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: 1. September

Continuation Examination in the subject TDT4240 Software Architecture

Saturday 9. Aug 2008
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:

Not available

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!

Problem 1: Various questions (15 points)

Answer these questions in 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 What is the purpose of the ATAM architecture evaluation?

The purpose of the ATAM is to evaluate how a software architecture addresses the most important quality scenarios (requirements) of a system.

1.3 What is an architectural pattern (style)?

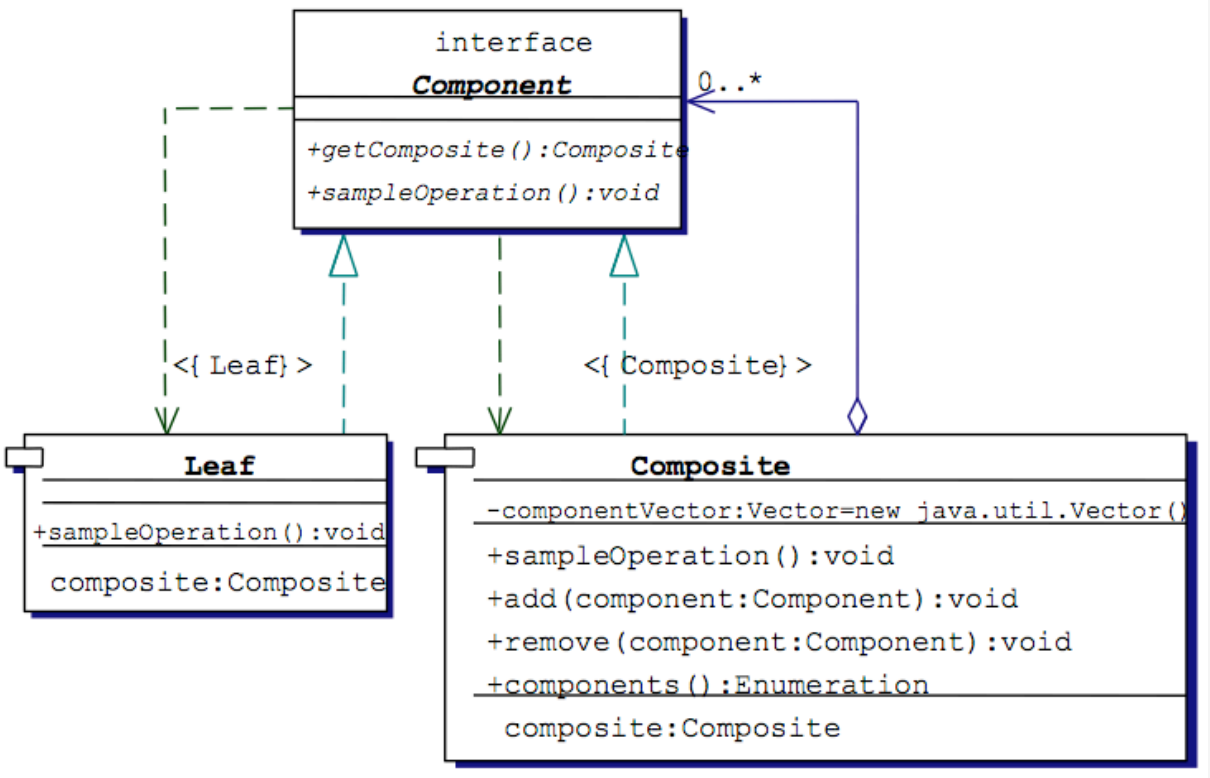
An architectural pattern (style) expresses a fundamental structural organization or scheme for software system and provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing for relationships between them.

1.4 What is architectural erosion?

Architectural erosion is that the implementation violates the architecture.

1.5 Describe the *Composite* design pattern and advantages of using it.

The Composite pattern is used to obtain a transparent treatment of single and multiple objects that simplifies the client code and allows arbitrary grouping of elements and uniform treatment of groups and primitives.



