PHILIPS

sense and simplicity

Code generation with ASD at Philips Healthcare iXR

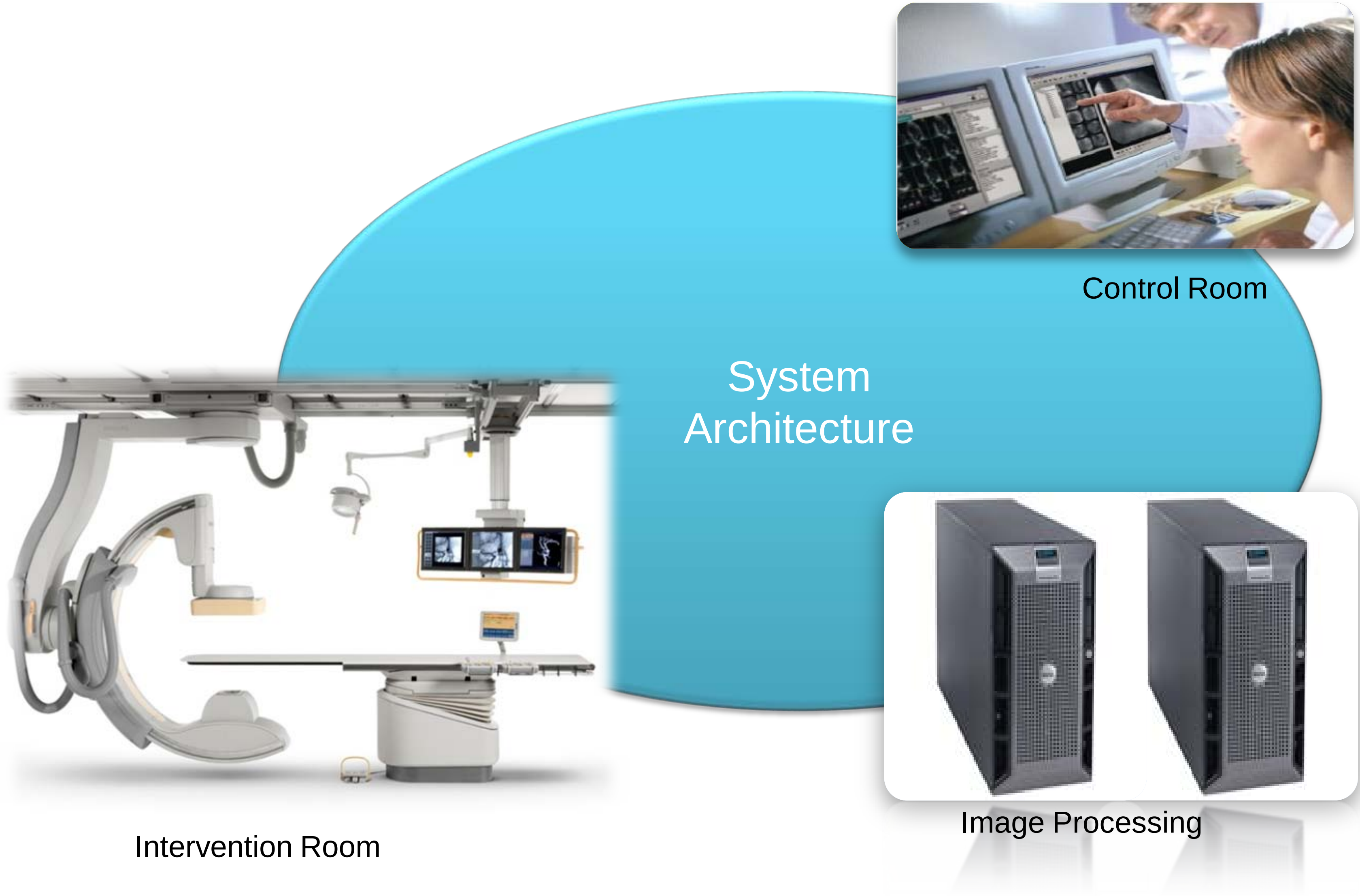
Erik Oerlemans, Hans van Wezep
Philips Healthcare

October 04, 2011

Overview

- Philips interventional X-Ray (iXR): Introduction
- ASD Code Generation applied in:
  - Back End and Front End subsystems
- Experiences and Observations of the FE subsystem

