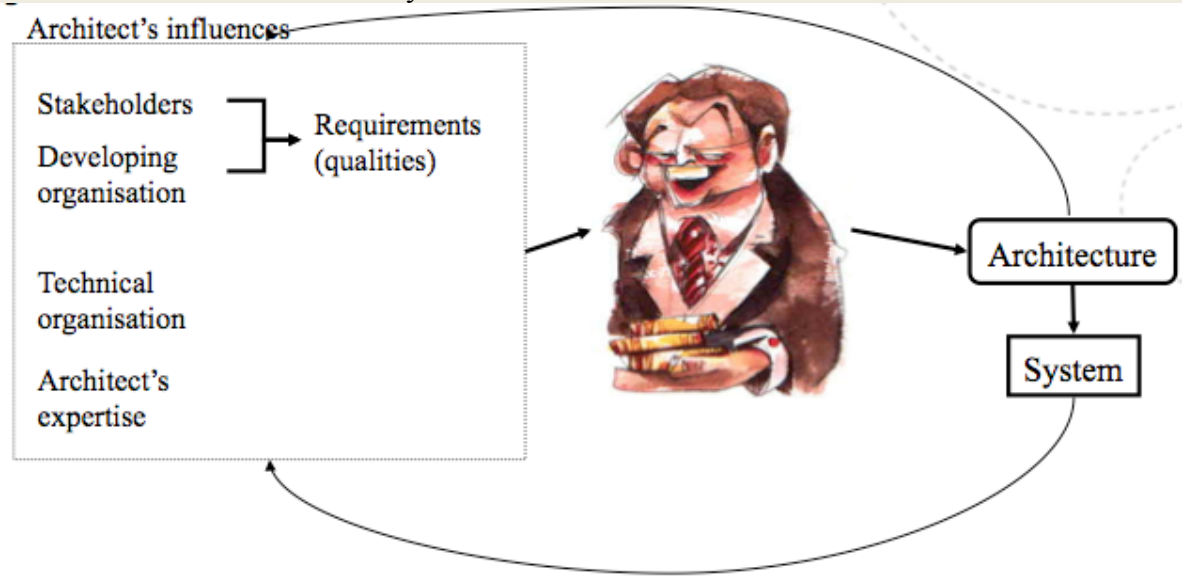
1.10 Resource demand, Resource management, Resource arbitration.

1.11 The Composite pattern makes is possible to handle single objects and objects that consists of several sub-objects in the same way.

1.12 Task time, Number of errors, Number of problem solved, User satisfaction, Gain of user knowledge, Ration of successful operations of total operations, Amount of time/data lost.

1.13 Implementation drifts away from the architectural documentation.

1.14 The architectural business cycle:



1.15 Description of element and relation types together with a set of constraint on how they may be used.

1.16 Encapsulate components with an alternative abstraction with an interface translation that includes translating an element of a component interface into an alternative element, hiding an element of a component interface, and preserving an element of a component's interface.

1.17 Architectural drivers for massive-multiplayer games:
- Performance: Huge amount of simultaneous users (players) – high server demands
- Performance: Low response times for players (network play) – high server/network demands
- Availability: Downtime on servers will cause player losses and thus business losses
- Availability: Players must rolled back to a consistent state if the server crashes
- Security: The game must withstand cheating or hacking the game's state values (levels, attributes, currency etc.)
- Modifiability: It should be possible to update servers without disturbing the players as little as possible.

1.18 The ADD process:
1 Choose module to decompose
2 Refine the module
  a Choose architectural drivers
  b Choose/Create architectural patterns to satisfy the drivers
  c Instantiate modules and allocate functionality
  d Define interfaces of the child modules
  e Verify and refine use cases and quality scenarios
3 Repeat the steps above for every module requiring further decomposition

1.19 The purpose of IEEE1471 standard is to describe what you should put into an architectural documentation without saying anything about the diagram language used.

1.20 A design pattern is a reusable solution or experience of a known problem.

Problem 2: Choose most appropriate design pattern (5 points)

Nominees:

a) Singleton
b) Abstract Factory
c) Factory method
d) Composite
e) Observer
f) Template method

Choose the *most appropriate design pattern* (one) for dealing with the 5 problems described below. Motivate for your choices (write reasons for choosing a pattern):

1. Need a mechanism to provide easy management of several configurations of look and feel in a software framework for graphical user interfaces that provides the same core functionality (buttons, windows, text-entry fields etc.).
2. Need a mechanism to manage various shapes in a graphics editor where complex shapes (e.g. a rectangle) are made out of simple shapes (e.g. lines).
3. Need a mechanism to notify clients when a server finds news gathered from the web related to the clients' specifications.
4. An application for a travel agency to manage packages of trips should be developed. You should provide a mechanism to allow all the trips follow the same structure (transport to location, visiting a location, transport back), but each trip package provides variations in transports (to and back), and locations.
5. Provide a global logging mechanism that can be accessed by all parts of a system.

