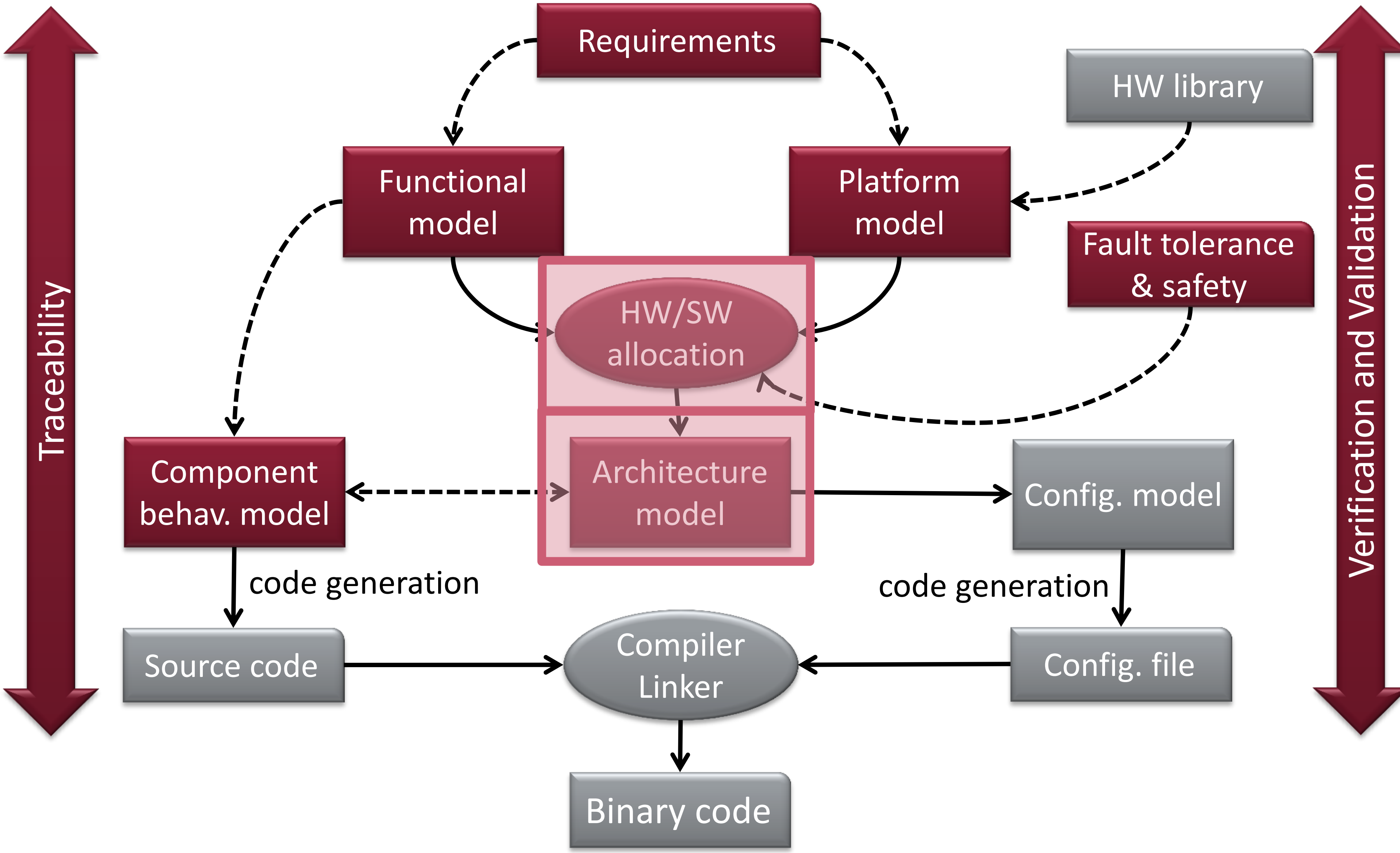
Architecture Languages and Integrated Analysis

Systems Engineering BSc Course

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Platform-based systems design



Learning Objectives

Function-platform allocation

- Summary of extra-functional system properties
- Brief overview of platform modeling in SysML
- Describe allocation specification in the SysML language

Case-studies

- See approaches to capture allocation information on models from different domains
- Analyze extra-functional properties of the integrated allocation model

Architecture Description Languages

ADLs

Abstract

- ”The architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.”
- (no universal agreement on what ADLs should represent)
- Software Architecture in Practice,
  Bass, Clements, and Kazman

Architecture Analysis and Design Language (AADL)

AADL

- Architecture Analysis and Design Language (AADL) is a standard architecture modeling language
  - Avionics
  - Aerospace
  - Automotive
  - Robotics
- Component based notation
  - Task and communication architecture
- Designed for modeling and analysis in mind
- SAE standard (AS 5506A)

- First was called Avionics Architecture Description Language
  - Derived from MetaH created by Honeywell
- V1 version in 2004
- V2 version in 2009

AADL

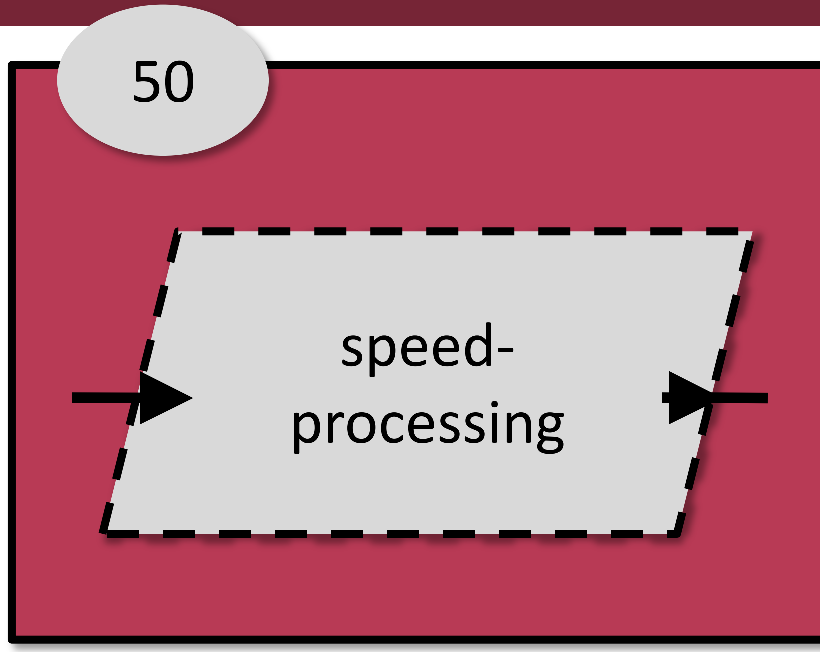
- Based on the component-connector paradigm
- Key Elements:
- Core AADL language standard (V2-Jan,2009, V1-Nov 2004)
  - Textual & graphical, precise semantics, extensile
- AADL Meta model & XMI/XML standard
  - Model interchange & tool interoperability
- Annexes Error Model Annex as standardized extension
  - Error Model Annex addresses fault/reliability modeling, hazard analysis
- UML 2.0 profile for AADL
  - Transition path for UML practitioner community via MARTE
- EMF representation also available (without EFeatureMap!)

AADL

- Precise execution semantics for components
  - Thread, process, data, subprogram, system, processor, memory, bus, device, virtual processor, virtual bus
- Continuous control & event response processing
  - Data and event flow, synchronous call/return, shared access
  - End-to-End flow specifications
- Operational modes & fault tolerant configurations
  - Modes & mode transition
- Modeling of large-scale systems
  - Component variants, layered system modeling, packaging, abstract, prototype, parameterized templates, arrays of components and connection patterns
- Accommodation of diverse analysis needs
  - Extension mechanism, standardized extensions

AADL Representation Forms

```
thread speed_processing
features
        raw_speed_in: in
data port;
        speed_out: out data
port;
properties
        Period => 50 ms;
end data_processing;
```




```
<ownedThreadType name=„speed_processing">
    <ownedDataPort name="raw_speed_in"/>
    <ownedDataPort name="speed_out" direction="out"/>
    <ownedPropertyAssociation property="Period"
        <ownedValue xsi:type="aadl2:IntegerLiteral"
        value=‟50" unit="ms"
        </ownedValue>
    </ownedPropertyAssociation>
</ownedThreadType>
```

AADL Language Elements

- Core modeling
  - Components
  - Interactions
  - Properties
- Engineering support
  - Abstractions
  - Organization
  - Extensions
- Infrastructure

- Strong modeling capabilities for embedded SW and Computer systems

AADL Components

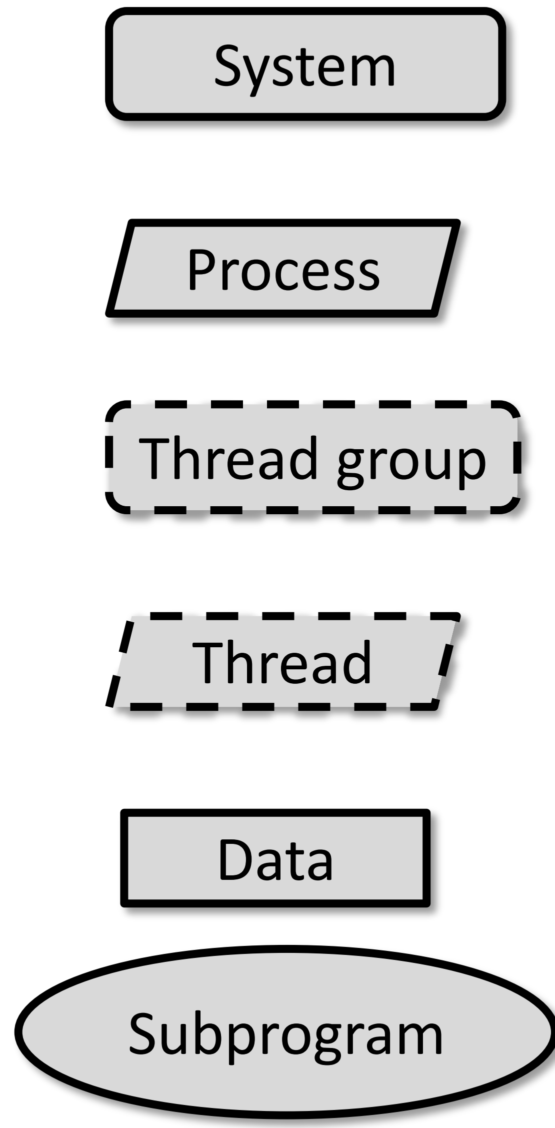
- Top element **system**

Example:

package F22Package
  public
   system F22System
   end F22System;
   system WeaponSystem
   end WeaponSystem;
   system implementation F22System.impl
    subcomponents
     weapon: **system** WeaponSystem;
   end F22System.impl;
end F22Package;

AADL SW Components

- **System** – hierarchical organization of components
- **Process** – protected address space
- **Thread group** – logical organization of threads
- **Thread** – a schedulable unit of concurrent execution
- **Data** – potentially sharable data
- **Subprogram** – callable unit of sequential code



AADL SW Components

- **Process**
  - Protected virtual address space
  - Contains executable program and data
  - Must contain 1 thread
- **Thread**
  - Concurrent tasks
  - Periodic, aperiodic, sporadic ,background, etc.
  - Interaction through port connection, subprogram calls or shared data access
  - errors: recoverable, unrecoverable

AADL SW Components

- **Ports and Connections**
  - Data (non queued data), Event (queued signals) or Event data (queued messages)
  - Complex Connection hierarchies through components
  - Timing
  - Feature groups
- **Data**
  - Optional but makes the analysis more precise
- **Flows**
  - Logical flow of data and control

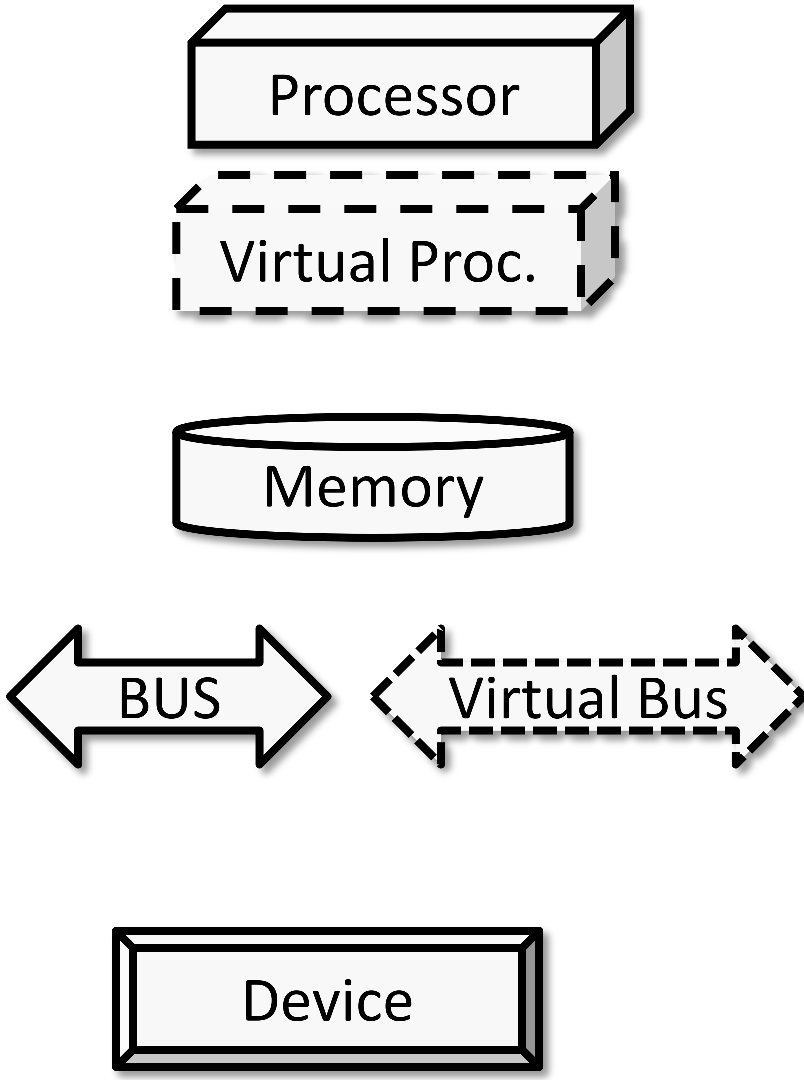
AADL Computer Components

- **Processor / Virtual Processor** – Provides thread scheduling and

- **Memory** – provides storage for data and source code

- **Bus / Virtual Bus** – provides physical/logical connectivity between

- **Device** – interface to external environment



AADL Computer Components

- ”Real” HW components
  - Bus transmission time, latency,
  - Processor timing, jitter
  - Memory capacity
  - Etc.
- Logical resources
  - Thread scheduling of a processor
  - Communication protocol overt network connection (modeled as bus)
  - Transactional memory (modeled as memory)

