TUTORIAL: Adding non-traditional properties for embedded systems design

Indira Jayaram, 2011

This tutorial is intended to provide a step-by-step description of the design process for the embedded systems by taking non-functional properties into consideration. A camera system is chosen as the example to explain the design steps.

1. Elicit functional requirements:
   List the functional tasks that the system is expected to perform. For a camera, this might have the following:
   - Take picture of given resolution
   - View a picture
   - Save a picture on to the computer
   - Charge batteries

2. List non-functional requirements:
   List requirements that are not functional in nature but are mostly related to how the system performs. For a camera, the traditional non-functional properties include:
   - Size
   - Power
   - Speed
   - User friendliness

   The non-traditional non-functional properties include cost and reliability.
   In general, the following are the other non-traditional, non-functional properties:
   -Reliability
   - Security
   -Safety
   -Ease of use
   -Real time responsiveness
   -Ease of upgrading

3. Analyze requirements and generate specification:
   Once the requirements are collected from the user, analyze them to determine if the system design is feasible with those requirements.
   Then, draw the use-case diagram for the system demonstrating the actors of the system

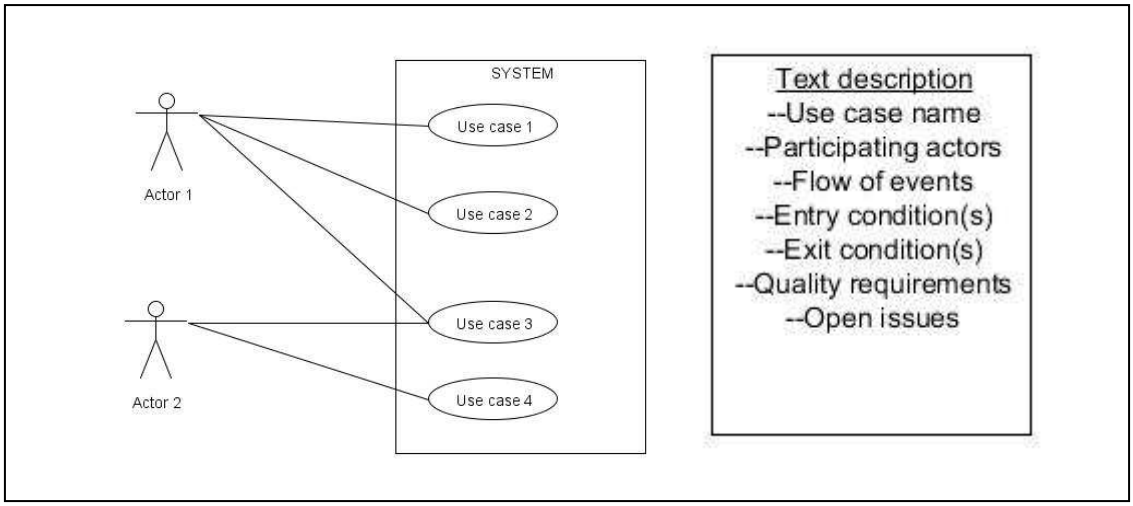

Figure A1: Example use case diagram

The use case diagram for a camera is as shown in Figure A2.

