NTNU
Norwegian University of Science and Technology

Faculty of Physics, Informatics and Mathematics

Department of Computer and Information Sciences

ENGLISH

Sensurfrist: 22. June

Exam in the subject
TDT4240 Software Architecture

Tuesday 27. May 2004
9:00 am – 1:00 pm

Aids code C:

Simple calculator allowed.

These 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 a English thesaurus (English-English).

Contact person during the exam:

Associate professor Alf Inge Wang, phone 73594485, mobile phone: 92289577

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 4 problems giving a total of 60 points.

Good Luck!

Problem 1 (10 points): Various questions

Answer these questions short:

1.1 What is a design pattern?

A: Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. One person's pattern can be another person's building block.

1.2 What are the advantages by using the Singleton design pattern in a software architecture?

A: Control the number of instances of a class and to avoid misuse.

1.3 What is Bass, Clements and Kazman's definition of Software Architecture?

A: 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.

1.4 What is a variation point in a software product line?

A: Places in the architecture that we can point to that capture the variation.

1.5 What is a *wrapper* related to Off-The Shelves components and architecture?

A: Encapsulation of a component within an alternative abstraction. Used for wrapping software e.g., legacy systems, in a component-based framework.

1.6 What is *View fusion* in reconstructing software architecture?

A: View fusion is to combine information in a database to create new views in order to reconstruct a database.

1.7 Why can it be necessary to reconstruct a software architecture?

A: Discover that a legacy system does not have a coherent architectural design, check conformance against "as-design" architecture, enable maintenance, integration, expansion reuse of a legacy system, and enable comparison of different systems.

1.8 What is an architectural driver (give examples)?

A: Things that make big impact on the architecture, like the developing organization, technical environment, business environment, requirements and qualities from stakeholders etc. Example: Time to market is 5 months.

1.9 What is an architectural strategy?

A: A collection of architectural tactics, which are fundamental design decisions.

1.10 What is the motivation for introducing software architecture into software development?

A: More predictable development of systems, software architecture analysis is cheap, to ensure extensibility, performance, availability, etc. Software architecture is the link between the system's quality attributes and design.

Problem 2 (10 points): Architecture Pattern, Reference Model, Reference Architecture or Architecture

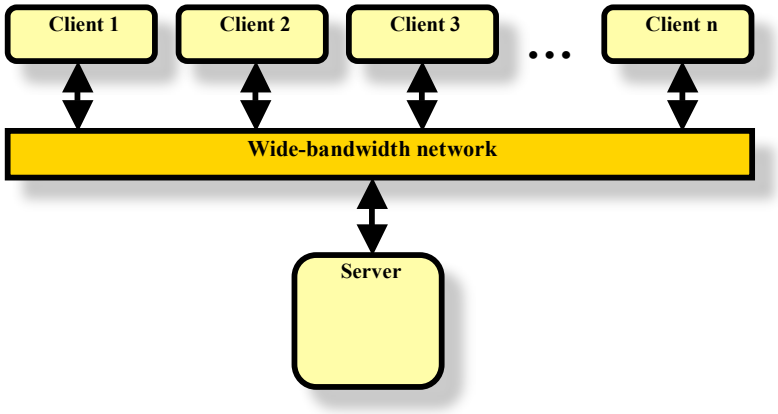
Decide if the following model examples are Architecture Pattern (AP), Reference Model (RM), Reference Architecture (RA) or Architecture (A) according to the book Software Architecture in Practice. Explain and motivate your choice.

A: Definitions.

- *Architectural pattern/style:* Description of element and relation types together with a set of constraints on how they may be used.
- *Reference model:* A division of functionality together with data flow between the pieces.
- *Reference architecture*: Reference model mapped onto software elements and the data flows between them. Standard decomposition of system components. Software elements that cooperatively implement the functionality defined in the reference model.
- *Software architecture*: See problem 1.3.