Solution Problem 2:
2.1 Abstract Factory – b
2.2 Composite – d
2.3 Observer – e
2.4 Template method – f
2.5 Singleton – a

Problem 3: Software Process and Software architecture (5 points)

Describe short how software architectural practices and concerns affect the following phases of the software process. Give concrete examples from the software architecture project in the TDT4240 course.

a) Requirement phase
b) Design phase
c) Evaluation phase
d) Implementation phase
e) Testing/Validation phase

Solution Problem 3:
3a) Quality requirements, COTS and other architectural constraints
3b) Architectural drivers, Stakeholders, Selection of view points, Architectural tactics, Architectural/Design patterns, Views, Consistency among views, Rationale
3c) ATAM – Quality requirements, Architectural description
3d) Implementation according to architecture, using patterns
3e) Test of quality requirements, Relationship to architecture, Lessons learned

Problem 4: CBAM (10 points)

From Table 1, 2 and 3 and the information below, find:

a) **Total benefit** from the 3 architectural strategies
b) **Return-On-Investment** for the 3 architectural strategies
c) **Rank the 3 architectural strategies** according to best investment.

Use straight lines between the data points in utility-response curves.

The data is results from applying the Cost Benefit Analysis Method (CBAM) on an information system for NTNU named TWOTS (Total Waste of Time System) for managing students, teachers, courses, equipment etc.

Table 1: Results from prioritizing scenarios with worst, current, desired and best response levels.

Scenario	Vote	Worst	Current	Desired	Best
1. Availability: Downtime per week	30	60 min	10 min	2 min	0 min
2. Performance: Response time on user requests	20	5 sec	3 sec	1 sec	1 sec
3. Performance: Number of simultaneous user transactions per sec	15	100	500	1500	5000
4. Modifiability: Time to add support for short courses for companies	10	30 days	15 days	1 day	1 day
5. Usability: Average mistakes out of correct operations made by users	20	30%	10%	1%	0%

Table 2: Results from assigning utility to the various scenarios.

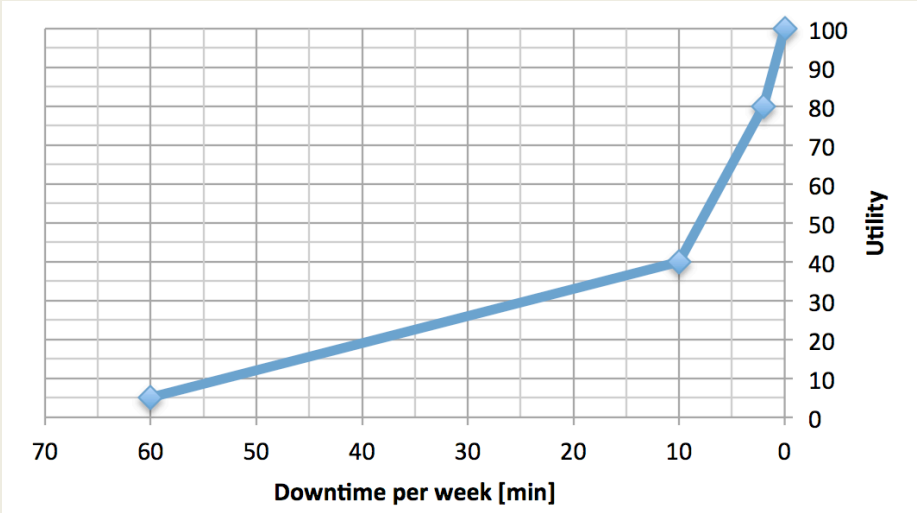
Scenario	Vote	Worst	Current	Desired	Best
1. Availability	30	5	40	80	100
2. Performance	20	10	60	100	100
3. Performance	15	5	30	50	100
4. Modifiability	10	5	20	100	100
5. Usability	20	10	40	80	100

Table 3: Effect and cost of using architectural strategies.