AADL Computer Components

- **Processor**
  - As HW
    - MIPS rating, size, weight, clock, memory manager
  - As Logical resource
    - Schedule threads → scheduling policies and interruption
    - Execute SW
- **Bus**
  - As HW
    - Physical connection inside/between HW components
  - As logical resource
    - Protocol, which are used for the communication
- **Memory**
  - Processes must be in memory
  - Processors need access to memory
- **Device Components**
  - Represents element that are not decomposed further
  - Sensors/Actuators
  - Device Driver

AADL Binding

- Binding
  - Bringing SW models and the execution platform together
  - Virtual processors → can be subcomponents of other virtual processors → ARINC653 partitioning
  - Hierarchical Scheduling
  - Virtual buses to physical ones
    - One-to-one
    - Many-to-one

Summary

- After 15 years of mainly DoD research it is getting mature enough
- Many pilot project uses AADL
  - FAA
  - DoD
  - Lockheed Martin
  - Rockwell Collins (Steven P. Miller)
- Many research paper on formal analysis, simulation and code generation
- Ongoing harmonization with SysML and MARTE

AUTOSAR

History

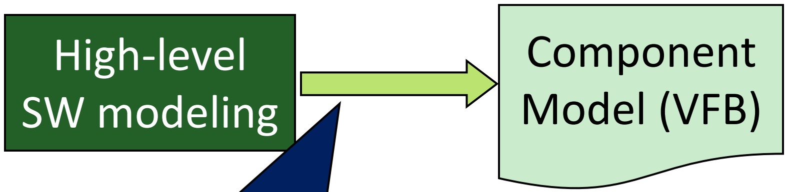
- AUTomotive Open System ARchitecture
- Started in 2002
- BMW, Bosch, Daimler, Conti, VW, + Siemens
- Industrial standardization group
  - Current standard version: 4.0 (end 2009)
  - Currently we use 3.1 (end 2008)
- Members: OEMs, Tool vendors, Semiconductor manufacturers ⍰ Europe-dominated
- Scope
  - Modeling and implementation of automotive systems
  - Distributed
  - Real-time operating system
  - String based interaction with HW and environment
- Out of scope
  - GUI, Java, internet connectivity, File systems, Entertainment systems, USB connectivity etc.

Key Concepts of AutoSAR

- A standard runtime architecture
  - component-oriented
  - layered
  - extensible
    - New functionalities
    - New components (component implementations)
  - all major interfaces standardized
  - Standardized Run Time Environment (RTE)
- A standard modeling and model interchange approach
  - follows the principles of model-driven design
  - supports the interchange of designs
  - supports the collaborative development
    - Between different developers,
    - Teams,
    - And even companies
- Conformance test framework
  - assuring the conformance to the standard
  - Still evolving – new in version 4.0

High-level design flow

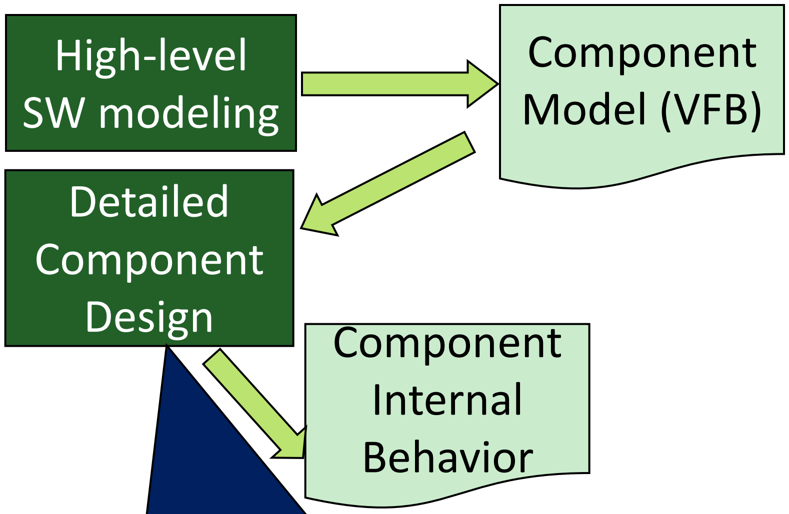
High-level design process



High-level software modeling

- Definition of
  - components
  - component ports
  - port interfaces
  - data types – logical
- Result
  - Virtual Functional Bus (VFB)-level software model

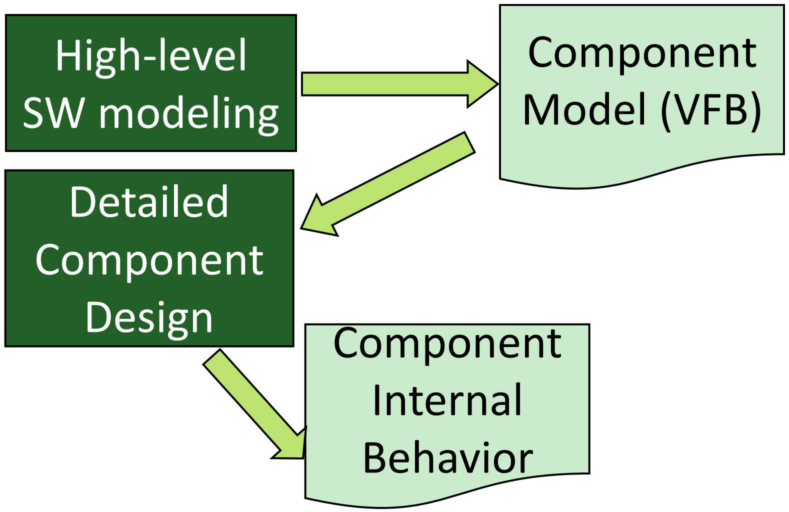
High-level design process

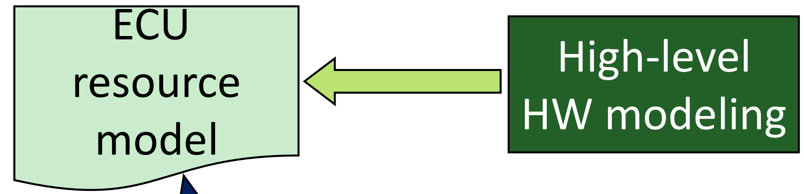


Detailed component design

- Specification of
  - component internal behavior
  - functional breakdown
  - implementation/use of ports
- Non-AutoSAR
  - specification of detailed behavior
  - any tool can be used
    - UML
    - Simulink
    - etc.
- Result
  - AutoSAR component internal behavior model
  - Non-AR: behavioral models/design

High-level design process

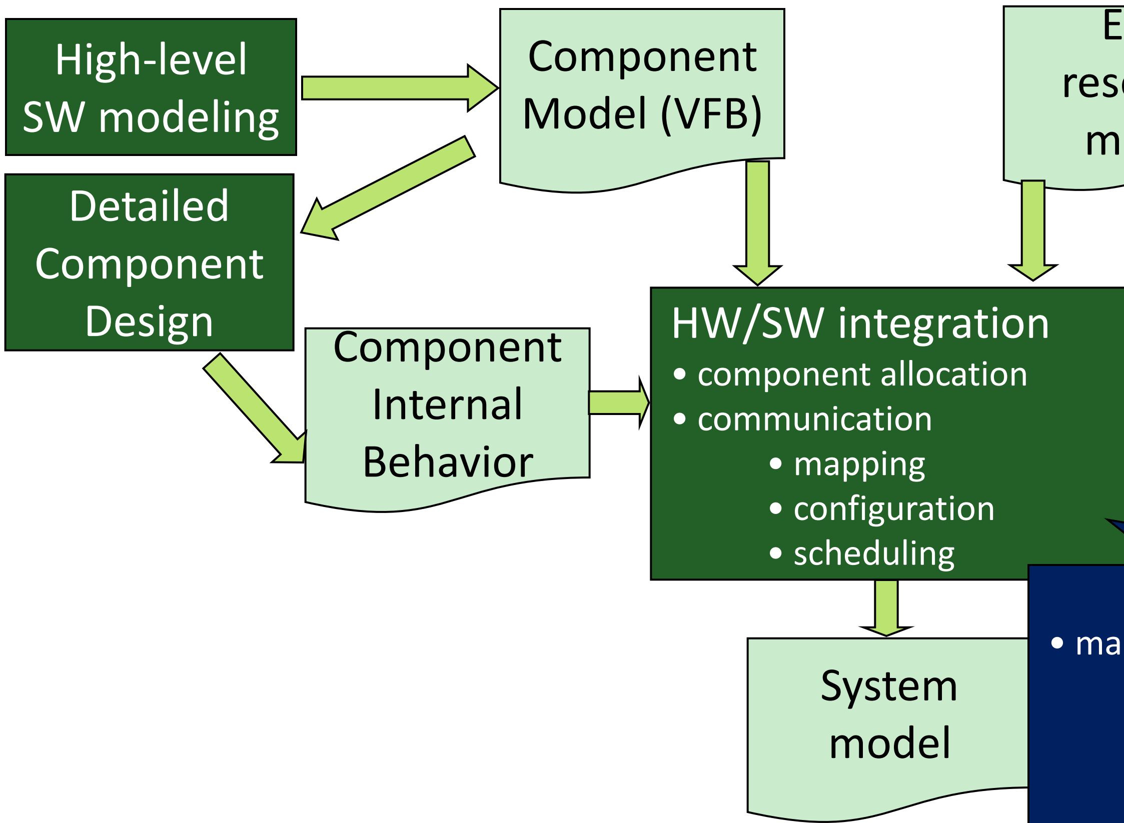




High-level hardware modeling

- Specification of
  - Electronic Control Unit (ECU) resources
    - CPU
    - memories
    - peripherals
    - communication hw
  - system topology
    - ECU instances
    - clusters
    - connections
- Result
  - ECU resource model – for all ECUs
  - System topology model

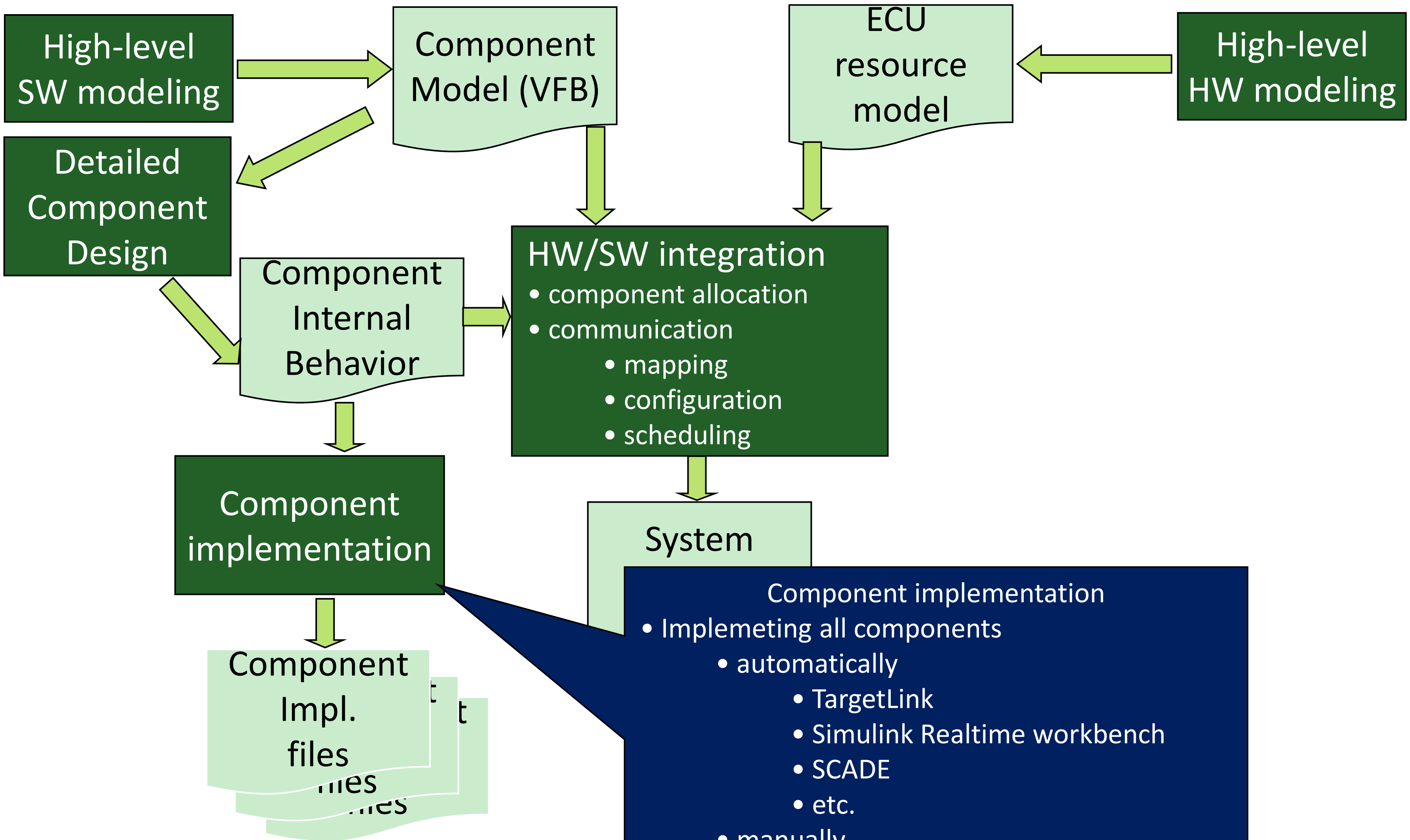
High-level design process



Hardware-software integration

- mapping
  - software component allocation
  - component implementation selection
  - data-element to signal mapping
    - inter-ECU communication
- communication configuration
  - signal to Protocol Data Unit (PDU) mapping
  - PDU to frame mapping
  - Signal, PDU, Frame triggering
  - Cluster and controller configuration
  - Frame scheduling (LIN, FlexRay)
- Result
  - System model describing the integrated HW/SW system

