NTNU
Det skapende universitet

Department of Computer and Information Science

SOLUTION for TDT4240 Software Architecture Exam

Academic contact during examination: Professor Alf Inge Wang

Phone: +47 9228 9577

Examination date: Friday May 31^{s} 2013

Examination time (from-to): 09:00 – 13:00

Permitted examination support material:

- 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).

Other information:

- Simple calculator or a calculator approved by NTNU allowed.
- The exam has 5 problems giving a total of 70 points. For each problem, each question has the same weight unless otherwise stated. The remaining 30 points are credits awarded from the software architecture project.

Language: English (Answers can be given in Norwegian or English)

Number of pages: 11

Note! Students will find examination results at Studentweb. If you have questions regarding the results, you have to contact your department.

Problem 1: Various questions (20 points)

Answer these questions *briefly*:

1.1	What is Bass, Clements and Kazman's definition of Software Architecture (the definition in the textbook)?
1.2	What is idioms when we discussion terms related to design patterns?
1.3	Describe briefly the three types of view described in the textbook: Allocation, Component-and-Connector, and Module?
1.4	What is the purpose of a utility tree, and how is it used in an ATAM evaluation?
1.5	Which architectural view describes in detail how a module is coded?
1.6	Discuss whether this statement is true or false: "Every software system has a software architecture".
1.7	Name one example of an architectural pattern that fits into the type *module*, one example that fits into the type *component-and-connector*, and one example that fits into the type *allocation*.
1.8	Give five reasons why software architecture is important according to the textbook.
1.9	Briefly explain how the four contexts *Technical*, *Project life cycle*, *Business,* and *Professional* affect the software architecture according to the textbook.
1.10	Write one fully specified quality attribute scenario for availability, using the template given in the textbook.
1.11	Describe the main types of availability tactics in the textbook, and give at least one specific example of an architectural tactic for each type.
1.12	Briefly explain the quality attribute interoperability.
1.13	Describe and explain the design pattern Abstract Factory using a class diagram.
1.14	What parameters (inputs) are typically used in models for analyzing performance?
1.15	What is an architectural pattern?
1.16	What analytic models are typically used for modeling availability?
1.17	What is a tradeoff point in an ATAM evaluation?
1.18	What is the challenge with a software product line where the scope is too narrow?
1.19	Briefly describe the steps in the Attribute-driven design method.
1.20	What techniques exist to help keep the code and the architecture consistent according to the textbook?

References:

- Bass, Clements & Kazman: "Software Architecture in Practice"
- Coplien: "Software Design Patterns: Common Questions and Answers"

Solution Problem 1:

1.1 The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

1.2 Low-level pattern specific to a programming language.

1.3 *Allocation* structures describe how the system will relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.).
Component-and-connector structures describe how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).
Module structures describe how the system is to be structured as a set of code or data units that have to be constructed or procured.

1.4 The utility tree is a way of organizing quality requirements into different quality attributes as well as it shows difficulty in achieving requirements and the importance of the quality attributes.

1.5 None.

1.6 True, as every system can be shown to consist of elements and relations among them to support some type of reasoning, even though the system might not have software architectural documentation.

1.7 *Module*: Layered pattern.
Component-and-connector: Broker, Model-view-controller, Pipe-and-Filter, Client-Server, Peer-to-Peer, Service-Oriented architectural, Publish-Subscribe, or Shared-Data pattern.
Allocation: Map-reduce or Multi-tier pattern

1.8 Candidates: Enabling a system's quality attributes, Reasoning and managing change, Predicting system qualities, Enhancing communication among stakeholders, Carrying early design decisions, Defining constraints on an implementation, Influencing the organizational structure, Enabling evolutionary prototyping, Improving cost and schedule estimates, Supporting transferable, reusable model, allowing incorporation of independent components, Restricting the vocabulary of design alternatives, and Providing a basis for training.

