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: 28. June

Exam in the subject
TDT4240 Software Architecture

Wednesday 7. June 2006
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 5 problems giving a total of 70 points. The remaining 30 points are credits awarded from the robot project.

Good Luck!

Magne Syrstad and Finn Olav Bjørnson

Controlled 29th of May 2006

Problem 1: Various questions (10 points)

Answer these questions short:

1.1 What is Bass, Clements and Kazman's definition of Software Architecture (the definition in the textbook)?

Def: A software architecture is the structure or structures of a system consisting of software components, their external visible properties and the relationship between them.

1.2 How can information be extracted when you are reconstructing a software architecture?

Information can be extracted by reading requirement and design documentation, talking to developers and architects, reading code, using tools to generate call-graphs from code, looking at file structure of source code, analyzing run-time system, analyzing inputs and outputs from the system etc.

1.3 What is architectural drift?

Architectural drift is when the implementation drifts/moves away from the initial architecture (documented). This means that the implementation does not reflect the architecture anymore.

1.4 What is an architectural pattern and give one example of an architectural pattern?

A description of element and relation types together with at set of constraints on how they can be used. Client server is one example of an architectural pattern.

1.5 What of the following quality area(s) is/are related to scalability: Availability, modifiability, performance, security, testability, and/or usability?

Scalability is mainly related to performance, as a scalable system is possible to e.g. handle more simultaneous user without getting long response times. In addition, scalability is partly related to modifiability, as a scalable system is modifiable in terms of performance.

1.6 Why is an enterprise architecture important for big companies like Telenor?

An enterprise architecture describes how all the important systems for a big company plays together. An enterprise architecture is important for big companies because it makes it possible to know how all the various systems affect each others, how it is possible to extend the portfolio provided, make risk and effort analysis before providing new services, provide the correct organization structure to support all systems, make it possible to track failures caused by interplay of several systems etc.

1.7 Where does the architectural design fit into the waterfall process model for software development?

The architectural design should be started after the requirement phase is completed and before the detailed design is started: Thus after requirement phase and before or part of the initial stage of the design phase.

1.8 Why is it useful to describe an architecture through different viewpoints and views?

By using different views and viewpoints, it is possible to concentrate on specific aspects of the system one at a time. Most systems are too complex to be captured in only one view, and by using multiple views, you get a less complex vision of the system. In addition, the different views address different concerns of the various stakeholders of the system.

1.9 How can interface mismatch of using Off-The-Shelves components in a software architecture be repaired?

Interface mismatch can be repaired by using wrappers, bridges or mediators. These means makes it possible to provide standard interfaces, to bridge the gap between interfaces and to remove parts that should not be provided in the system.

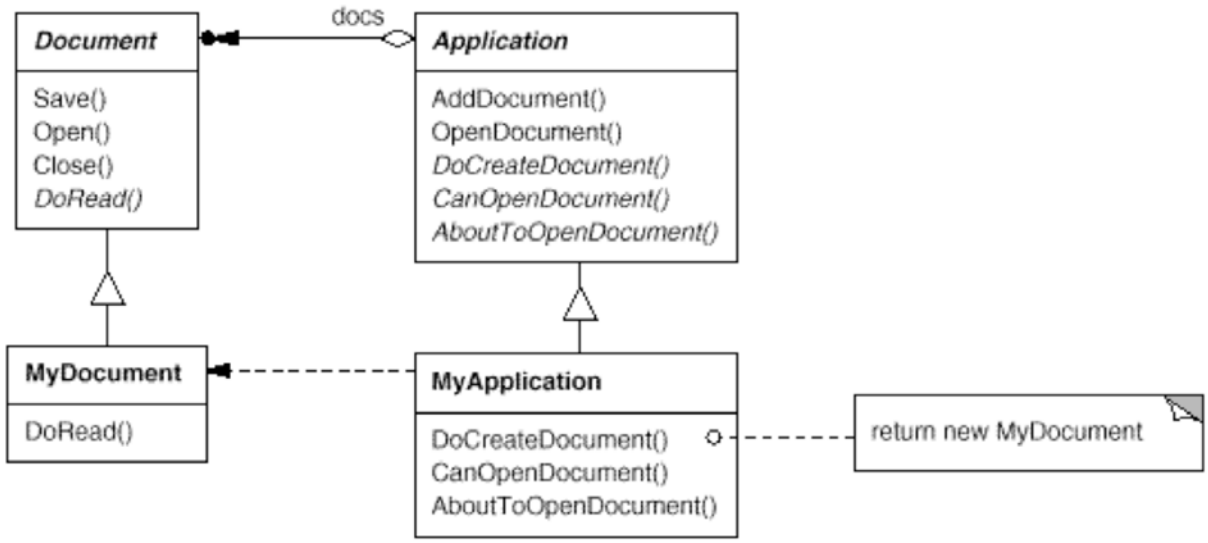
1.10 How should you design your architecture to support variation points in a product line?