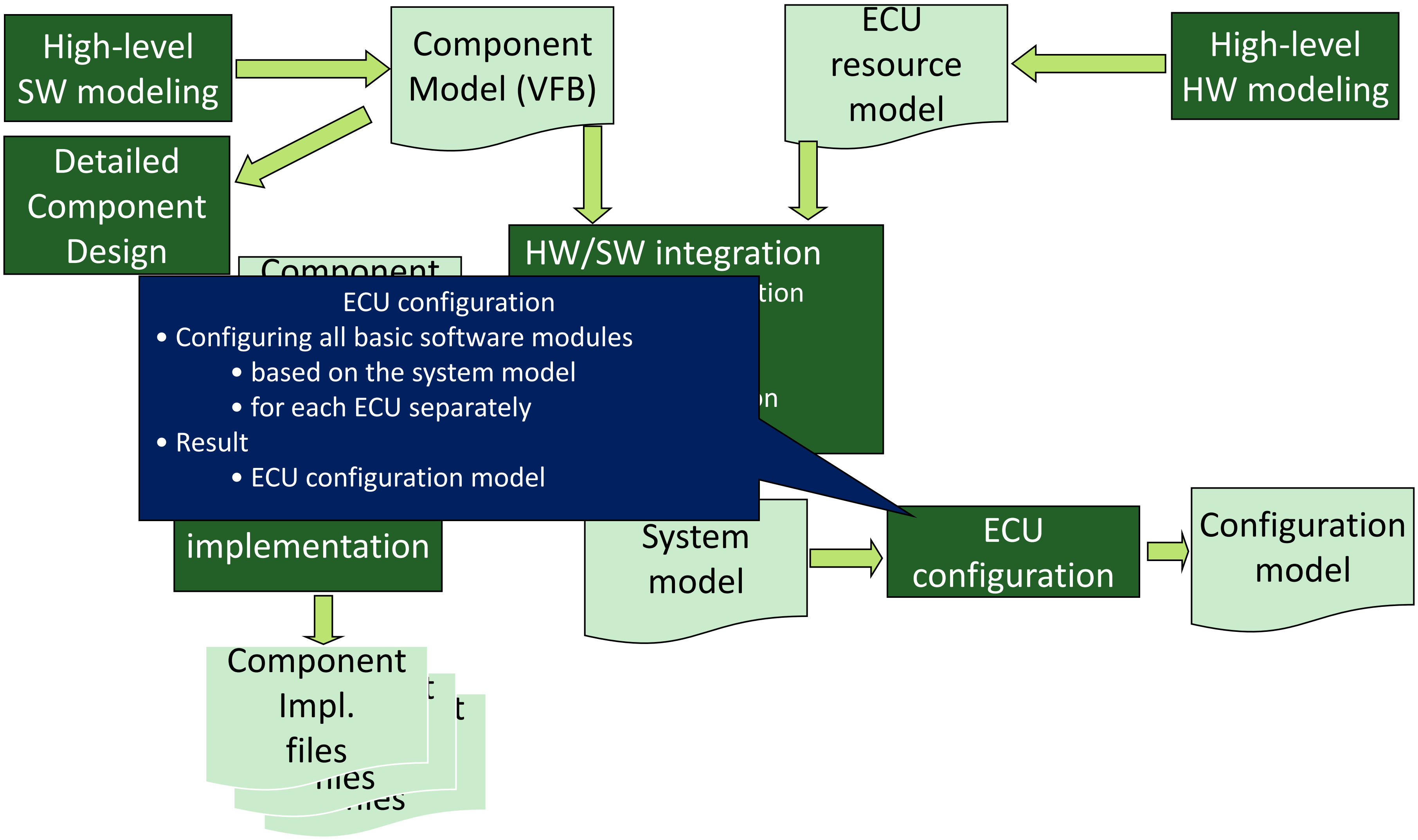
High-level design process



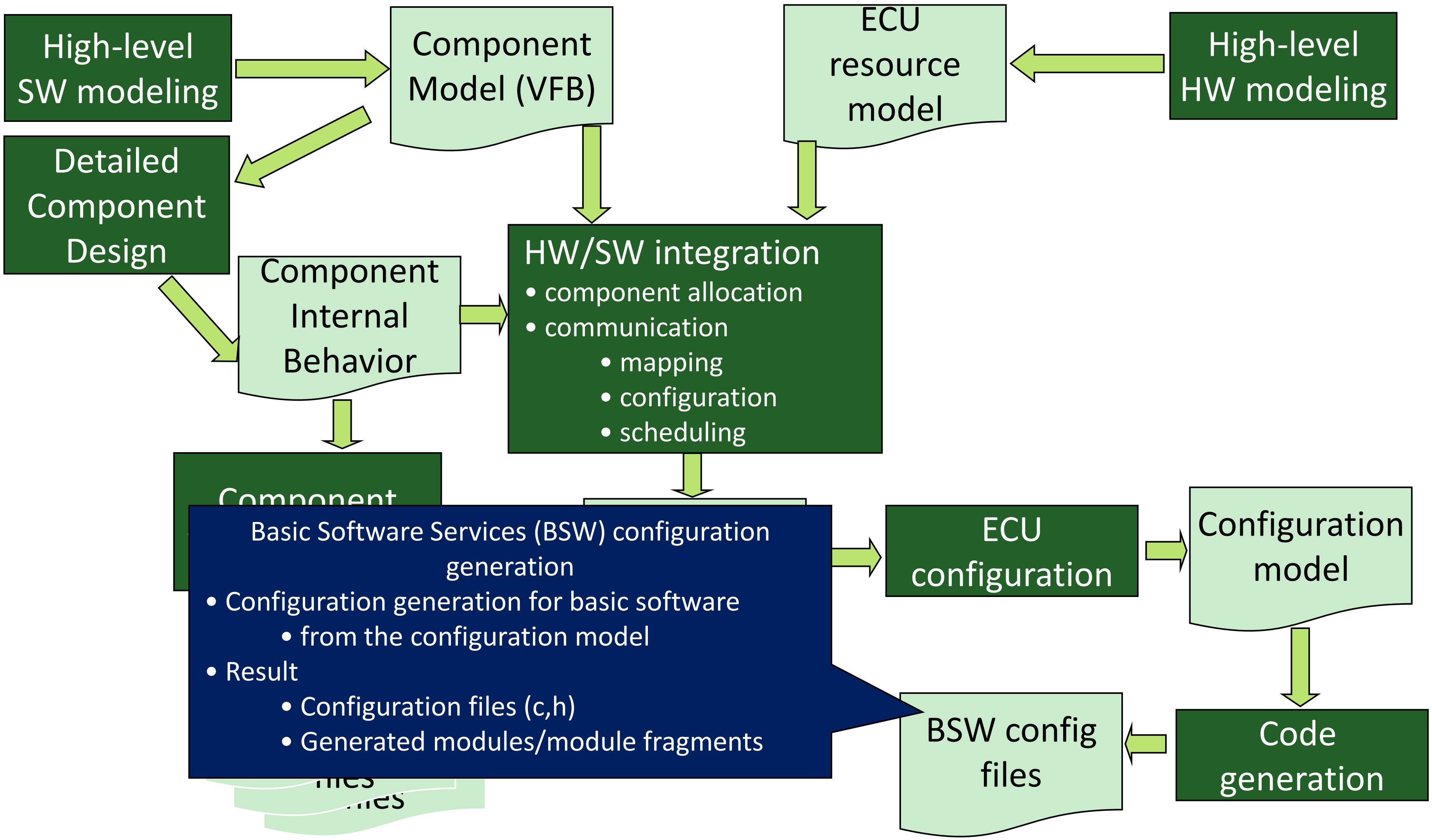
Component implementation

- Implemeting all components
  - automatically
    - TargetLink
    - Simulink Realtime workbench
    - SCADE
    - etc.
  - manually
- Result
  - implementation of the components
    - C/C++/…

High-level design process



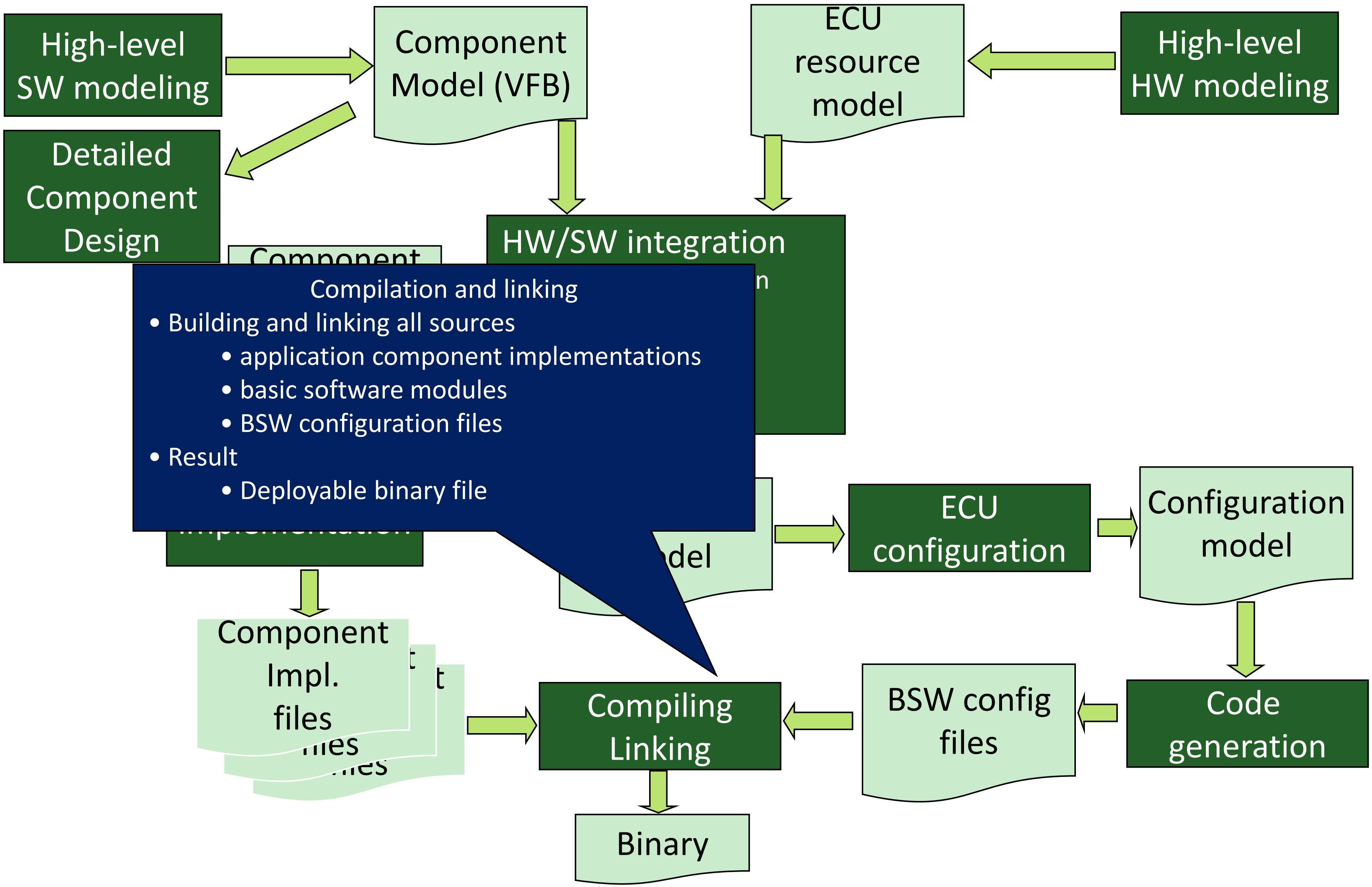
High-level design process



Basic Software Services (BSW) configuration generation

- Configuration generation for basic software
  - from the configuration model
- Result
  - Configuration files (c,h)
  - Generated modules/module fragments