1.6 Describe the CBAM process (9 steps)

The CBAM process:
1. Collate scenarios (prioritize top 1/3)
2. Refine scenarios (worst, best, current and desired)
3. Prioritize scenarios (eliminate)
4. Assign utility for current and desired levels
5. Develop architectural strategies for scenarios and determine quality attribute response levels
6. Determine the expected utility value of architectural strategy using interpolation.
7. Calculate total benefit obtained from an architectural strategy.
8. Choose architectural strategies based on ROI subject to cost constraints
9. Confirm results with intuition

1.7 What is the purpose of Utility-Response Curve in CBAM?

The Utility-Response Curve describes the usefulness (utility) various responses in the quality scenario have for the stakeholders.

1.8 How do you calculate the total benefit obtained from an architectural strategy in CBAM?

Use the formula:
$B_i = \sum(b_{i,j} \times W_j)$
Where:
B_i = Benefit for each architectural strategy i.
$b_{i,j}$ is the benefit by using strategy i to its effect on scenario j.
$b_{i,j}$ is $U_{expected} - U_{current}$.
W_j is the weight of scenario j.

1.9 Describe the *process* in the textbook for reconstructing a software architecture.

1. Information extraction: Extract information from various sources.
2. Database construction: Convert information from step 1 to a database format.
3. View fusion: Combines information in the database.
4. Reconstruction: Building abstractions and various representations.

1.10 What kind of information can be useful to extract to reconstruct a software architecture?

Useful information for reconstruction of software architecture can be static relationship information like file-file, file-function, file-variable, directory-directory, directory-file, function-function (call), function-variable (read), function-variable (write), build sequence, design models. Also runtime information can be used such as polymorphism, function pointers, runtime parameterization etc.

1.11 Name the different types of architectures that are included in an enterprise architecture according to Telenor (Sigurd Thunem)?

An enterprise architecture consists of strategy architecture, business architecture, portfolio architecture and information architecture.

1.12 Name the three main areas the textbook describes for availability tactics.

The three main areas are fault detection, fault recovery and fault prevention.

1.13 What is the purpose of the GRASP (design) patterns?

The GRASP patterns describe the fundamental principles of assigning responsibilities to objects, expressed as patterns.

1.14 What is a (design) pattern language?

A pattern language is a collection of patterns that build on each other to generate a system. A pattern in isolation solves an isolated design problem, while a pattern language builds a system.

1.15 What are the three techniques described in the textbook for repairing interface mismatch?

The three techniques are wrappers, bridges and mediators.

Problem 2: Choose the correct architectural pattern (10 points)

Buff Tore is hired as a software architect in the Pear software company, where his job is to choose the correct architectural pattern for the following two systems Pine and Smash. Buff Tore can choose among the following architectural patterns:

1. Pipe and filter
2. Layered
3. Blackboard
4. Task Control
5. NASREM

2.1 Choose and motivate for your choice of architecture pattern for the Pine system (5 points)

The Pine system should have the following characteristics:

- The main parts of the system will remain the same for many years (stable).
- The Pine system should be possible to tailor for each customer, typically the user-interface and look & feel of the system.
- The Pine system should be possible to run on various operating systems and hardware platforms.

The architectural pattern that fits best to the Pine system is the **layered pattern**. The layered pattern makes it easy to create a stable part of the system implemented through several layers of abstraction. The bottom layers will handle the hardware and operating specific features making it possible to run the system on various hardware and operating systems. The top layer is the customer specific layer that is easy to change to tailor customer specific needs.

2.2 Choose and motivate for your choice of architecture pattern for the Smash application (5 points)

The Smash system is an application for changing audio in real-time by applying audio effects. The system has the following characteristics:

- Several audio effects can be applied to the input audio one after another.
- The implementation of the audio effects might be changed to improve performance.
- New audio effects might be added to the application in later releases.

The architectural pattern that fits best to the Smash application is the **pipe-and-filter pattern**. The pipe-and-filter pattern fits very well as there is audio input that will be transformed through various filters. The pipe-and-filter pattern will make it easy to update filters both in terms of performance and sound, and it will also be easy to add new audio effects in the future by just adding another filter.