2.1 Client-server model:

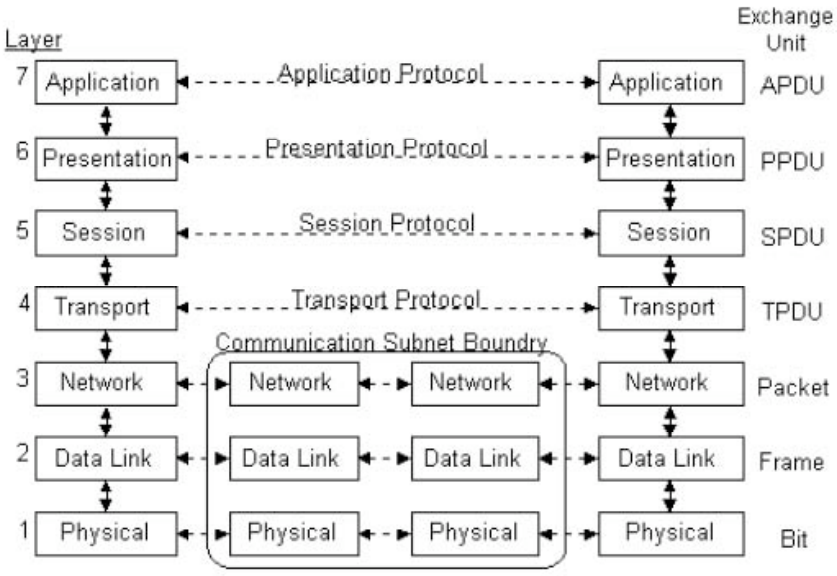
Description: A model describing a server providing services to clients through a wide-bandwidth network.



A: This is an **Architectural pattern** because it describes some software elements without any specific functionality and how it is structured.

2.2 OSI model:

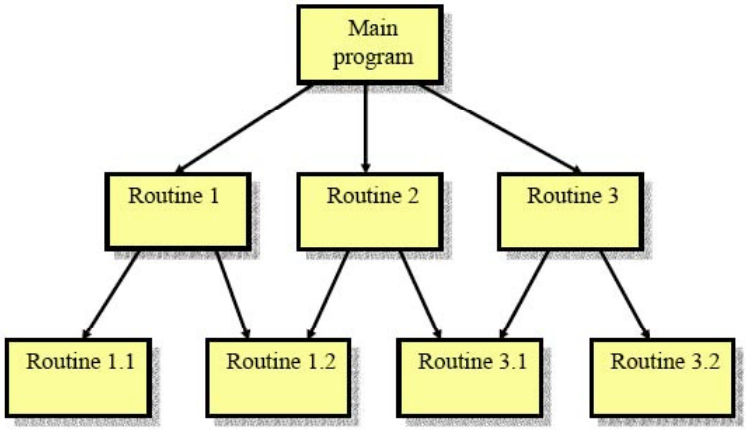
Description: Describing how applications can communicate through network using layers:
7) Provides services to the applications
6) Converts the information
5) Handles problems which are not communication issues
4) Provides end to end communication control
3) Routes the information in the network
2) Provides error control between adjacent nodes
1) Connects the entity to the transmission media



A: According to the description of the OSI model above, there are three possible. The OSI model can be viewed as a **Reference architecture** because it describes software elements and the structure between them, but also the functionality of each software elements. Another correct answer would be to call it **an Architecture**.

2.3 Call-return model:

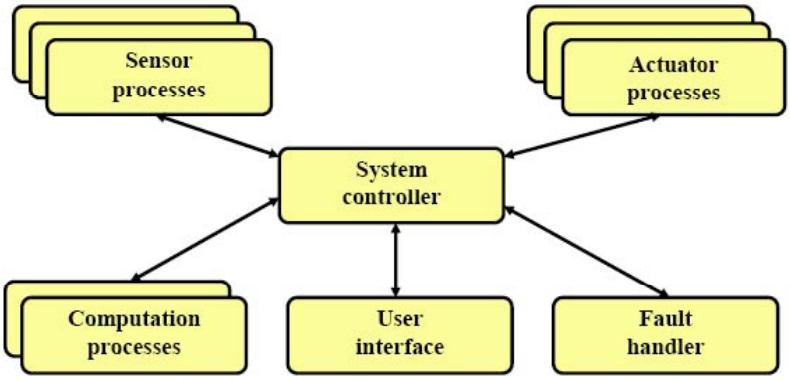
Description: Top-down subroutine model where control starts at the top of a subroutine hierarchy. The model is only applicable to sequential systems. The main program can call Routines 1, 2 and 3. Routine 1 can call 1.1 or 1.2 etc.



A: This is an **Architectural pattern** because it describes some software elements without any specific functionality and how it is structured.