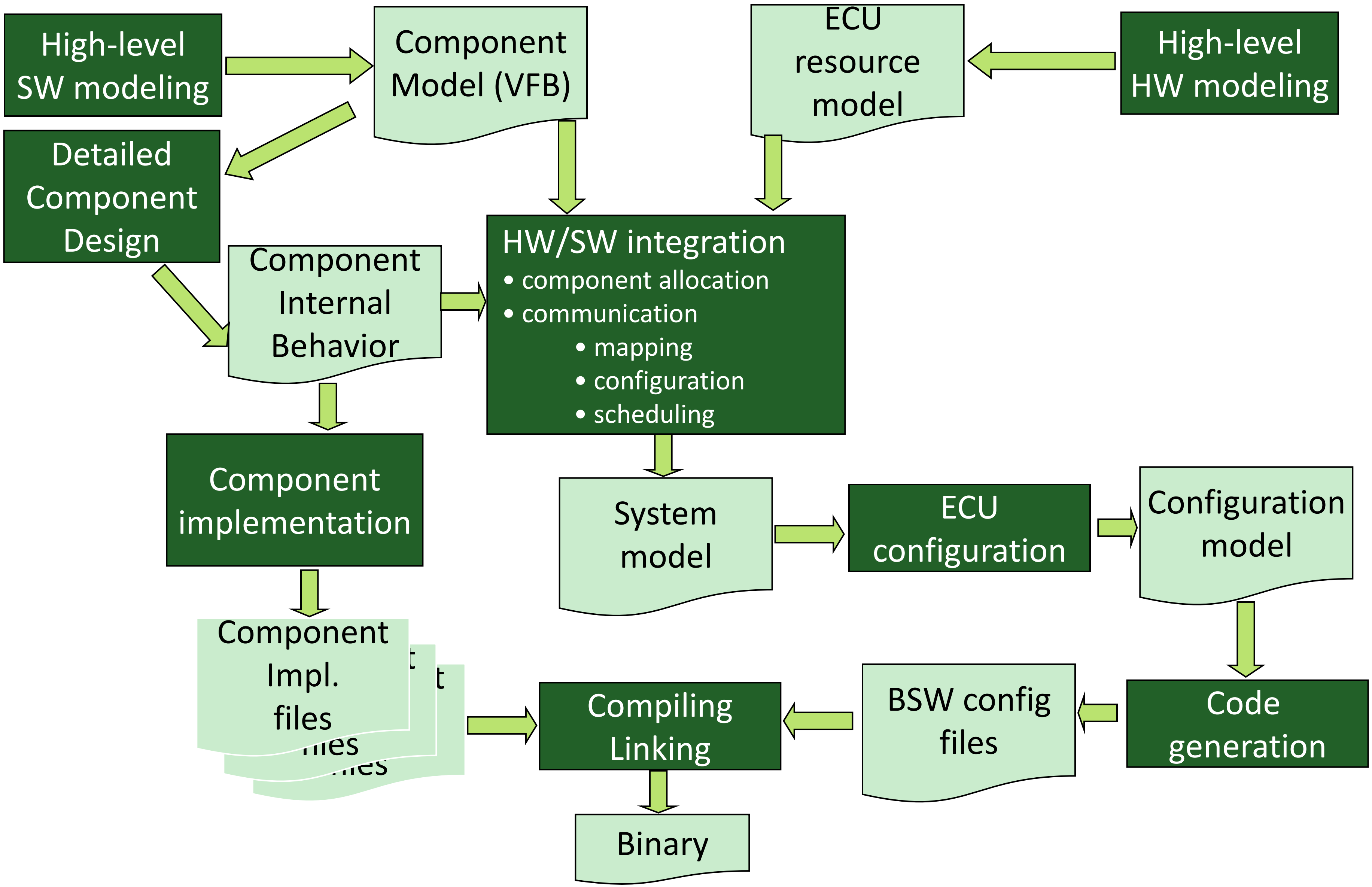
High-level design process



Compilation and linking

- Building and linking all sources
  - application component implementations
  - basic software modules
  - BSW configuration files
- Result
  - Deployable binary file

High-level design process



Models in the design flow

- Software Component Template
  - Components, ports, interfaces
  - Internal behavior
  - Implementation (files, resource consumption, run time, etc.)
- ECU Resource Template
  - Hardware components, interconnections
- System Template
  - System topology, HW/SW mapping
  - Comm. matrix

Models in the design flow 2

- Basic Software Module Template
  - BSW modules
    - Services
    - Schedulable entities
    - Resource consumption
- ECU Configuration Parameter Definition Template
  - Configurable parameters of BSW modules
- ECU Configuration Description Template
  - Actual configurations of BSW modules
  - Based on the ECU Parameter Definition

AUTOSAR vs. UML/SysML/... modeling

- AUTOSAR defines models with
  - Domain Specific Constructs
  - *Precise* syntax
  - Synthesizable constructs
    - Direct model -> transformations
    - Direct model -> detailed model mappings
  - Different abstraction levels
    - From Virtual Function Bus to configuration
- Result
  - Models *are* primary design *and* implementation artifacts
    - More precise, consistent modeling should be done

AUTOSAR Components

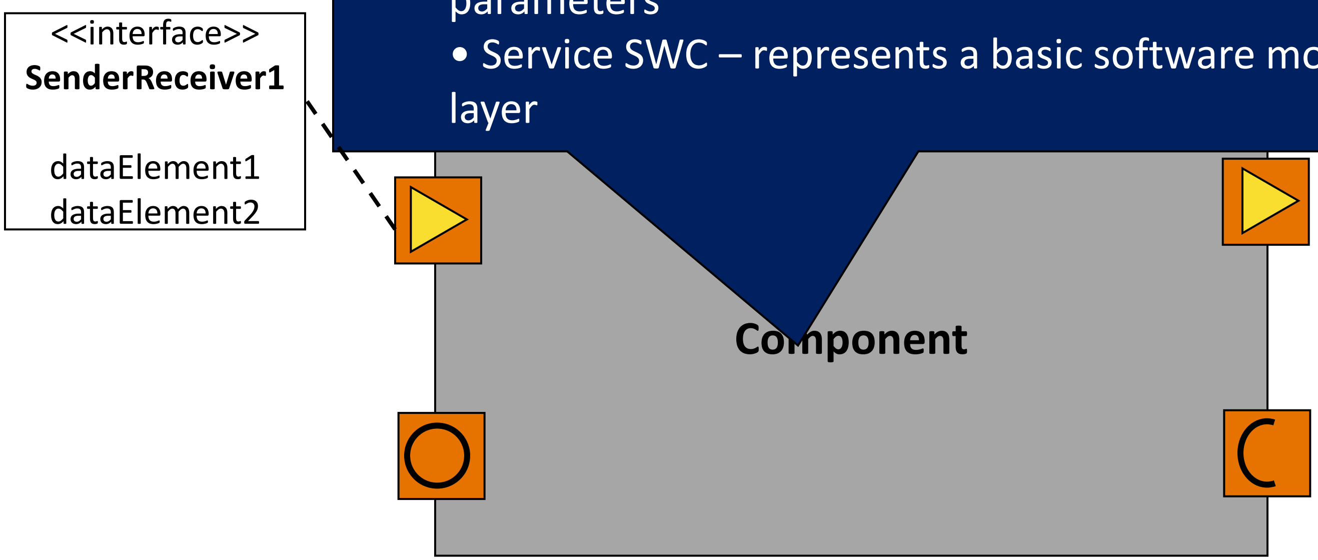
Component-oriented design

- What is a component?
  - *"A component is a self contained, reusable entity that encapsulates a specific functionality (and/or data), and communicates with other components via explicitly defined interfaces."*
- AUTOSAR uses the term *component* for application-level components
  - Elements related to the high-level functionality of the system under design
- Basic software (middleware) components are called *modules*.
  - Standard elements of the AUTOSAR architecture

Component-based approach

Component

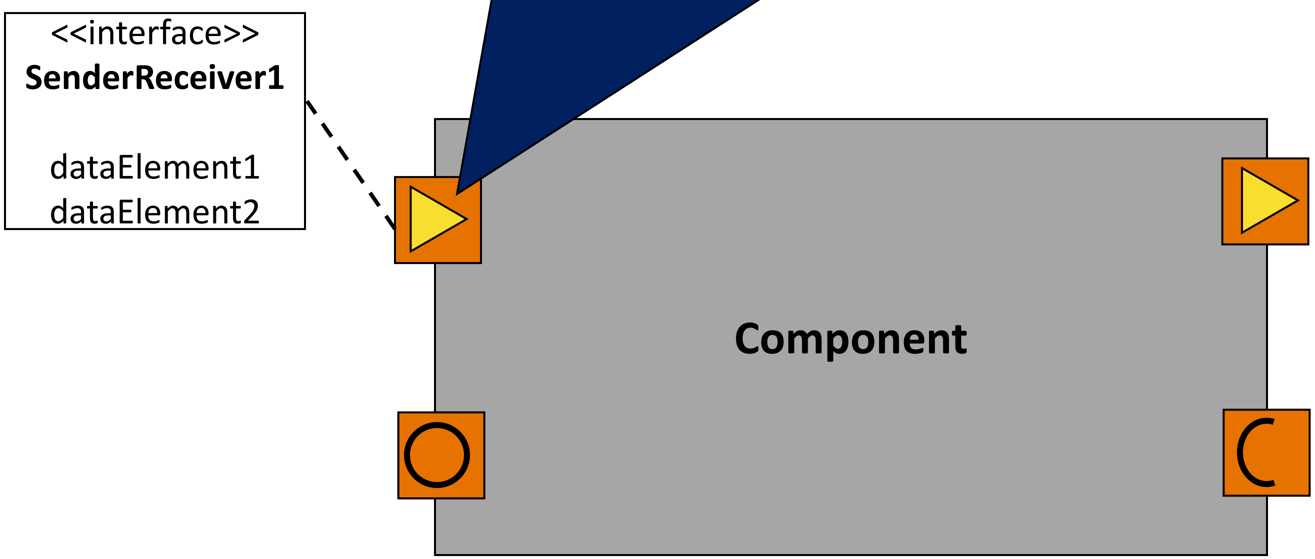
- Encapsulates a specific functionality
- Different kinds
  - Composite component – hierarchical refinement
  - Application SW component – generic, high level functionality
  - Sensor/actuator SW-C – handling sensor or actuator data
  - ECU HW abstraction – higher level device driver and abstraction
  - ComplexDeviceDriver – time-critical, low-level driver
  - Calibration parameter SWC – collects system calibration parameters
  - Service SWC – represents a basic software module from the service layer



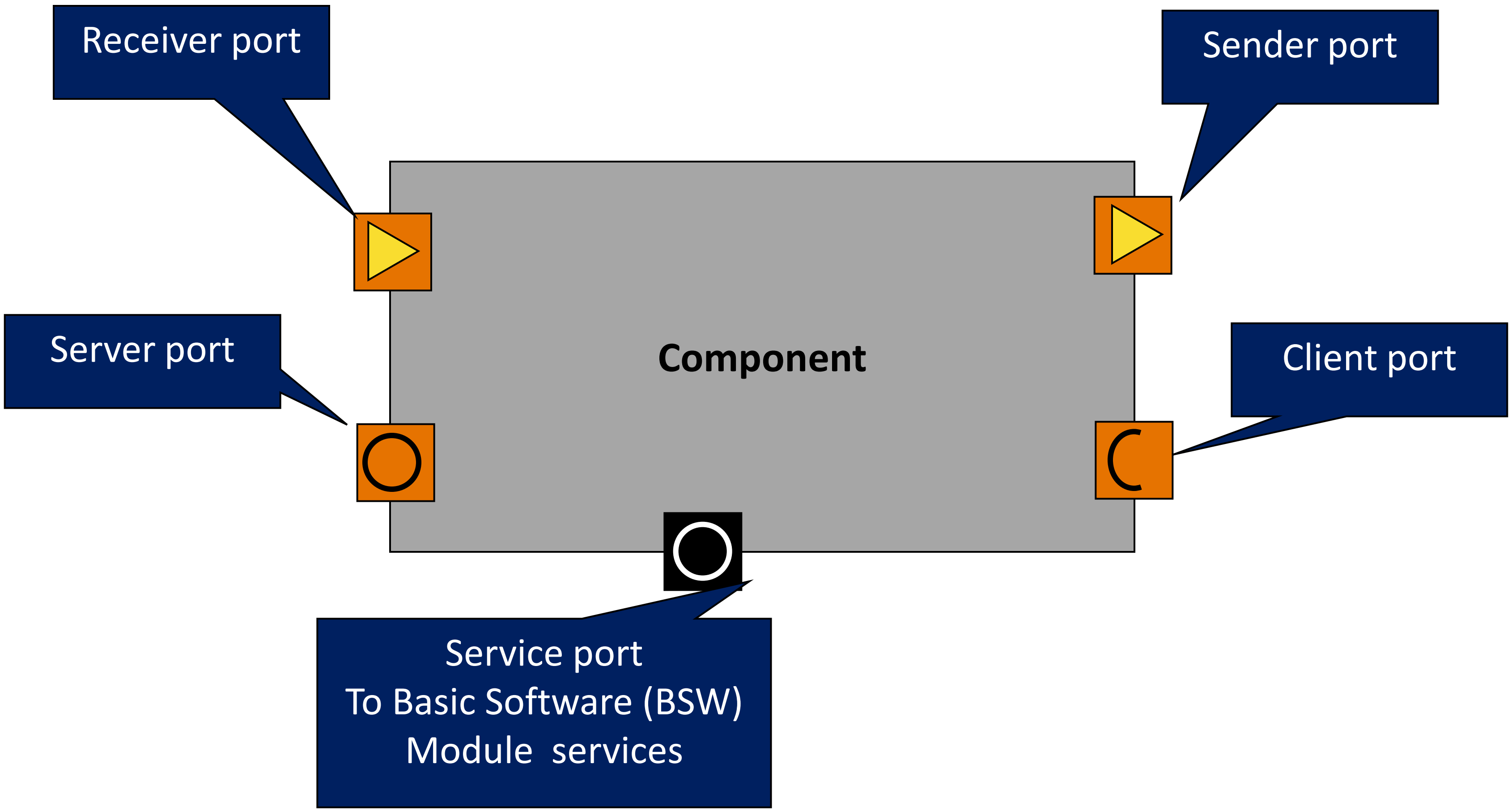
Component-based approach

Ports

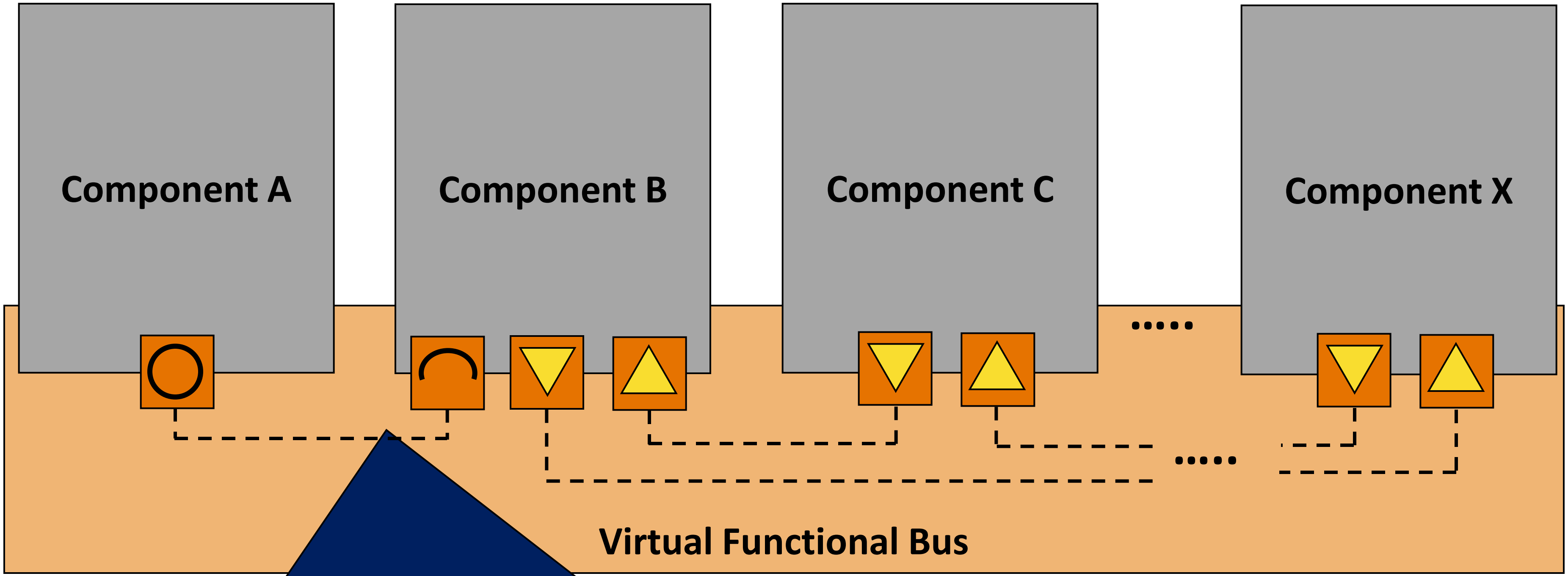
- The only interaction points between the component and its environment
- Are implementing *port interfaces*
    - sender receiver (message-based unidirectional communication)
    - client-server (remote procedure call)