Philips Healthcare: iXR introduction



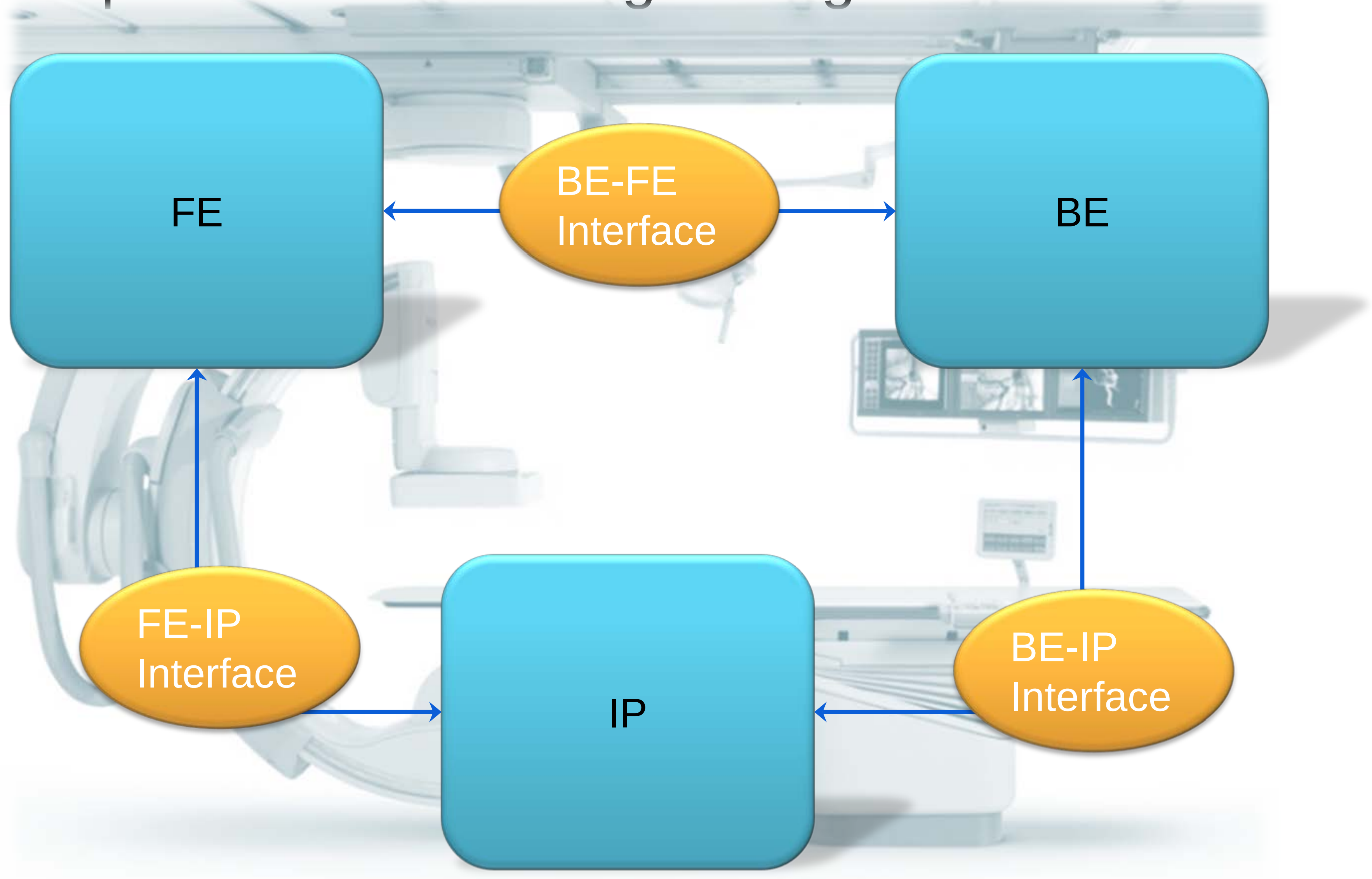
System Architecture

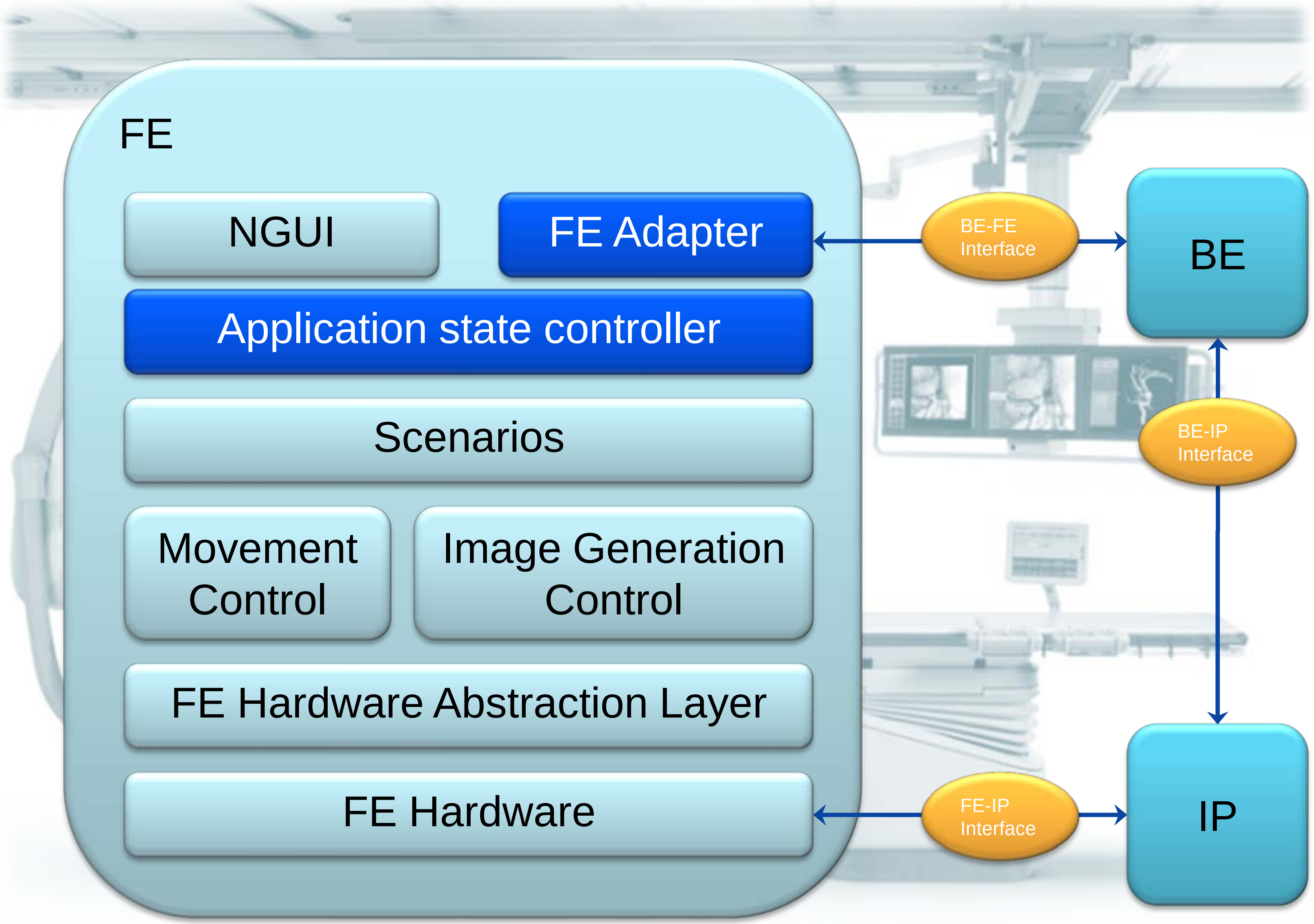
Control Room

Image Processing

Intervention Room

Component-based Design using ASD





FE

NGUI