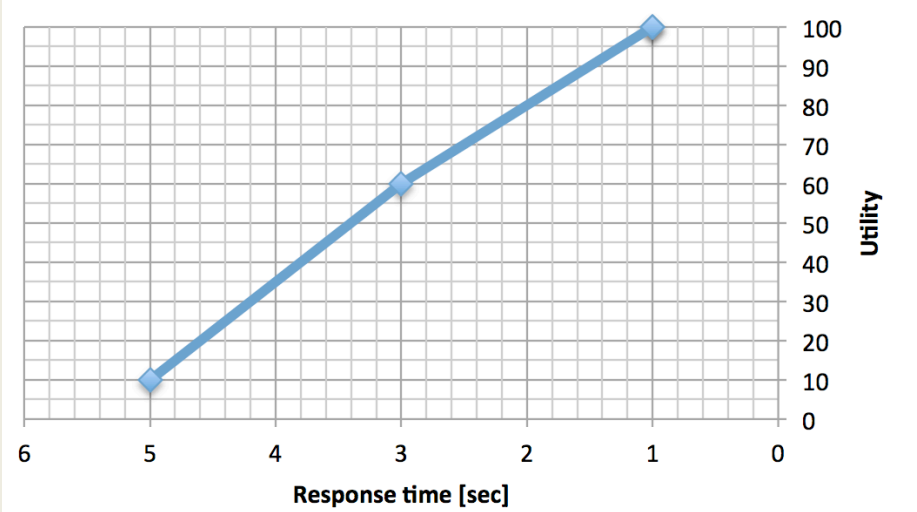
Strategy	Scenario	Cost	Current response	Expected response
1. Replicated servers (hardware and software)	1 2 3	100000	10min 3sec 500	4min 2sec 1000
2. Faster hardware	2 3	20000	3sec 500	2sec 750
3. Improved exception handling	1	5000	10min	8min

Solution Problem 4:
We do only have to create utility-response curves for Scenario 1-3, as the architectural strategies only affect these scenarios.

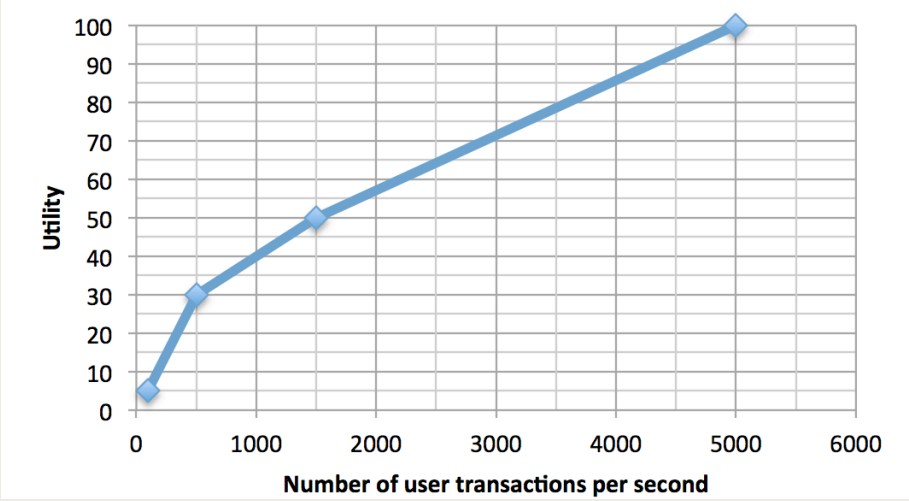
Utility-response curve scenario #1:



Utility-response curve scenario #2:



Utility-response curve scenario #3:



Effects of strategy #1 Replicated servers (read from the graphs):
- Scenario 1: From 10 min to 4 min: 75% utility (from 40%)
- Scenario 2: From 3 sec to 2 sec: 80% utility (from 60%)
- Scenario 3: From 500 to 1000 trans: 40% utility (from 30%)

Effects of strategy #2: Faster hardware (read from the graphs):
- Scenario 2: From 3 sec to 2 sec: 80% utility (from 60%)
- Scenario 3: From 500 to 750 trans: 35% utility (from 30%)