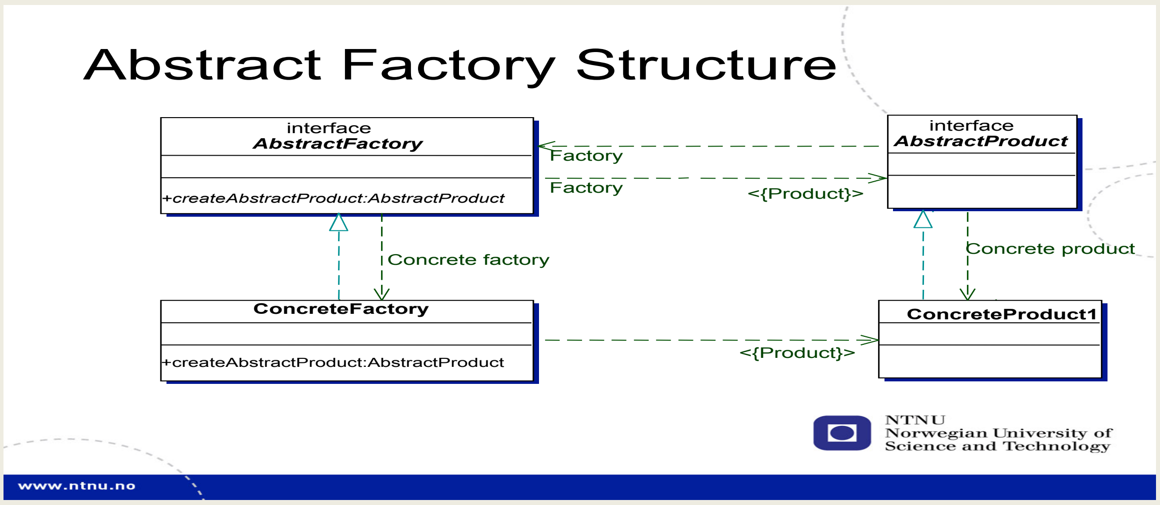
1.9 How contexts affect the software architecture:
* Technical context: Set of quality attributes highly affect the software architecture, but the architecture is affected also by current technical environment, and the technology and framework used such as web, object-oriented, service-oriented, mobility-aware, cloud-based, social-network friendly.
* Project-life cycle context: The architecture will limit or dictate what kind of life-cycle model that can be used and vice-versa (e.g. incremental development vs. waterfall vs. scrum etc.)
* Business context: The business goals will highly affect the software architecture and vice-versa, such as time-to-market, roll-out schedule, changing business environment, time-to-launch etc.
* Professional context: Software architecture is highly affected by knowledge and skills of the architects such as managerial skills, communication skills, knowledge of patterns, frameworks etc.

1.10 Quality attribute scenario for availability
Source: Hardware
Stimulus: Failure
Artifact: Hard disk on server
Environment: Normal operation
Response: Backup-server takes over without any lost data
Response measure: Back to normal operation within 30 seconds

1.11 Availability tactics (more than one example is shown):
* Detect Faults: Ping/Echo, Monitor, timestamp, Voting, Self Test etc.
* Preparation and Repair: Active Redundancy, Passive Redundancy, Spare, Exception Handling etc.
* Reintroduction: Shadow, State Resynchronization, Escalating Restart, Non-Stop Forwarding.
* Prevent Faults: Removal from Service, Transactions, Predictive model, etc.

1.12 Interoperability: The degree to which two ore more systems can usefully exchange meaningful information.

1.13 Abstract factory: A family of related classes can have different implementation details and through the abstract interface the client does not need to know anything about which variant it is using.


Abstract Factory Structure
interface AbstractFactory
+createAbstractProduct:AbstractProduct
interface AbstractProduct
Factory
Factory
<{Product}>
Concrete factory
Concrete product
ConcreteFactory
+createAbstractProduct:AbstractProduct
ConcreteProduct1
<{Product}>
NTNU Norwegian University of Science and Technology
www.ntnu.no

1.14 Parameters (inputs) for analytic modeling of performance (candidates): Arrival rate of events, queuing discipline, scheduling algorithm, service time for events, network topology, network bandwidth, routing algorithm etc.

1.15 Architectural pattern (candidate answers):
* Package of design decisions that is found repeatedly in practice
* Reusable structures that can be applied on construction of systems
* Defines the element types and their forms of interaction used in solving the problem.

1.16 Analytic models for availability: Markov models and statistical models.

1.17 Trade-off point in ATAM: An architectural decision that will affect two or more quality attributes at the same time (both positive and negative effects)

1.18 Narrow scope in software product line: Too few variants of the product can be produced.

1.19 Attribute-driven design process:
1. Choose the module to decompose
2 Refine the module:
   2a) Choose architectural drivers
   2b) Choose/create architectural pattern satisfying the drivers
   2c) Instantiate modules and allocate functionality
   2d) Define interfaces of the child modules
   2e) Verify and refine use cases and quality scenarios
3 Repeat steps above for every module requiring further decomposition

1.20 Keep code and architecture consistent:
* Embedding the design in the code
* Frameworks
* Code templates
* Keeping code and architecture consistent by using tools, rules, mark out of date documents and schedule doc/code synchronization