Problem 3: ATAM (5 points)

Do the step 6 (Analyze the architectural approaches) in the ATAM process based on the following information about a system for selling tickets over the web:

Utility tree:

- *Usability:*
  - *Scenario U1. The client user should be able to correctly use 90% of the functionality of the application after using the system for 5 minutes (M,H).*
- *Performance:*
  - *Scenario P1: 10000 simultaneous users should have a response time less than 1 second under normal operation (H,M).*

Identified architectural tactics:

- *Reduce the number of events processed* – when more than 8000 users use the system at the same time, the number of users above 8000 will get lover processing priority.
- *Replication of server PCs* – The server application is replicated on two PCs to process more user requests.
- *Scheduling of resources* – the scheduling of processing resources is based on first come – first served.

Scenario U1 is not relevant to the identified tactics.
Scenario: P1: 10000 simultaneous users requests (H,M)
Attribute: Performance
Environment: Normal operation
Stimulus: Client request
Response: Response in less than 1 second

Tactic	Sensitivity	Tradeoff	Risk	Nonrisk
P1.Reduce number events	S1		R1	
P2.Replication server	S1			N1
P3.Schedule resources		T1	R2	

Reasoning:
The tactics used are well-known and well-proven tactics for performance. P2 is considered safe and easy to implement. P1 and P3 make the system more complicated and must have extra attention to be designed and implemented robustly. T1 and T3 are very dependent on the choice of algorithm and design.

Sensitivity points/tradeoff points:
S1: Will improve performance and availability.
T1: This tactic improves performance but might cause negative effect on availability.

Risk/Nonrisks:
R1: P1 is a risk as it only guarantees for 8000 users and does not guarantee for the extra 2000 according to the quality scenario.
N1: P2 is considered a safe tactic with no or little negative effects other than extra costs.
R2: P3 can be a risk depending on the design and implementation. Also another scheduling algorithm might be better for this application.
Risk themes:
A risk theme for this scenario is how the tactics are design and implemented.

Problem 4 (10 points): The CBAM

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

- **Total benefit** obtained from the 3 architectural strategies.
- **Return-On-Investment** for the 3 architectural strategies
- **Rank the 3 architectural strategies** according to best investment.

Use straight lines between the data points in the graph.

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

Scenario	Vote	Worst	Current	Desired	Best
1. Performance (Time to pick up and deliver 4 balls to the light)	30	20min	10min	4min	3min
2. Availability (How many times the robot get stuck during the mission picking up balls)	20	30 % fail	10% fail	4% fail	0 % fail
3. Availability (How many times the robot controller software crashes)	15	20% crash	5% crash	0% lost	0% lost
4. Modifiability (Time to add readymade module)	20	60min	30min	15min	5min
5. Testability (Time to test robot movements)	15	5hours	2hours	20min	5min

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

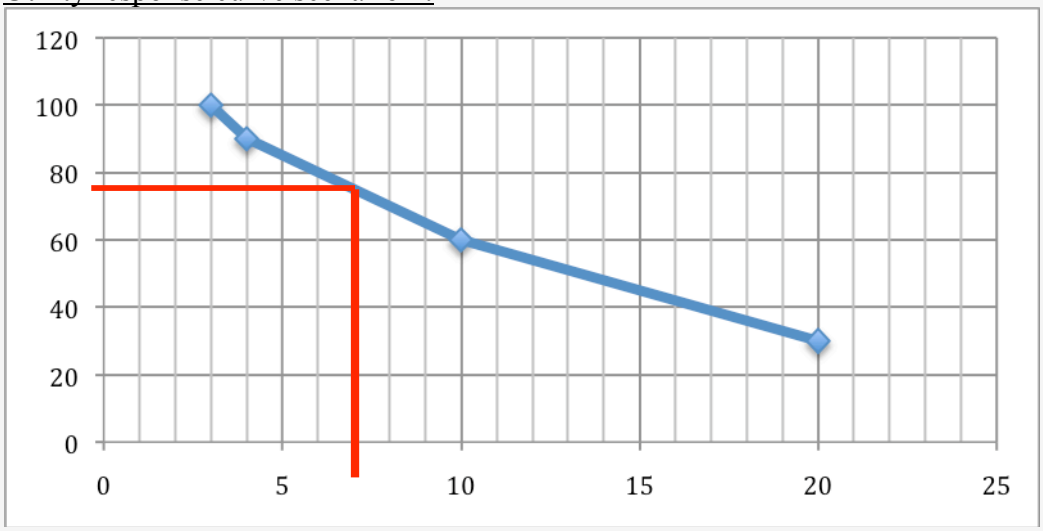
Scenario	Vote	Worst	Current	Desired	Best
1. Performance	30	30	60	90	100
2. Availability	20	20	70	90	100
3. Availability	15	30	80	100	100
4. Modifiability	20	15	60	70	100
5. Testability	15	5	30	90	100