FE Adapter

Application state controller

Scenarios

Movement Control

Image Generation Control

FE Hardware Abstraction Layer

FE Hardware

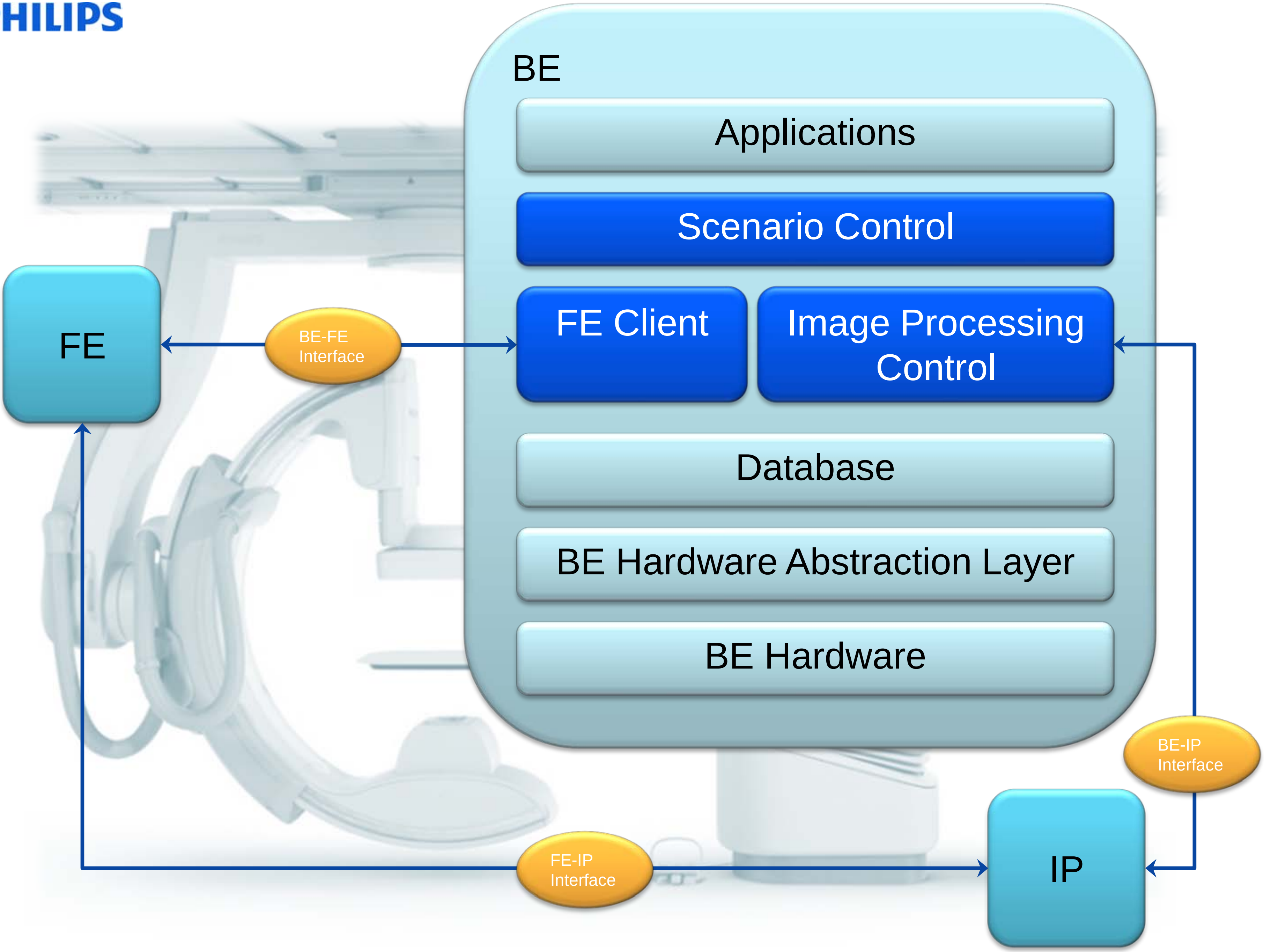
BE-FE Interface

BE

BE-IP Interface

FE-IP Interface

IP