Problem 2: Choose the most appropriate architectural pattern (5 points)

Nominees:

a) Layered
b) Broker
c) Model-view-controller
d) Pipe-and-Filter
e) Client-Server
f) Peer-to-peer
g) Service-Oriented
h) Publish-Subscribe
i) Map-Reduce
j) Multi-tier

Choose the *most appropriate architectural pattern* (one) for the 5 descriptions below. Motivate for your choices (give reasons for choosing the pattern):

1. Wants to split a system into a number of computationally independent execution structures (groups of software and hardware) such as database, business logic, web interface and client, connected by some communication media. The structure is chosen to provide a specific server environment optimized for operational requirements and resource usage.
2. Wants to set up a set of equal distributed computational entities that are connected via a common protocol to share their services and provide high availability and scalability.
3. Wants a system that can be divided into reusable, loosely coupled components that can be flexibly combined and arranged to transform between various data formats.
4. Wants a distributed system with a structure that enables that service users do not need to know the nature or location of service providers.
5. Wants a system that quickly can analyze enormous volumes of data by sorting the data and then analyzing the grouped data.

Problem 3: Edge-dominant system (8 points)

a) What is an edge-dominant system and what characteristics do edge-dominant systems have? (2 points)
b) Draw and explain the metropolis structure of an edge-dominant system (3 points)
c) What are the implications of edge-dominant systems for the software architecture? (3 points)

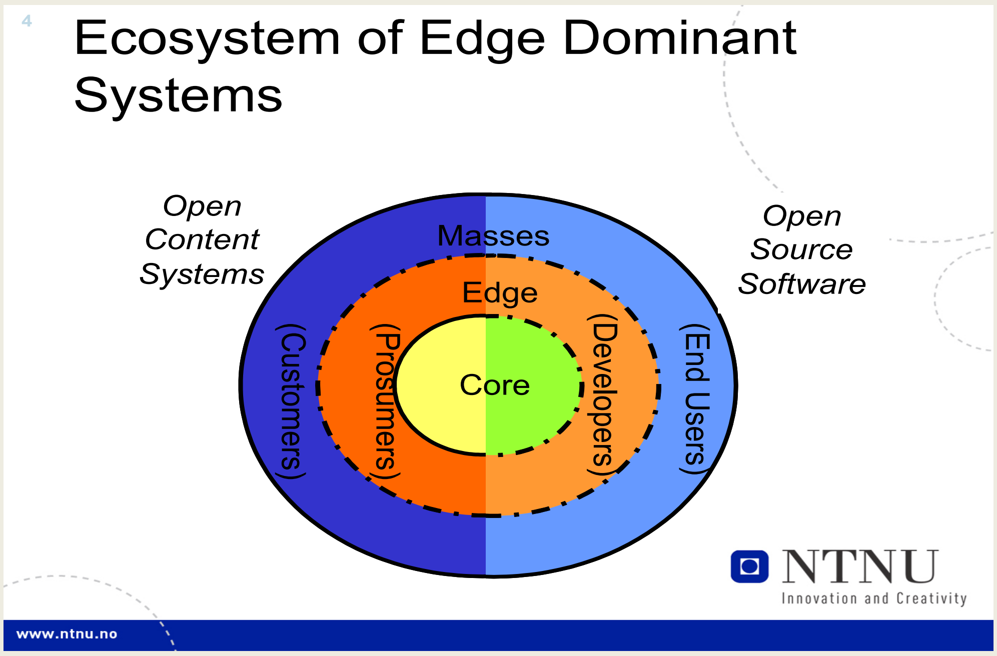
Problem 4: Cloud Architecture (7 points)

a) Explain the three Cloud Service Models described in the textbook: 1) Software as a Service, 2) Platform as a Service, and 3) Infrastructure as a Service (3 points)
b) What are the economic benefits of using a cloud based architecture? (2 points)
c) What is the purpose of the hypervisor in a cloud-based architecture? (2 points)

Solution Problem 2
2.1 Multi-tier (j)
2.2 Peer-to-peer (f)
2.3 Pipe-and-filter (d)
2.4 Broker (b)
2.5 Map-Reduce (i)

Solution Problem 3
3a) An edge-dominant system is one that depends on input from users for its success. Characteristics can be built-in social media functionality, easy to add functionality and services, parts or the whole system might be open-source, core software developed by small group of developers, many users & prosumers.