Table 3: Effect and cost of using architectural strategies.

Strategy	Scenario	Cost	Current response	Expected response
1. Use route-planning	1	2000	10min	7min
2. Improved wall detector	2 3	800	10% fail 5% crash	6% fail 5% crash
3. Improved exception handling	2 3	400	10% fail 5% crash	10% fail 2% crash

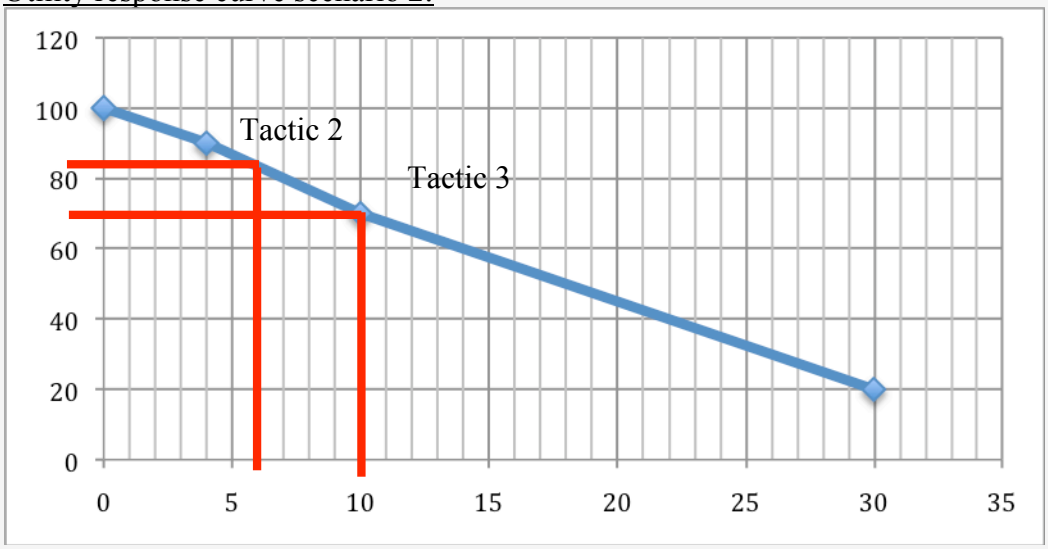
Solution:
To compute the total benefit obtained, the return-on-investment, and find what architectural strategy is best investment, utility response curve for scenarios 1-3 must be made (scenarios 4-5 can be ignored as there is not defined any tactics that will affect them). From these curves the utility 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. However, the conclusion should be the same.