Variation points in a product line can be supported in a software architecture by several approaches. One approach is to provide sub-classing, where you have a general class covering the main functionality, and sub-classes for each specific product. Another approach is to use virtual machines that provide the same interface into the system, but can support various interfaces outside the system. Other possible options here can be to use design patterns like template method, abstract-factory, etc.

Problem 2: Design patterns (10 points)

2.1 Describe design pattern (5 points)

What is the name of the design pattern shown below and what is the purpose of this design pattern?



This design pattern is called *template method* and is used for cases where the algorithm or behaviour has a stable core and there are several variations of the core. Typically you define the core functionality in the template, and you can have several variations that override some of the functions in the template.

2.2 Description of design patterns (5 points)

Name five items (parts) that should be in a description of a design pattern and describe why these items (parts) are important in a design pattern description.

A description of a design pattern should consist of these elements (prioritized):

- *Pattern name and classification*: What type of design pattern, e.g. behaviour etc.?
- *Intent*: What is the purpose of the design pattern, what problem does it solve?
- *Applicability/Context*: Where or for what purposes can the design pattern be used?
- *Implementation*: How is the design pattern implemented?
- *Sample code*: Examples of how the design pattern can be used.
- *Structure*: How is the design pattern structured?
- *Consequences:* What can the consequences be by using the design pattern?
- *Also known as:* Other names for the design pattern.
- *Motivation:* What is the background/motivation for the design pattern?
- *Collaborations:* What other design patterns can be used in collaboration with current design pattern?
- *Participants:* What roles are involved when using the design pattern?

Problem 3: Quality Scenarios (10 points)

Create one quality scenario for availability and *one quality scenario for performance* for the system described below. The scenarios should be fully specified according to the textbook, you can describe the scenario as a diagram or as a table, and the scenarios should be useful and understandable for the stakeholder *user* (player).

SmashYourFriends ™ is a multiplayer game where up to 5000 players (on the same server) can play against each other using their personal computer as clients. The player can choose between different vehicles like tanks, trucks, jeeps, helicopters and planes, and the goal is to shoot all the other players and to be the last survivor. All the players connected to the same server will play in the same play area. Different servers will provide different play areas like desert, arctic, city, mountain, moon, etc. An illustration of the game is shown below.