3b) Metropolis structure:



The whole system is based on a stable and high-quality core developed by a dedicated small team. In the Edge layer, prosumers and developers will add content and services to the system based on the APIs from the core. This layer is essential for the survival of the system. The Masses layer is the layer of many users that use the services and the content created in the Edge layer.
There is also movement between the Masses layer and the Edge layer, but not to the Core.

3c) Implications for the software architecture:
It is very important that the Core is developed by a small excellent team, which will produce a core which is highly modular and layered, highly robust with respect to errors, and can produce high-performance and scalability. Good documentation for the service APIs must be available. Another implication is that the requirements come from web-forums and the like. The service layer must be very flexible and at the same time easy to test and product from harmful activities.

Solution Problem 4
4a) Cloud Service Modes:
- Software as a Service (SaaS): Consumer uses applications running on a cloud.
- Platform as a Service (PaaS): Provides a development and deployment platform in the cloud.
- Infrastructure as a Service (IaaS): Provides a virtual machine for developer or system administrator.

4b) Economical benefits of using cloud-based architecture:
Large data centers are cheaper (cost of power, infrastructure labor cost), security and reliably, hardware cost)
Utilization of equipment (virtualization, adapt to time of day, year, usage patterns)
Multi-tenancy: Single application used by multiple consumers saves costs related to user-support, simple upgrade, single version of software.

4c) Purpose of the hypervisor:
Operating system to create and manage virtual machines. Manage a pool of hardware to provide an even larger pool of virtual machines through managing memory, processes, file access and network.

Problem 5 Design a software architecture (30 points)

Read the description below and do an architectural design. Your answer must include:

a) Architectural drivers – 2 points
b) Architectural tactics and patterns – 3 points
c) Process view – 8 points
d) Logical view – 14 points
e) Architectural rationale – 3 points

Motivate for your choices and state your assumptions.

Software for sea buoys support for navigation at sea

There exists a collection of free-floating buoys that provide navigation and weather data to air and ship traffic at sea. The buoys collect air and water temperature, wind speed, and location data through a variety of sensors. Each buoy may have a different number of wind and temperature sensors and may be modified to support other types of sensors in the future. Each buoy is also equipped with a radio transmitter (to broadcast weather and location information as well as an SOS message) and a radio receiver (to receive requests from passing vessels. Some buoys are equipped with a red light, which may be activated by a passing vessel during sea-search operations. If a sailor is able to reach the buoy, he or she may flip a switch on the side of the buoy to initiate an SOS broadcast. Software for each buoy must:

- maintain current wind, temperature, and location information; wind speed readings are taken every 30 seconds, temperature readings every 10 seconds and location every 10 seconds; wind and temperature values are kept as a running average.
- broadcast current wind, temperature, and location information every 60 seconds.
- broadcast wind, temperature, and location information from the past 24 hours in response to requests from passing vessels; this takes priority over the periodic broadcast
- activate or deactivate the red light based upon a request from a passing vessel.
- continuously broadcast an SOS signal after a sailor engages the emergency switch; this signal takes priority over all other broadcasts and continues until reset by a passing vessel.

Solution Problem 5

Design Considerations:

The problem statement defines a set of separate functions with relatively little in common. They share the communications equipment and a number of current sensor readings.

The software architecture must permit the integration of these loosely coupled functions (requirement R1).

At the same time, it must respect their priorities and timing constraints (R2).

