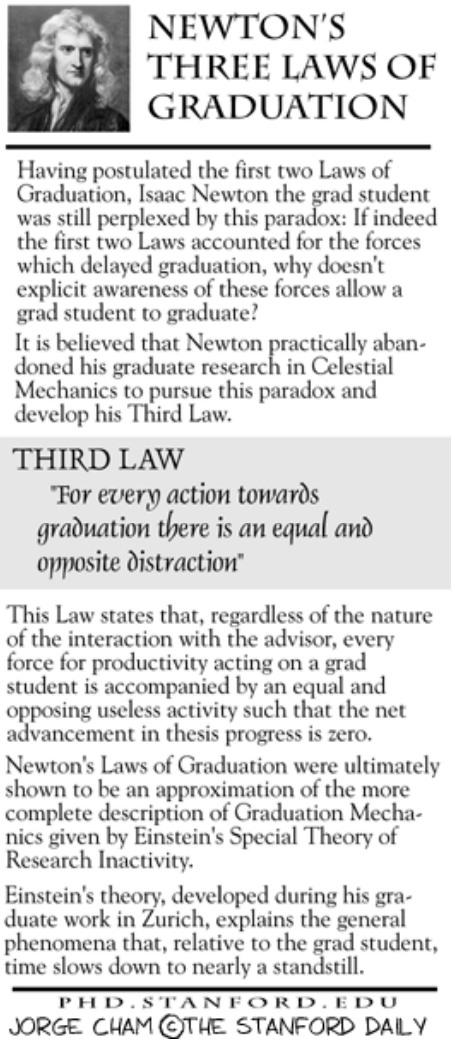
2015.03.15

Create Software Architecture in Automotive

Eugen Constantinescu

Is this introduction ?



Building Architecture vs Software Architecture

Open architecture: The developers didn't finish half of what was in the spec.

What do the weather and architectures have in common?
A safe prediction is they will both eventually change!

What does architecture and strategic planning have in common?
They both tend to be black holes that suck in every thing around them and never split anything out!

What do sharks and architecture have in common?
If they stop moving forward, you're dead!

What can architects learn from 80/20 rule (Pareto principle) ?
Complete 80% of the requirements before beginning architecture and 80% of the architecture before beginning detailed design, etc.
20% of the application deliver 80% of the business value!

What’s the Index of this presentation ?

- What’s the meaning of … *Architecture* ?
- Why do we have to use Architecture ?
- What questions should I ask before doing the architecture ?
- When it is the right time for developing SW Architecture ?
- How to build an Architecture ?
- **What is an Automotive Architecture ?**
- What about "Design and Patterns" ?
- Hi SW Architect...what do you do ?
- Where do you find more information ?

What's the meaning of … *Architecture* ?

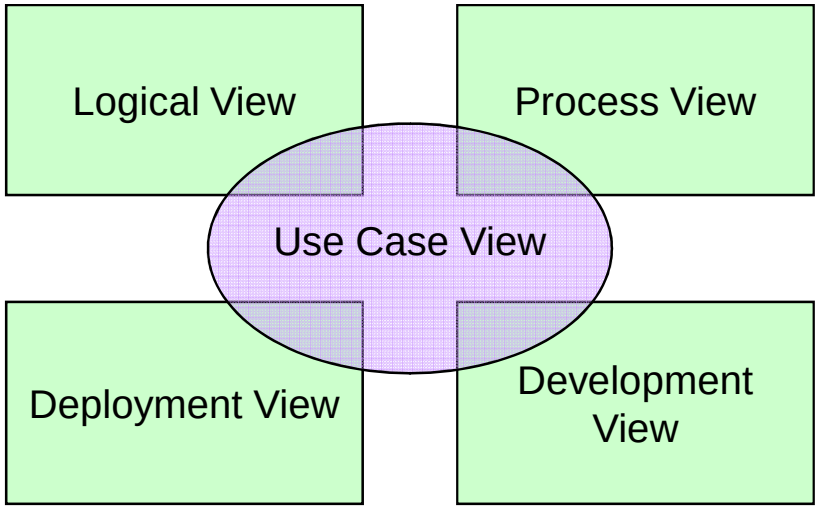
Definition

The Architecture is the organizational structure of the system describing the decomposition in its parts with their connectivity and interaction mechanisms. It includes details related to principles and decisions used to build the system.

What does it contain ?

- Architecture is the base for developing the system design, it must describe the most important parts and their relationships.
- It includes different views of the system
  1. Logical View (Functionality)
  2. Deployment View (Physical Components, Installation)
  3. Process View (Performance, Scalability)
  4. Development View (Configuration Management)
  5. Use Case View (Behavior)

Philippe Kruchten



Why do we have to use Architecture ?

What questions should I ask before doing the architecture ?

1. Is it part of the essence of the system?

Is the feature really **core** to what a system actually is? Think about it this way: can you imagine the system without that feature? If not, then you've probably found a feature that is part of the **essence** of a system.

Note from marketing: suggest replacing profanity with "heck."

MEANING

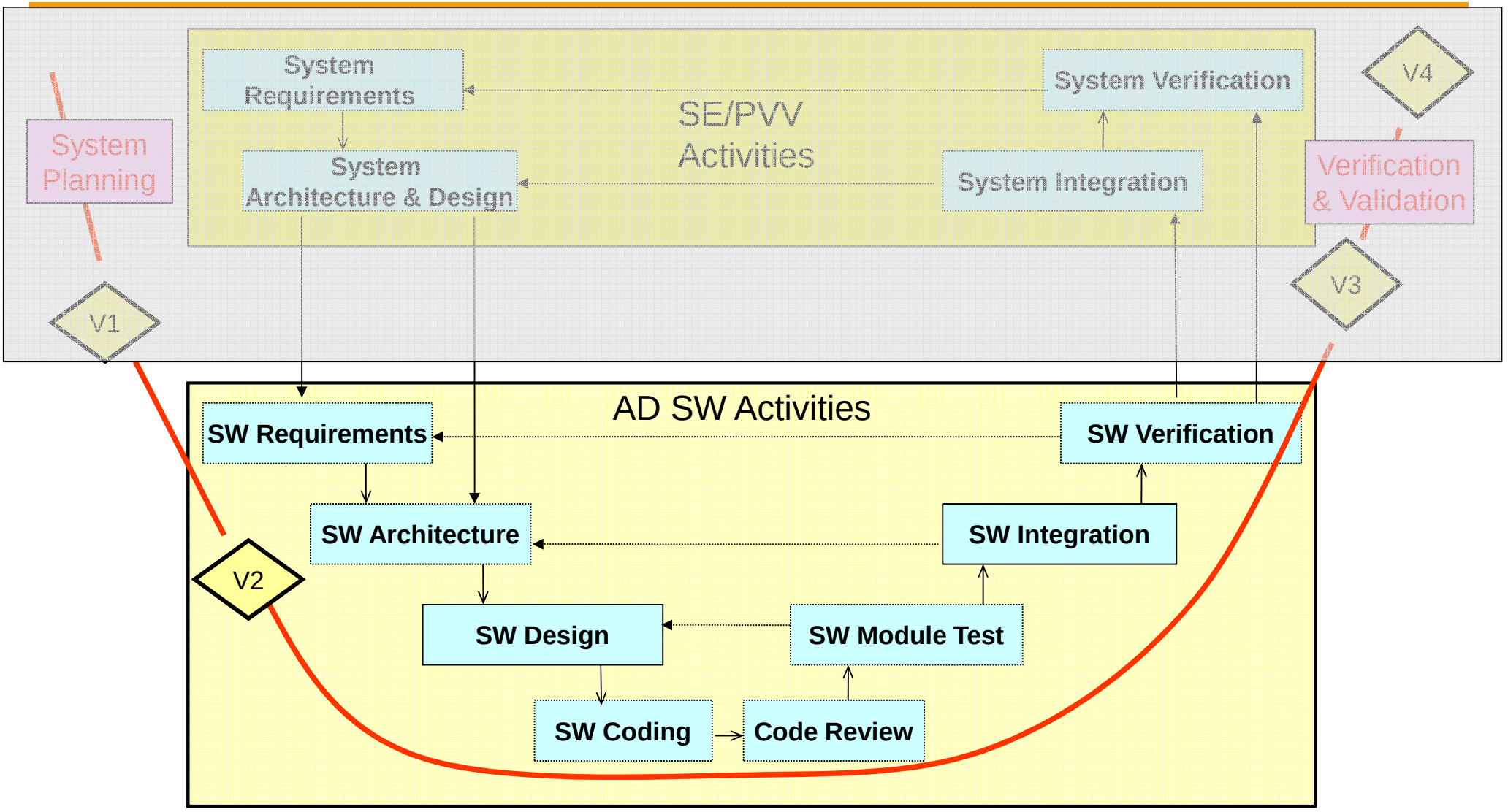
2. What the [redacted] does it mean?

If you're not sure what the description of a particular feature **really means**, it's probably pretty important that you pay attention to that feature. Anytime you're unsure about what something is, it could take lots of time, or create problems with the rest of the system. Spend time on these features early, rather than late.

3. How the "heck" do I do it?

Another place to focus your attention early on is on features that seem **really hard** to implement, or are **totally new** programming tasks for you. If you have no idea how you're going to tackle a particular problem, you better spend some time up front looking at that feature, so it doesn't create lots of problems down the road.

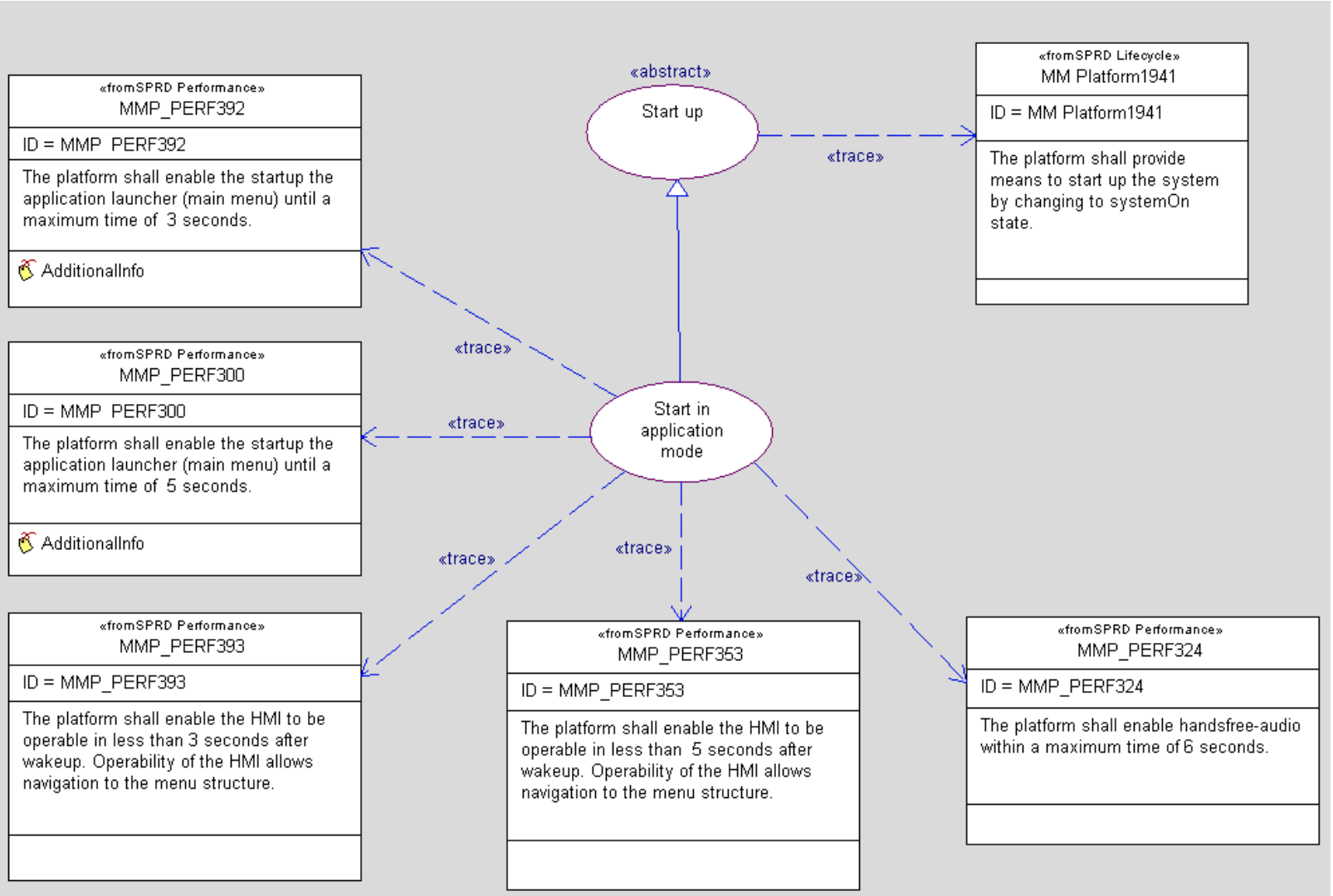
When it is the right time for developing SW Architecture ?
V - Cycle



How to build an Architecture ?
System Essence (4 + 1 … views)

- A system is developed based on ***requirements***. But if the system is really big, then we have to group requirements in ***features***.
- We have to identify the ***most important (System Essence)*** features and to group them (Ex: Audio, Video, Broadcast, Connected Devices, Speech, HMI, Navigation). Grouping will help to split the effort.
- Let's ***focus first*** on those features and develop the ***5 views*** model for them
- Describe the requirements behind the system essence as detailed as possible, everybody will use them. This means ***Logical and Use Case Views*** but only as a ***Black Box*** model.
- Start to develop the first level of ***White Box*** model with enough details as needed by ***Deployment and Development Views***. Complete this ***White Box*** model for ***Logical and Use Case Views*** too.
- Build the basic model for Process View. We have used ***SysML*** to model the above things.
  We get the System Architecture.
- Go into the next level of details for ***White Box*** model and expand ***ALL 5 Views*** as needed. Use UML and focus on ***Interfaces and Interaction Mechanisms***.
  We get the Software Architecture.
- What do we get if we keep on going deeper and deeper ? … ***Design***

Requirements in UML: Start-Up Performance



Types of UML diagrams in Rhapsody

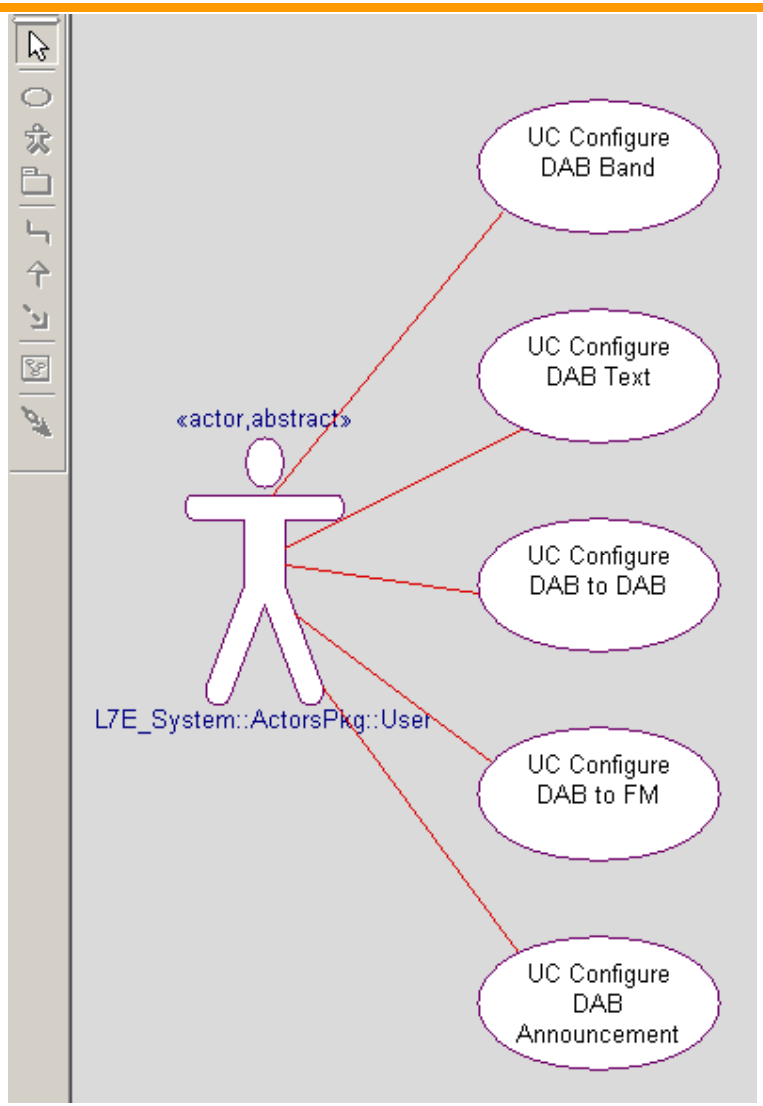
- **Use case diagrams** show diagrams show typical interactions between the system that is being designed and the external users or actors. You can use IBM® Rational® Rhapsody® to generate code for actors in use case diagrams to test models.
- **Class diagrams** show the static structure of a system: the classes in the system and their associations and operations, and the relationships between classes and any constraints on those relationships. A class diagram is the basic diagram in UML. A class diagram is the equivalent of the object model diagram in Rational Rhapsody. However, a class diagram is specified for the class structure, while an object model diagram is specified for the object structure.
- **Object model diagrams** show the static structure of a system: the objects in the system and their associations and operations, and the relationships between classes and any constraints on those relationships. In Rational Rhapsody, an object model diagram is the equivalent of a class diagram. However, an object model diagram is specified for the object structure, while a class diagram is specified for the class structure.
- **Sequence diagrams** show the message flow of objects over time for a particular scenario.
- **Collaboration diagrams** provide the same information as a sequence diagram but emphasizes structure, whereas a sequence diagram emphasizes time.

Types of UML diagrams in Rhapsody

- **Statecharts** define all the states that an object can occupy and the messages or events that cause the transition of the object from one state to another.
- **Activity diagrams** specify a workflow or process for classes, use cases, and operations. Activity diagrams are like statecharts; however, activity diagrams are better for showing linear step-by-step processes, while statecharts show non-linear and event-driven processes.
- **Component diagrams** describe the organization of the software units and the dependencies among these units.
- **Deployment diagrams** depict the nodes in the final system architecture and the connections between them. Nodes include processors that run software components, and the devices that those components control.
- **Structure diagrams** model the structure of a composite class; any class or object that has an object model diagram can have a structure diagram. Object model diagrams focus more on the specification of classes, whereas structure diagrams focus on the objects used in the model.

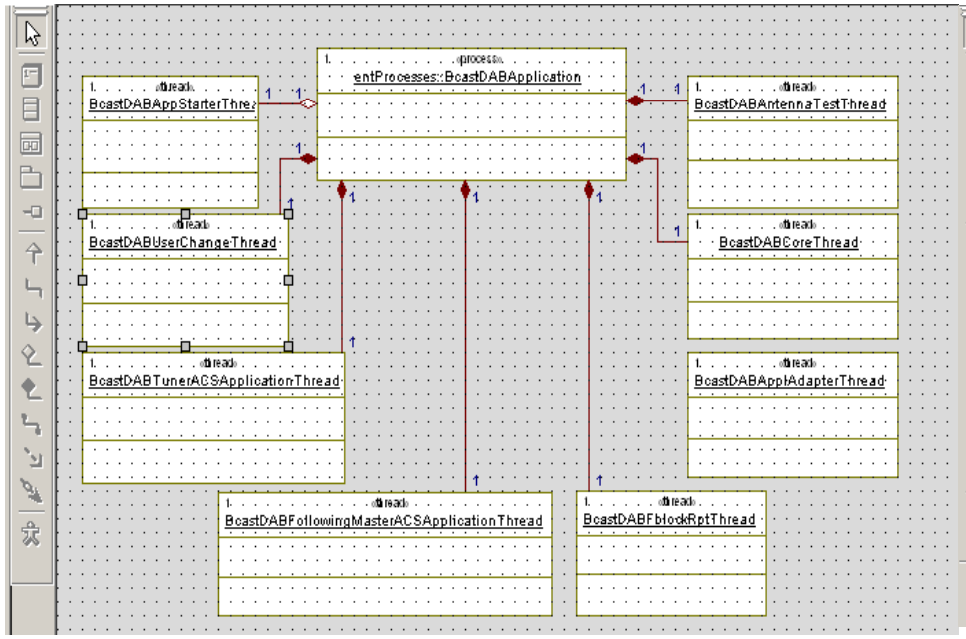
Use case diagram

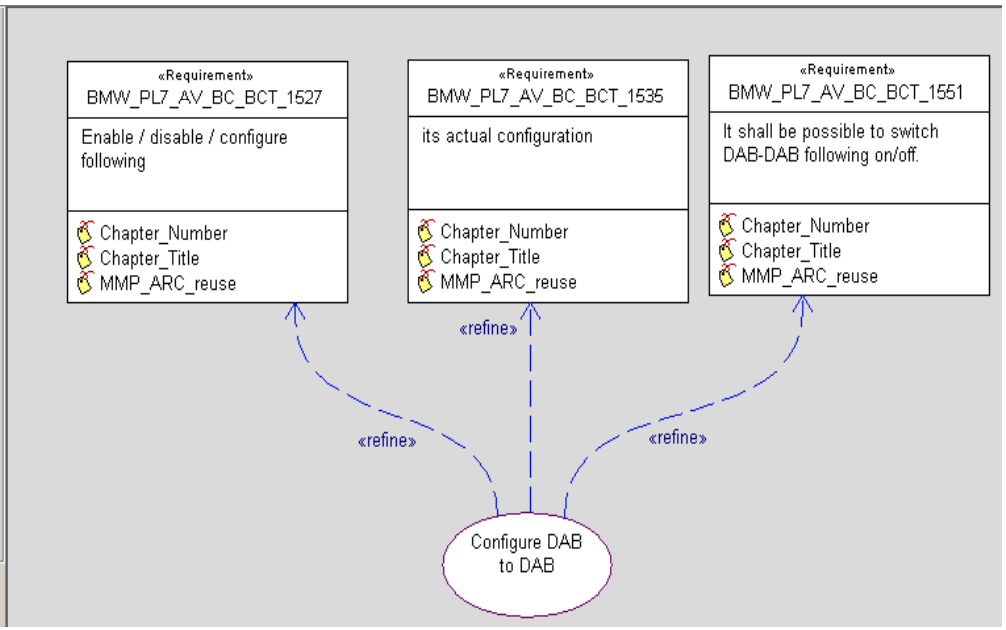
- Commonly used in system architecture
- Who is responsible for creating these diagrams?
- Where are they commonly found ?
- What are they related to ?



Object Model Diagram

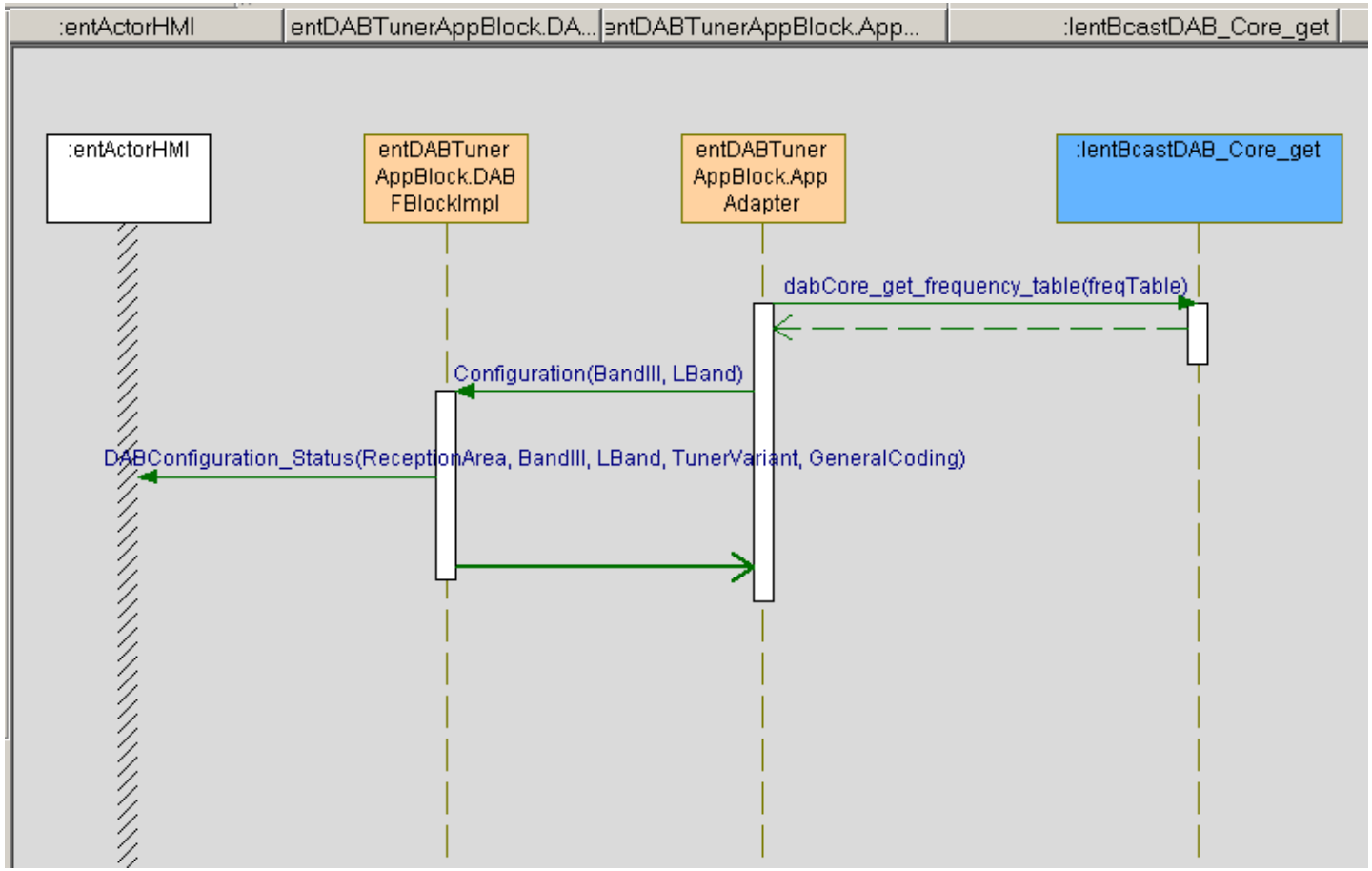
- Is used to show a static view of relevant relations between classes or objects
- Commonly used to make software design
- They are also used to show how use cases cover requirements (stereotype requirements diagram is used here)



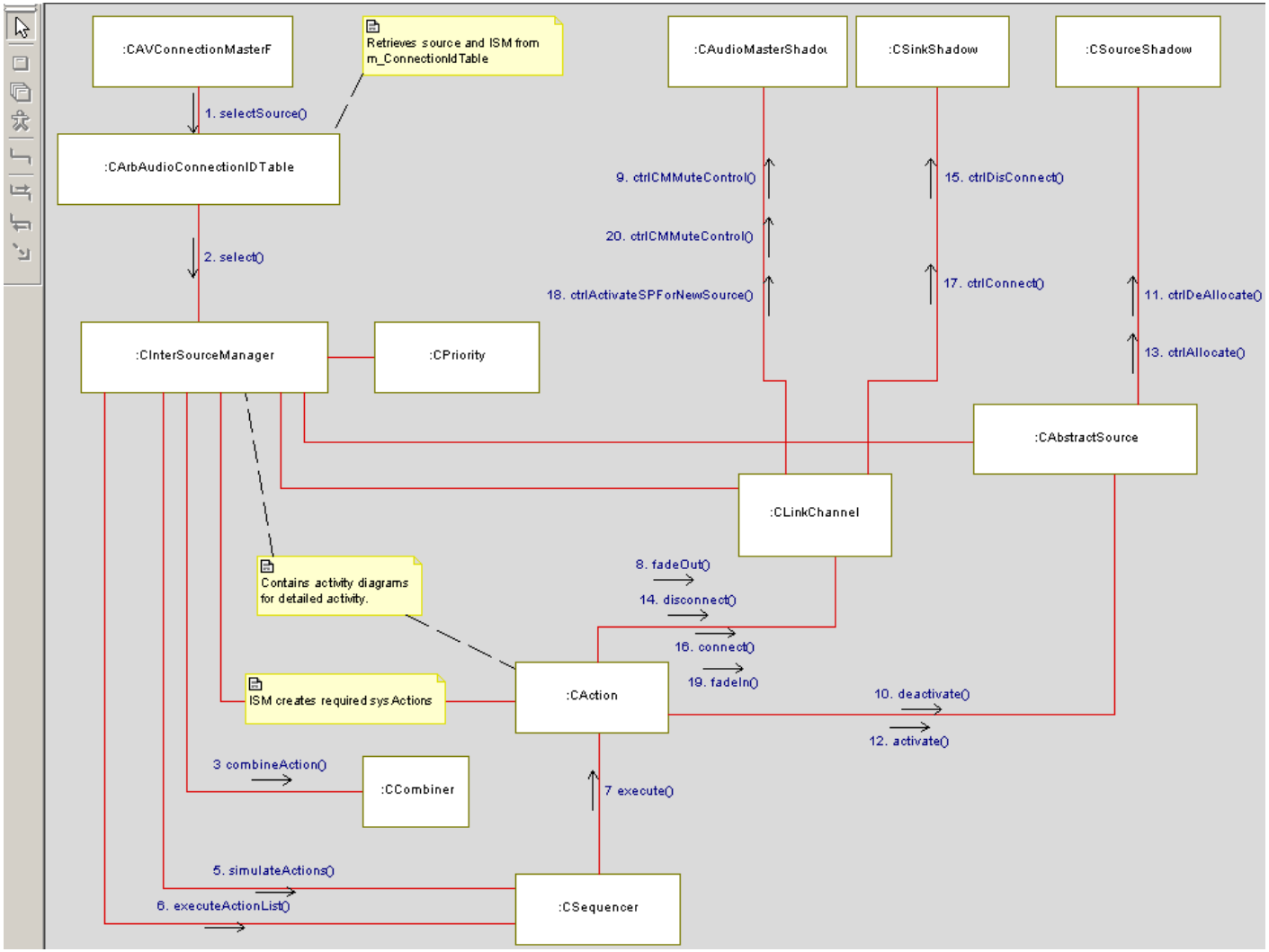


Sequence diagram

- Is the most commonly used diagram to show how objects (may be actors, components , classes, etc.) interact with each other to satisfy desired behavior

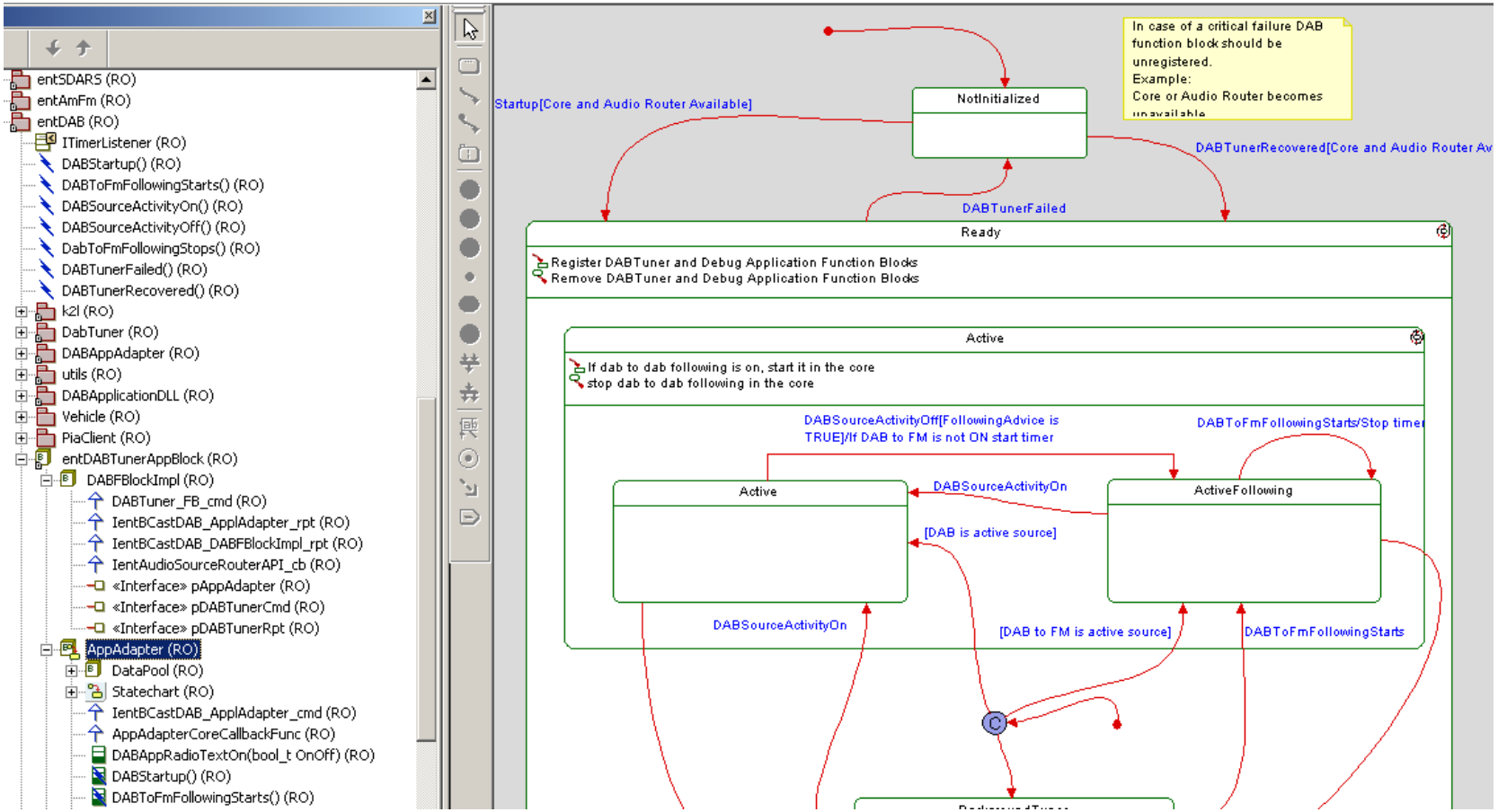


Colaboration Diagram

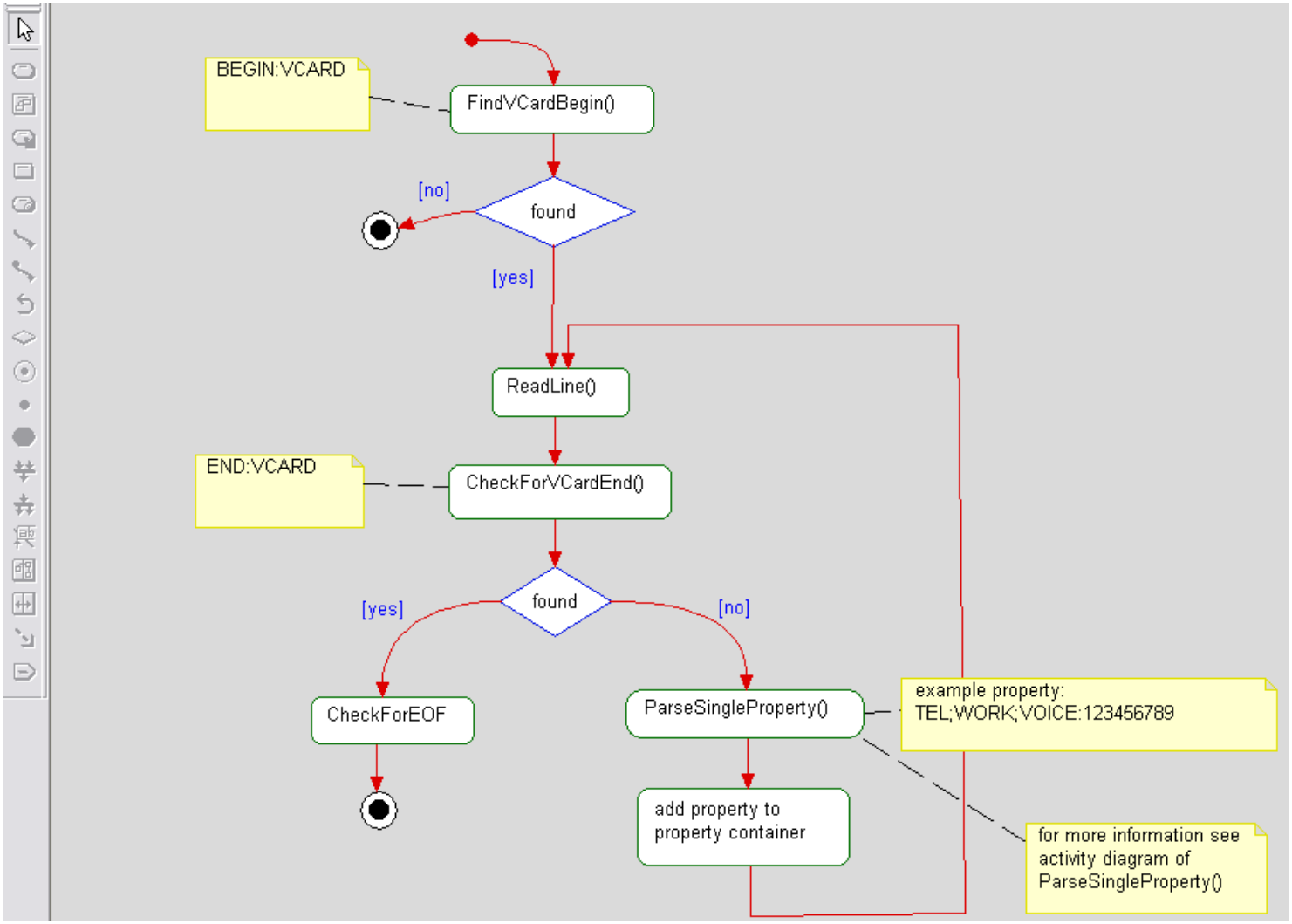


Statechart Diagram

- Will show you how a block or a class will behave.
- A class or a block may have only one associated state or activity diagram

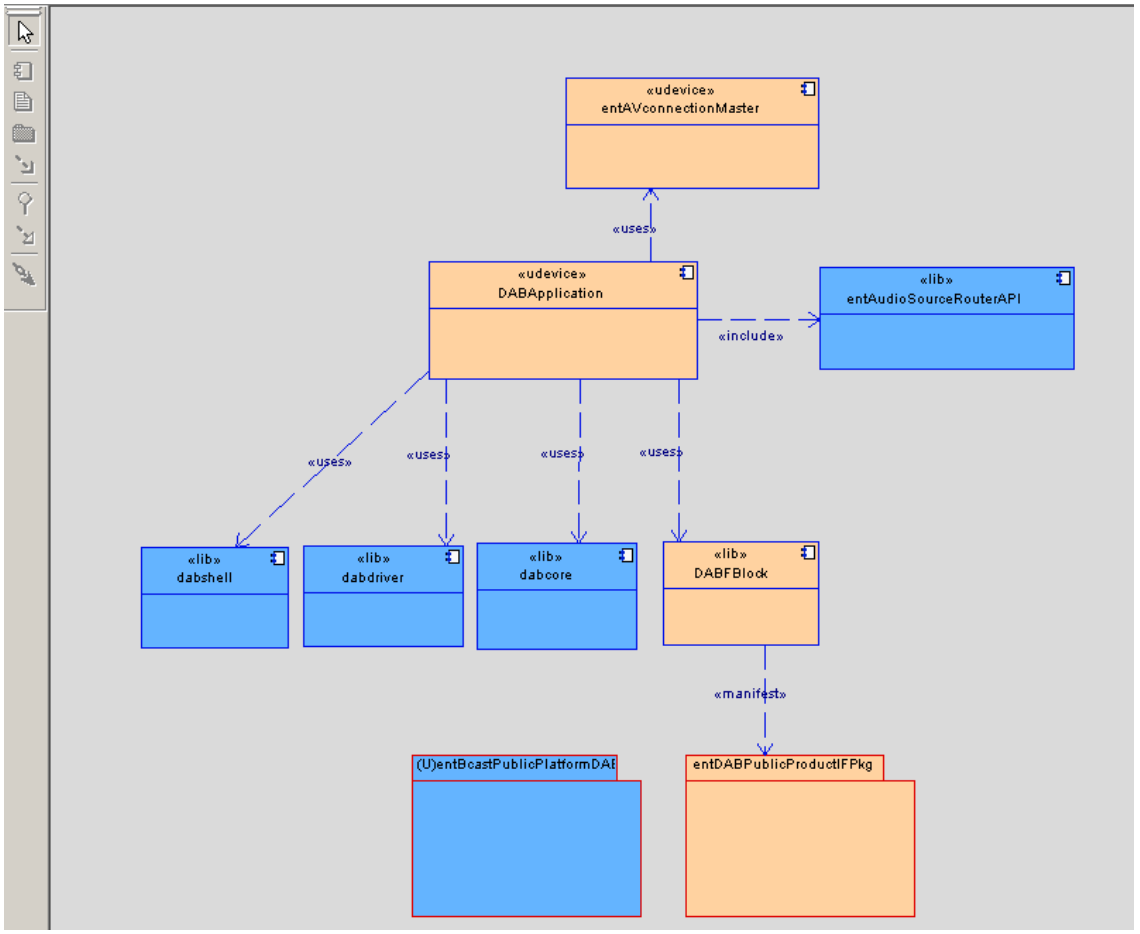


Activity Diagram

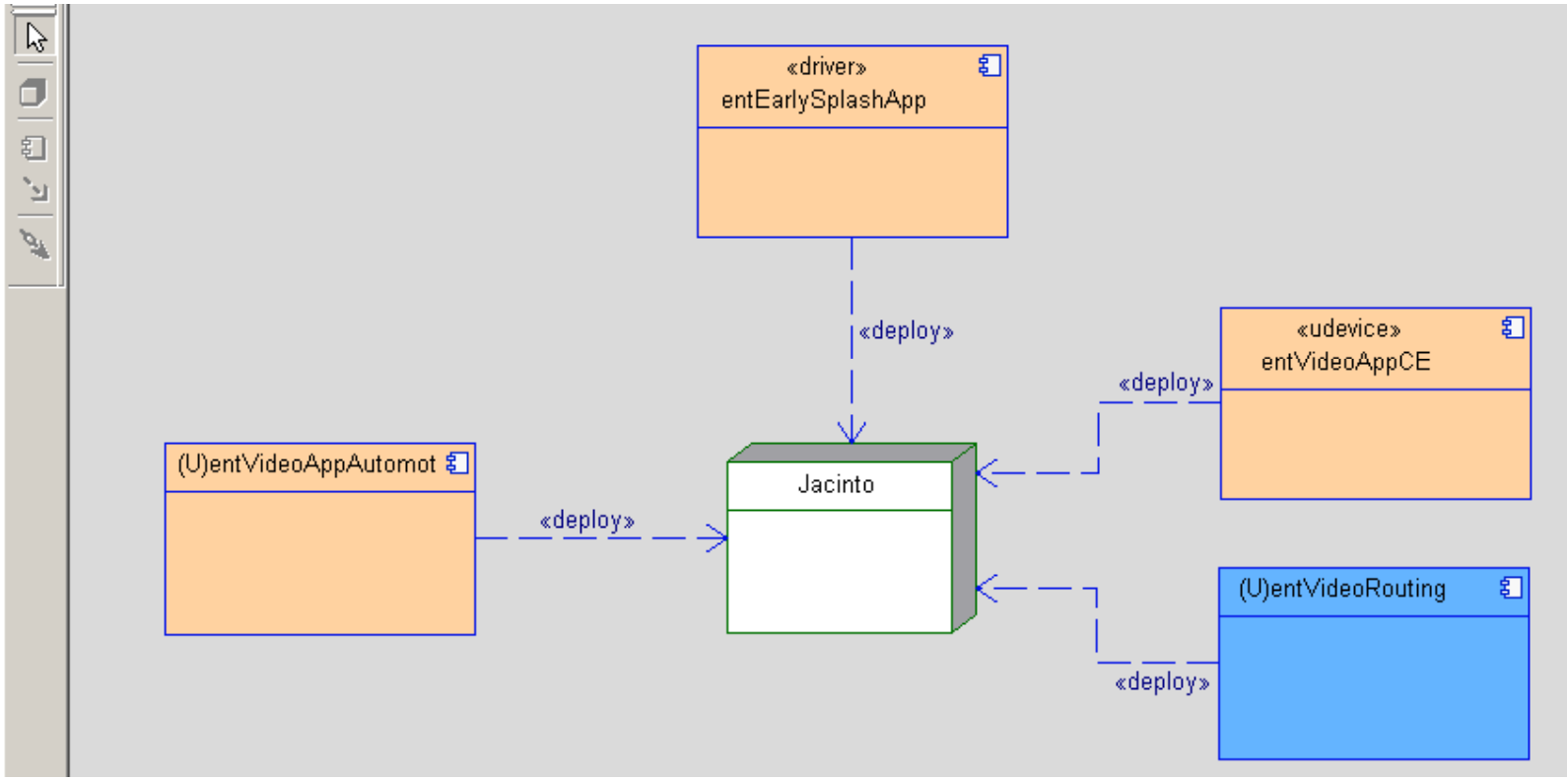


Component Diagram

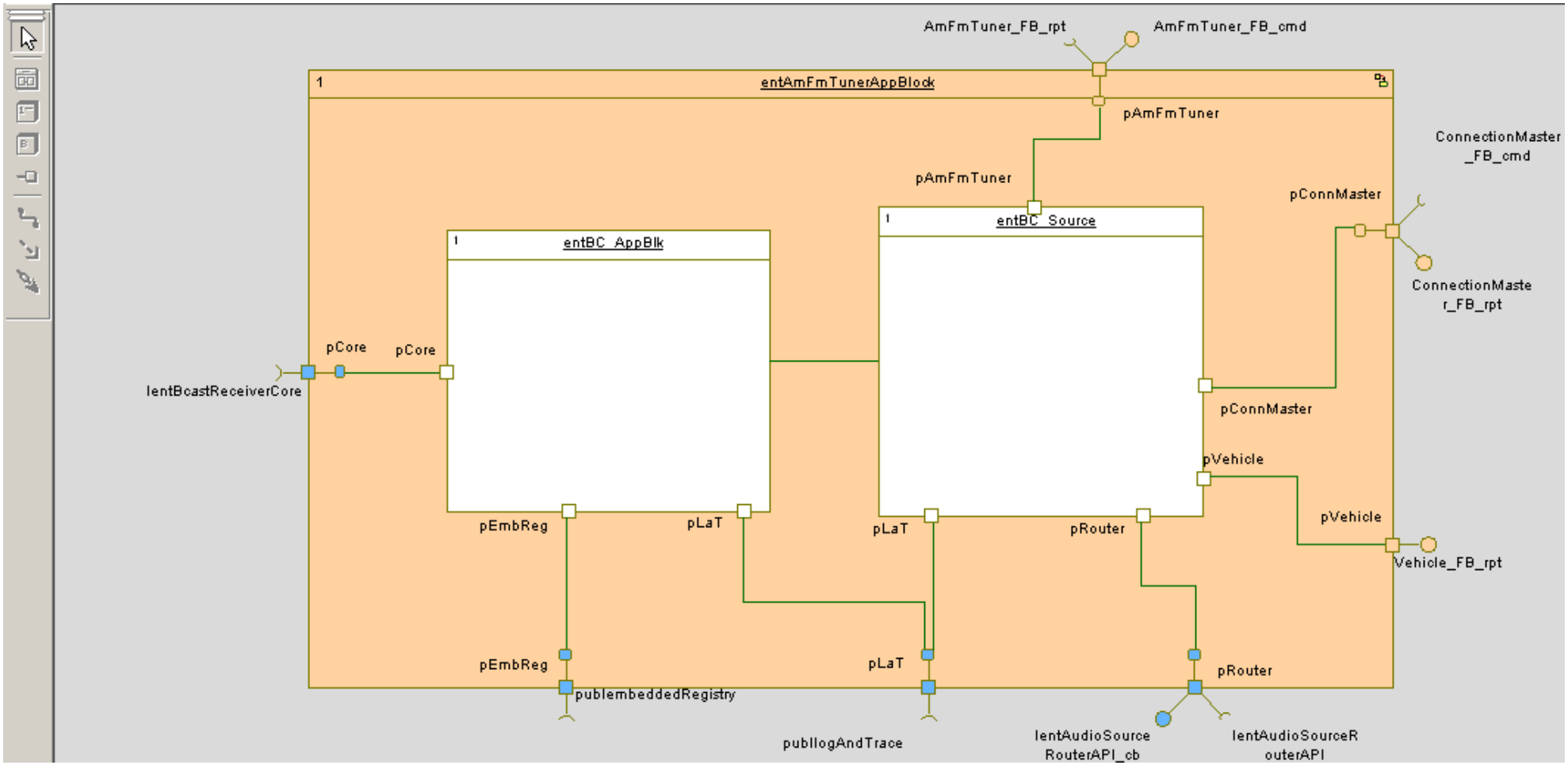
- Shows relations between components



Deployment diagram



Structure diagram



What is an Automotive Architecture ?

- Most of the requirements are coming from **Automotive Domain**.
- One platform, many products
- We have to use embedded systems (they fit best into the auto vehicles)
  Deployment and Process views have details about it.
- Multimedia Automotive Protocols
  - We have to use the buses available in auto vehicles (CAN, MOST). There are gateways from HW to SW. There are interfaces defined for each bus. Also drivers and specific binaries.
  - We have to use busses available for embedded devices (USB, BlueTooth, I2C, I2S)

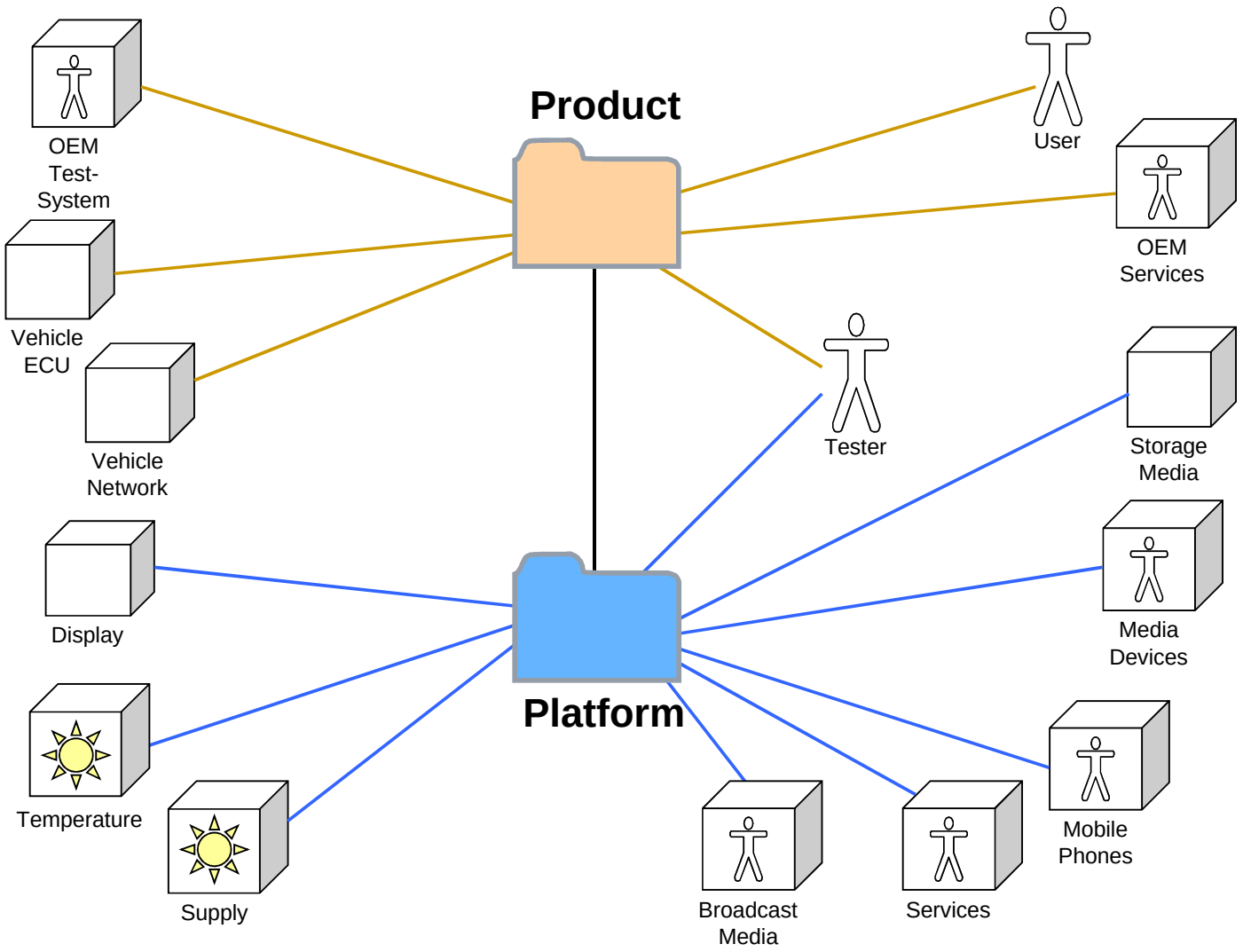
Platform and Product

- What's platform ?
- What's product ?
- What's Product Line Engineering ?
- Why do we need this type of architecture ? (reusability, costs, development process compliance)
- How to split what we keep in platform and what will go in products

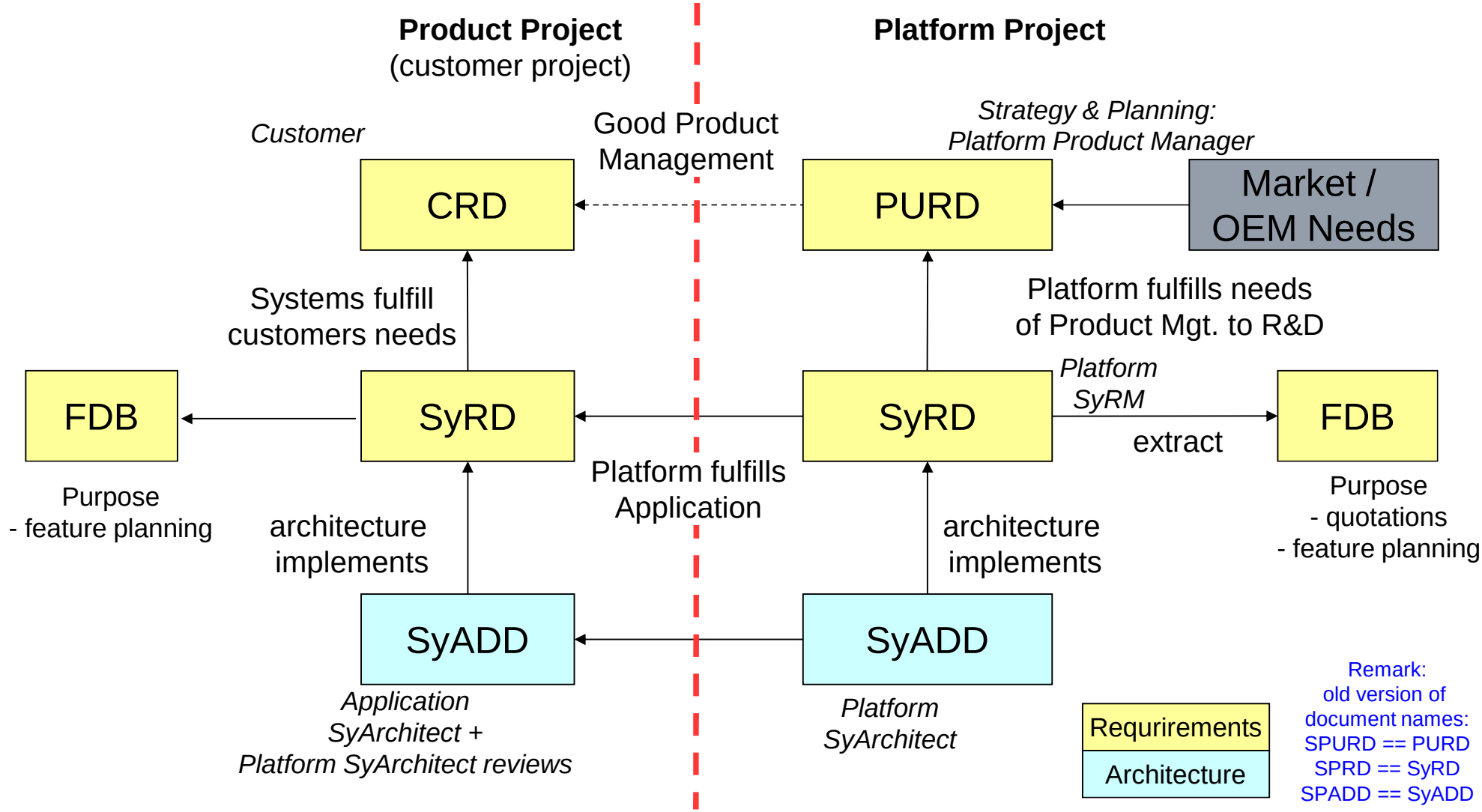
Product Line Engineering

Domain	Is a field of knowledge driven by business requirements and characterized by a set of concepts and terminology understood by stakeholders in that area.
Core asset	The basis of a the product line: reference architecture, reusable software components, domain models, requirements, documentation, performance models, test plans, test cases, ...
Platform	Core assets common to all products.
Product-Line	A set of systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment and that are developed from a common set of core assets.
Product-Line Engineering	Paradigm to develop software applications using platforms.