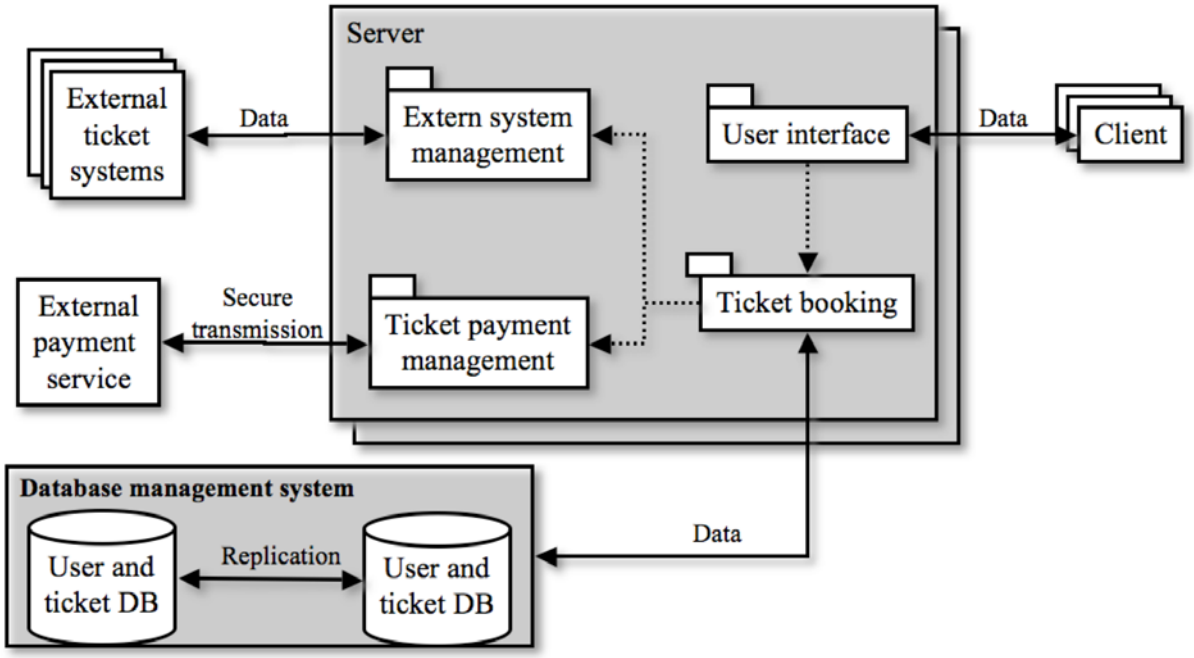
Availability scenario #A1: Downtime		**Performance scenario #P1: Latency**	
Source:	Internal	Source:	End users
Stimulus:	Fault	Stimulus:	Press a key on the keyboard
Artifact:	Process	Artifact:	Game server
Environment:	Normal operation	Environment:	Under normal operation
Response:	Continue normal operation	Response:	Change play area according to action
Response measure:	Less than 5 minutes downtime in a week.	Response measure:	With average latency of 10 milliseconds.

Problem 4: ATAM (10 points)

Read the description below (including the figures of the architecture and the utility tree) and *carry out an analysis of the architectural approaches for the two most important quality scenarios* (step 6 in the ATAM process).

The company MacroSoft ™ wants to develop a booking system called GetTick. This system makes it possible for users to order/buy tickets over the web using GetTick. The users should be able to buy tickets from various sources like cinemas, theatres, sports arenas, and concert halls. The tickets are paid in the system using credit cards.

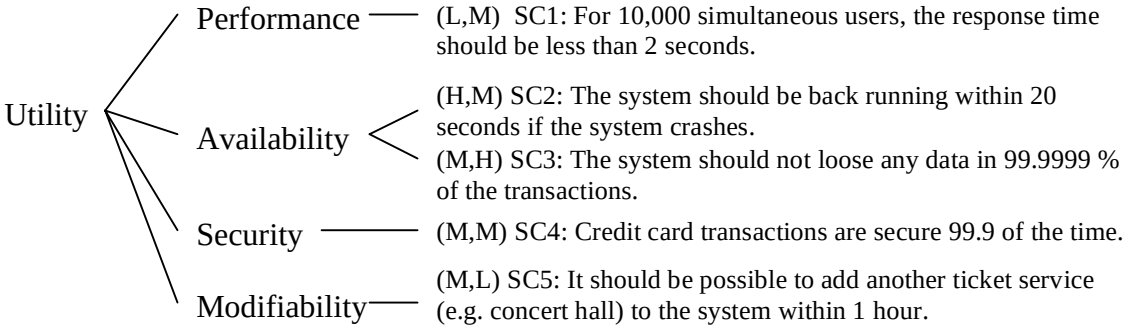
The figure below shows a logical & deployment view of the architecture of GetTick.



The following architectural tactics are a part of the architectural plan:

- AT1: Provide general interface for all external system communication.
- AT2: Use model-view controller pattern.
- AT3: Passive redundancy (warm restart).
- AT4: Replication of the database.
- AT5: Heartbeat (between primary and secondary server).
- AT6: Authenticate users.
- AT7: Data encryption.