Component-based approach – port notation



Component interconnection – the Virtual Functional Bus



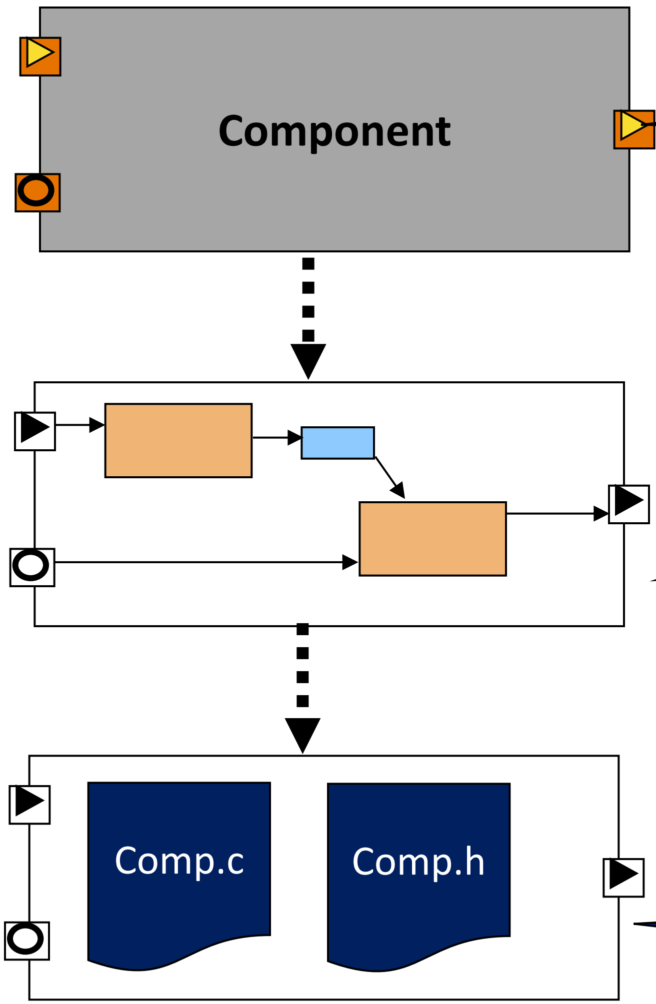
Virtual Functional Bus (VFB)

- Abstract interconnection layer
  - Implementation of data/control transport between components
  - No hardware/network dependency
  - Hides the details of the implementation
- Allows high-level integration *and simulation* of components
  - Before hardware architecture is chosen

Software Components

- On high-level, *atomic* components are black boxes
- Detailed design “looks into” these black boxes
- Main goals
  - Detail the behavior to get schedulable entities
  - Specify the semantics of port handling
  - Specify any service needs
  - Specify any RAM, nvRam needs

Refinement of a component



Black box definition of a component

Definition of component *internal behavior*
Schedulable entities, connections to the ports

Component implementation.
Specification of source and header files

Component internal behavior

- Specification of the internals of an atomic SWC
- Schedulable elements
  - Called: runnable entities
- Connection of ports
  - Port semantics
  - Port API options
- Inter-runnable communication
- Runnable activation and events

Component internal behavior – runnable entities

- Smallest code-fragments considered by RTE
- Subject to scheduling by the OS
- Abstraction of a schedulable function
- Communicates
  - Using the SWC ports
  - Using inter-runnable communication facilities
- Is activated by
  - An RTE event
    - Communication-related event
    - Timing event

Summary of AUTOSAR

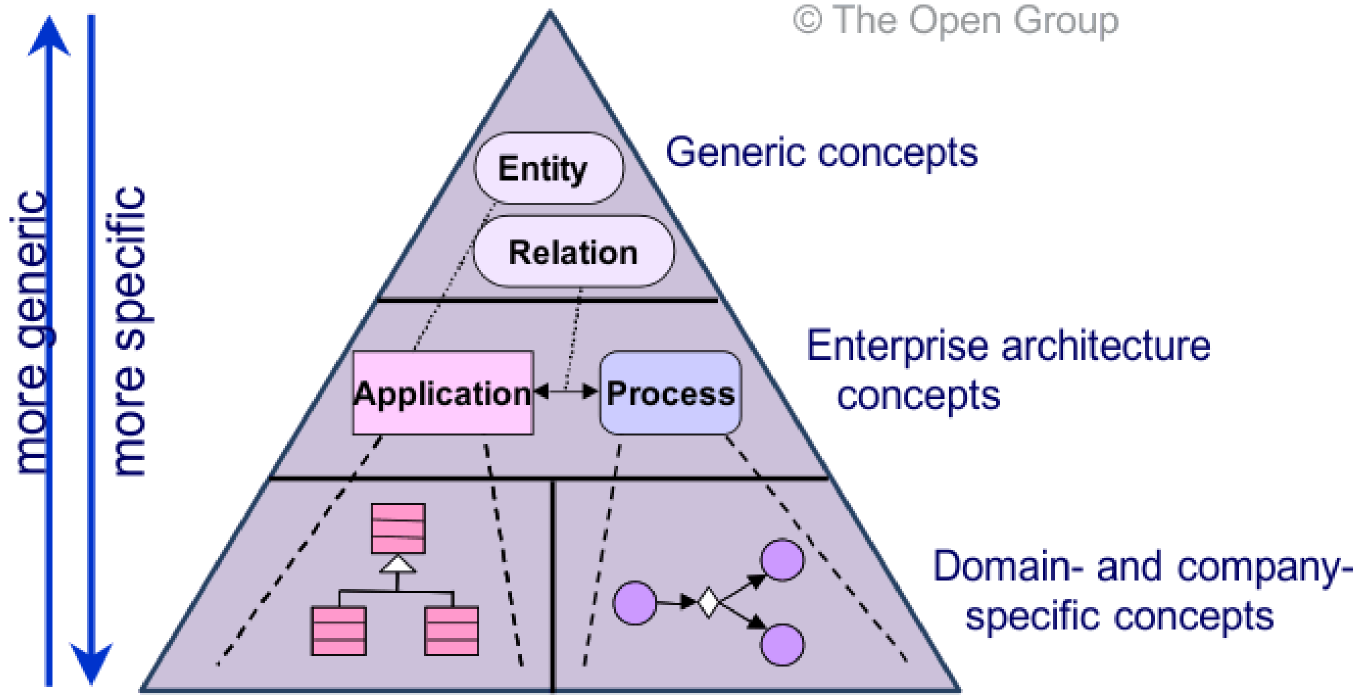
- AUTOSAR defines
  - A component-oriented system design approach
    - Domain specific modeling language
    - A high level design process
    - Standard middleware (basic software) stack
      - Standard interfaces
      - Standard configuration descriptors
- AUTOSAR compliant ECU software
  - Includes several BSW and application components
  - RTE provides the integration (glue) between these
  - Configuration and glue code is mostly auto-generated

Case study

Modeling IT infrastructure using ArchiMate

IT system and infrastructure

- Challenge: find a modeling language that is not too general neither too specific for a given domain



- Applies multi-level allocation

ArchiMate – infrastructure modeling

- The ArchiMate language defines three main layers
  - The *Business Layer* offers products and services to external customers, which are realized in the organization by business processes performed by business actors.
  - The *Application Layer* supports the business layer with application services which are realized by (software) applications.
  - The *Technology Layer* offers infrastructure services (e.g., processing, storage, and communication services) needed to run applications, realized by computer and communication hardware and system software.

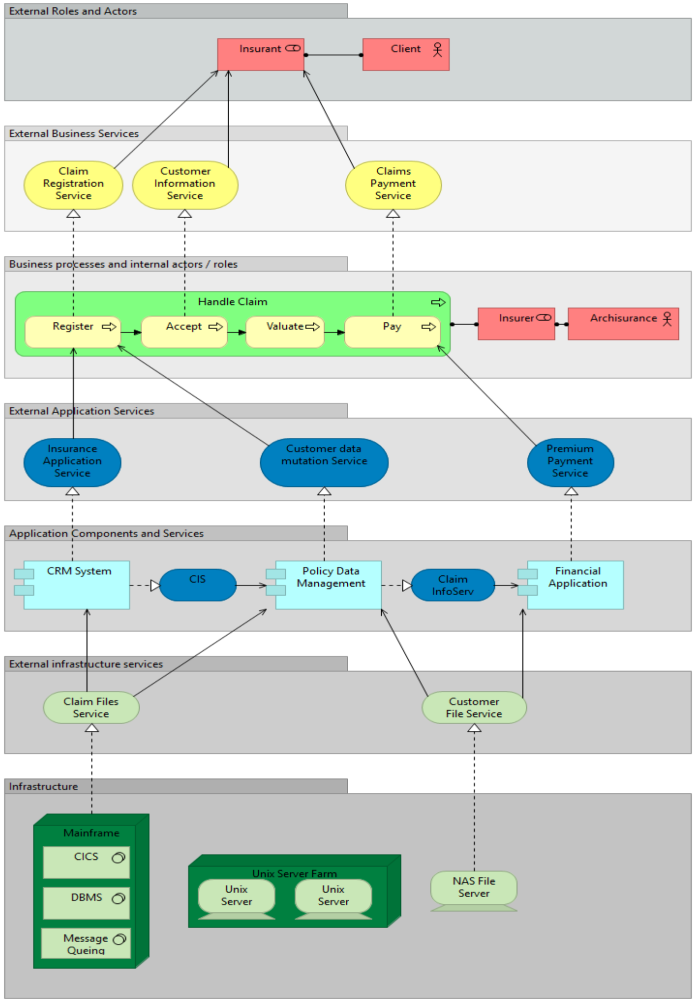
ArchiMate example – big picture

- An example of a fictional Insurance company.