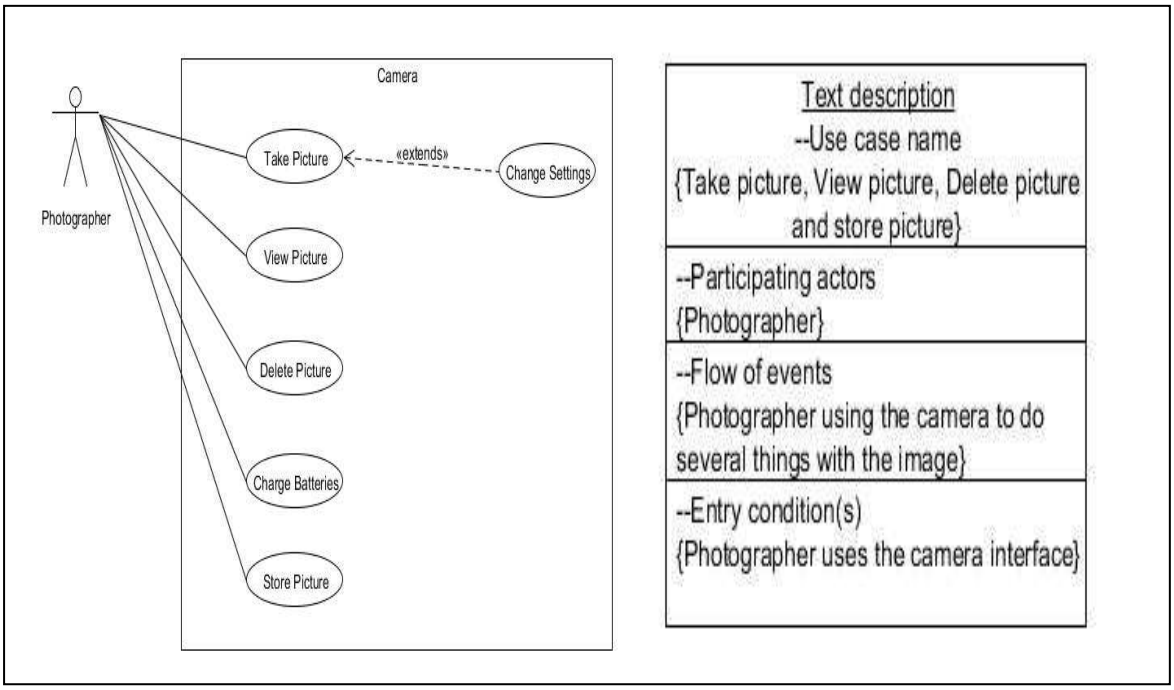

Figure A2: Use case diagram for a camera

Next, draw the class diagram for the system showing the separation of different tasks among the classes. A typical class diagram looks like Figure A3.

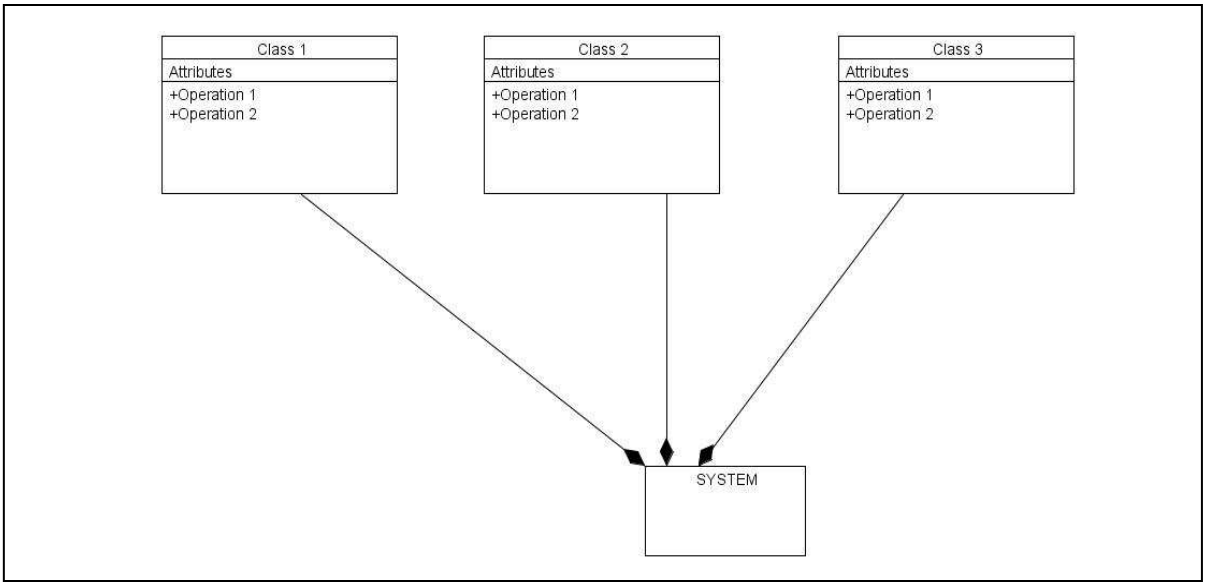

Figure A3: Example class diagram

The class diagram for the camera system is as shown in figure A4.