The utility tree for the GetTick system:



Scenario SC2: The system should be back running within 20 seconds if the system crashes.				
Architectural decisions:	**Sensitivity**	**Tradeoff**	**Risk**	**Nonrisk**
AT3: Passive redundancy (warm restart)		T1		N1
AT4: Replication of database		T2	R1	
AT5: Heartbeat	S1		R2	
Reasoning:	The most important tactic here is AT3 and AT5. AT3, AT4 and AT5 are positive for availability for the system. AT3 and AT4 could have negative effects on the performance (especially AT4). A possible problem with the architecture is that the responsible for the heartbeat mechanism is not delegated to an independent part or one of the servers.			

Scenario SC3: The system should not loose any data in 99.9999% of the transactions.				
Architectural decisions:	**Sensitivity**	**Tradeoff**	**Risk**	**Nonrisk**
AT4: Replication of database		T2	R1	
Reasoning:	Only AT4 is relevant for this scenario to ensure that data is not lost. AT4 could have a negative effect on the performance. It is important that the system have mechanisms to store transactions in the databases immediately after the transaction has been completed.			

Sensitivity/ Tradeoff/ Risk/ Nonrisk	
S1	AT5 is positive for availability without other negative effects.
T1	AT3 is positive for availability, but could have negative effect on performance (state information must be exchanged between servers).
T2	AT4 is positive for availability but could be negative for performance, as all data must be written simultaneously on two databases.
R1	Depending on how the replication is implemented, it could be a problem for performance.
R2	The architectural description does not say anything about who is in charge for the heartbeat mechanism.
N1	AT3 could have some minor drawback on performance, but this effect is minimal.

Risk themes: The architecture should state who is in charge of the heartbeat mechanism. It is also unclear how replication is implemented that could be a problem in terms of performance. Finally, the architecture does not say anything about transactions or rollback mechanisms.

Evaluation: There are several issues that need to be clarified in order to know whether the two scenarios are supported by the architecture. The main problem is with the SC3 scenario.

Problem 5 Create an architecture (30 points)

Read the description below and do the following:

5.1 Identify the most important quality attribute(s) and the architectural drivers for the system described below (5 points)

5.2 Choose and describe suitable architectural tactics for the problem described below, and describe how the tactics affect the quality attributes (5 points)

5.3 Create architecture views of the system described below. The architecture must be described in two views according to the 4+1 view model: *Process and Logical view* (20 points)

Motivate for your choice of quality attributes, architectural drivers and the architectural tactics used in your architecture.

Software for House Alarm System BigBrother

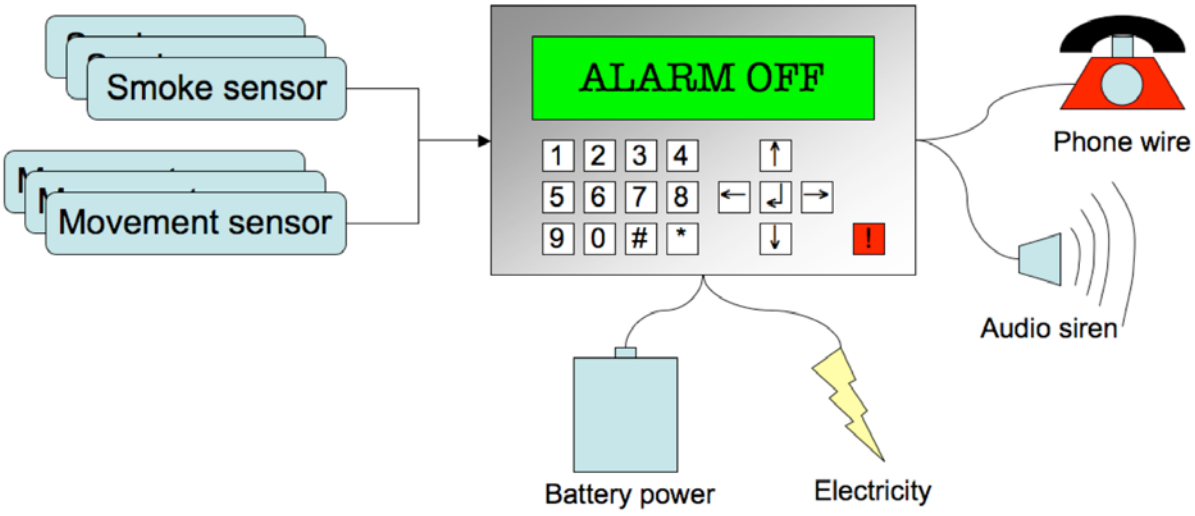
The software described here is software for controlling an alarm system called BigBrother sold to households. The software should be able to run different configurations consisting of sensors from various producers, variations in types of displays and keyboard/button configurations.