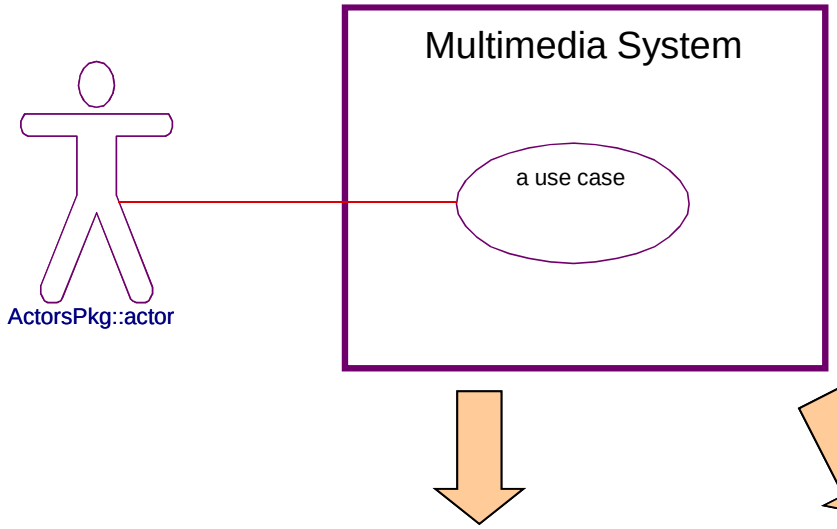
Platform and Product Concept Overview



Platform and Product
Requirements, Features & Planning goes to Architecture

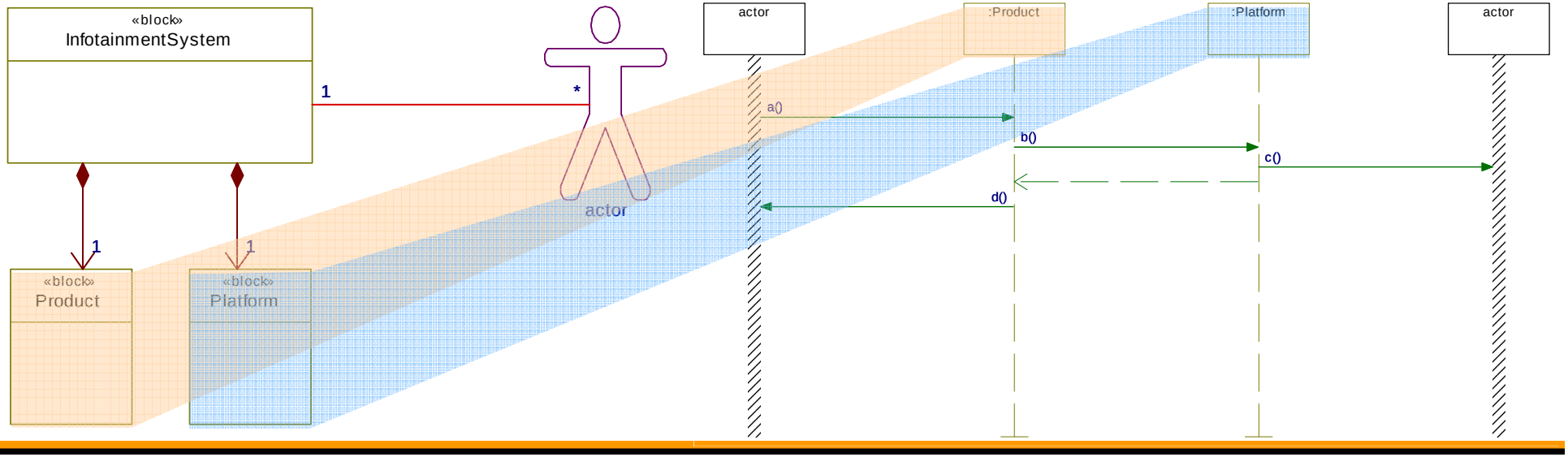


Platform and Product Concept Overview in UML

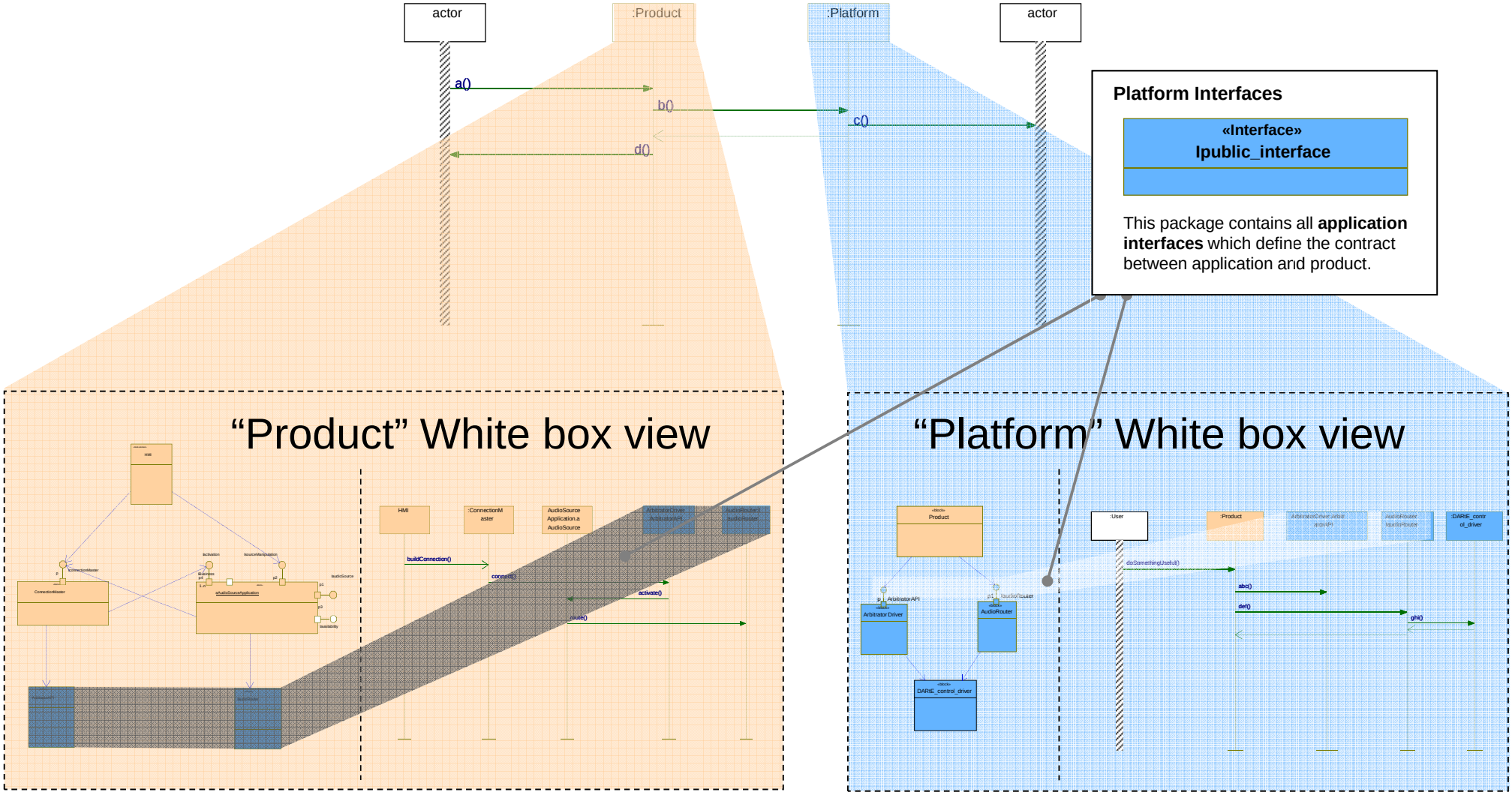


Functional analysis:

- Use cases are shown in the context of the system.
- System is composed of two black boxes.
- Use cases are realized in black box view.
- Flows between actors, product and platform are shown in sequence diagrams.



Platform and Product
Concept Overview in UML

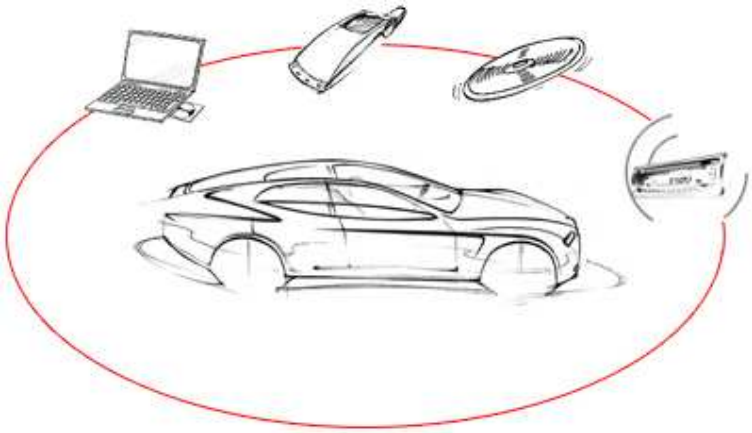


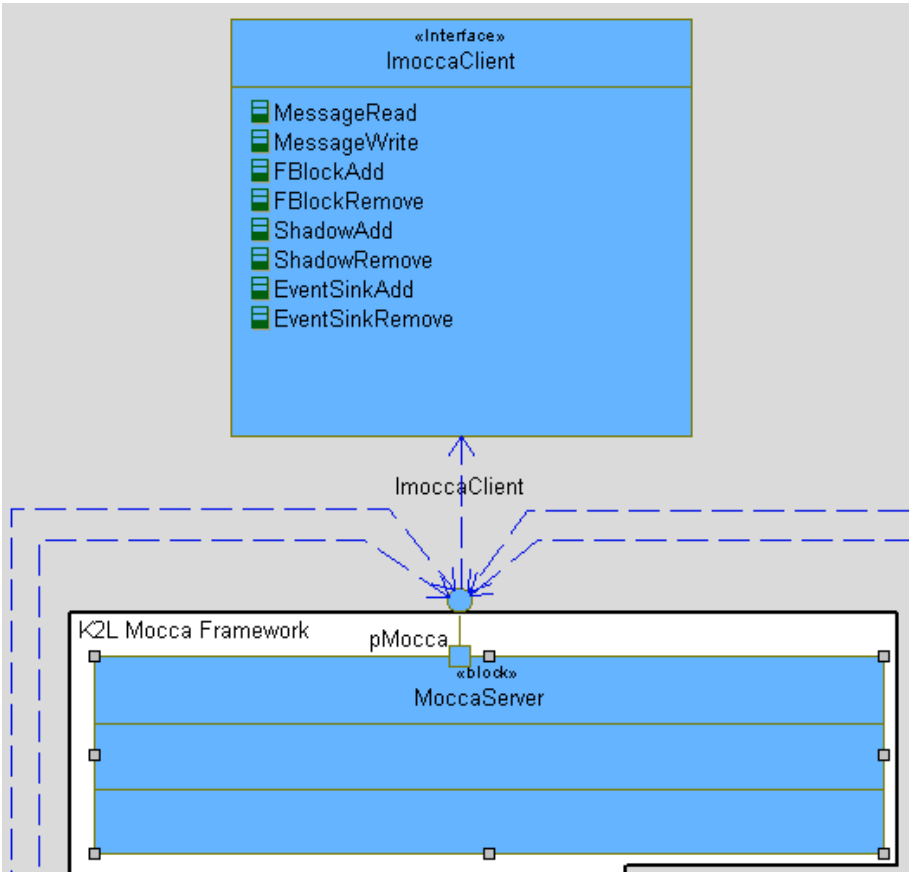
Multimedia Automotive Protocols

MOST (Most Oriented System Transport) ✓

Networking protocol which provides an optical solution for interconnecting automotive multimedia.

- Ring topology
- Wide application range
- Asynchronous and synchronous data transfer
- Plug and Play feature enabling easy adding and removing of devices





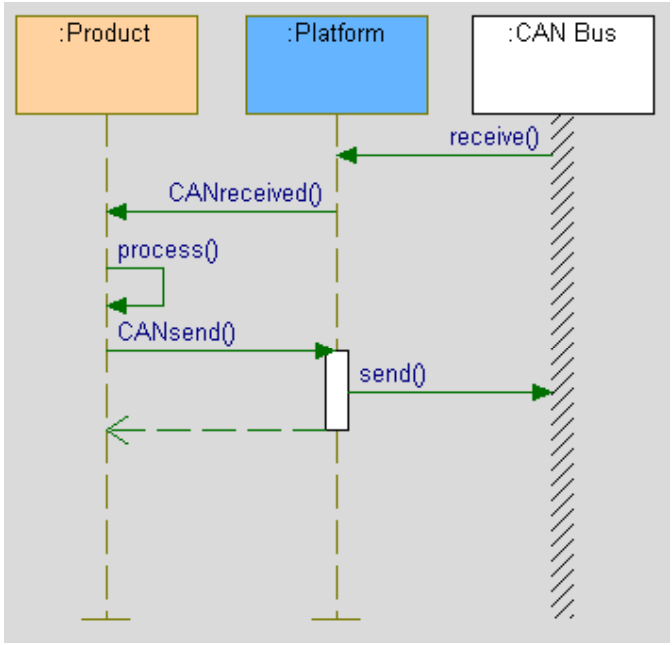
Multimedia Automotive Protocols

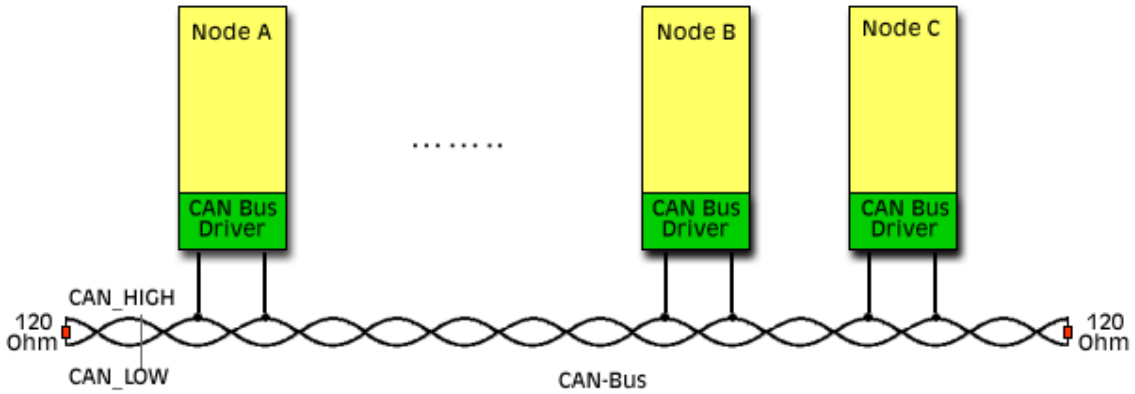
CAN (Controller Area Network) ✓

Serial communication protocol which supports distributed real time control with a very high level of security.

- mid-range speed (up to 1Mbps)
- single channel, dual wire
- prioritization of messages
- time synchronization
- error detection and signaling
- automatic retransmission

Widely used not only in cars, but also in many industrial applications.





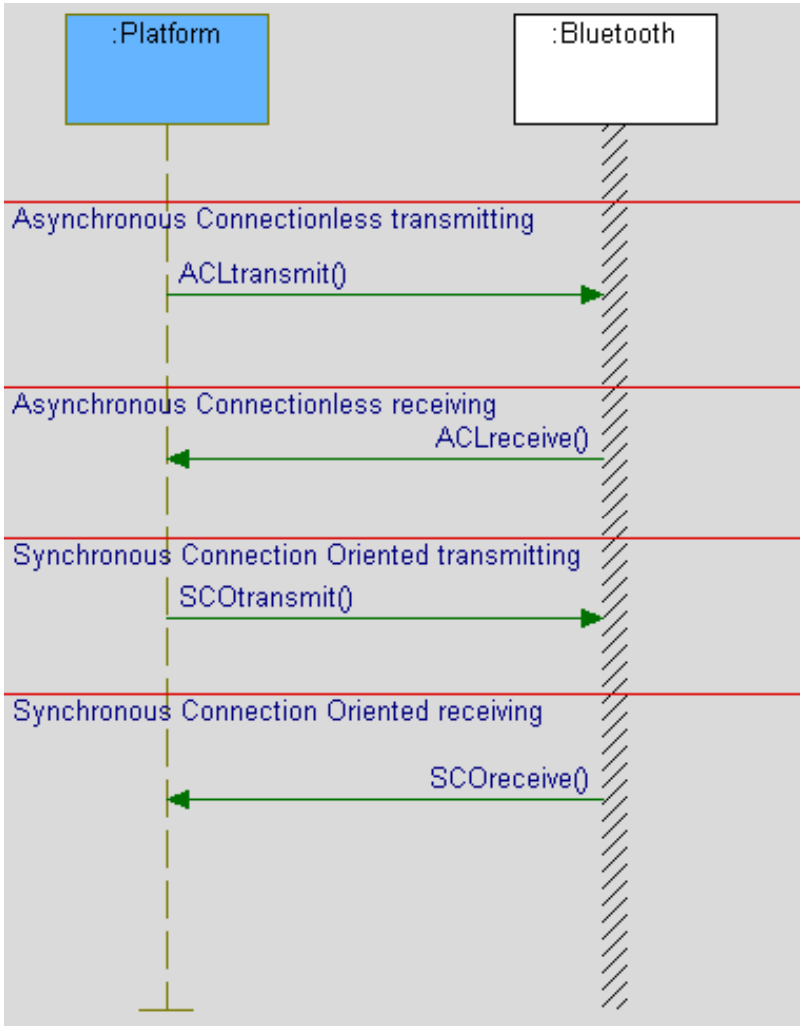
Multimedia Automotive Protocols

Bluetooth

✓

Protocol which provides a way of exchanging information between wireless devices (PDAs, mobile phones, laptops, headsets).

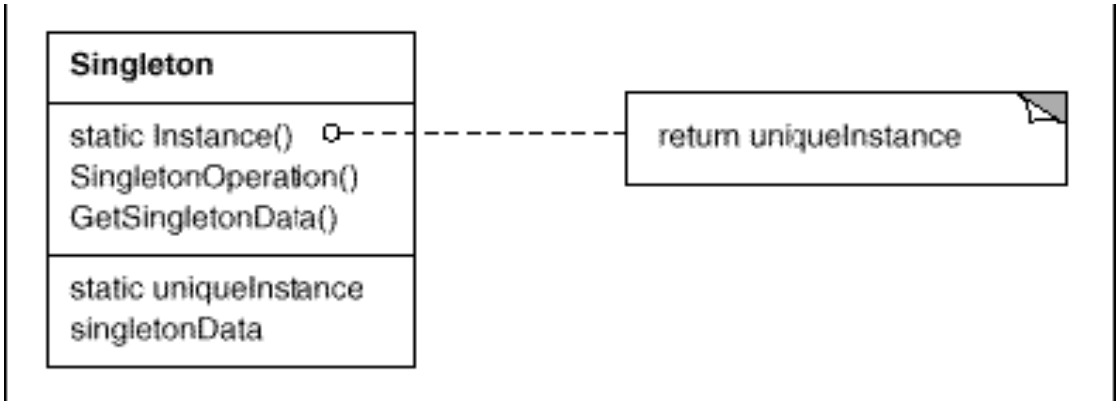
- short-range radio frequency
- based on profiles
- low-cost transceiver microchips



Design & Patterns

- What are the most common patterns used in Automotive ?
  - Singleton and Factory
  - Observer
  - Interceptor
  - Decorator
- Why to use them when doing architecture ?
- How are them applied in Automotive Architecture ?

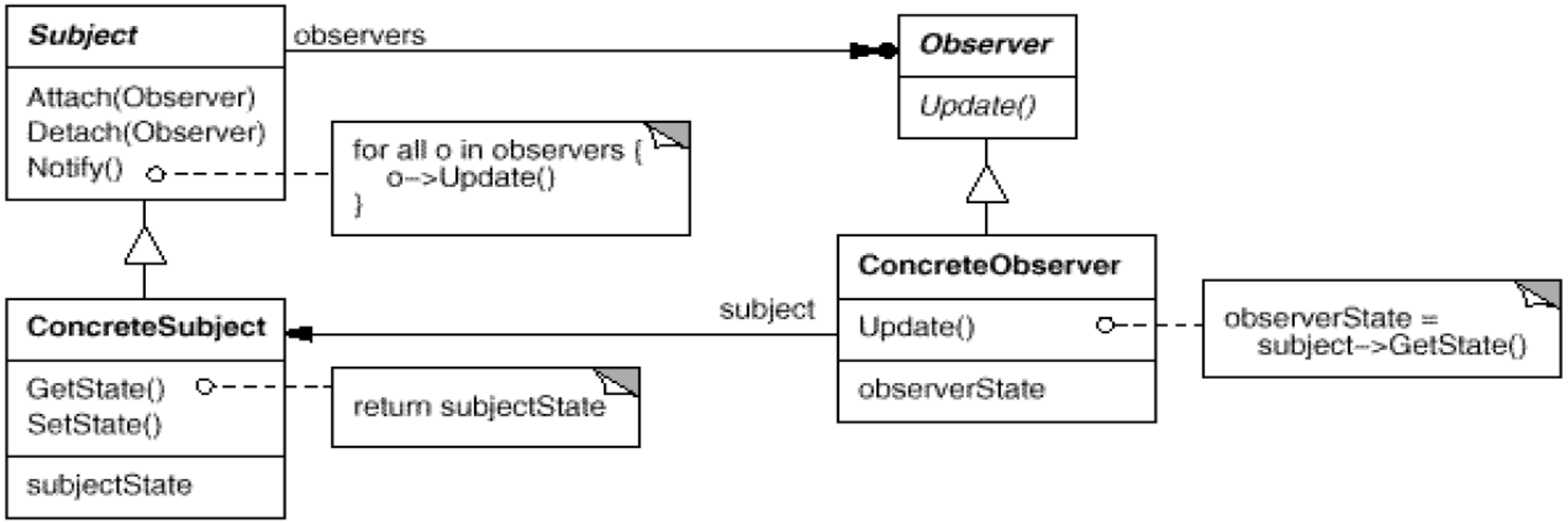
Design & Pattern - Singleton



- **The class of the single instance object** - responsible for creation, initialization, access, and enforcement. Declare the instance as a private static data member. Provide a public static member function that encapsulates all initialization code, and provides access to the instance.