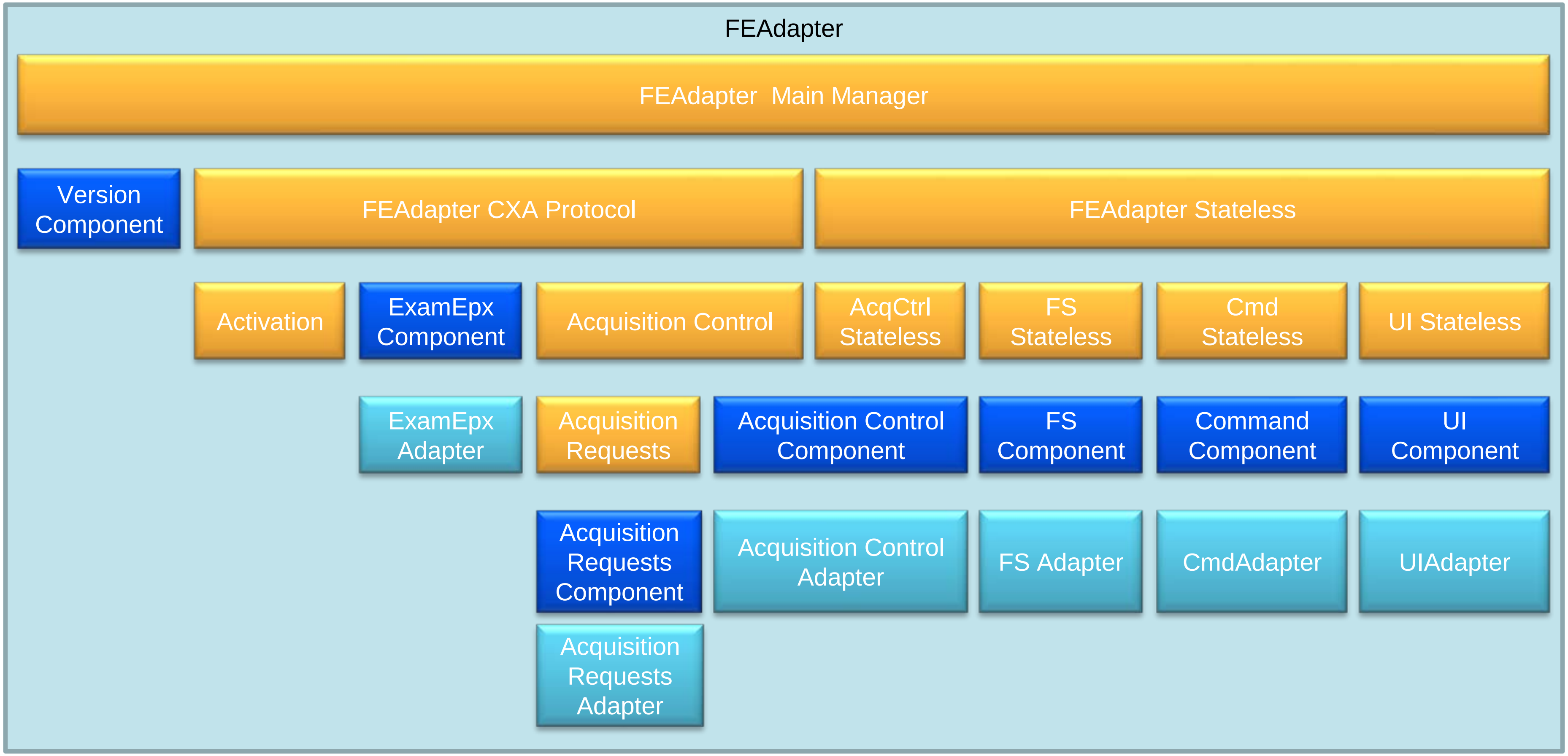

BE
Applications
Scenario Control
FE Client
Image Processing Control
Database
BE Hardware Abstraction Layer
BE Hardware
FE
BE-FE Interface
BE-IP Interface
FE-IP Interface
IP

Some numbers

	Back-End	Front-End
Nbr of models	66	55
Nbr of generated code files	110	72
Nbr of LOCs	36 000 (C#)	46 000 (C++)
Nbr of people	6	4

PHILIPS

FE Adapter - internal design



Legend: 🟧 Fully generated ASD code 🟦 Partly generated ASD code 🟦 Fully hand-written code