2.4 Centralized management of control model:

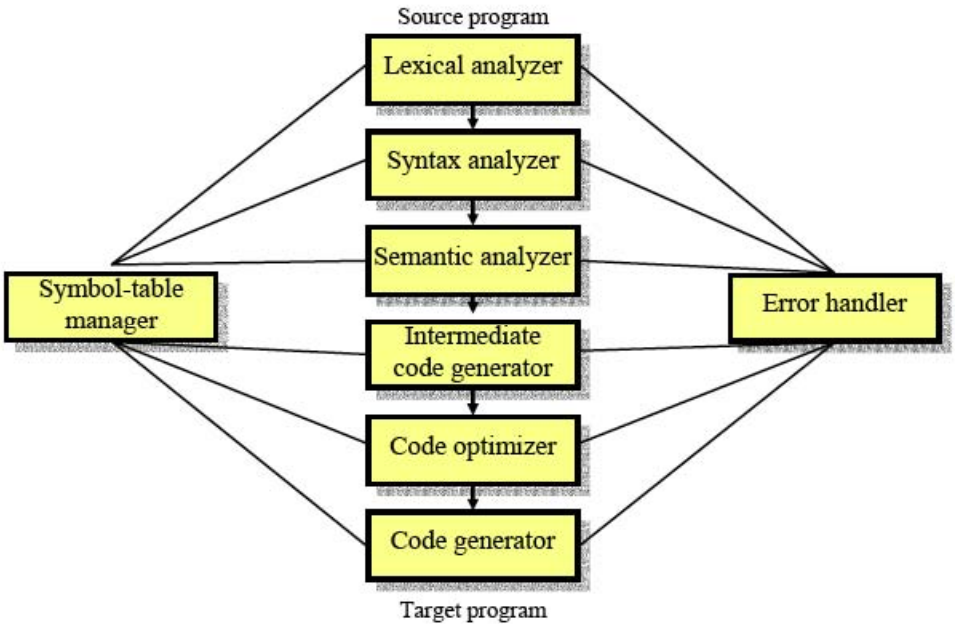
Description: This model is often used in "soft" real-time systems which do not have a very tight time constraints. The central controller manages the execution of a set of processes associated with sensors and actuators.



A: This should be classified as a **Reference architecture** because it presents grouping of functionality and also identifies some software component structures as parallel instances of sensor and actuator processes etc.

2.5 Compiler model:

Description: This model describes the phases of a compiler. The phases represent the necessary functionality to implement a compiler.



A: The compiler model is a **Reference model** as it identifies the grouping and functionality of a compiler and does not say much more about the software elements in it.

Problem 3 (10 points): The CBAM

From the table 1, 2 and 3 and the information below, compute (use straight lines between the data points in the graph):

- Total benefit obtained from the 3 architectural strategies.
- Return-On-Investment for the 3 architectural strategies and find what architectural strategy is the best investment.

The data is results from applying the Cost Benefit Analysis Method (CBAM) on a web-system for selling tickets for cinemas, theatres, music concerts, and sports events.

Total Benefit Obtained can be computed using this formula: $B_i = \Sigma (b_{i,j} \times W_j)$ where:

- B is benefit for each architectural strategy *i*
- $b_{i,j}$ is the benefit by using strategy I to its effect on scenario j
- W_j is the weight of scenario j

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

Scenario	Vote	Worst	Current	Desired	Best
1. Performance (simultaneous users)	20	1000	3000	5000	10000
2. Availability (server failure)	40	10 % fail	5% fail	1% fail	0 % fail
3. Availability (transactions lost)	25	4% lost	2% lost	0% lost	0% lost
4. Usability (% of users need help)	15	40%	20%	0 %	0%