Business layer

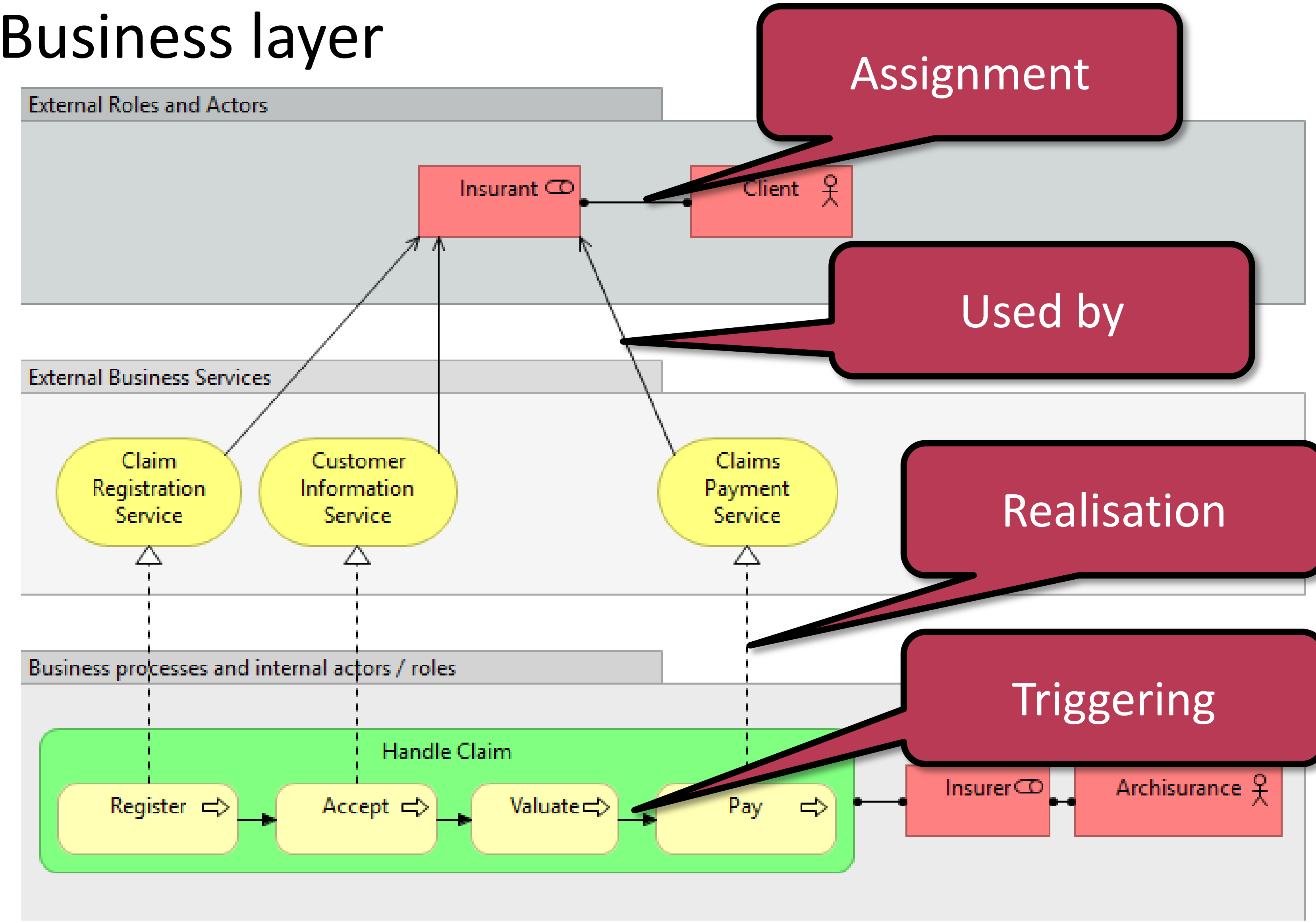
Application layer

Technology layer



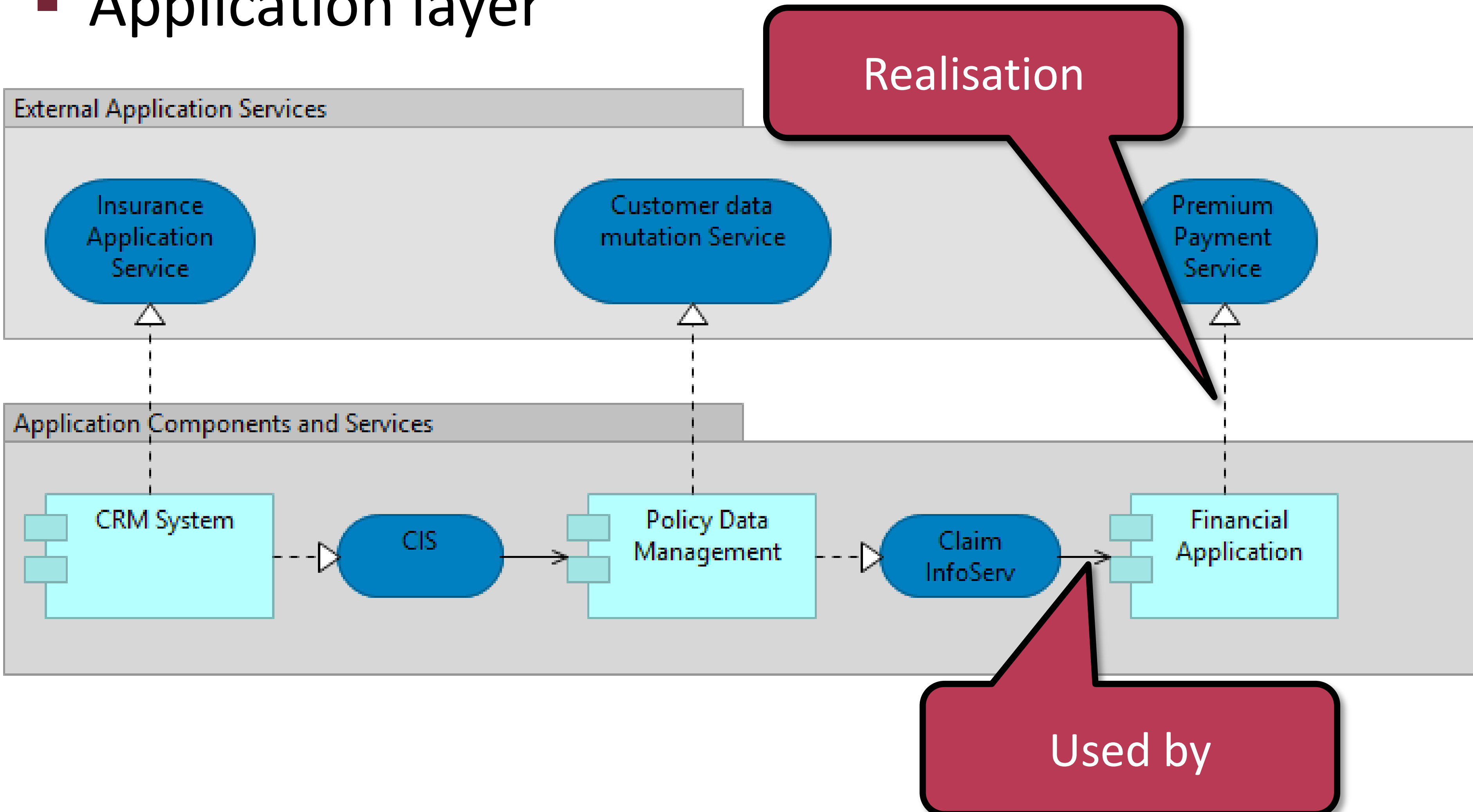
ArchiMate example: fictional Insurance company

- Business layer



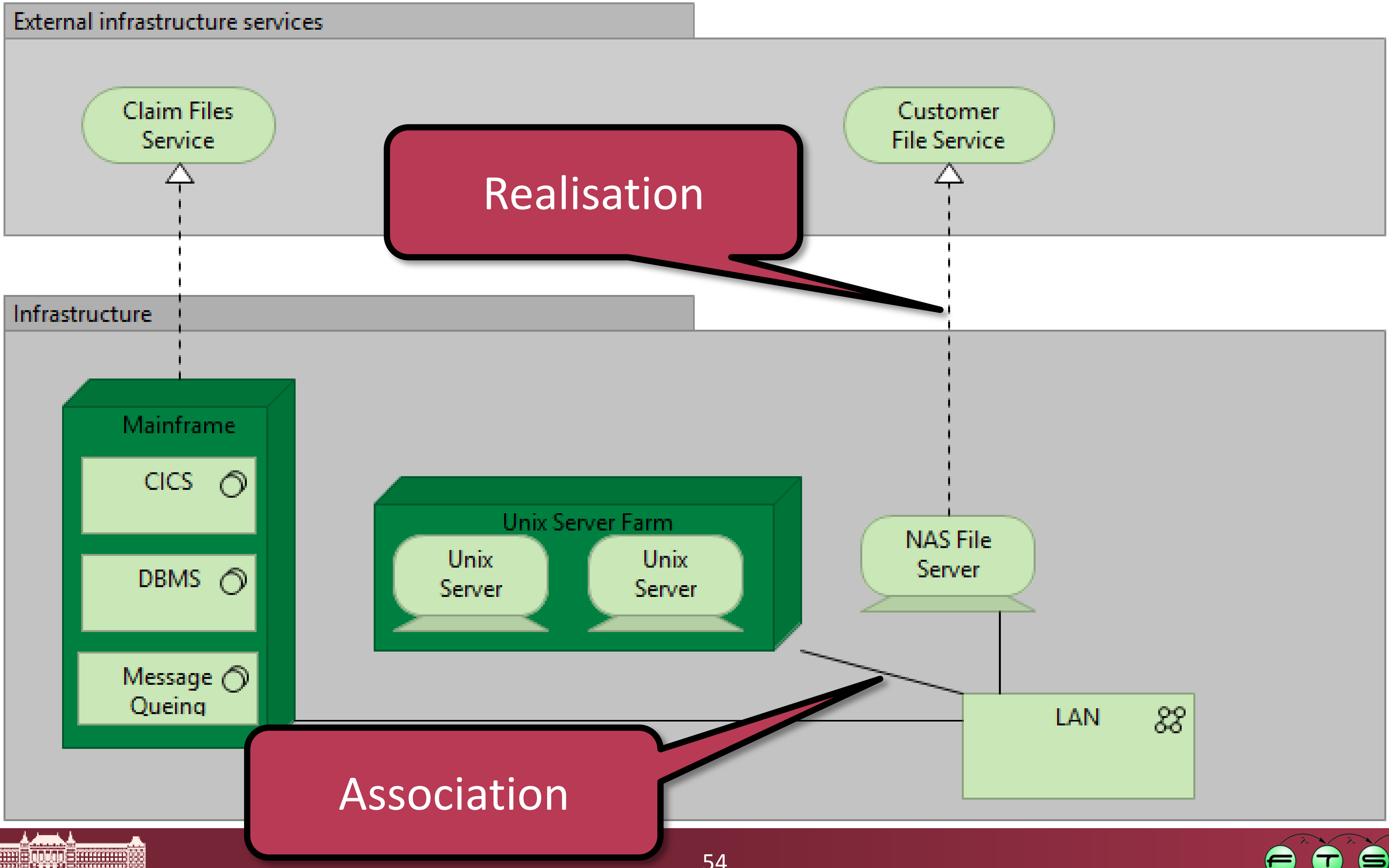
ArchiMate example: fictional Insurance company