Effects of strategy #3: Improved exception handling (read from the graphs):
- Scenario 1: From 10 min to 8min: 45% utility (from 40%)

a) Total benefit on tactics 1, 2 and 3:
B_1 = (75-40) x 30 + (80-60) x 20 + (40-30) x 15 = 1600
B_2 = (80-60) x 20 + (35-30) x 15 = 475
B_3 = (45-40) x 30 = 150

b) Return-on-investment on tactics 1, 2 and 3:
ROI_1 = 1600/100000 = 0.0160
ROI_2 = 475/20000 = 0.0238
ROI_3 = 150/5000 = 0.0300

c) Ranking of tactics (from best to worst):
1. Tactic 3 Improved exception handling
2. Tactic 2 Faster hardware
3. Tactic 1 Replicated servers

Problem 5 Design a software architecture (30 points)

Read the description of AI-Drillo and do an architectural design. Your answer must include:

a) Architectural drivers – 2 points
b) Architectural tactics and patterns – 3 points
c) Process view (only state diagram of the states of the car) – 5 points
d) Logical view – 17 points
e) Architectural rationale – 3 points

Motivate for your choices and state your assumptions.

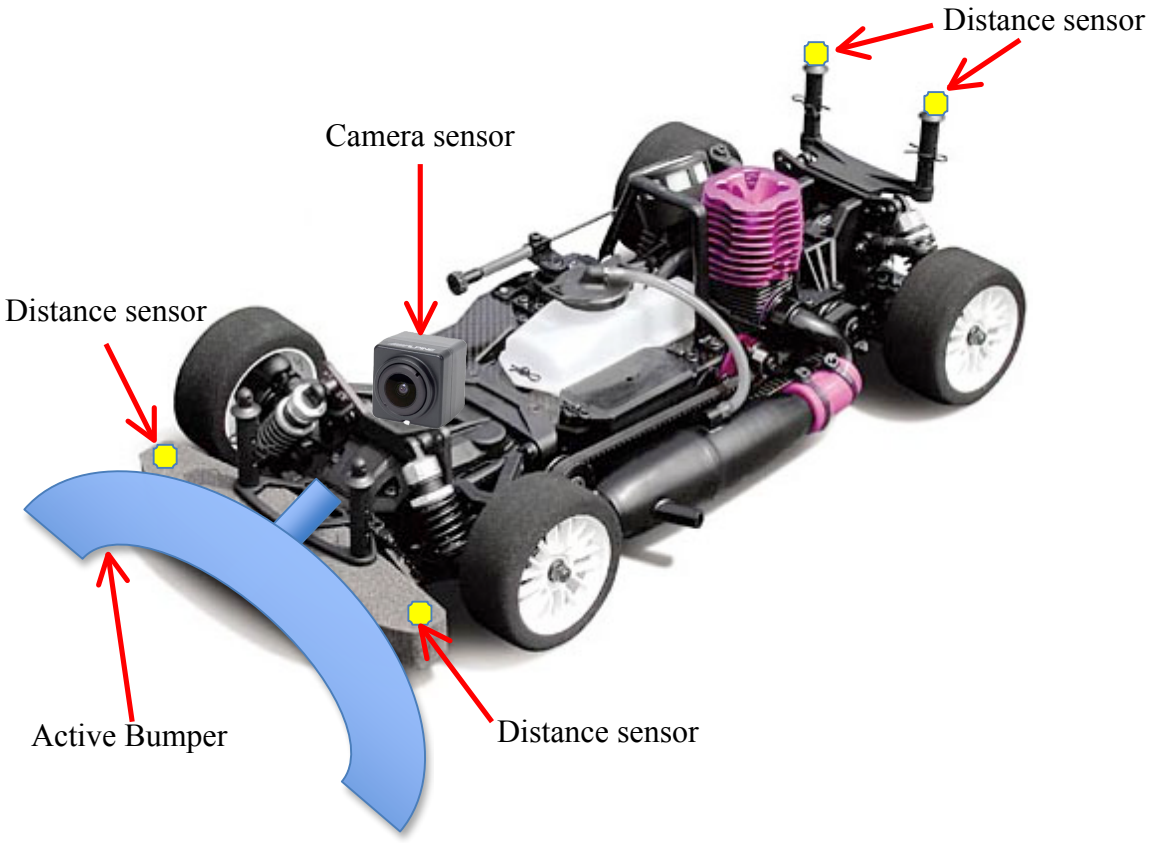
Software for managing a computer-controlled soccer RC car - AI-Drillo

AI-Drillo is computer-controlled car equipped with four distance sensors at each corner of the car and a bumper in front that can hold the ball and be activated to shoot the ball forward. AI-Drillo has also a front camera that can recognize a soccer ball, sidelines of the soccer field and the goals. Further, the car has activators to control the speed of the engine transferred to wheels in the back of the car (forward and reverse), as well as controlling the angle on the front wheels (for turning). It should be possible to use various activators to enable different engines and steering mechanics for the car. The wheels also have a counter that increases by one every time the left front wheel has turned 360 degrees. The goal of the car is to drive around the field and find a ball, then take the ball to the correct goal and score. The car should stop when it has scored 5 goals. Patterns on the sidelines will help the car to find the right goal.