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

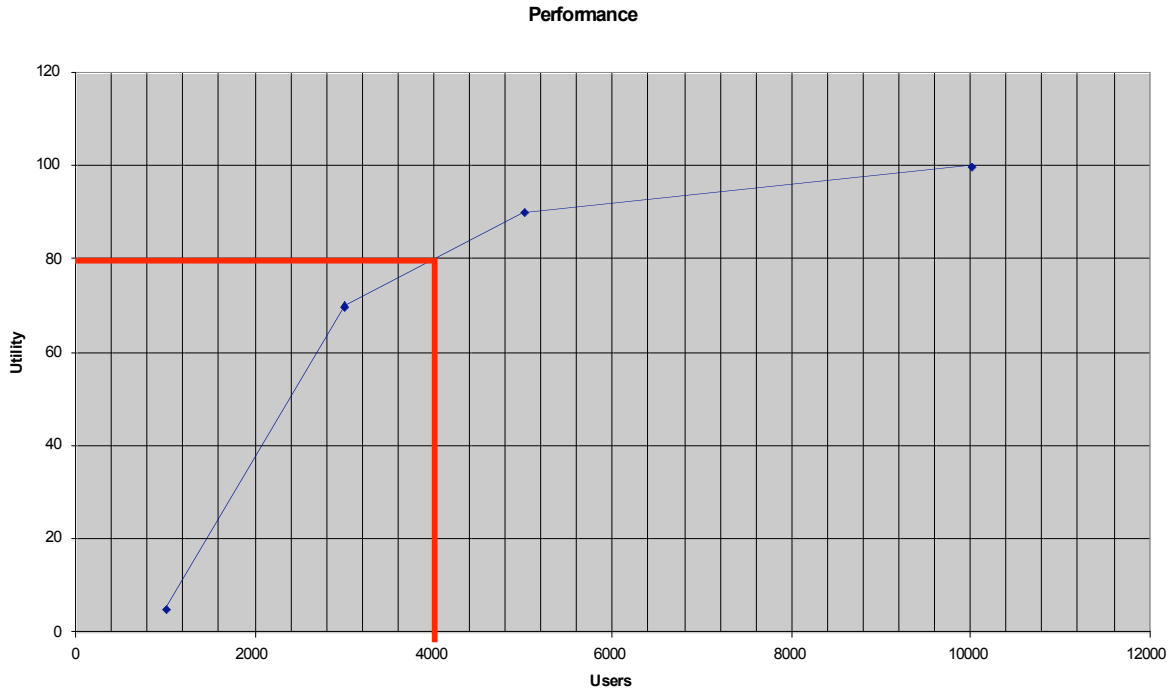
Scenario	Vote	Worst	Current	Desired	Best
1. Performance	20	5	70	90	100
2. Availability	40	0	60	80	100
3. Availability	25	15	80	100	100
4. Usability	15	20	60	100	100

Table 3: Effect and cost of using architectural strategies.

Strategy	Scenario	Cost	Current response	Expected response
1. Increase computational efficiency	1	500	3000	4000
2. Active redundancy	2 3	1000	5% fail 2% lost	2% fail 1% lost
3. Support for cancel/undo	4	900	20%	5%

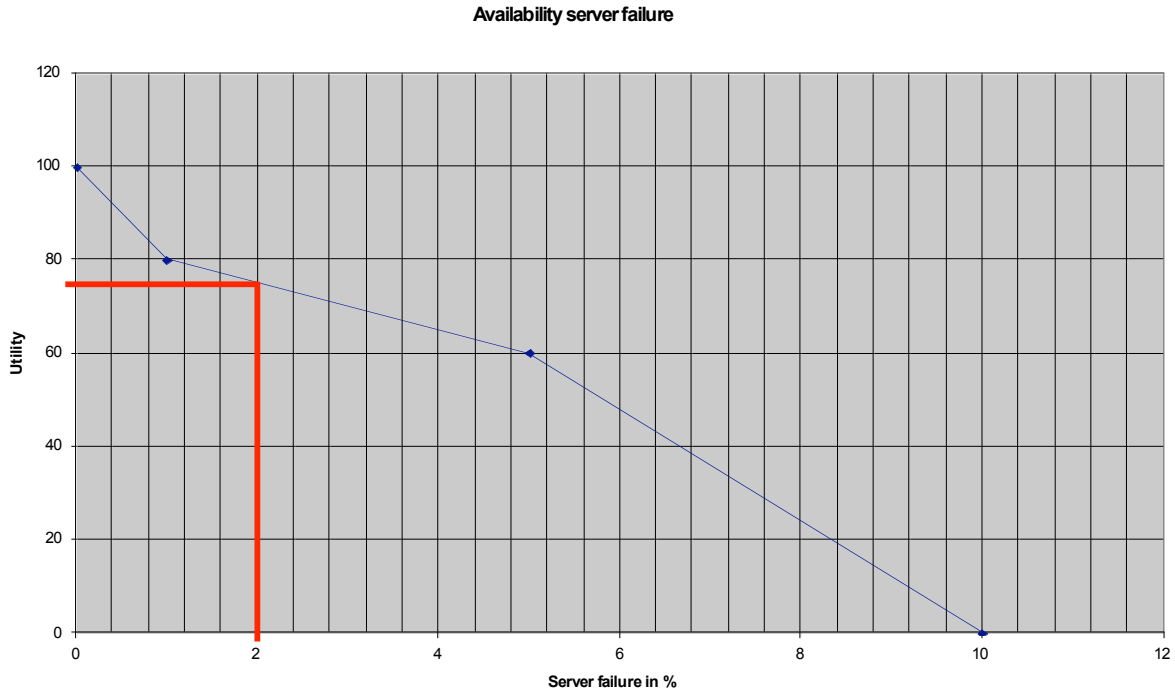
A: To compute the total benefit obtained, the return-on-investment, and find what architectural strategy is best investment, utility response curve for all 4 scenarios must be made. From this curves the utility of expected response when applying the architectural strategy can be read from the graph. Note that the numbers can be slightly different depending on how the graphs are drawn. The conclusion however should be the same.