Observations - Race conditions

- Model checking frequently shows **race conditions**, showing design problem to deal with “simultaneous”
  - client call and
  - callback from used interface

  which lead to different states

- Forwarding valued calls leads to additional race conditions

ASD Code generation applied

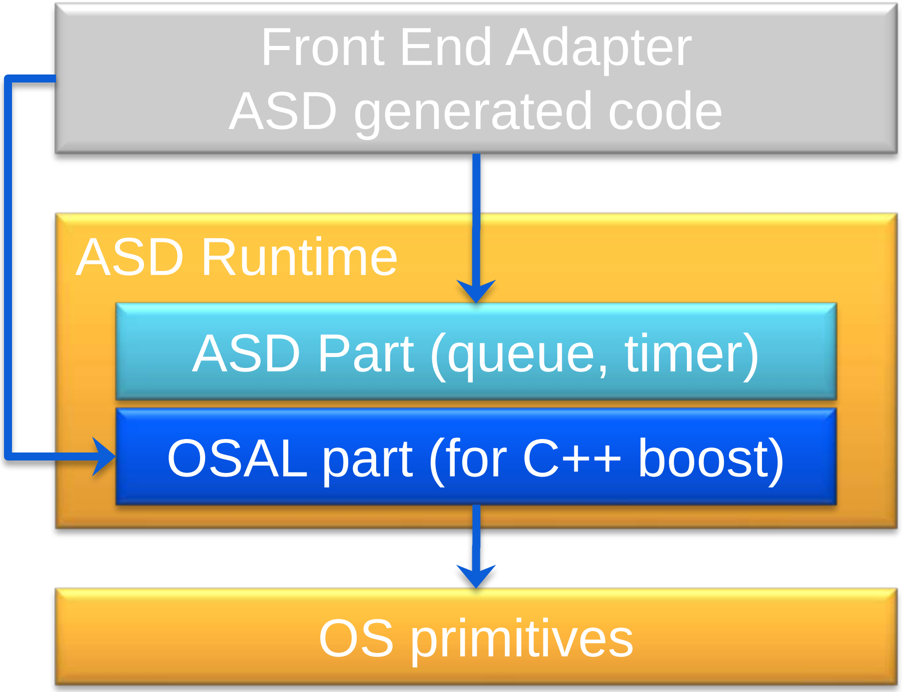
ASD code generation of the Frond End Adapter:

- C++/C# source code is generated from verified ASD Interface and Design Models:
  - Source code is guaranteed to be correct and defect-free (according to the specified ASD models)
  - Source files that are ready to compile, link, and execute.
  - Generation of both singleton and non-singleton components
  - Full support for parameter passing (but no data handling!)

ASD Code generation applied

ASD code generation, additional components:

- A language specific ASD:Runtime software package
- For C++ boost library OS abstraction



Front End Adapter
ASD generated code

ASD Runtime

ASD Part (queue, timer)

OSAL part (for C++ boost)

OS primitives

Observations – Code generation

- Interface models provide type information for API arguments, design models provide the control behavior.
  - As a consequence Interface model is *programming language specific*

- Custom trace and logging handlers are needed to fulfill trace and logging requirements of the system

- Data handling components need to be written

- Wrapper code has to be made to realize IPC between generated components and other (executable) components.

Configuration management approach

- The following items are put into our configuration management tool Clearcase:
    - CXA C# interface assemblies and CXA ASD interface models
    - ASD interface and design models
    - Source code (including the generated ASD source)
    - ASD runtime
    - BOOST library

Observations – Configuration management

- CXA C# interface assemblies and CXA ASD interface models should be placed in a centralized place to avoid duplication and mismatches
  - Is being worked on.

- One button ‘Generate & build’ not in place
  - Main reason is that our build environment is virtualized without any connection to the Internet ; which is required for code generation

PHILIPS

Lessons learned

- Requirements Capturing:
  - Requirements have to be very clear and complete before using the ASD tooling.
  - Our system has a lot of hidden/implicit requirements; these were made explicit for our ASD components.
  - Mindset change; get the needed info, instead of waiting till it is handed. (designer vs. engineer/programmer)

- Design/Modeling:
  - Provide a complete specification of the control behavior of a component (both happy and non-happy flows) at design time.
  - The control behavior of each component can be verified early in the process, and each component can be verified in isolation (separation of concerns).
  - When requirement change, do not be afraid to re-factor your ASD design. It is often easier to start over than to force new requirements in your existing design
  - Designing the interface of a component is different from designing the component itself.