The different configurations also represent different price segments from the very simple and cheap alarm systems to the expensive and advanced. The BigBrother software system is supports both smoke (fire) and movement sensors (theft).

In normal mode, the system is running on electrical power from a standard power socket in the wall. However, in case a power outage, the system can operate on battery power. All the sensors are powered by the BigBrother system. In case of a detection of fire or theft, the BigBrother system will start a siren (alarm sound) and the display information about what caused the alarm, in what area of the house. How the information is shown is dependent on the capabilities of the display used in the system from only simple text to graphical description of the situation.

For the more expensive configurations, the system can call the fire department or a security company through a telephone connection. The system can also be set up to call the mobile phone of the owner of the house. The system will also warn the security company if the alarm system is running on battery.

The software of BigBrother is running on custom made computer with a CPU, memory and various input/output interfaces. A physical illustration of the system is shown below.



5.1 Most important quality attribute(s) and architectural drivers for the system:
This system is a product line system with many variations of configurations. In addition, such a system must be reliable as it concerns safety of the users. The two most important quality attributes for this system are availability and modifiability.
Architectural drivers for the system:

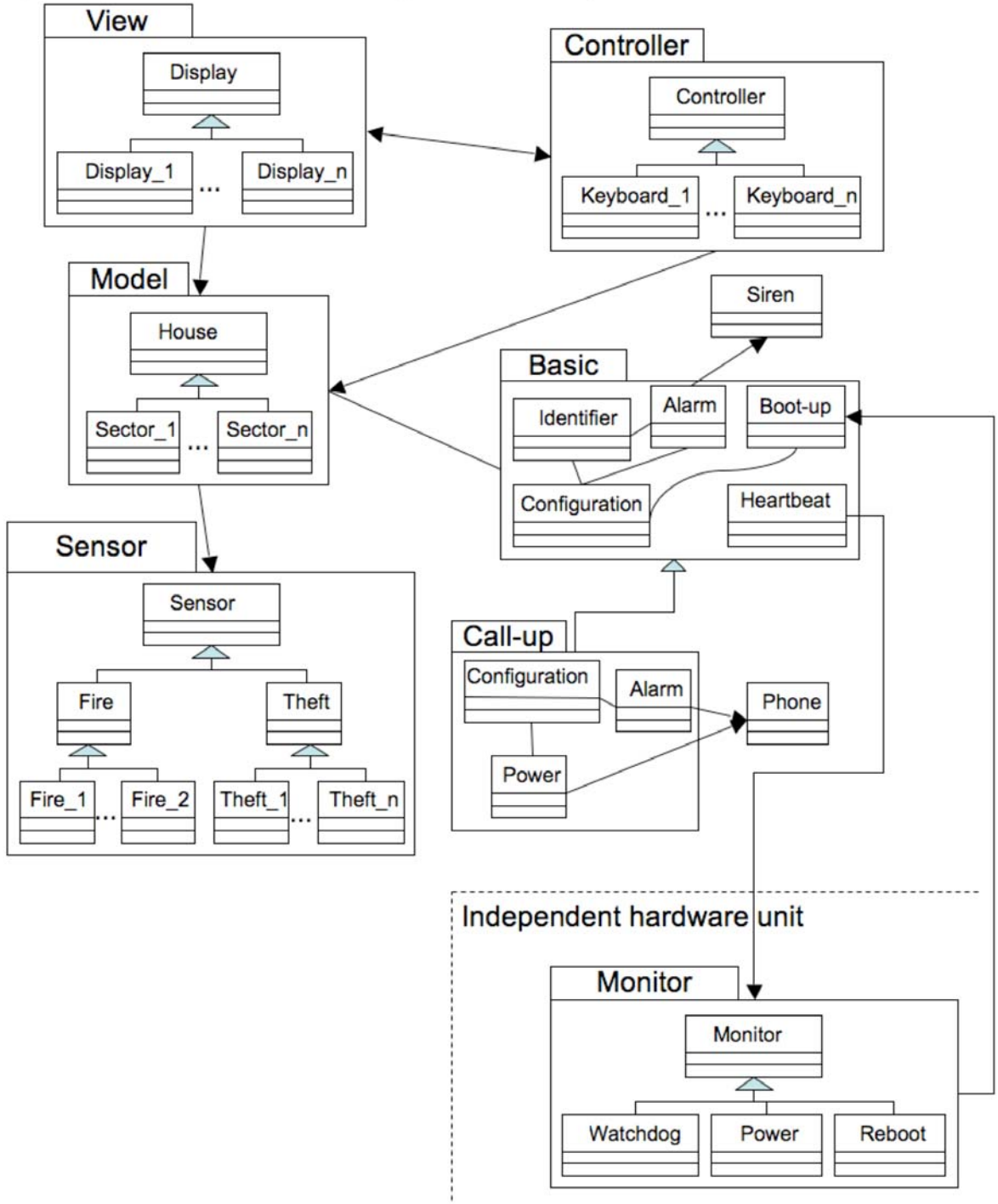
- The architecture must provide high availability because the system can possibly save lives or keep people out of danger.
- The architecture must provide interfaces that can handle various types of sensors.
- The architecture must be able to support various types of displays and keyboard/button configurations.
- The architecture must be flexible in such way that it provides different types of functionality based on the price segment of the product.
- The architecture must be able to automatically switch to battery-operated mode in case of a power outage.