Utility response curve for scenario 1:



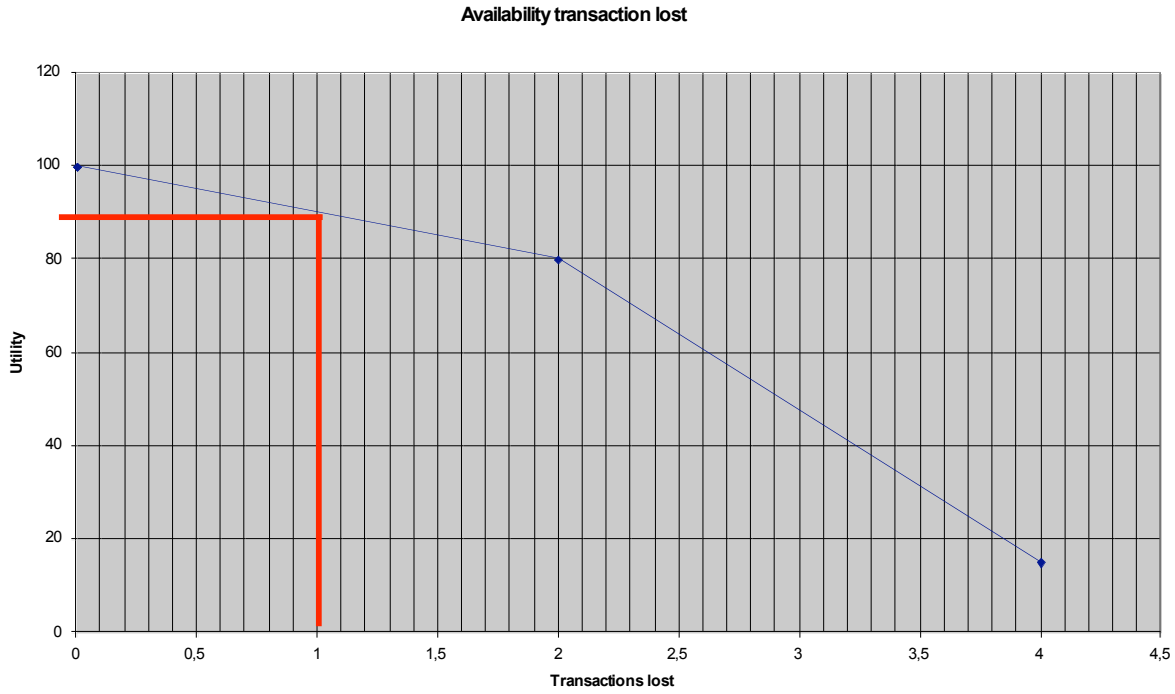
By applying strategy 1, we get the expected response **4000** (table 3). By looking at the utility response curve we get that the utility **80%** by applying strategy 1.

Utility response curve for scenario 2 (availability server failure):