Utility response curve scenario 1:



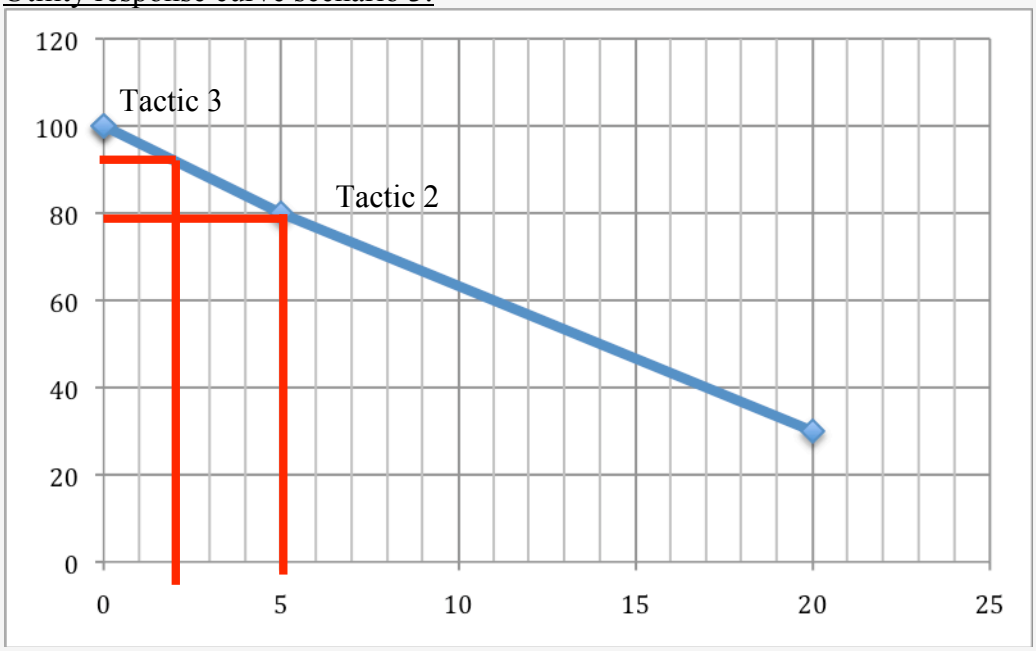
- Using tactic 1 on scenario 1: 7 minutes pickup time gives 78% utility.

Utility response curve scenario 2:



- Using tactic 2 on scenario 2: 6% fail gives 82% utility
- Using tactic 3 on scenario 2: 5% fail gives 70% utility

Utility response curve scenario 3:



- Using tactic 2 on scenario 3: 5% crash gives 80% utility
- Using tactic 3 on scenario 3: 2% crash gives 92% utility

Total benefit on tactics 1, 2 and 3:

B_1 = (78-60) x 30 = 540
B_2 = (82-70) x 20 + (80-80) x 15 = 240
B_3 = (70-70) x 20 + (92-80) x 15 = 180

Return-on-investment on tactics 1, 2 and 3:

ROI_1 = 540/2000 = 0,27
ROI_2 = 240/800 = 0,30
ROI_3 = 180/400 = 0,45

Ranking of tactics (from best to worst):

1 Tactic 3: Improve exception handling
2 Tactic 2: Improve wall detector
3 Tactic 1: Use route-planning

Problem 5 Create an architecture (30 points)

Read the description below and do the following:
5.1 Identify the ***architectural drivers*** for the system described below (5 points)

The architectural drivers of the system are variation in hardware components and configurations (modifiability), availability of the software (the software on the camera should not crash) and time-to-marked (hard competition in bringing new models to the marked first).

5.2 Choose and describe suitable ***design and/or architectural patterns*** for the problem described below, and describe how the patterns affect the quality attributes (5 points)

The most obvious architectural patterns are model-view-controller pattern to make it easier to change the GUI. In general, it is important that the software architecture is flexible so that all the main components can have different variations that can be configured when the software is compiled (e.g. allow various storage technologies, etc). Various design patterns can be used, but one candidate is the observer-pattern to implement the model-view-controller.

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: *Logical and Scenario view* (20 points)

Motivate for your choice of quality attributes, architectural drivers, design patterns and the architectural patterns used in your architecture. State your assumptions.