Lessons learned

- Code generation:
  - Code can be generated automatically from ASD design models.
  - Generated code from verified models is guaranteed to be correct.
  - ASD generated code does not ‘fit’ directly into an existing execution architecture.
  - Additional C++ libraries (ASD:Runtime and boost) required.
  - Design of interface determines types in code; interface model is language dependent!!

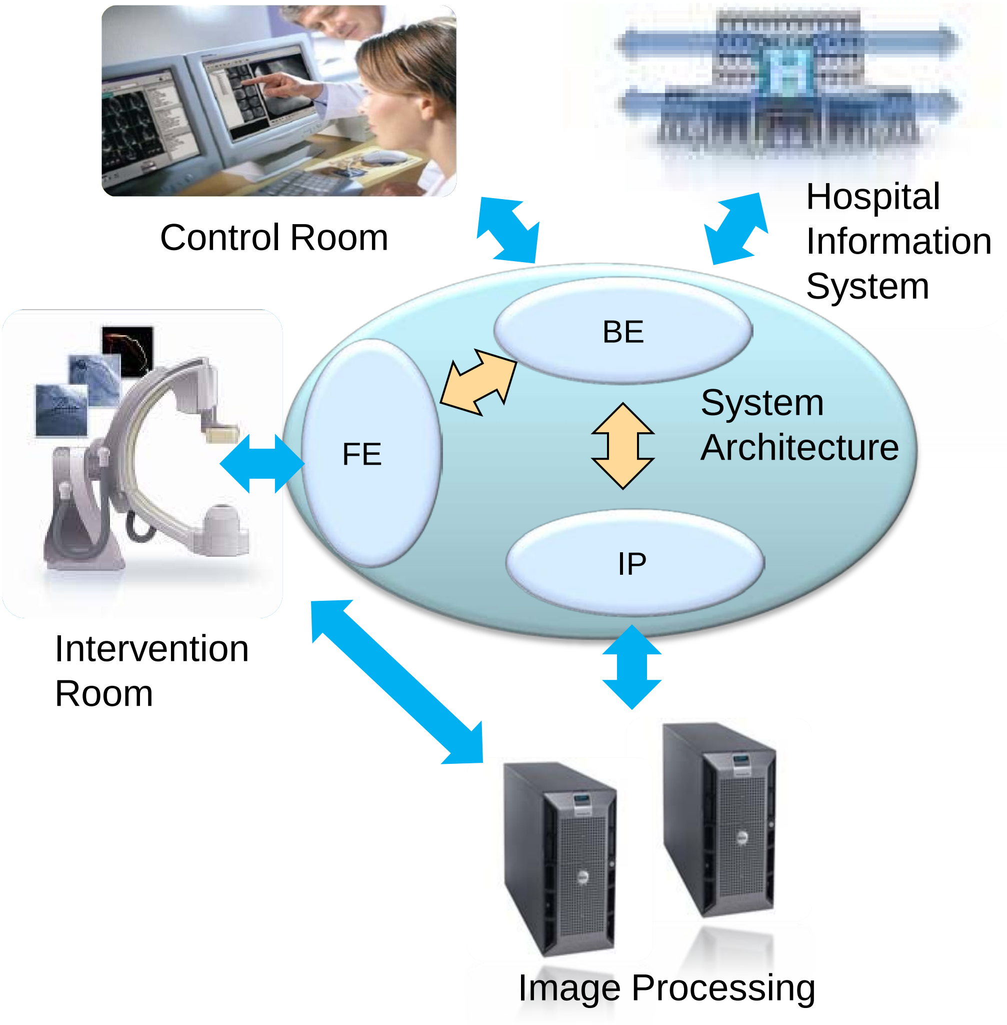
- Testing:
  - Using ASD does not eliminate the need for testing! ‘Foreign’ components still need to be tested as well as their interaction with the ‘pure’ ASD components.