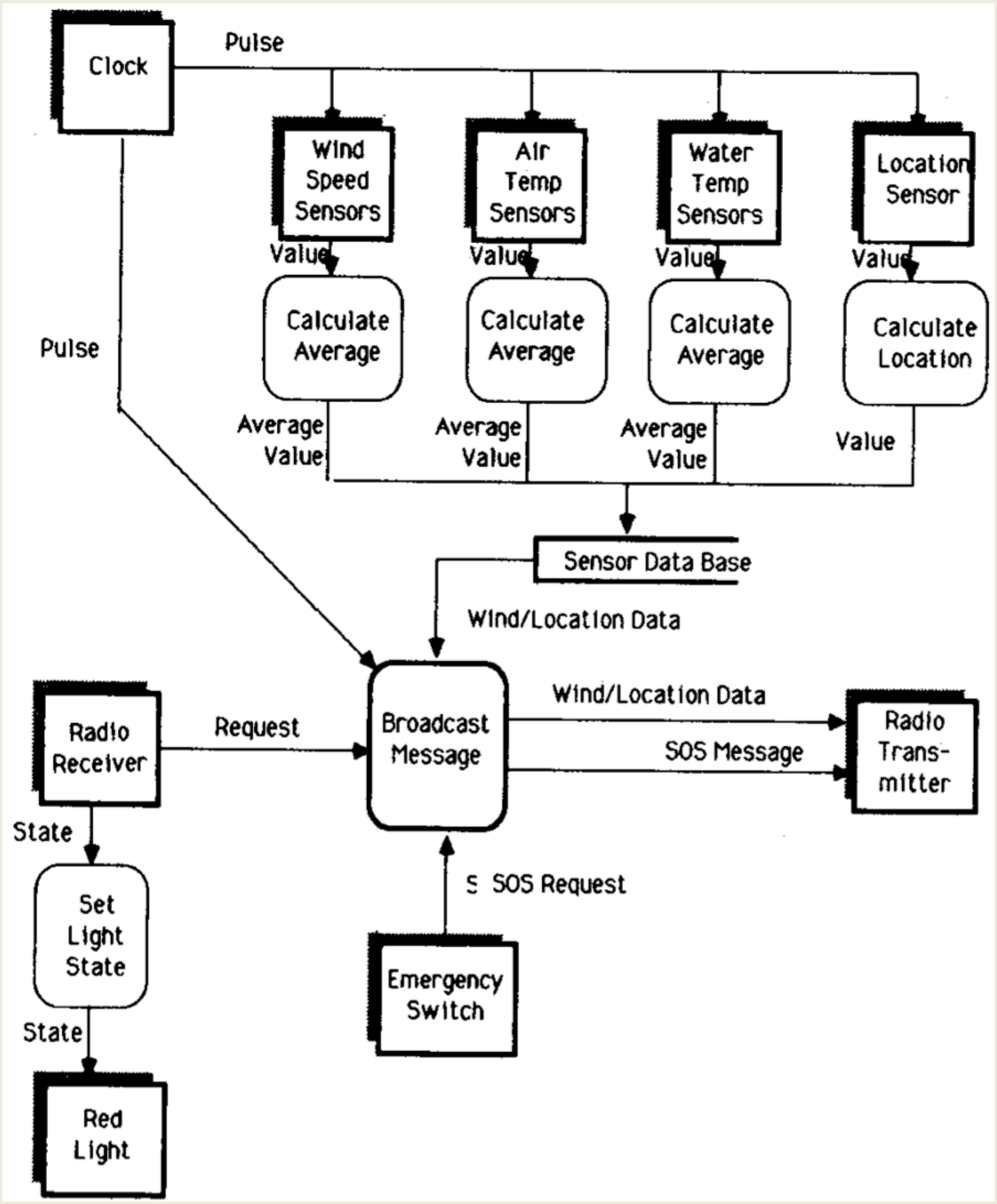
Clearly the system may be extended further by additional functions (e.g., more sensors) or that the priorities and timing constraints may be modified. The architecture should therefore allow modifications to the overall system parameters (R3).

Finally, sea buoys must operate for long periods without maintenance, and they are numerous enough for cost to be a major consideration. As a result, the architecture should provide hints for its implementation on the most basic platform (R4).

5a) Architectural drivers:
Important quality attributes are
* Availability: Operating on its own out in the sea
* Performance: Timing requirements
* Modifiability: Work with various sensors, might add new sensors in future with additional new functionality.