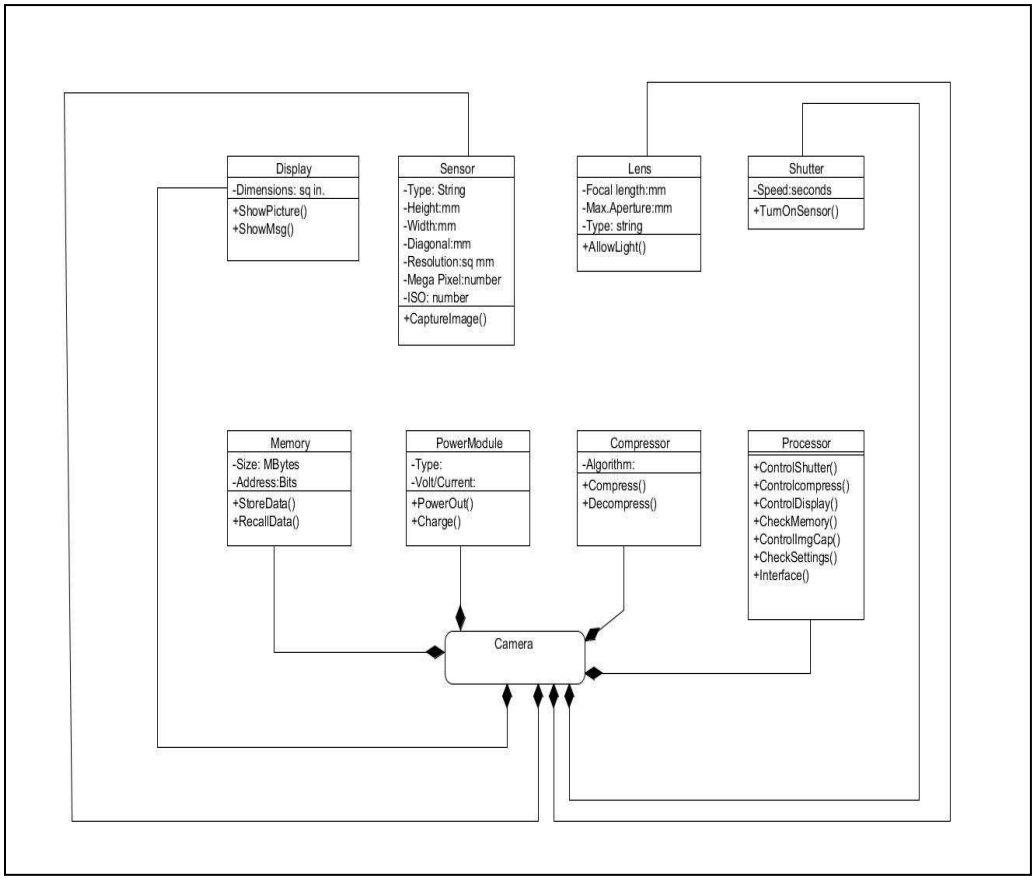

Figure A4: Class diagram for a camera

Next, draw the sequence diagrams for different sequences of operation that the system can perform. A typical sequence diagram looks like figure A5.

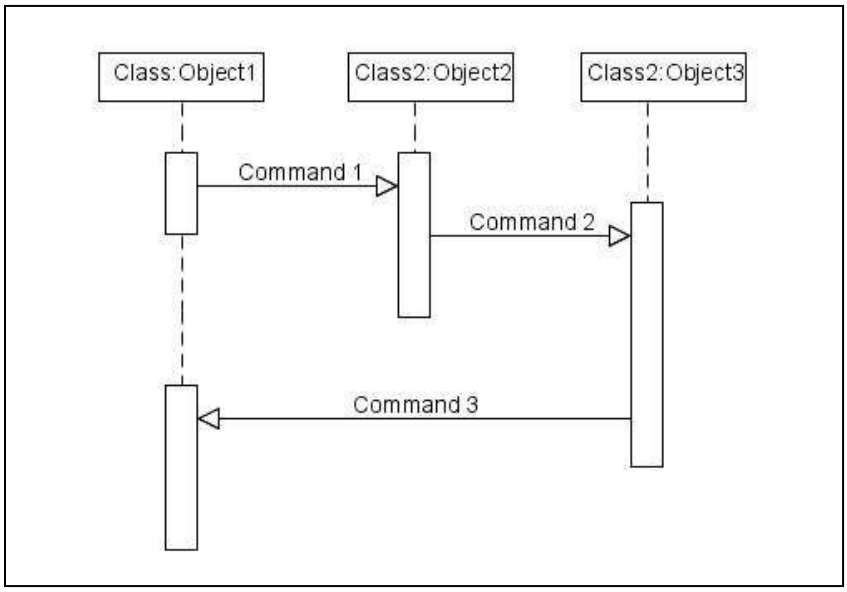

Figure A5: Example sequence diagram

One of the sequences for the camera system is shown in Figure A6.