Software for a simple digital video camera

The software described here is software that is used in the controller of various simple video cameras. The software should be able to provide different levels of functionality depending on the price segment of the video camera and the software should be able to be used for various kinds of hardware configurations (buttons, screen, data storage and optical components).
The video camera consist of these hardware components:

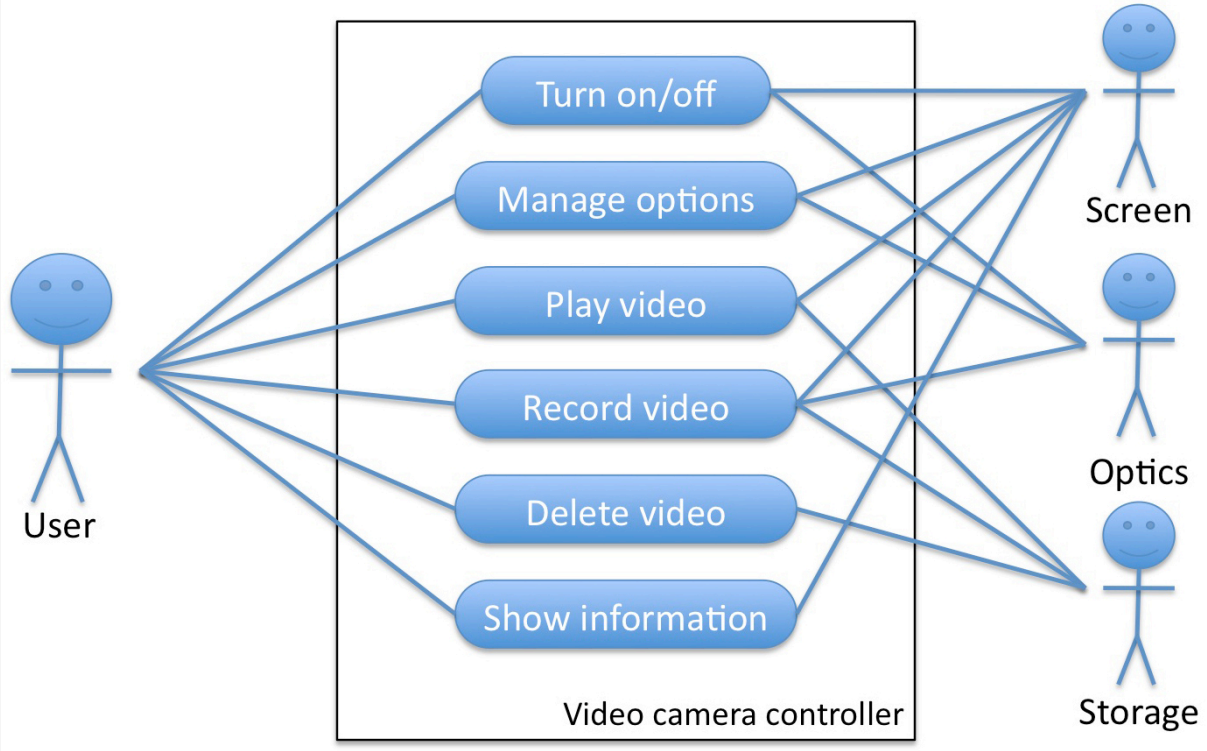
- Controller (CPU, memory): managing the other components, provide interface with the user etc.
- Controller buttons (vary from camera to camera). Typically buttons for on/off, video record, video player, night shot, menu, navigation, zoom etc.
- Digital screen (can vary from camera to camera in size, colour depth etc).
- View finder (small screen in the back of the camera).
- Permanent data storage (typically hard drive, flash-memory, memory stick, SD-cards etc).
- Optical component with an interface to control zoom, focus, etc....

Here is a list typical functionality offered by the video camera:

- Turn on/off camera.
- Playback video on screen.
- Record video to permanent data storage.
- User controlled optical functionality (zoom out, zoom in etc).
- Camera controlled optical functionality (auto focus, lens opening etc).
- Power save functionality (shut down camera if not used etc).
- Storing, retrieve and delete videos.