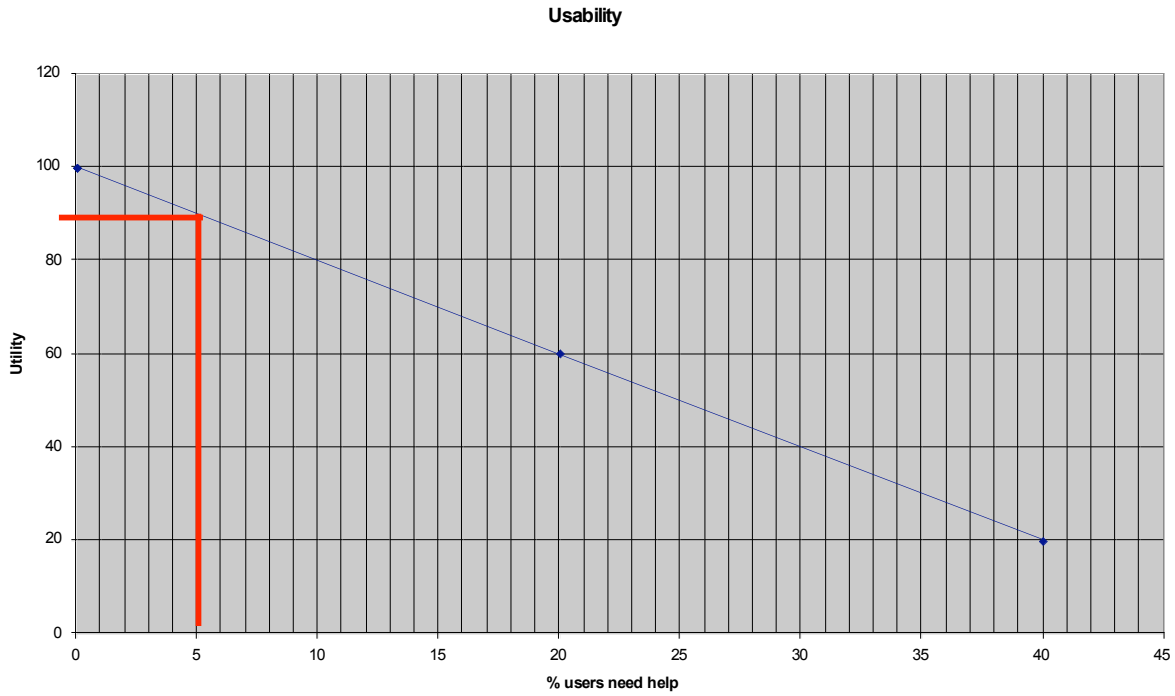
Note that this curve can be flipped horizontally for better readability. By applying strategy 2 on scenario 2 we get the expected response **2% fail** (table 3). From the response curve we find the utility for expected response to be **75%**.

Utility response curve for scenario 3 (availability transaction lost):



Note that this curve can be flipped horizontally for better readability. By applying strategy 2 on scenario 3 we get the expected response **1 % lost** (table 3). From the response curve we can find the utility for expected response to be **90%**.

Utility response curve for scenario 4:



Note that this curve can be flipped horizontally for better readability. By applying strategy 3 on scenario 4 we get the expected response of **5%** (table 3). From the response curve we can find the utility of the expected response to be **90%**.

The total benefit obtained from the 3 architectural strategies is (expected utility-current utility)* Weight:

- B for strategy 1 on scenario 1: (80%-70%)*20 = **200**
- B for strategy 2 on scenario 2 and 3 : (75%-60%)*40+(90%-80%)*25 = **850**
- B for strategy 3 on scenario 4: (90%-60%)*15 = **450**

- Return-on-invest strategy 1: 200/500 = **0,4**
- Return-on-invest strategy 2: 850/1000 = **0,85**
- Return-on-invest strategy 3: 480/900 = **0,53**

Architectural strategy 2 is the best investment.

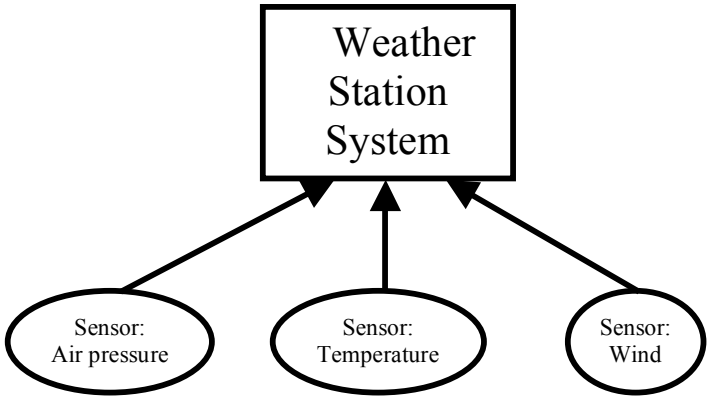
Problem 4 (30 points): Create an architecture

Read the description below and do the following:

- Identify the most important quality attribute(s) for the system described below.
- Identify architectural driver for the system described below.
- Choose and describe suitable architectural tactics for the problem described below, and describe how the tactics affect the quality attributes.
- Create a software architecture for the system described below. The architecture must be described in two views according to the 4+1 view model: Logical view and process view.
- Motivate for your choice of quality attributes, architectural drivers and the architectural tactics used in your architecture.