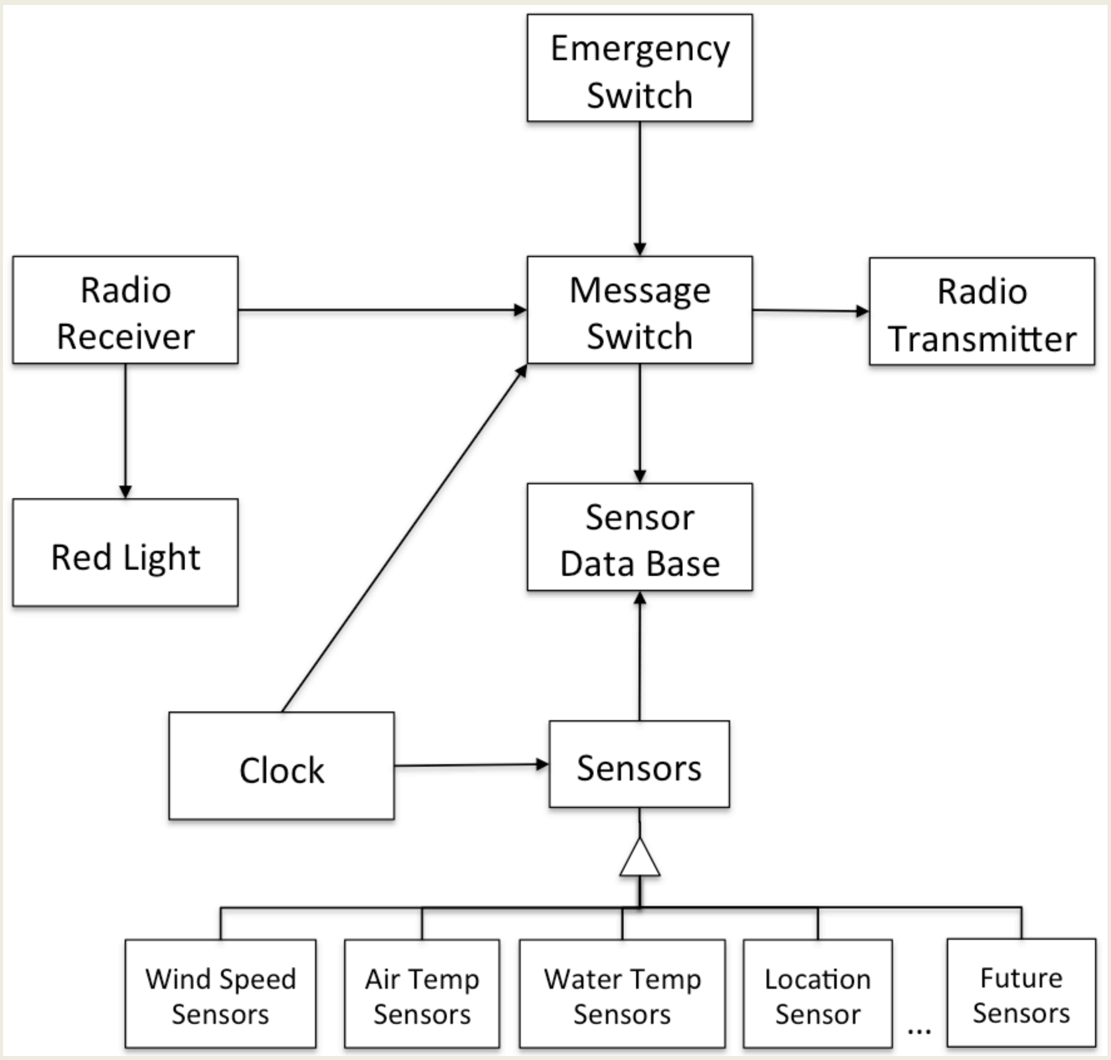
5b) Architectural tactics and patterns
The architecture is based on the *Task Control Architectural patterns*, which basically is a way of communicating with various tasks by sending messages to a central server (named Message Switch in the architecture).
Various tactics are used to achieve quality in various areas:
* *Modifiability*:
  - Generalized interface for sensors: Makes it easier to add and change sensors
  - Increase semantic coherence: Clear separation of concerns and all classes has distinct and not overlapping responsibilities
  - Encapsulation: Interfaces only provides API to methods accessed by other classes. Rest remain private.
* *Availability*:
  - Heartbeat: The Clock of the system is provided by separate hardware and can be used as a heartbeat and force restart if system does not react within time.
  - Self-test: System will do a self-test when starting up.
  - Exception Handling: Catch exceptions in the system, which can cause a restart of the system or broadcasting error if unfixable error.
  - Simplicity: Keep the system simple to make it easier to test and find errors.
* *Performance*:
  - Reduce overhead: Keep the code simple
  - Schedule resources: Use the Clock to trigger events at the right time. Schedule so that sensors are read in sequence and not at the same time.

5c) Process view (there are several possible process view, but the view must show how the system is operating while running and how the different parts of the system interact).



This process view shows an activity diagram with events and shows how the different parts of the system interact and that they all are timed by a clock, which outputs a pulse. The clock synchronizes all sensors as well as the broadcasting of messages. An SOS request triggered by the Emergency Switch will cause SOS Message send by the Radio Transmitter.

5d) Logical view



5e) Architectural rationale

The architecture has focused on modifiability, performance and availability. The Clock provides both performance by scheduling when various tasks are performed, and is used as a heartbeat to check that the system is up and running (external hardware unit). Modifiability is provided through the Task Control Architecture pattern, which makes it easy to extend and change the design, clear separation of concerns between various parts, and abstraction for common services for sensors, which makes it easy to replace sensors as well as adding new sensors in the future.