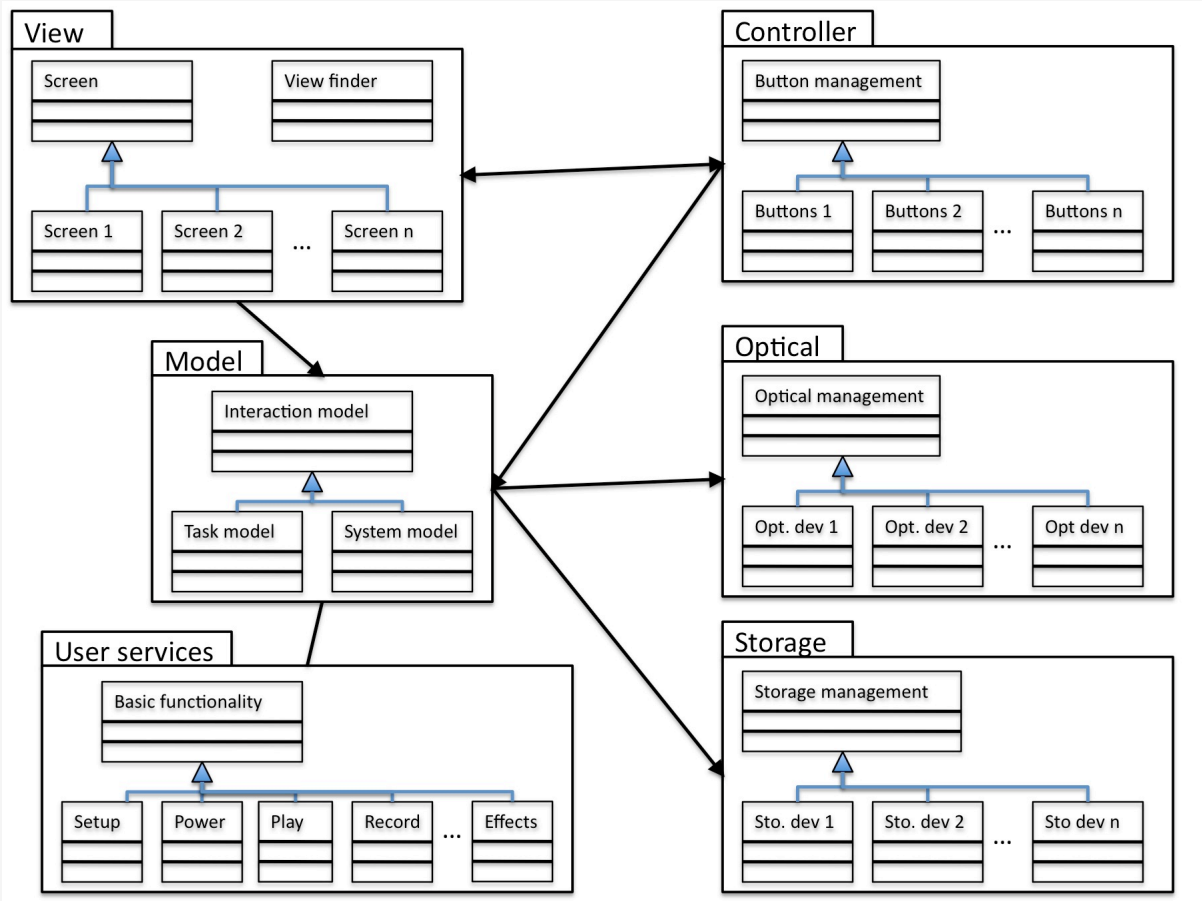
- Processing video effects.
- Display information to the user on the camera's screen.
- Video camera set up (storage options, GUI-options, language options etc.)

Scenario view:



The scenario view shows the main user interactions (use cases) involving the main parts of the system (screen, optics and storage).

Logical view:



The logical view is based on the model-view controller architectural pattern. In addition, the logical view models the variation points in the architecture allowing various configurations in terms of screens, buttons, optical devices and storage devices. The logical view also identifies the main user services provided by the controller.