Weather Station System (WSS)

The weather station system (WSS) is a software system for producing weather data and weather reports gathered from physical weather sensors where the WSS is located. The WSS operates without any human operator, and gets weather data from three sensors measuring temperature, wind, and air pressure as shown in the Figure to the right.



Stakeholders of WSS can be divided into three main groups:

- *Web users* accessing weather data as web pages (HTML) provided by WSS using a web-browser.
- Web sites accessing WSS to receive weather data as XML. These web sites use the weather data to present weather information on their own web pages.
- People walking in the mountains or people in boats accessing the WSS through mobile devices (Personal Data Assistant or mobile phones) or portable PCs using the WAP-protocol on a GSM mobile network. These stakeholders are dependent on weather data from WSS in short intervals, to travel safely in the mountains or at the sea. This information is provided in Wireless Markup-Language (WML).

WSS can provide the following information to other systems (in HTML, XML or WML):

- The temperature, wind speed, and/or air pressure right now.
- The temperature, wind speed, and/or air pressure at a given time.
- Weather report (temperature, wind speed and/or air pressure) for a given time interval (e.g. from 1200-1500 every 30 minutes, 19th of May 2004).

The weather data is also used by the Norwegian Weather Association to create yearly weather reports that are used to compare weather year-by-year, month-by-month, and day-by-day. This data is also used to look at trends in global heating and other weather effects that can be results of environmental changes.

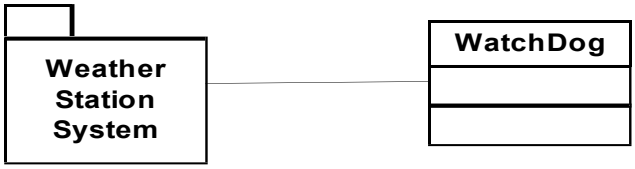
A: There is no standard solution for this problem, but a sketch of a solution will be given.

- **Most important quality attribute(s):** Availability and Modifiability. Availability is important since travellers in the mountains or at sea is dependent on the system to be safe. Modifiability is important since the system provides several interfaces and it is likely that this can be extended in the future. Also it is likely that the functionality can be extended. Also other sensors can possibly be added in the future.
- **Architectural driver(s):** The availability is clearly an architectural driver for this system because this system should be able to operate on its own without human interaction. Also the weather station can be placed somewhere remote which means that availability must be top priority.
- **Choose and describe architectural tactics:**
    - Suitable availability tactics: Ping/echo, heartbeat, exceptions, voting and passive redundancy. Redundant hardware both system hardware and sensors is also useful.
    - Suitable modifiability tactics: Maintain semantic coherence, Anticipate expected changes, hide information.