- Application layer

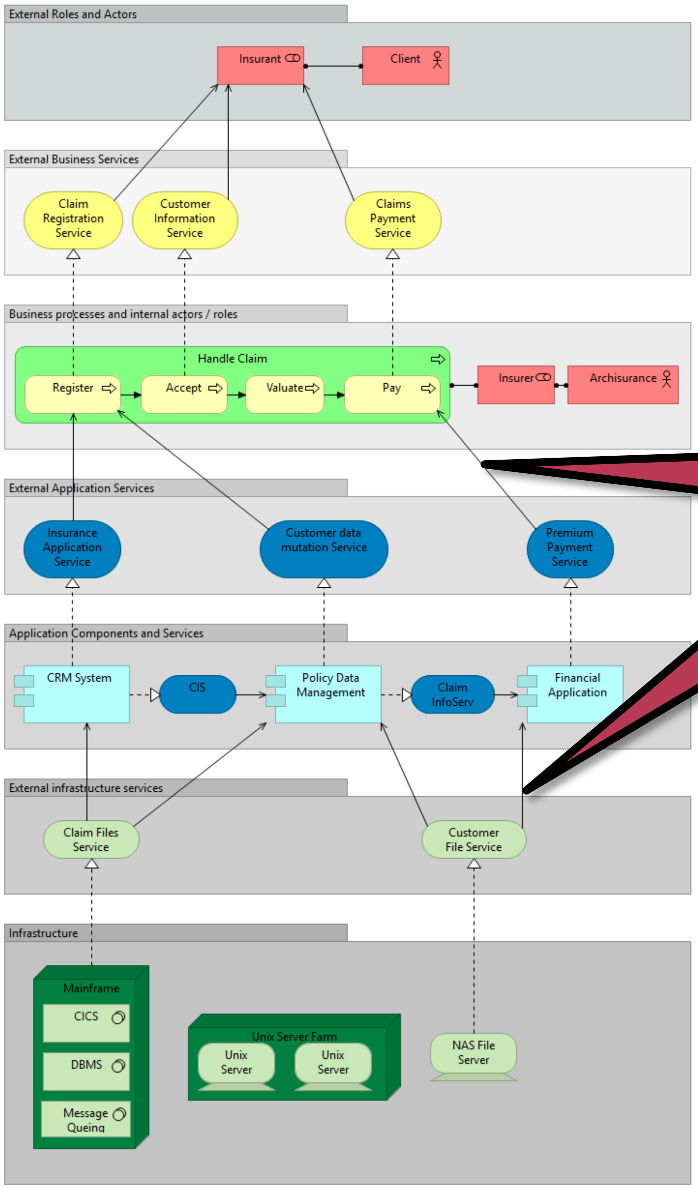


ArchiMate example: fictional Insurance company

- Technology layer



ArchiMate example – big picture



Used by – across layers

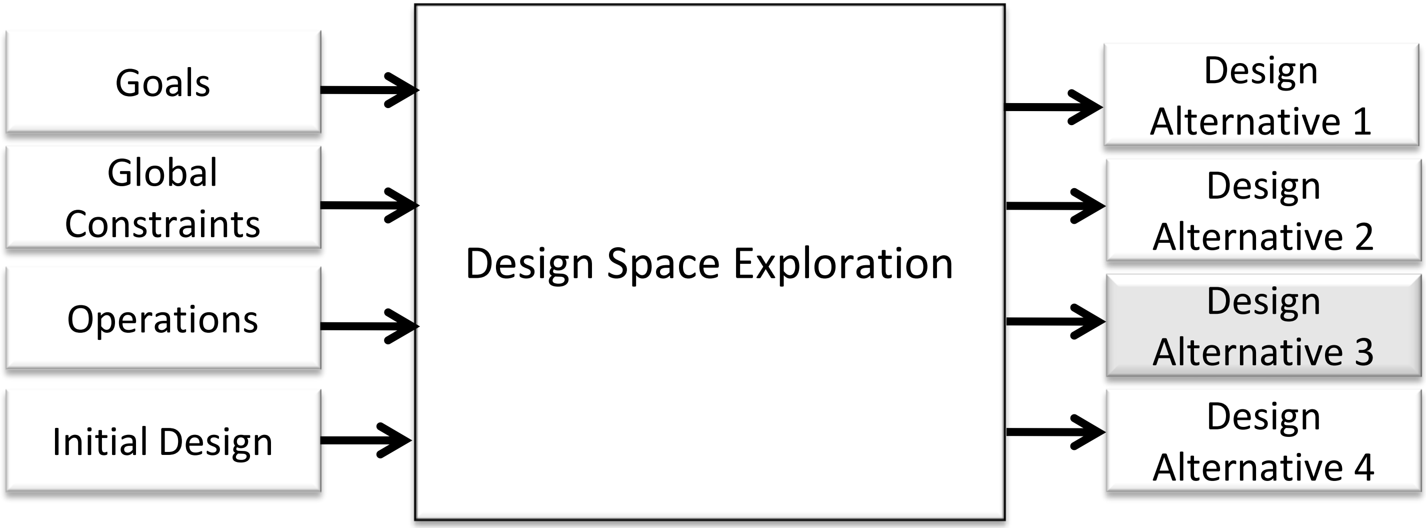
Case study

Application of DSE for allocation

Motivating example: Smart Building

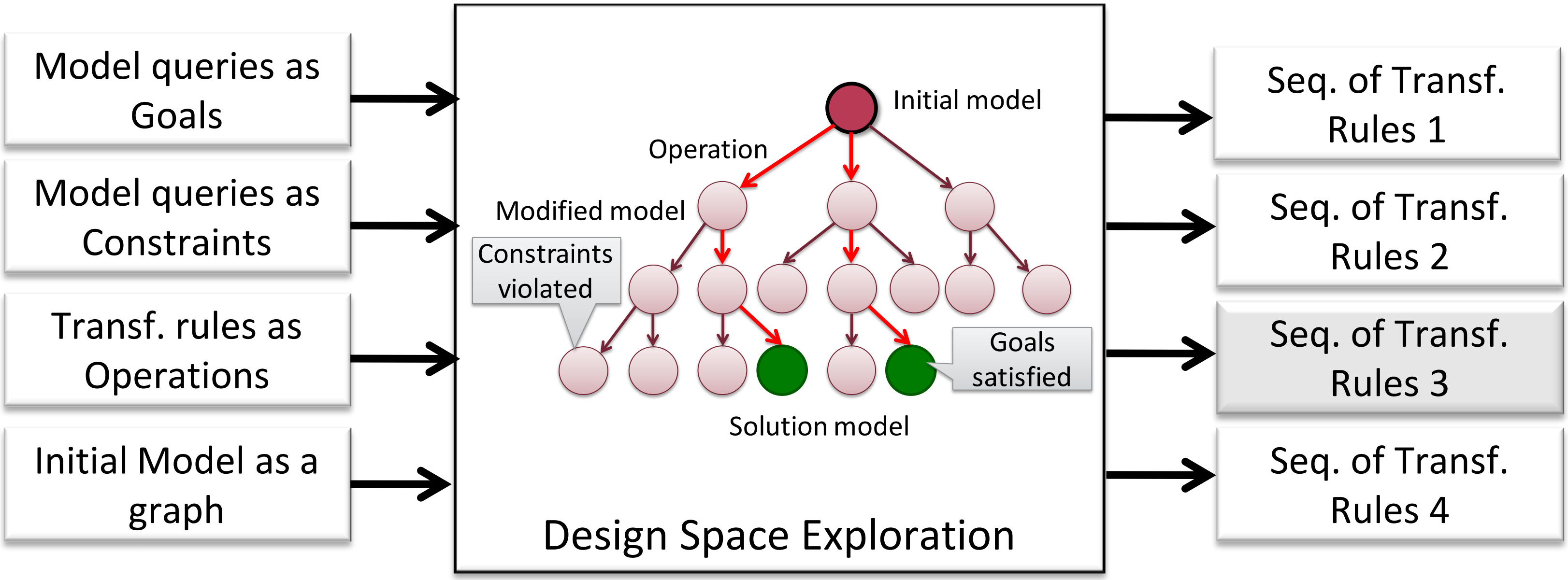
- Reconfiguration of supervising cyber-physical systems (CPS)
  - Offices to rent with highly configurable services
  - Services to deploy on both embedded and virtual computational units
  - Requests may change over time
  - Certain faulty devices may no longer function

Design Space Exploration (DSE)



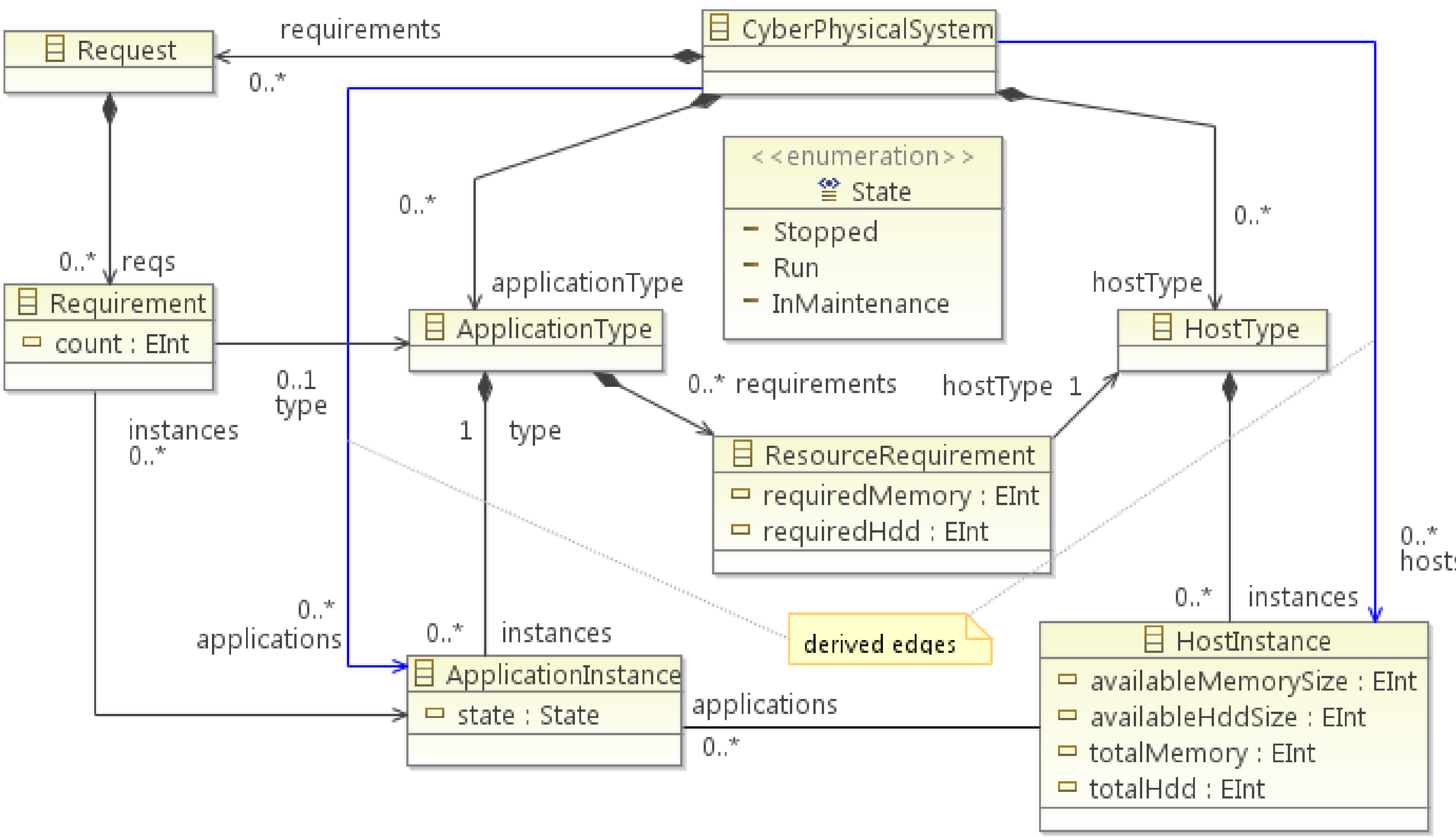
- Special state space exploration
  - Potentially infinite state space
  - cannot put upper bound on the number of model elements used in a design candidate (elements are created and deleted during exploration).

Rule-based Design Space Exploration

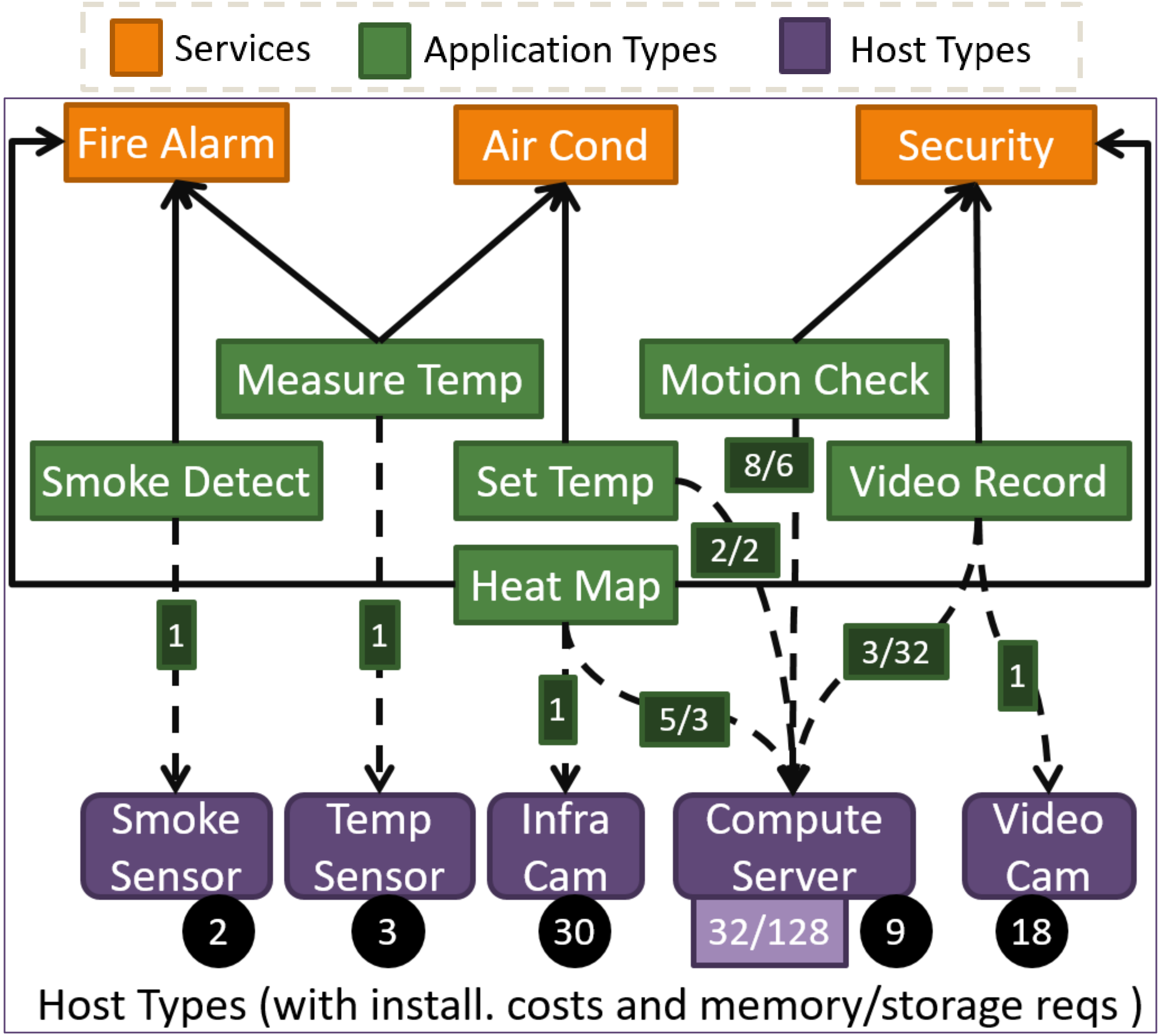


- Objectives : complex model metrics calculated by model queries
- Cost calculations may depend on the seq. of transf. rules
- Multiple objectives

Motivating example: Smart Building

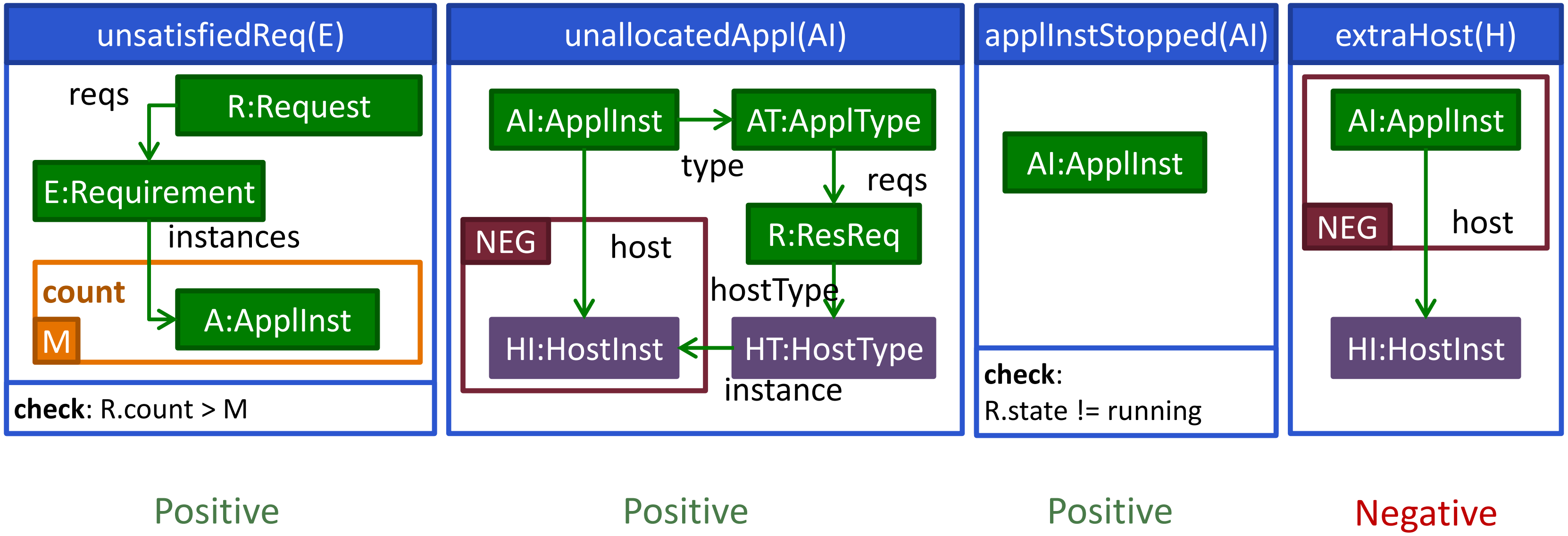


Motivating example: Smart Building



Smart building: constraints

- Constraints
  - Graph patterns to search for with model queries
  - For smart buildings
    - Constraints define valid or invalid configurations