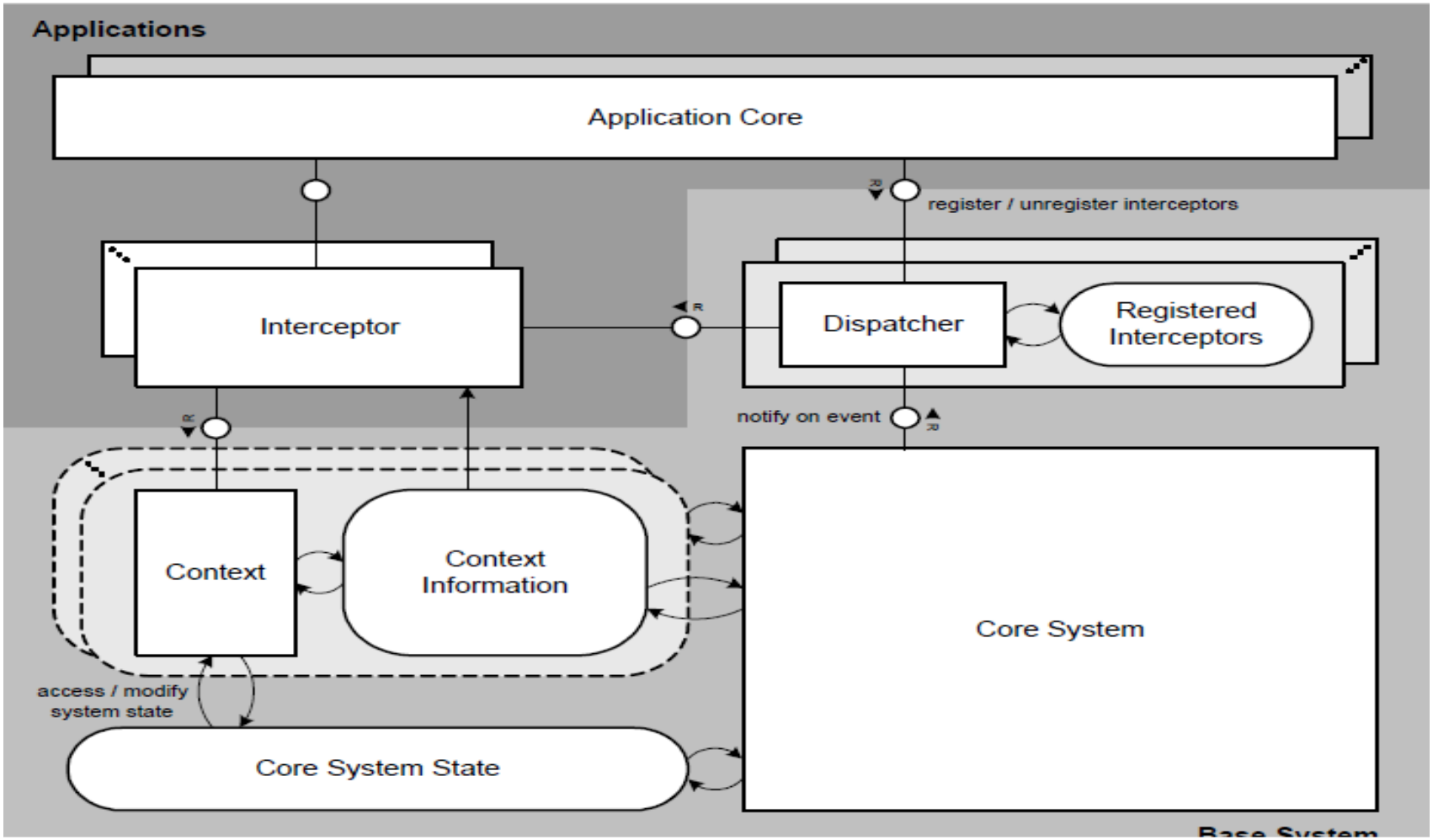
- **The client** - calls the accessor function - (using the class name and scope resolution operator) whenever a reference to the single instance is required.

Design & Pattern : Observer

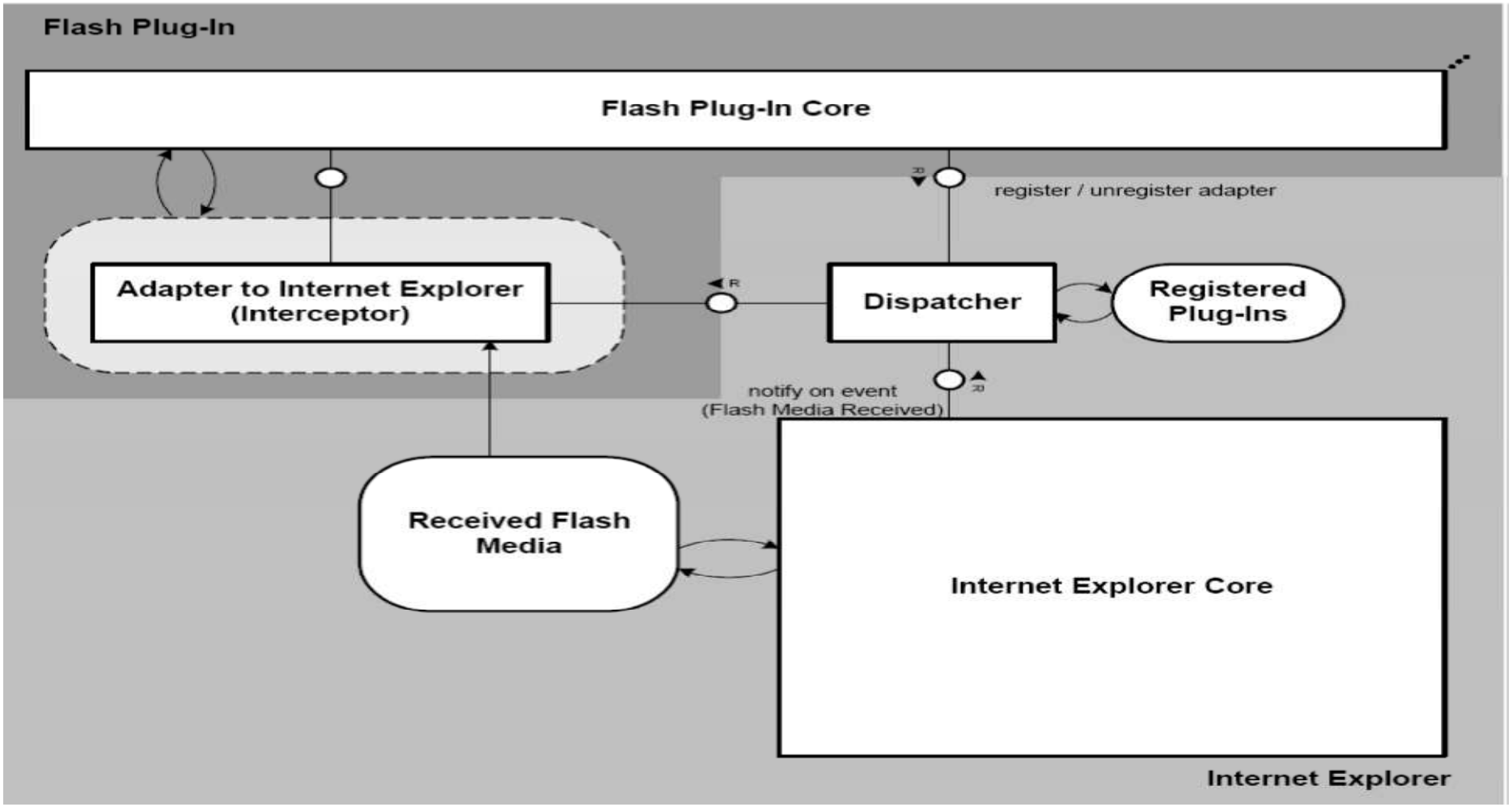


- A ***subject*** may have any number of dependent ***observers***.
- All ***observers*** are notified whenever the ***subject*** undergoes a change in state. Synchronization procedure follows.
- This kind of interaction is also known as publish-subscribe.
  - The ***subject*** is the ***publisher*** of notifications.
  - Any number of ***observers*** can ***subscribe*** to receive notifications.

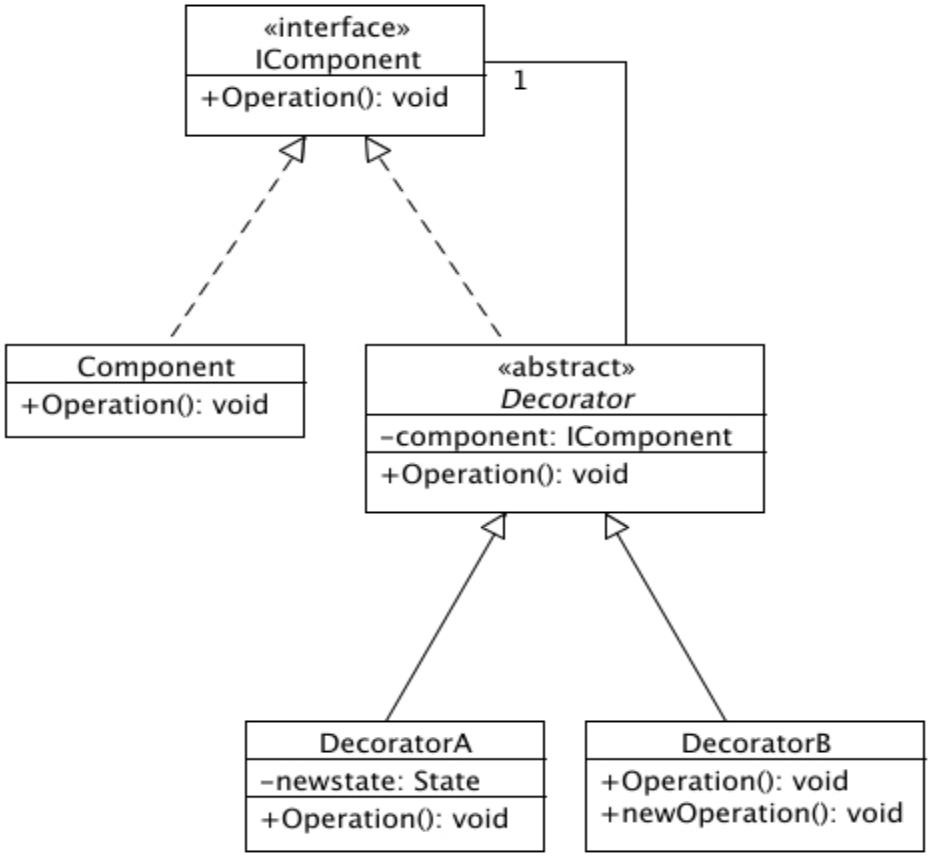
Design& Pattern - Interceptor



Design& Pattern - Interceptor



Design & Pattern - Decorator



- **IComponent**
  - interface for component to be decorated
- **Component**
  - concrete component to be decorated
- **Decorator**
  - decorator interface: implements IComponent and contains an IComponent
- **DecoratorA**
  - concrete decorator
- **DecoratorB**
  - concrete decorator

Hi SW Architect...what do you do ?

- Create the Concept of Operation documents
- Analyze and review the requirements, sometimes even requirements management
- Create and update the System Architecture (UML)
- Create and update the Software Architecture (UML)
- Quotation for projects
- Estimation for work-packages
- Review the code written by SW Developers
- Help everybody with technical information, even mentoring new people
- Optimize the development process, look for innovations

Where do you find more information? (References)

- Continental's documents
- Internet (Architecture definitions, Design and Patterns)