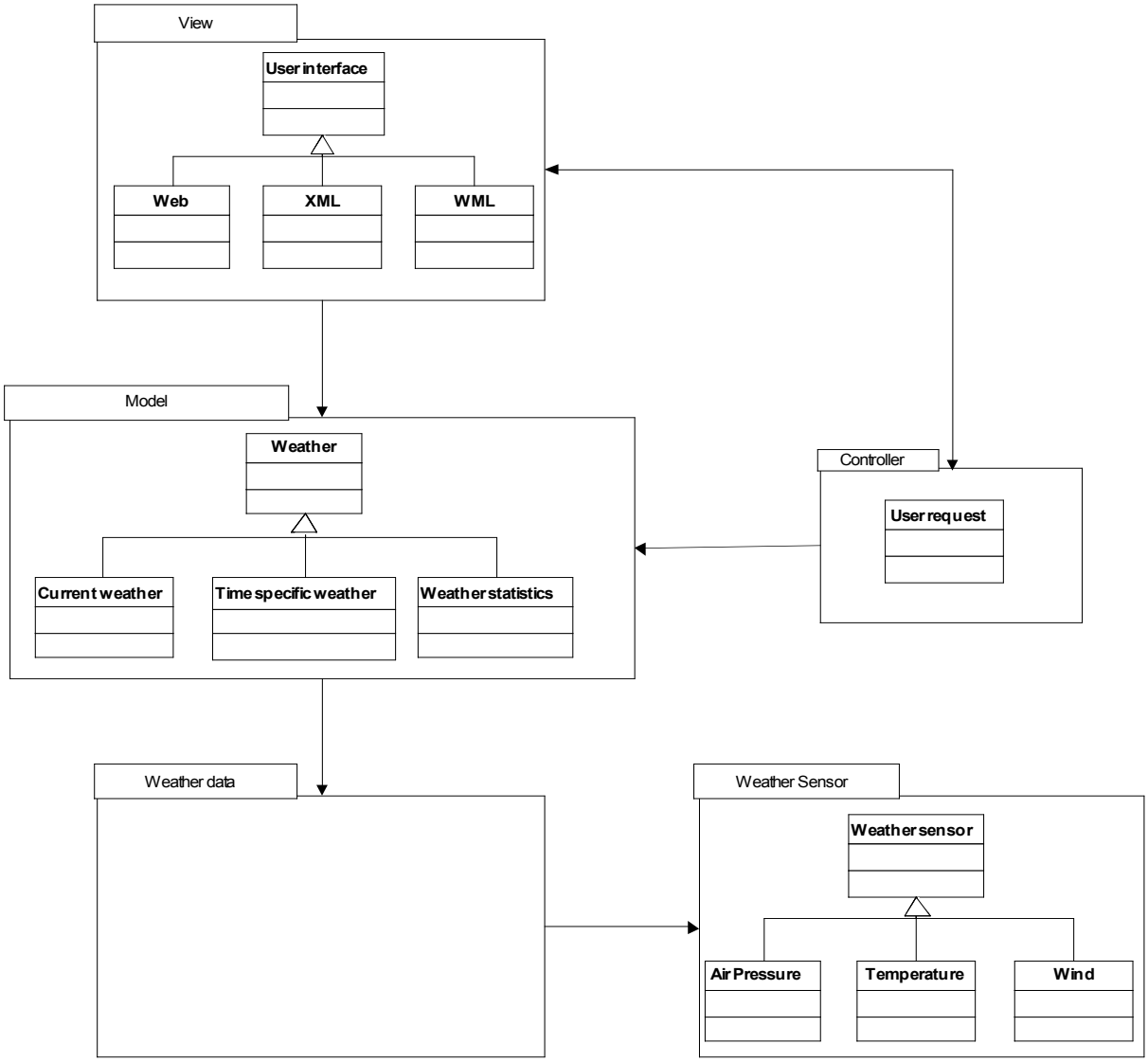
- Suitable usability tactics: Use model view controller to separate different interfaces of the system.

A possible **logical view** of the weather station system:

The logical view is described in two class diagrams. The first diagram shows how the architecture should attack the availability problem by using a watchdog (running on a separate process, preferably on a separate hardware) to ensure that the system is alive. If there is no life signal from the system, the system should be restarted in a consistent state.

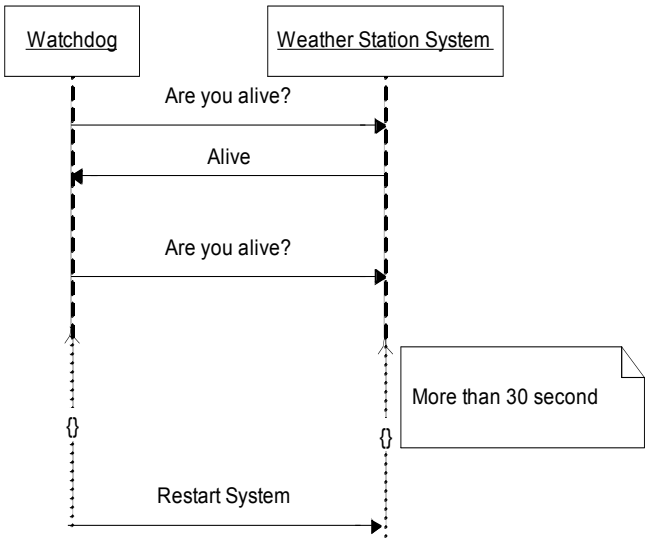


The second class diagram shows how the architecture attacks modifiability and usability:



Process view of the weather station:

The process view is also divided into two parts covering availability and modifiability/usability respectively.
The process view for availability is shown below:



For modifiability and usability the system is divided into four running processes:

- A controller process handling user interaction.
- A model process handing the weather model of the system.
- A sensor data process handling the sensor inputs and storing the sensor information in a database. Because of the tight coupling of the sensors and the database, one process is used for this purpose.
- A view process updating the user view.