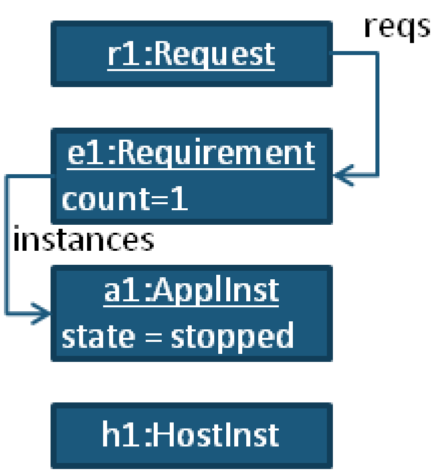


Smart building: constraints con't

- Constraints
  - Constraint fulfillment

$$ConstFulfillment(M) = \sum_{\forall p \in P} w_p \times matches(p, M)$$

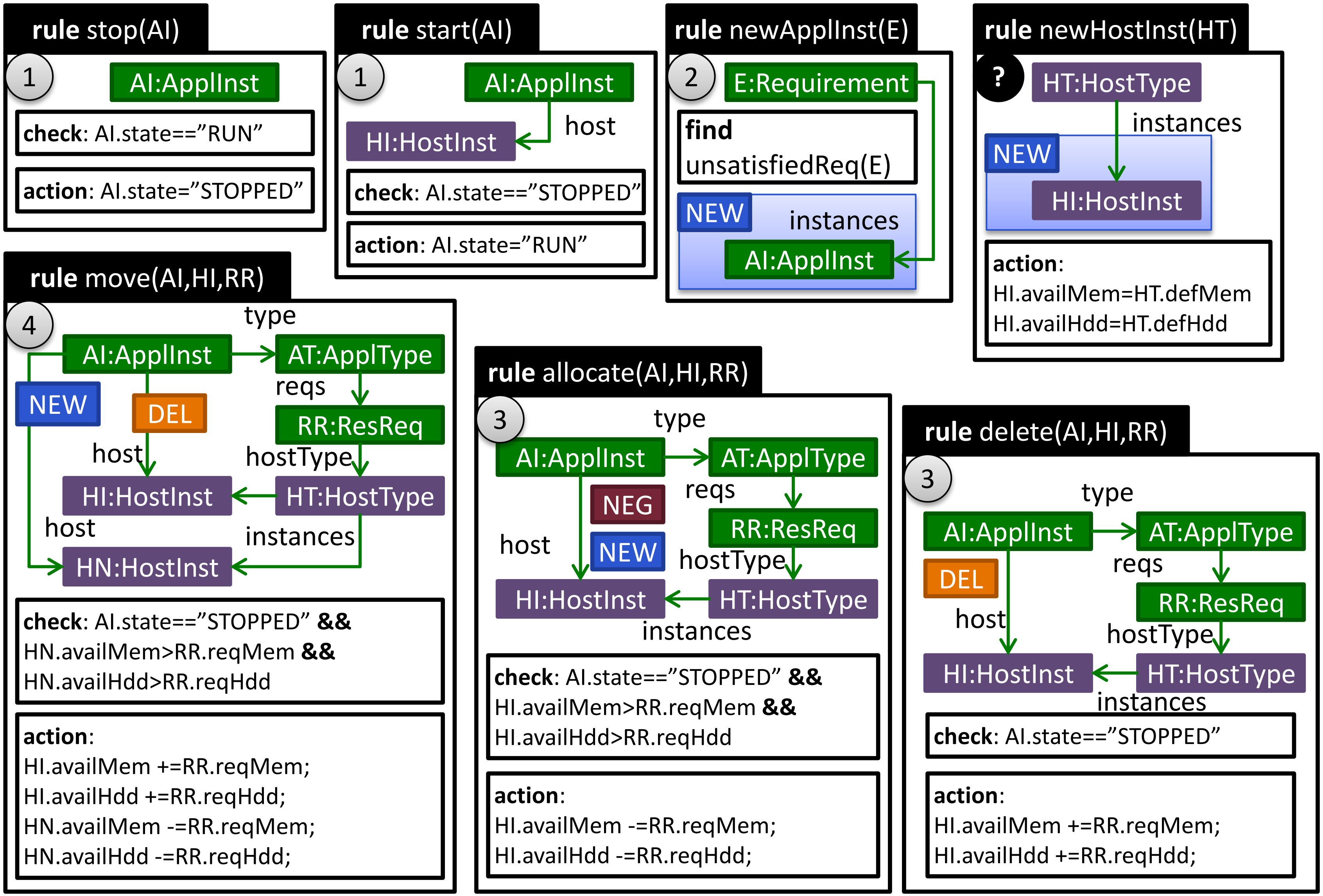
Positive for well-formedness constraints
Negative for ill-formedness constraints



Reqs	#	W	Score
satisfiedReq	1	2	2
allocatedAppl	0	3	0
applInstRun	0	4	0
extraHost	1	-1	-1
			1

$$ConstFulfillment(M) = 1 \times 2 + 0 \times 3 + 0 \times 4 + 1 \times -1 = 1$$

Smart building: rules



Smart Building: configuration model Services and Requests

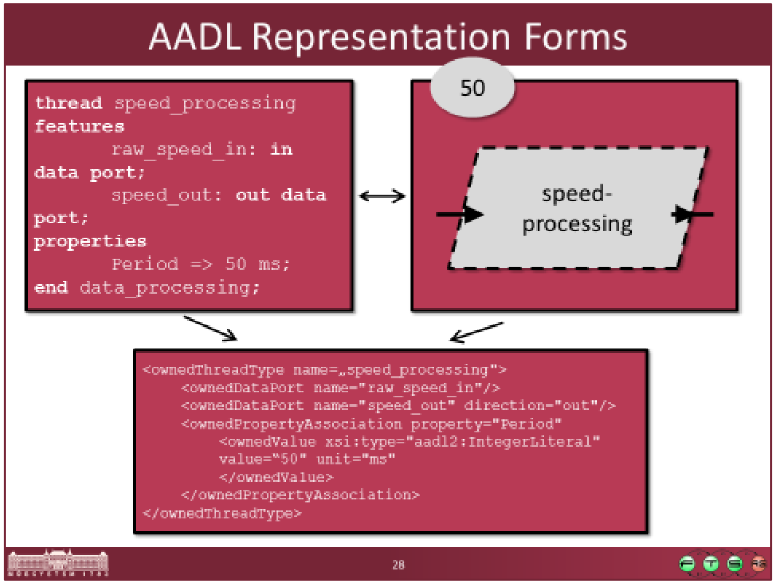
Package	Services	Appl Types
Basic	Fire Alarm	Smoke Detect MeasureTemp
Comfort	+ Air Cond	+ SetTemp
Secure	+ Security	+MotionCheck +VideoRecord
Max		+HeatMap

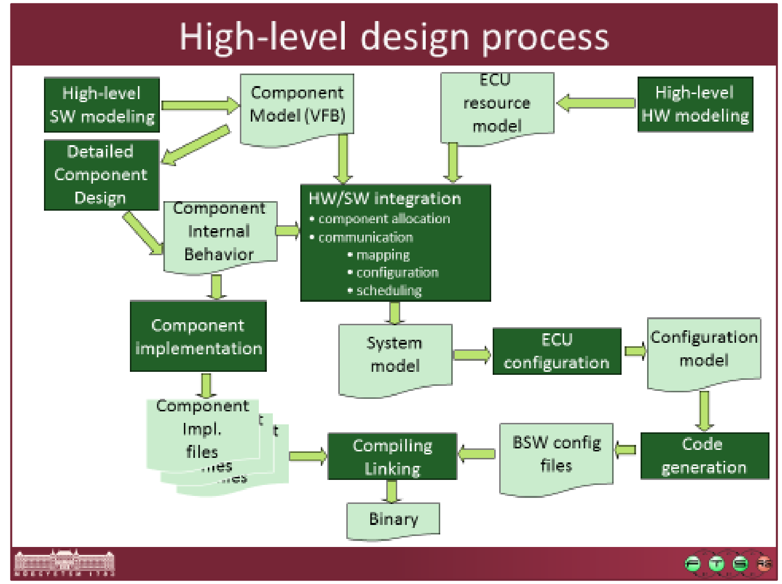
(a) Services

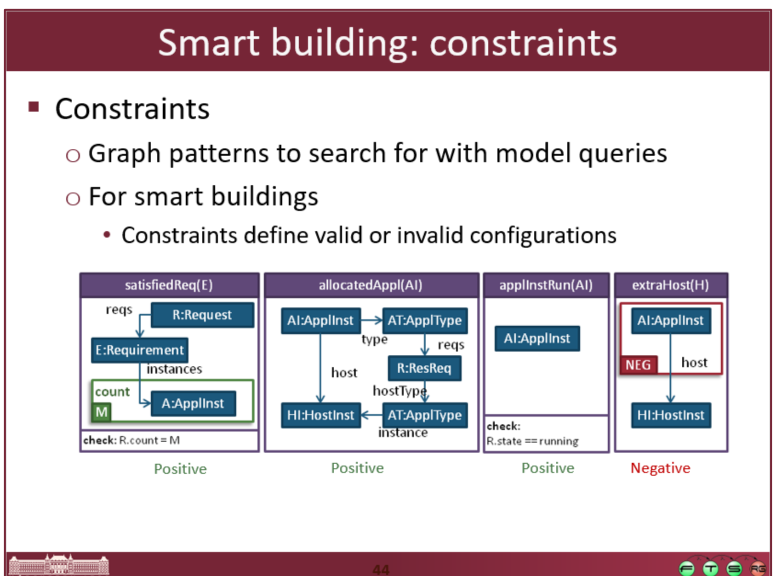
R	Packages	ApplInst	HostInst
1	Comfort (2) Basic(1)	3xSD, 2xMT, 2xST	3xSS,6xTS, 2xCS,
2	Max (2)	2xSD, 6xMT, 2xST, 2xMC, 2xVR, 2xHM	2xSS,6xTS, 8xCS, 2xIC, 2xVC,

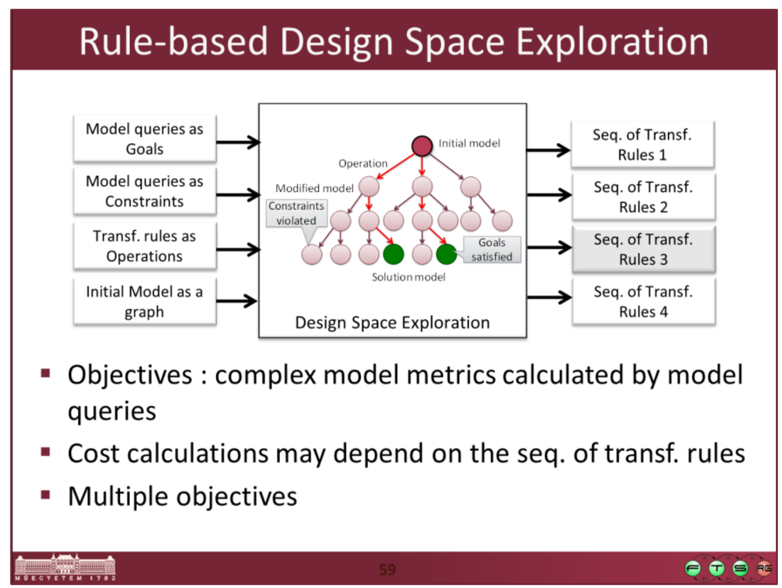
(b) Two examples on company requests

Summary


AADL Representation Forms
thread speed_processing
features
raw_speed_in: in data port;
speed_out: out data port;
properties
Period => 50 ms;
end data_processing;
50
speed-processing
<ownedThreadType name=„speed_processing">
<ownedDataPort name="raw_speed_in"/>
<ownedDataPort name="speed_out" direction="out"/>
<ownedPropertyAssociation property="Period"
<ownedValue xsi:type="aadl2:IntegerLiteral"
value="50" unit="ms"
</ownedValue>
</ownedPropertyAssociation>
</ownedThreadType>
28


High-level design process
High-level SW modeling
Component Model (VFB)
ECU resource model
High-level HW modeling
Detailed Component Design
Component Internal Behavior
HW/SW integration
• component allocation
• communication
• mapping
• configuration
• scheduling
Component implementation
System model
ECU configuration
Configuration model
Component Impl. files
Compiling Linking
BSW config files
Code generation
Binary


Smart building: constraints
Constraints
Graph patterns to search for with model queries
For smart buildings
Constraints define valid or invalid configurations
satisfiedReq(E)
reqs
R:Request
E:Requirement
instances
count
M
A:ApplInst
check: R.count = M
Positive
allocatedAppl(AI)
AI:ApplInst
AT:ApplType
type
reqs
host
R:ResReq
hostType
HI:HostInst
AT:ApplType
instance
Positive
applInstRun(AI)
AI:ApplInst
check:
R.state == running
Positive
extraHost(H)
AI:ApplInst
NEG
host
HI:HostInst
Negative
44


Rule-based Design Space Exploration
Model queries as Goals
Model queries as Constraints
Transf. rules as Operations
Initial Model as a graph
Initial model
Operation
Modified model
Constraints violated
Goals satisfied
Solution model
Design Space Exploration
Seq. of Transf. Rules 1
Seq. of Transf. Rules 2
Seq. of Transf. Rules 3
Seq. of Transf. Rules 4
Objectives : complex model metrics calculated by model queries
Cost calculations may depend on the seq. of transf. rules
Multiple objectives
59

References

- http://www.ptidej.net/courses/log3410/fall11/Lectures/Article_6.pdf
- https://hal.archives-ouvertes.fr/hal-00110453/document
- http://pubs.opengroup.org/architecture/archimate2-doc/toc.html
- https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf
- http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1675654