The software of AI-Drillo should manage driving around the soccer field, avoiding other cars or driving out of the field, getting the ball when it is detected, find the goal, and score. Several balls are out in the field for the car to pick up.

It should be possible to replace sensors and activators of the car. The AI-Drillo is a prototype where it must be possible to change the behavior of the car at runtime by updating/replacing software component while the car is running.

Here is a conceptual illustration of the AI-Drillo RC car:

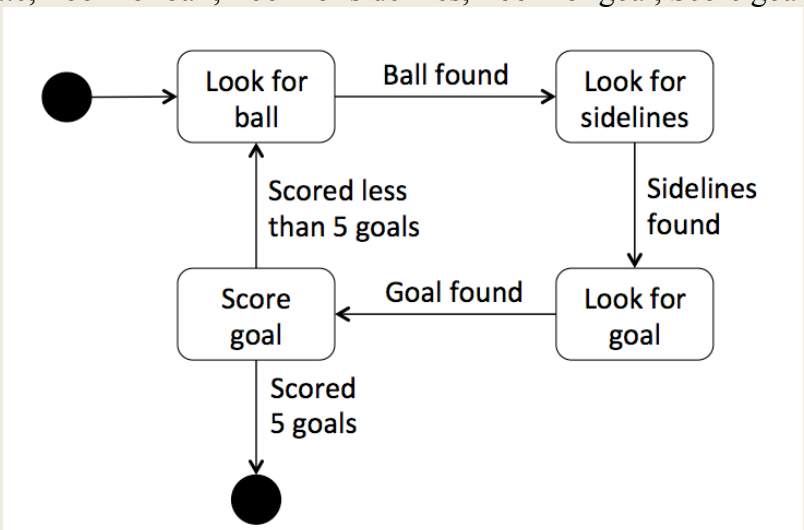


Solution Problem 5:
5a: Architectural tactics:
Modifiability (real-time): The software components can be replaced in run-time.
Modifiability (design time): The software architecture should support various sensors and activators.
Availability: The software architecture should ensure high availability to avoid that the car crash into things due to software faults or failures.
Performance: The software architecture should ensure that important decisions such as turning and speed of the car have fast enough reaction-time.

5b: Tactics:
Modifiability tactics: Assign functionality that are related to same classes, Hide internal complexity, Well-defined interfaces, defer binding time by communicating through well defined interfaces and protocols.
Availability tactics: Exception handling, and restart if system does not respond within 5 seconds.
Performance tactics: Schedule and priorities motor and wheel controls.