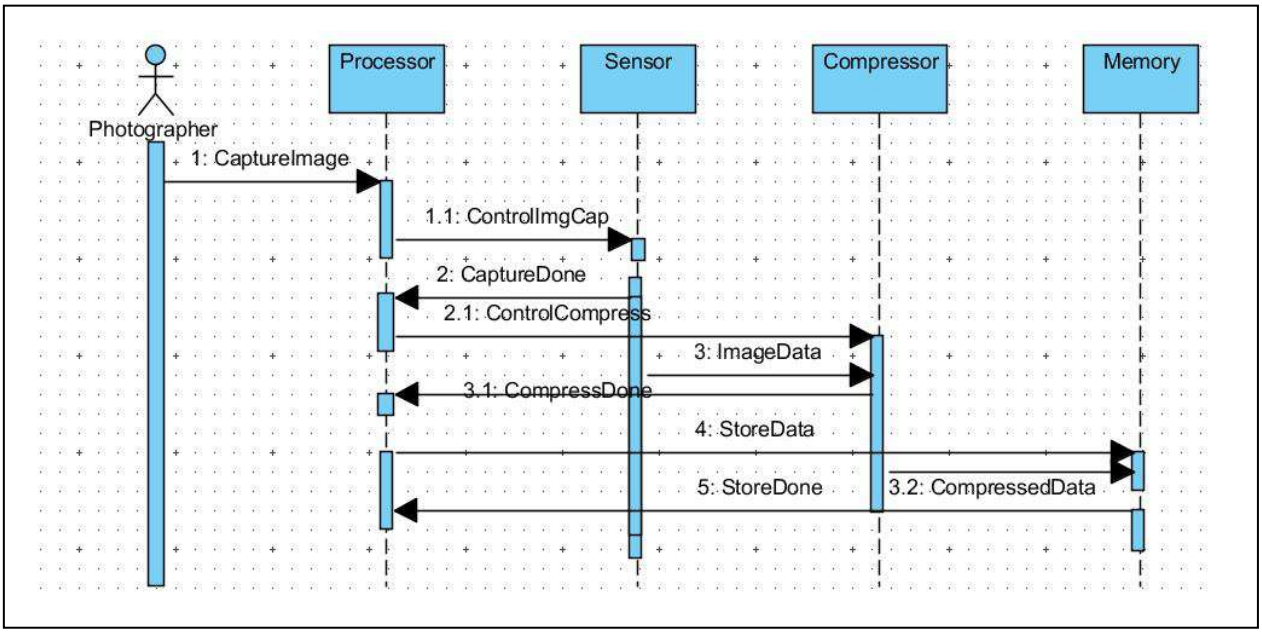

Figure A6: Camera image capture sequence

4. Weight the non-functional requirements

Generate weighted-constraint charts for the system

A typical weighting table looks like Figure A7

Requirements->	Req1	Req2	Req3	Req4	Req5
Weights->	w1	w2	w3	w4	w5

Figure A7: Example weighted constraint table

Here, the weights w1 through w5 should add up to 100. The most crucial requirement is given the highest weight.

The corresponding weighted constraint chart is as shown in Figure A8. Some sample weights are assigned

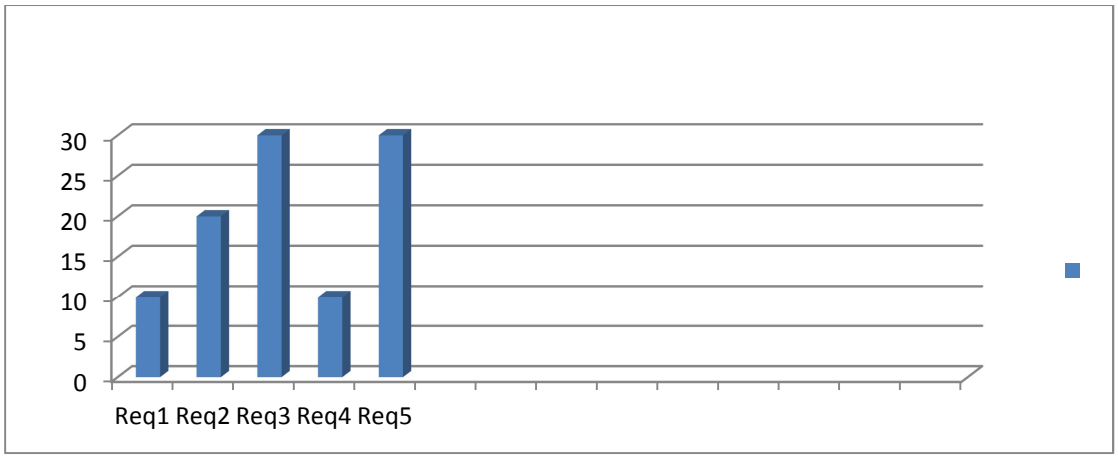

Figure A8: Example weighted-constraint chart

The weighted constraint chart for the camera system is shown in figure A9.

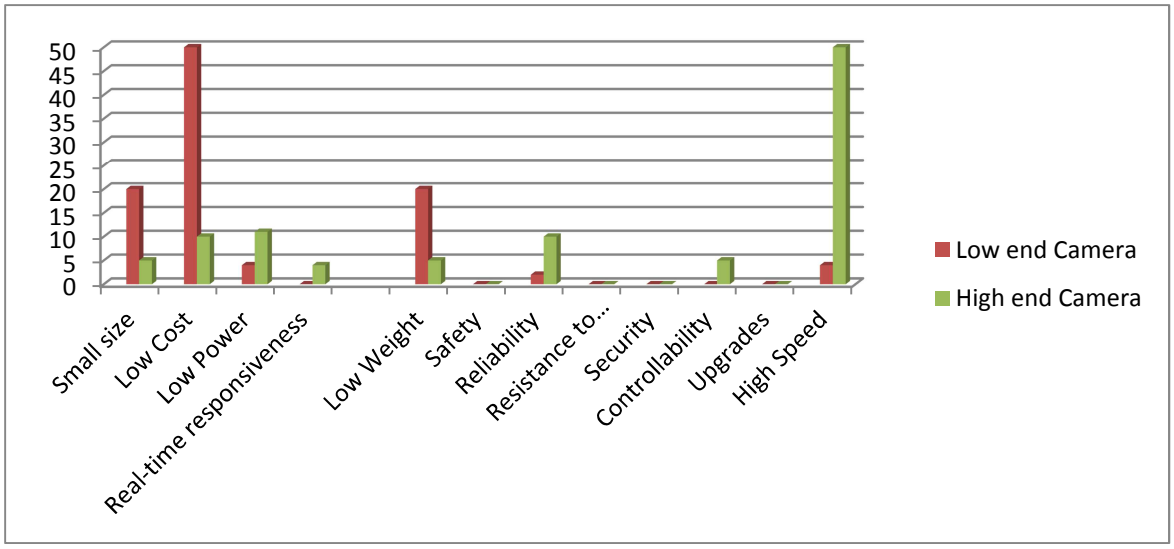

Figure A9: Weighted constraint chart for camera

5. Draw weighted constraint charts for all the classes in the system
   This should be done in such a way that, for any requirement, the average of the weights that is assigned to that requirement should be equal to the value of the weight assigned to that particular requirement at the system level.
   In the above example, Req 5 has the weight 30. Now, when Req5 is given weights in different classes of the system, the average of those weights should be 30.

   A typical weighted constraint table is shown in figure A10 and the corresponding chart is shown in Figure A11.

	Req 1	Req 2	Req 3	Req 4	Req 5
Class 1	15	15	50	10	10
Class 2	10	30	30	10	20
Class 3	5	15	10	10	60

Figure A10: Constraint table for classes

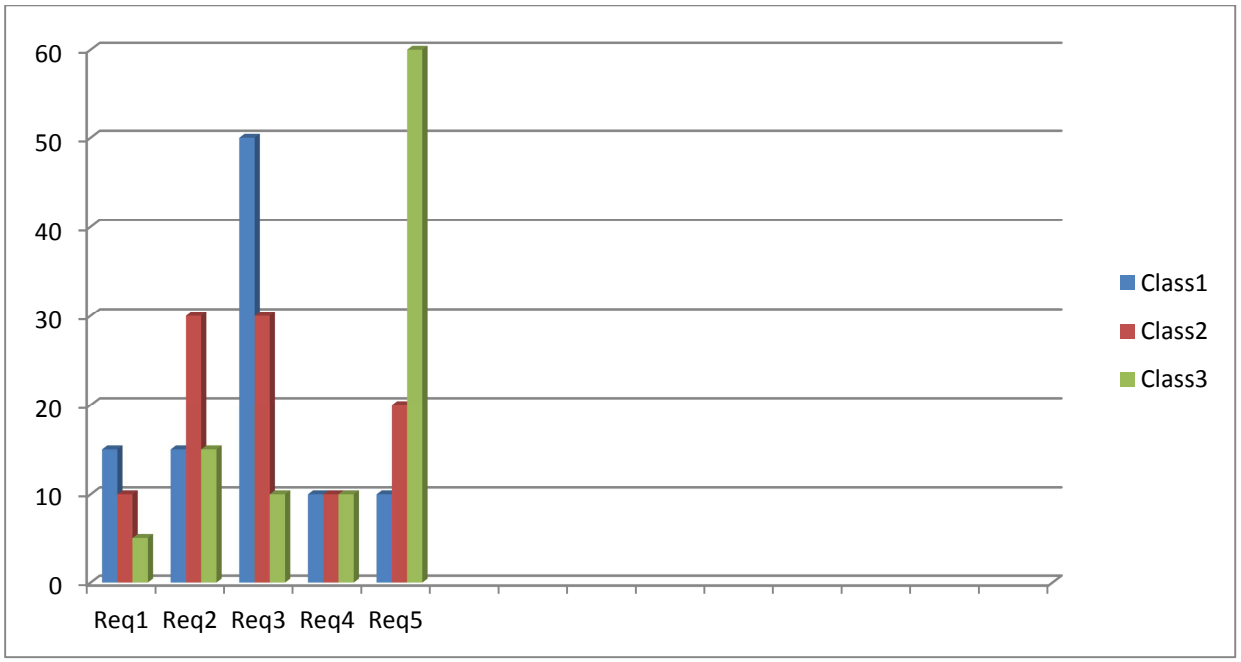

Figure A11: Weighted constraint chart for different classes

The weighted constraint chart for the classes of the camera is shown in Figure A12.

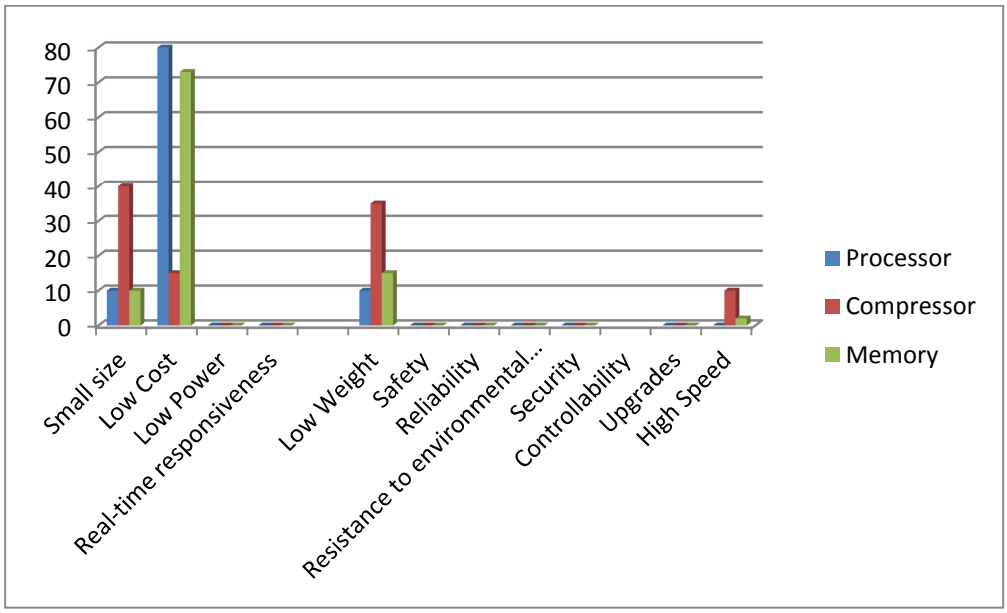

Figure A12: Weighted constraint chart for classes of camera system

6. Annotate the class diagram to reflect non-traditional properties

The non-traditional properties that are relevant to a particular class are annotated as notes attached to that class. Figure A13 shows how this typically looks.

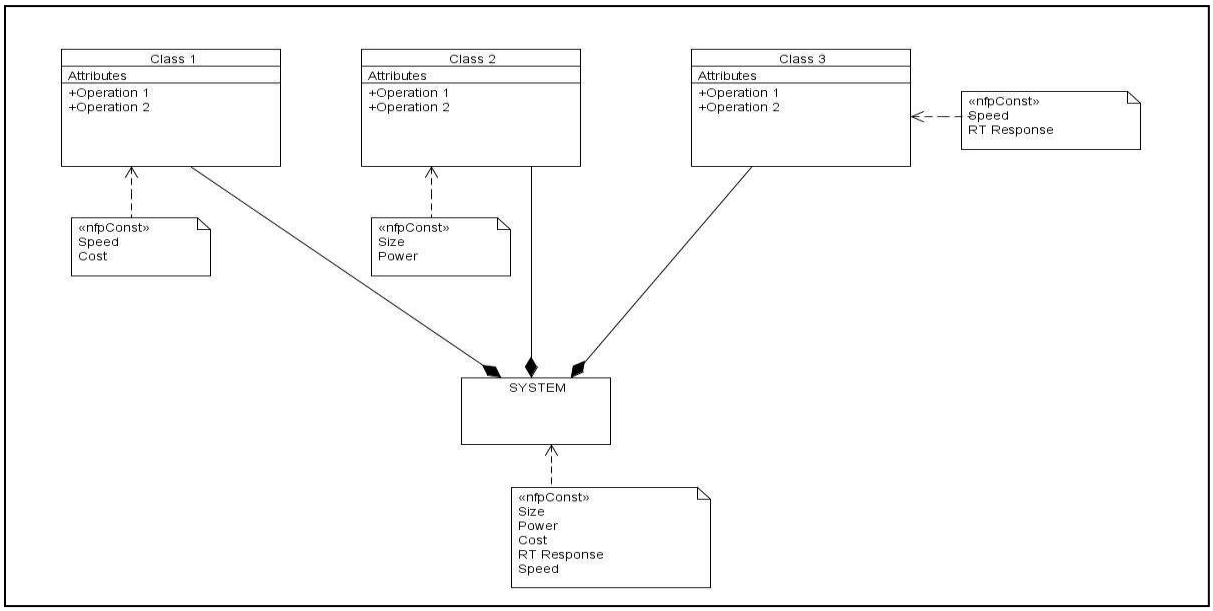

Figure A13: Annotated class diagram

Figure A14 shows an annotated class diagram for the camera

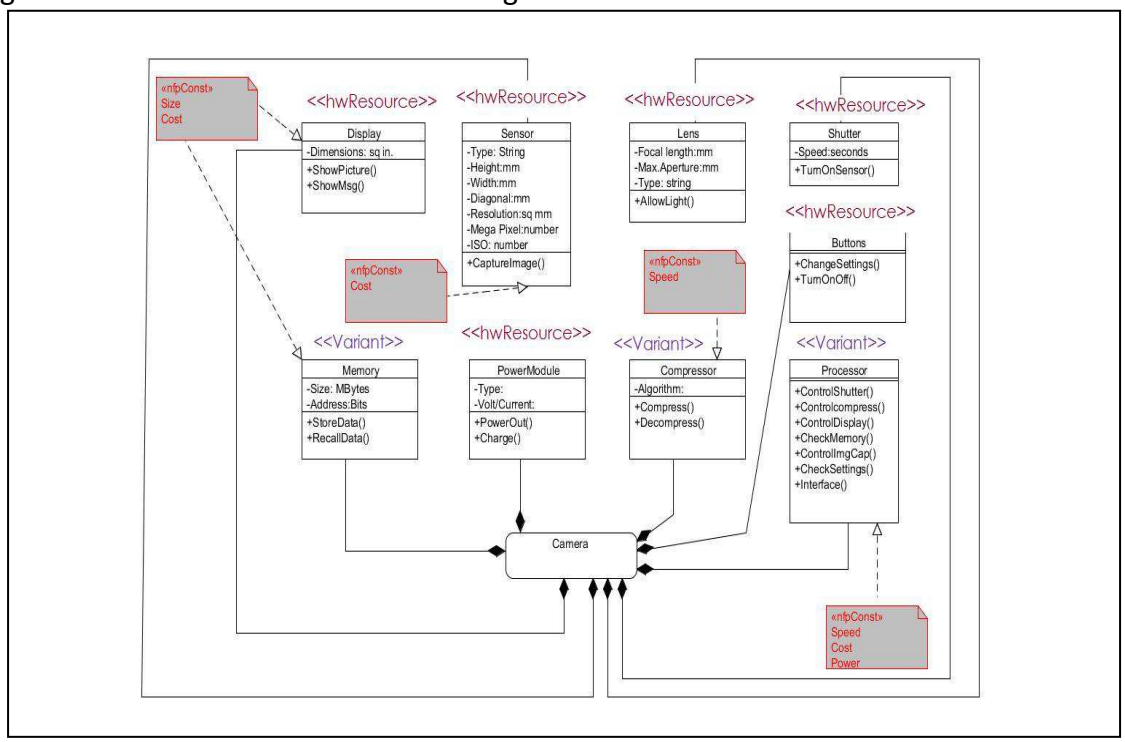

Figure A14: Annotated class diagram for camera

7. Annotate the sequence diagram to reflect time constraints
   The duration constrains are represented at several stages in the sequence. Figure A15 shows an example.