PHILIPS

Philips iXR: Architectural Challenge

Redesign the system architecture:

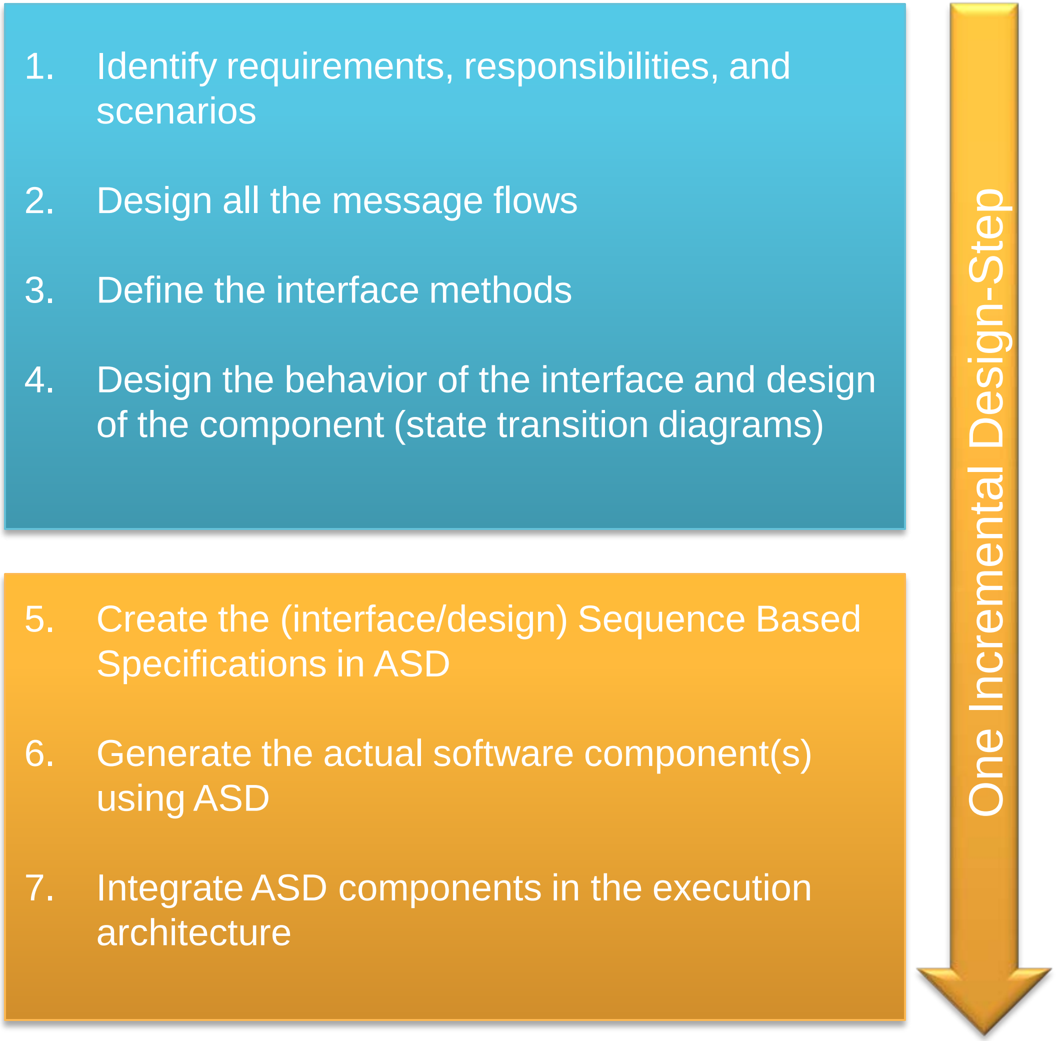
- To cater for over 1 million product variations
- Supporting fast the clinical segments (Cardio, EP, Neuro/Rad, and Surgery)
- That allows for 3rd party suppliers and warm integration with partners offering complementary solutions
- That allows for products that can be serviced for 10 years
- In incremental steps (no revolutionary design from scratch)



Development Process using ASD

- Component-Based Design and the ASD tool forces you to make complete designs.

- Incremental design is still possible as long as you specify the added functionality completely



1. Identify requirements, responsibilities, and scenarios
2. Design all the message flows
3. Define the interface methods
4. Design the behavior of the interface and design of the component (state transition diagrams)
5. Create the (interface/design) Sequence Based Specifications in ASD
6. Generate the actual software component(s) using ASD
7. Integrate ASD components in the execution architecture

One Incremental Design-Step