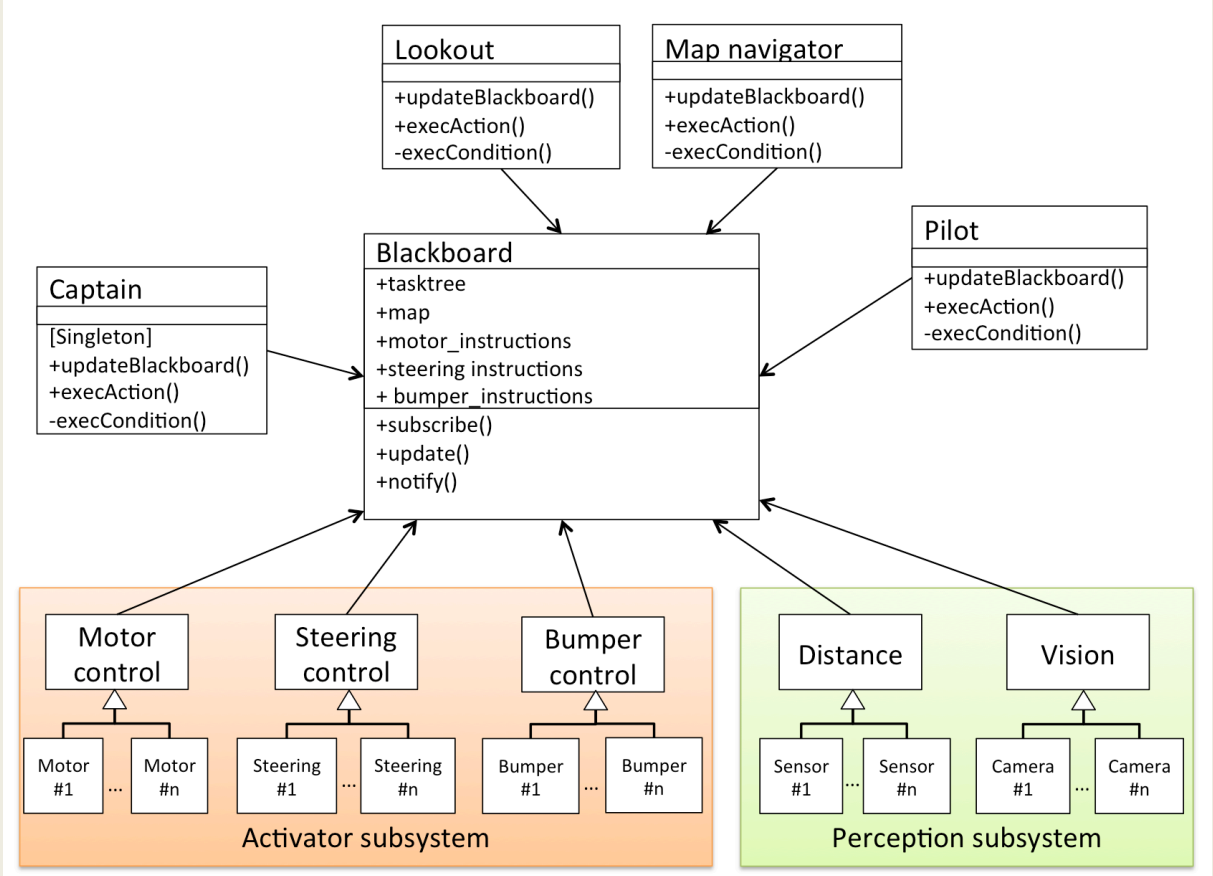
5b: Patterns:
The major architectural pattern used is the *Blackboard architectural* pattern, which allows components to be changed runtime, as all communication is handled by the blackboard through information objects. All components can publish and subscribe to information objects, the components can place observers that looks for certain characteristics. Information objects can be inserted, read, and removed. The format of information objects are predefined in such a way that one component can be changed with a new one in run-time. Further, the more specific *CODGER Reference architecture* can be used to specify the various roles for the components: Pilot, Lookout, Captain, Map navigator and a Perception subsystem.
Another architecture pattern is the *Task Tree* pattern that is used to represent the tasks of the AI-Drillo car in a hierarchical manner.
The *Observer* design pattern is used to implement the publish/subscribe mechanism. The *Singleton pattern* is used to ensure that there is only one instance of the controlling class.

5c: Process view
The process view is a state diagram that shows the six states of the software for controlling the car: Start state, Look for ball, Look for sidelines, Look for goal, Score goal, and End state.



5d: Logical view

The architecture is based on the CODGER reference architecture that utilize the Blackboard pattern. Lookout is responsible for looking for ball, sidelines, other cars and the goals. Pilot is responsible for low-level path planning (local navigation), while Map navigator is responsible for high-level path planning. Captain is in charge of controlling the software using information objects from the others. To ensure a consistent behavior, only one Captain is allowed by using the Singleton pattern. Blackboard is in charge for managing information objects, making it possible to subscribe to information objects and notify subscribers when updates occur (implemented by the Observer pattern). The information objects managed by the Blackboard are tasktree (tree representation of the tasks to be executed), a map, motor_instructions (removed after they have been executed), steering_instructions (removed after they have been executed), and bumper_instructions. A Perception subsystem is provided with one part for distance sensors (Distance) and one for camera (Vision). A similar Activation subsystem is provided for controlling the bumper, motor and the steering. Both subsystems use generalization to provide support for various sensors and activators.



The logical view does not show the infrastructure for removing and adding new components dynamically.

5e: Architectural rationale

The main focus of the architecture has been on modifiability. The Blackboard architectural pattern was chosen to provide a way of allowing run-time modification of system components. This is now possible, as the components have no direct associations but interact through the Blackboard using information objects. This also makes it possible to e.g. have more than one Pilot, Lookout or Map navigator. Generalization has been used to support various sensors and activators. The singleton pattern was used to ensure consistent behavior. The architecture does not show tactics on performance and availability.