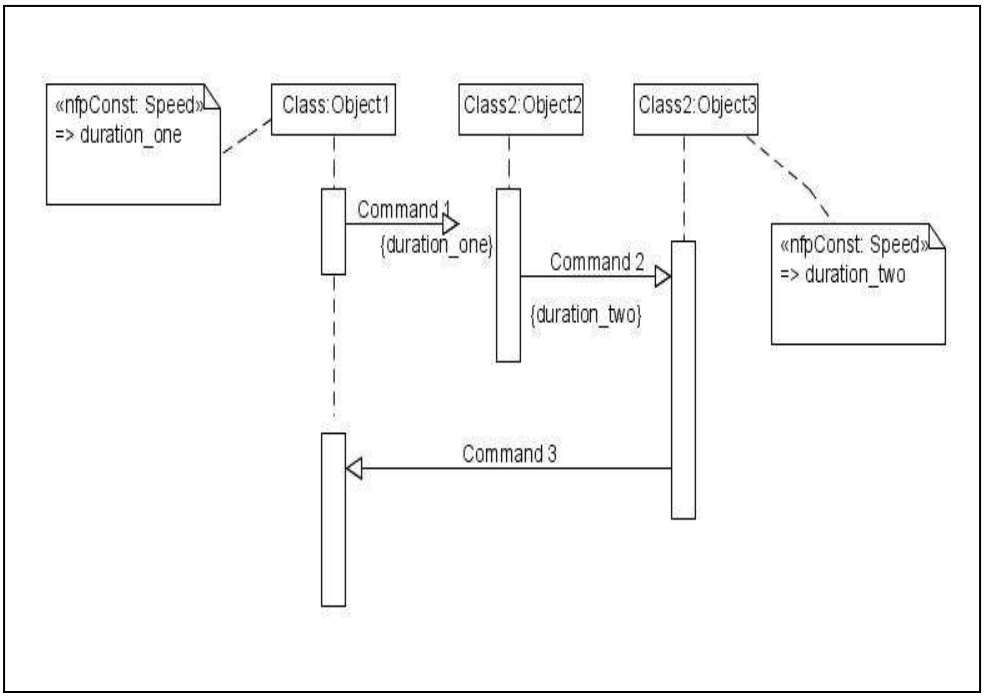

Figure A15: Example annotated sequence diagram

Figure A16 shows the annotated sequence diagram for the camera system

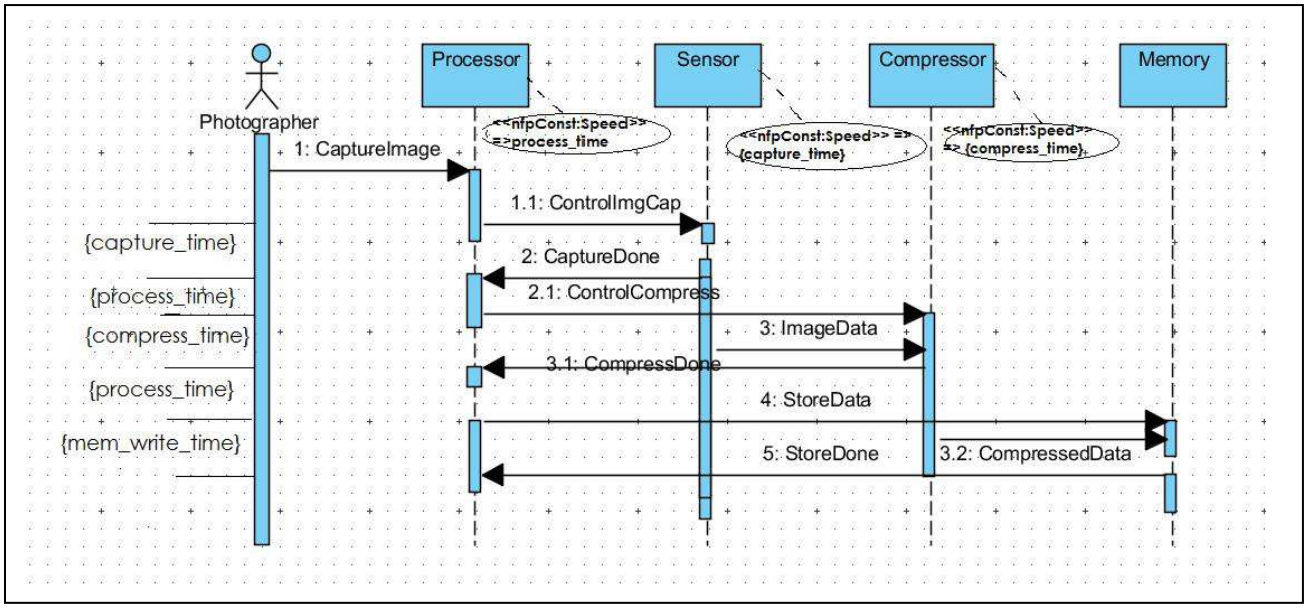

Figure A16: Annotated sequence diagram for camera system

8. Choose the implementation platform for the design

9. For the Altera UP3 platform, decide on whether you want a software or hardware implementation or a mixture of both taking into account both functional and non-traditional properties as discussed above.

10. Implement the design. If hardware implementation is chosen, write the program in Verilog/VHDL and synthesize it on the UP3 board through Quartus software.

11. If the software implementation is chosen, design the NIOS processor with essential peripherals through Quartus SOPC builder, synthesize and program the UP3 board. Through NIOS IDE, write the target program in C/ assembly and run it as NIOS hardware.

12. If a mixed implementation is chosen, design the software part of the system as described in step 11 and the hardware part of the system as described in step 10. Once the subsystems are developed, develop the schematic for the system consisting of both the hardware and software subsystems. Compile, synthesize and program the UP3 board through Quartus software.

Exercise questions:

1. Design a secure embedded internet system that would be used in a handheld device using the design techniques developed. Work out the implementation details for the system on Altera DE1/DE2 boards. Verify that the implementation met the design requirements on the non-traditional properties front.
2. Devise an implementation strategy for the camera design on a more general platform (i.e. making use of general purpose microprocessors/microcontrollers, DSPs etc.) and work out the various implementations possible. Through the set of implementations, try to arrive at the standard weights for the non-traditional constraints of relevance.
3. What test strategies would you propose to ensure that the non-traditional requirements have been met to a reasonable degree for any given embedded system? Demonstrate with an example system, say, a microwave oven.
4. Currently, the values of weights for constraints are chosen at random with higher weights representing more crucial requirements. Can you suggest ways to make the process more quantitative? What methods would be good to arrive at 'standard weight scales'? (The idea is to be able to pick a set of weights given the system and the type of the system you want to design).
5. Currently, the weighted constraint technique finds its use mostly in the design phase. Can you think of ways in which analysis information can be included in this technique? Also, can you think of how this technique would result in making the best implementation choice after generating the charts for multiple prototype systems?