5.2 Choose and describe suitable architectural tactics and describe how the tactics affect the quality attributes:
This solution will focus on the most important architectural tactics to achieve improved availability and modifiability:

- **Heartbeat** (availability)**:** The system should provide a heartbeat functionality to ensure that the system is running and alive (availability). The system should consist of a separate hardware unit (monitor) that has a battery backup hardware so it will never die in case of power outage. The main system will send a heartbeat signal in fixed time intervals (e.g. every second) to the separate heartbeat listener unit. If this unit does not get any heartbeat signal within some time (e.g. 5 seconds), the main unit will be restarted. If this fails, the main system will switch to battery power and restarted.
- **State resynchronization** (availability)**:** The heartbeat signal sent from main unit to the monitor unit consists of the state information of the main unit. This makes it possible for the monitor unit to reinitiate the main unit in previous state.
- **Exceptions** (availability)**:** Exceptions can be used to detect faults in the main unit and should be sent to the monitor unit. The monitor unit can restart the main unit, if the main unit is in an unstable state.
- **Use model-view controller** (modifiability): By using the model-view controller approach, it is possible to provide support for various displays and keyboard/buttons configuration in the same architecture.
- **Use an intermediary** (modifiability): To support the variation in sensors, intermediaries can be used to represent the different sensors to provide the same interface for different sensors into the system. Here it is possible to use the template method design pattern, where the core functionality of the sensors are provided in a template class and where sub-classes represents the different types of sensors.
- **Separate core functionality and advanced functionality** (modifiability): Since the functionality provided by the system can vary depending on the price segment of the system, the core functionality of all systems should be separated from the more advanced functionality provided by the more advanced systems.

5.4 Create logical and process view:

Logical view of the architecture of BigBrother Alarm System:



The logical view is based on the model-view controller pattern, where the *View* module provides variations for different types of displays and the *Controller* module provides support for different types of keyboards/buttons. The *Model* module represents the state of the house in terms of sectors. A separate module called *Sensor* provides support for various sensors of fire and theft sensors. The basic functionality is provided in the *Basic* module that consists of management of *Alarm*, *Configuration* of the alarm system, an *Identifier* part that takes care of user identification when turning off the alarm system and a *Heartbeat* sending status information to the hardware independent monitor unit. The *Call-up* module provides more

advanced functionality for calling security, fire department, or the house owner in case of an emergency. The module also provides functionality to call security if the system runs on battery power (power outlet). The *Call-up* module is only provided in the more expensive alarm systems. The *Monitor* module (running on separate hardware) is responsible for collecting heartbeat signals from the main alarm system and monitoring possible power outlets. This module monitors the state of the main unit (*Watchdog*) and can reboot the main system in case of a failure (software or power).

Process view of the architecture of BigBrother Alarm System:

Heartbeat sekvensdiagram: