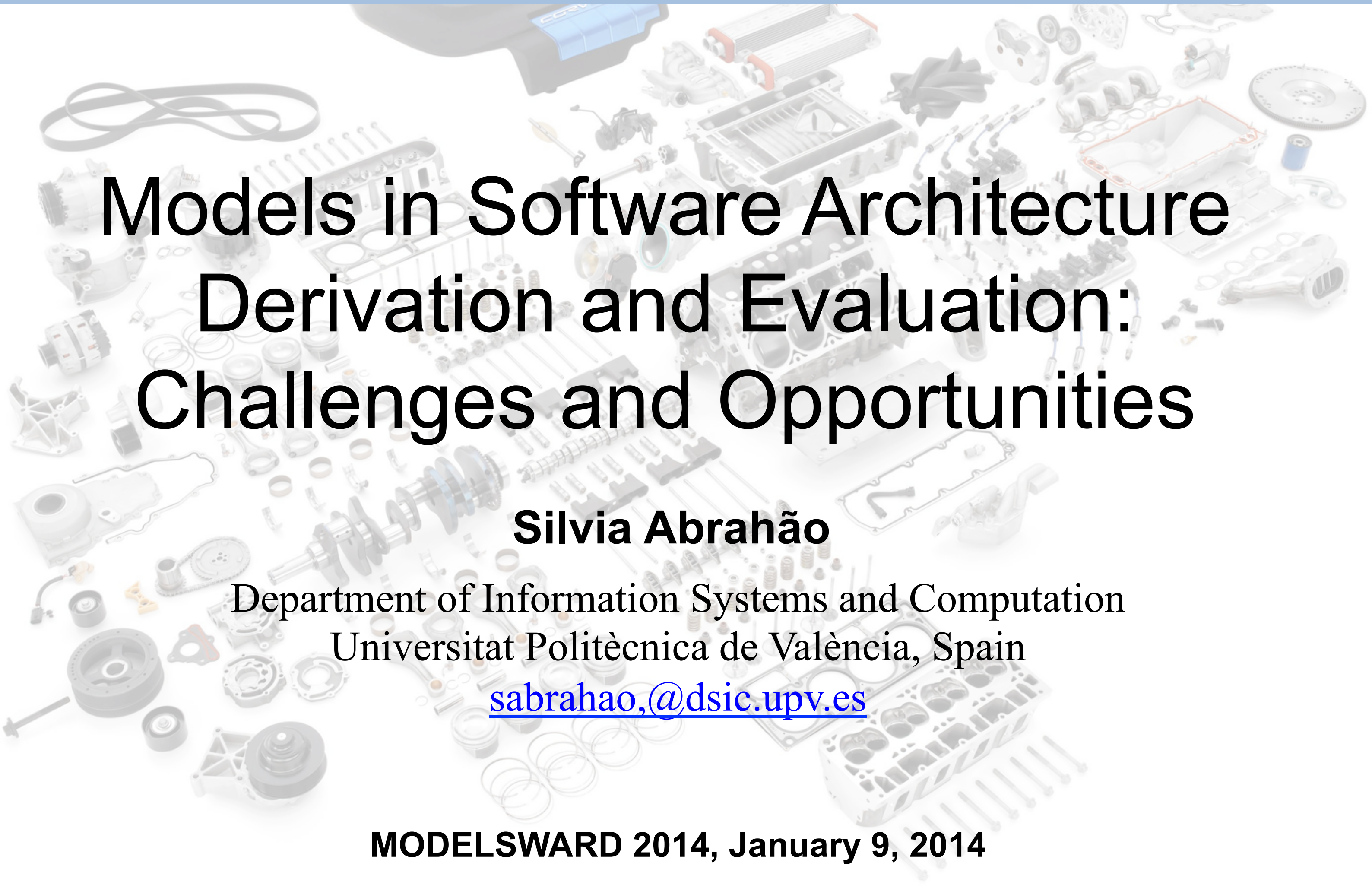


Models in Software Architecture Derivation and Evaluation: Challenges and Opportunities

Silvia Abrahão

Department of Information Systems and Computation
Universitat Politècnica de València, Spain
sabrahao,@dsic.upv.es

MODELSWARD 2014, January 9, 2014

Context

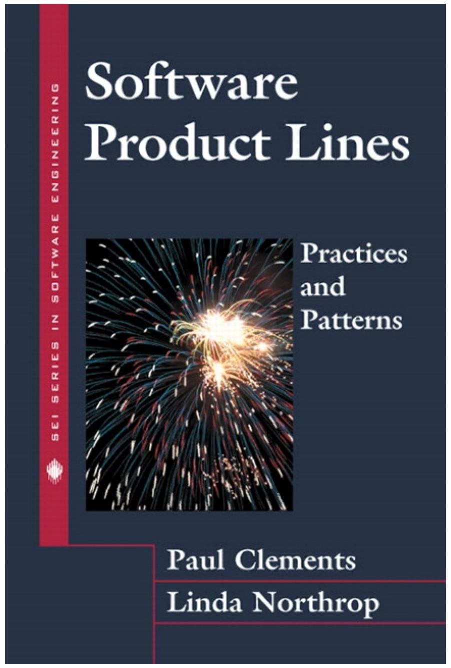
- **SEI** Software Engineering Institute | Carnegie Mellon
- **MULTIPLE Project** CICYT (TIN2009-13838)
  - MULTIPLE (*Multimodeling Approach for Quality-Aware Software Product Lines*)
  - From 2010 to 2013
  - 10 researchers at UPV (4 Professors and 6 PhD students)
  - 5 external researchers:
    - University of Leicester (UK), Universidad de Colima (Mexico)
    - LERO (Ireland), IT University of Copenhagen (Denmark)
    - Universidad Rey Juan Carlos (Madrid)
  - **EPO**: Rolls-Royce (UK)

    Goi Eskola Politeknikoa J.M.A.

Fundamentals

- Software product lines (SPL) emerged as a promising approach to improve software development processes so as to ***reduce costs*** and ***enhance productivity and product quality***.

*"A set of software-intensive systems **sharing** a common, managed set of features that satisfy the specific needs of a **particular market segment** or mission and that are developed from a **common set of core assets** in a prescribed way"*



SPL's and reuse

- A SPL is a strategic, "planned" reuse
  - Two processes: **Domain Engineering & Application Engineering**
  - (Base) software architecture
  - Support for commonality and variability
  - Core asset base
- ***Variability management*** encompasses:
  - Domain modeling and management (*Feature Model*)
  - Variability management as supported by **core assets**
  - **Production plan** that describes how the products are produced from the core assets

Quality in SPL development

- SPL adoption focuses mainly on managing a **single view** of the system (variability view).
- In practice, the variants are beyond the act of monotonically adding/removing functionality to the PL architecture.
  - **Interactions in the structure and behavior** of a software product to be developed can ***impact on its quality*** making the product inviable!
- Quality is a crucial factor in SPL development.
  - A defect in the PL architecture or in the core assets may impact the quality of many products within the SPL.

Core Asset Maturity

- ***Degree to which an asset is free from further modification***
- A low maturity asset is likely to be exposed to changes and depending on *when* they manifest, this can lead to high levels of effort to fix defects.
- Maturity becomes a sensitive issue for SPLs especially if products are using low maturity assets.
  - At Rolls-Royce, *50% of effort can be spent on scrap & rework* and 50% on the development of the assets.

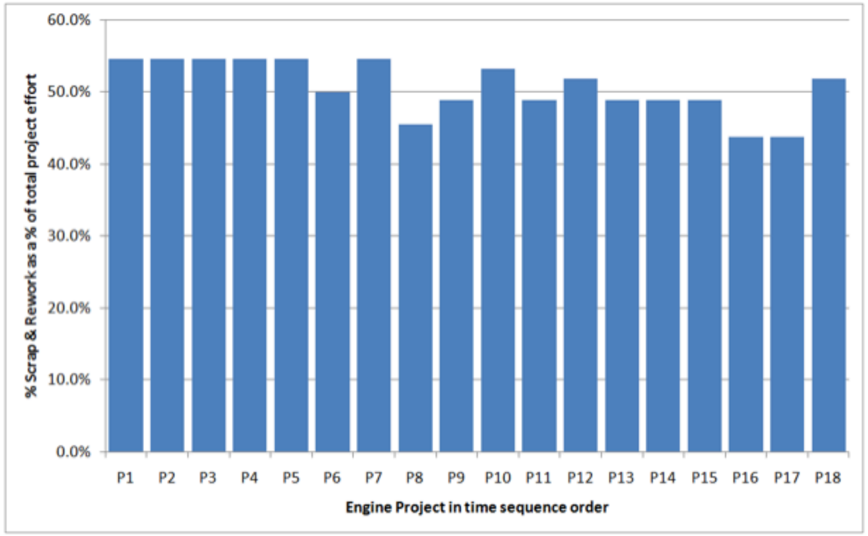
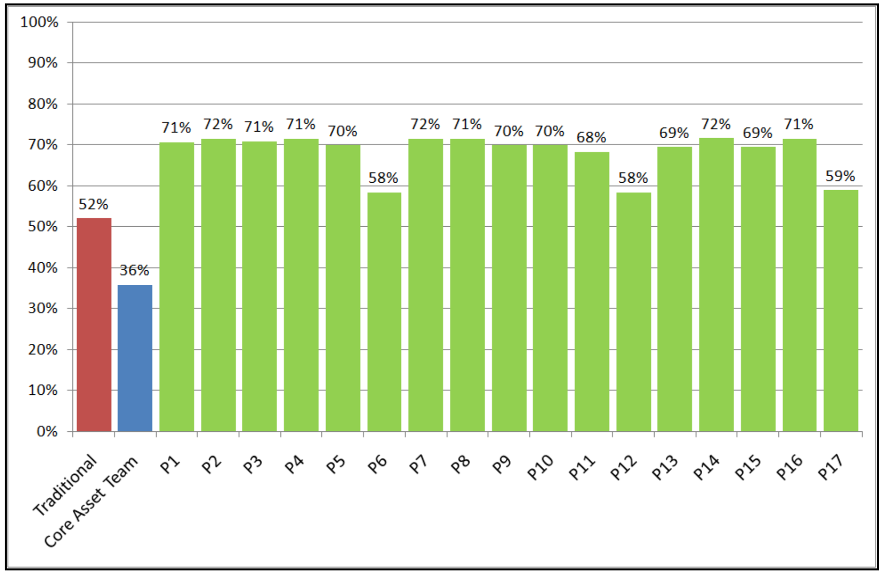


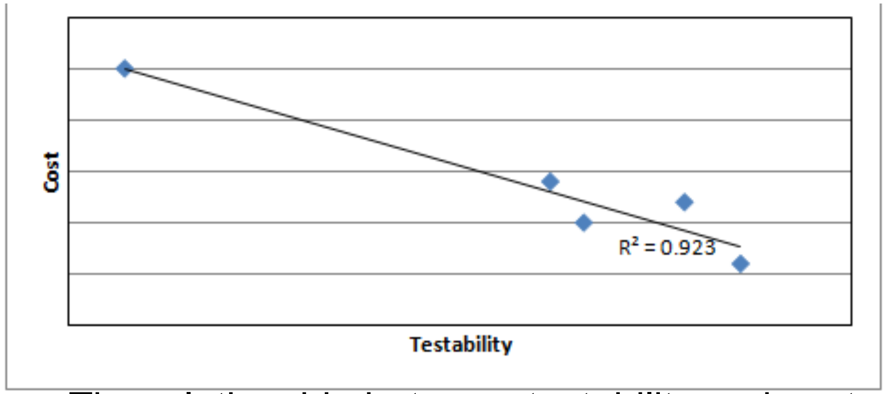
Figure 2. Percentage of Scrap & Rework for a range of engine projects. Scrap & rework is measured in terms of amount of hours' effort invested in modifying existing and configured core assets.

Andy Nolan, Silvia Abrahão, Paul Clements, John McGregor, Sholom Cohen: Towards the Integration of Quality Attributes into a Software Product Line Cost Model. SPLC 2011: 203-212

Testability

- A non-SPL safety-critical product invests 52% of its total development effort on some form of V&V.
- In a SPL at Rolls-Royce, data shows that up to 72% of a product's overall effort will be spent in some form of V&V
- Testability can be estimated from the #test cases (decision points) required to exercise the core asset.



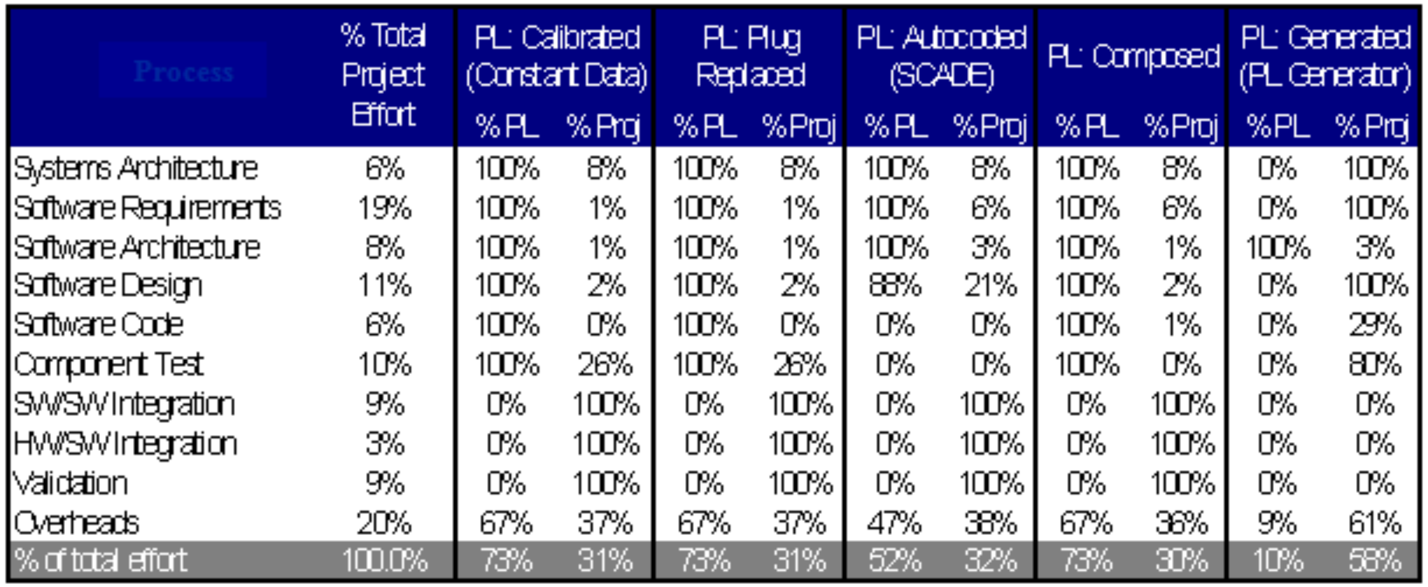


The relationship between testability and cost

[Nolan et al., 2011]

Variability

- The selection of a specific variation mechanism for a core asset can have an impact on the product development & deployment cost.
- **Cost of variability in a core asset** = cost of deploying the asset (in a specific process) * cost of using the different variation mechanisms.



Process	% Total Project Effort	PL: Calibrated (Constant Data)		PL: Plug Replaced		PL: Autocoded (SCADE)		PL: Composed		PL: Generated (PL Generator)	
		%PL	%Proj	%PL	%Proj	%PL	%Proj	%PL	%Proj	%PL	%Proj
Systems Architecture	6%	100%	8%	100%	8%	100%	8%	100%	8%	0%	100%
Software Requirements	19%	100%	1%	100%	1%	100%	6%	100%	6%	0%	100%
Software Architecture	8%	100%	1%	100%	1%	100%	3%	100%	1%	100%	3%
Software Design	11%	100%	2%	100%	2%	88%	21%	100%	2%	0%	100%
Software Code	6%	100%	0%	100%	0%	0%	0%	100%	1%	0%	29%
Component Test	10%	100%	26%	100%	26%	0%	0%	100%	0%	0%	80%
SW/SW Integration	9%	0%	100%	0%	100%	0%	100%	0%	100%	0%	0%
HW/SW Integration	3%	0%	100%	0%	100%	0%	100%	0%	100%	0%	0%
Validation	9%	0%	100%	0%	100%	0%	100%	0%	100%	0%	0%
Overheads	20%	67%	37%	67%	37%	47%	38%	67%	36%	9%	61%
% of total effort	100.0%	73%	31%	73%	31%	52%	32%	73%	30%	10%	58%

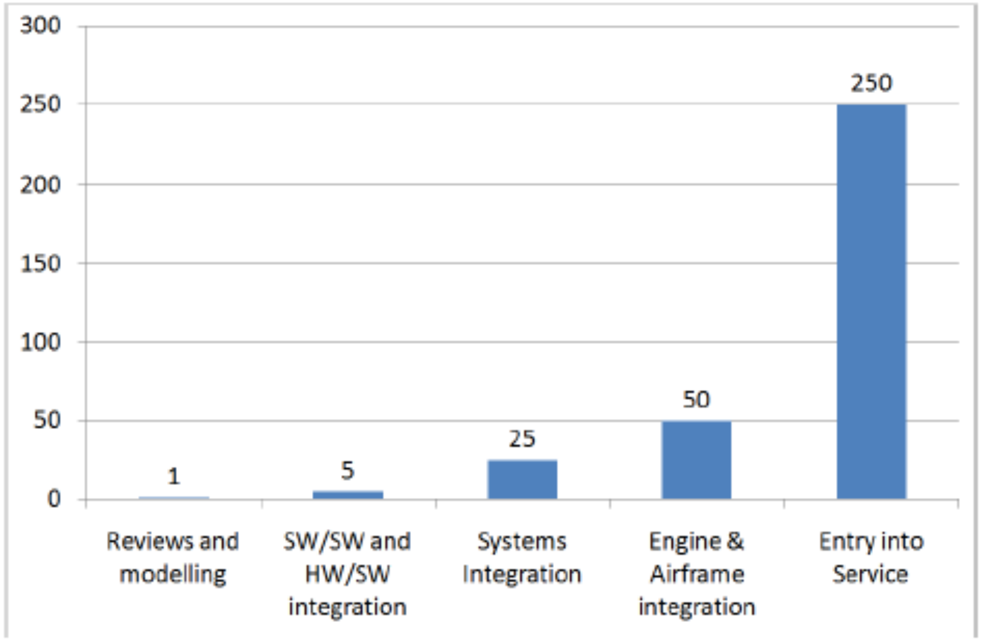
% effort per process and variation mechanism

[Nolan et al., 2011]

Variability

THE EFFORT EXPENDED BY THE CORE ASSET TEAM AND THE PROJECT TEAM FOR EACH VARIATION MECHANISM

Variation Mechanism	Core Asset Team	Product Team
Calibrated (Constant Data)	60.7%	39.3%
Plug Replaced	60.7%	39.3%
Autocoded (e.g. SCADE)	57.1%	42.9%
Composed	59.5%	40.5%
Generated (PL Generator)	21.3%	78.7%



Cost involved in discovering and correcting a defect depending on when that defect was detected.

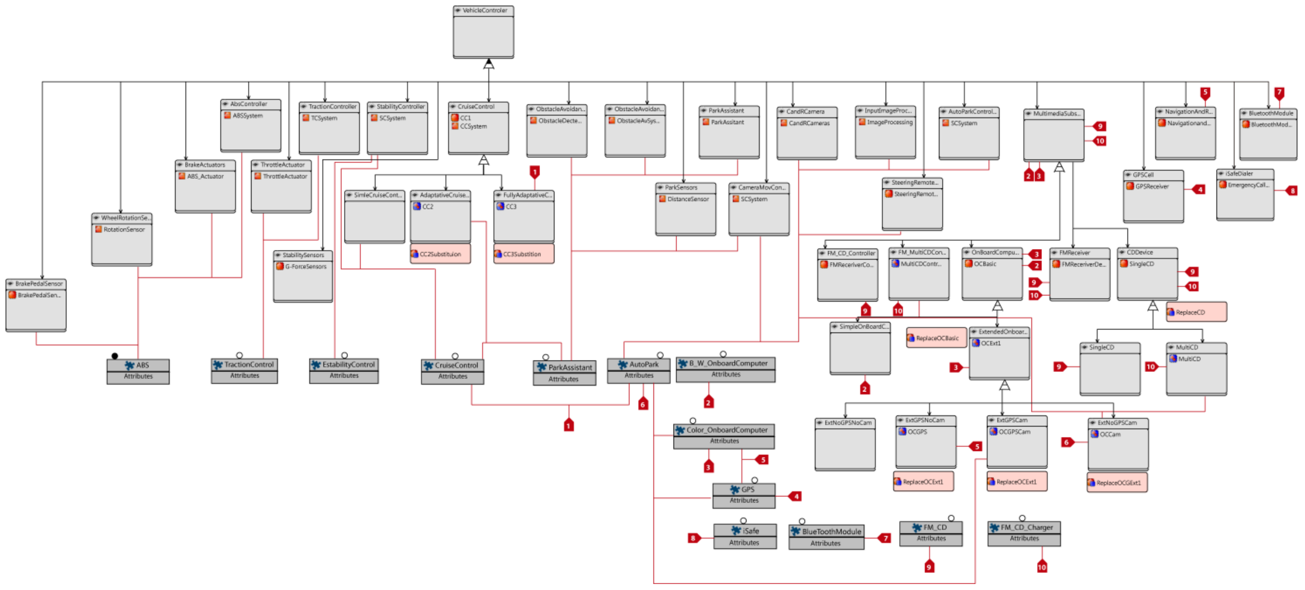
[Nolan et al., 2011]

Challenges

- Quality should be evaluated at both the Domain Engineering and the Application Engineering phases.
- Software architecture is a means to achieve the product quality attributes.
- In SPL, the architecture plays a dual role:
  - The ***PL architecture*** contains a set of variation mechanisms that support the functional and NFRs of the entire set of products that constitute the product line.
  - The ***product architecture*** is derived from the PL architecture by exercising its built-in architectural variation mechanisms.

Challenges

- When compared to the vast amount of research on developing SPLs, little work has been dedicated to the use of SPLs to derive individual products.
  - The architecture derivation and product configuration is a complex, time-consuming process.
  - Given a set of ***architectural variation points (PL architecture)***, how we decide which ones should be selected or which ones should not?



Challenges

- One of the most difficult tasks during product derivation is meeting the required quality attributes.
- Once derived, the product architecture should be evaluated to guarantee that it meets the product specific quality attributes.
- When the quality attributes of a product cannot be attained by using built-in variation mechanisms, certain ***architectural transformations*** should be applied to achieve these quality attributes.

This implies the following:

- ***Quality attributes related to each architectural transformation need to be represented and used for selecting the transformation to be applied.***
- ***The resulting product architecture has to be evaluated to asses if the required quality attribute levels are fulfilled.***

Key activities for product derivation in SPLs

Criteria	*Description*
***C1**[*]*	Non-functional requirements (NFRs) support
***C2**[*]*	Explicit representation of NFRs/quality attributes and their relationships with variability or architectural components
***C3**[**]*	Configuration support
***C4**[**]*	Automated derivation support
***C5**[***]*	Adaptability and extensibility (i.e., metamodel support, extension points for the integration of domain specific generators)
***C6**[***]*	Flexible and user-specific visualizations of variability (filtering, classification and ordering support based on tasks, users, roles etc.)
C7	Explicit representation of architectural variability
C8	Architectural views support
C9	ADL/Modeling language support
C10	Configuration consistency checking
***C11**[***]*	End-user guidance
***C12**[***]*	Project management support (task management, roles and users support)

[*] **C1 and C2** Adapted from the "Application requirements management support" [Rabiser et al. 2010]

[**] **C3 and C4** Adapted from the "Automated and interactive variability resolution" [Rabiser et al. 2010]

[***] **C5, C6, C11 and C12** Criteria proposed at the systematic review by [Rabiser et al. 2010]

Rick Rabiser, Pádraig O'Leary, Ita Richardsonc, Key activities for product derivation in software product lines, Journal of Systems and Software, 2010.

Existing approaches for product (architecture) derivation

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
Kobra (Atkinson et al. 2000)	-	-	-	-	+	-	-	C&C	Own	-	Partially	-
Koalish (Asikainen et al. 2003)	-	-	+	+	+	-	+	C&C	Own	-	-	-
Cabello (2008)	-	-	+	+	+	-	+	+	Own (PRISMA)	+	-	-
Botterweck et al. (2009)	-	-	+	+	+	-	+ (FM /C)	C&C	+	+ (FM /C)	-	-
Perovich et al. (2009)	+	-	+	+	-	-	-	C&C	+	-	-	-
Duran-Limon et al. (2011)	-	-	+	+	-	-	+ (OWL and FM)	C&C	+	+ (FM)	-	-
Guana y Correal (2013)	+	-	+	+	+	-	+	C&C	+	-	-	-
Czarnecki y Antkiewicz (2005)	-	-	+	+	+	-	+	+	+	+ (FM)	-	-
Ziadi y Jézéquel (2006)	+	-	+	+	+	-	On the model	+	UML	+	Partially	-
PLUS-EE- (Gomaa y Shin 2007)	-	-	+	+ (Executable code)	-	-	On the model	Multiple viewpoints	UML	+	-	-
Perrouin et al. (2008)	-	-	+	+	+	-	+	-	UML	+	Partially	-
Schaefer et al. (2009)	-	-	+	+	+	-	+	CoBoxes	CoBoxes	-	-	-
Tawhid y Petriu (2011b)	-	-	-	+	-	-	On the model	Structure	Marte	-	-	-
Sánchez et al. (2008)	-	-	-	+	+	-	+	+	+	-	For language definition	-
FeatureMapper (Heidenreich et al. 2008)	-	-	+	+	+	-	+ (FM and Models)	+	+	+	-	-
Haugen et al. (2010)	-	-	+	+	+	-	+	+	+	+	-	-

Legend:
FM: Feature Model; **C&C:** Component and Connector; **FM/C:** Feature Model and Component Model; **+:** Supported; **-:** Not Supported

Existing approaches for product (architecture) derivation

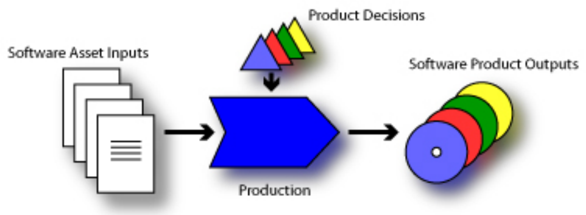
- Several methods for product (architecture) derivation have been proposed over the last few years, but:
  - They do not properly integrate quality attributes in the derivation process.
  - The derivation process is not properly integrated with the evaluation and quality improvement processes.
  - The derivation process is often not automated.
  - The architect knowledge is not well captured and represented.

Existing approaches for architecture derivation

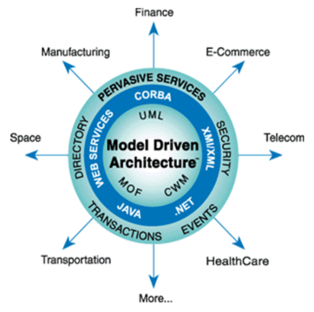
- Several methods for architecture evaluation (specific for SPLs):
  - **FAAM** (SAAM and ATAM extension): does not consider interactions among competing quality attributes, specific for ***interoperability*** and ***extensibility***.
  - **D-SAAM** (SAAM extension): no interactions among quality attributes.
  - **ALMA:** scenario-based method specific for ***modifiability***.
  - **ATAM** provides a principled way to evaluate the fitness of a software architecture with respect to multiple competing quality attributes (not for SPL).
  - **EATAM** and **HoPLAA** (ATAM extensions): lack a systematic mechanism for architectural improvement.
- There is still a need for
  - Modeling the impact among ***architectural design decisions*** and ***quality attributes*** and use this information to drive the derivation and evaluation of high-quality product architectures.

Our approach: QuaDAI

- An integrated method for the derivation, evaluation and improvement of software architectures in the development of Model-Driven SPLs.
- Based on the existence of **several models** (functionality, features, quality,...) that represent the different SPL views with relationships among them (**Multimodel**).
- The **views are "active" software artifacts** which drives the production plan by means of two **model transformation processes:**
  - Architecture derivation and product configuration
  - Architecture evaluation and improvement



Exploit Software Product Lines



Model Transformations Techniques

Multimodel

- A ***multimodel*** is a set of interrelated models that represents different viewpoints of a particular *system*[1].
- A ***viewpoint*** is an abstraction that yields a specification of the whole system restricted to a particular set of concerns.
- In any given viewpoint it is possible to define a model of the system that contains only the objects that are visible from that viewpoint. Such a model is known as a ***viewpoint model***, or a ***view*** of the system from that viewpoint **(NISTIR 6928, 2003)** [2].

[1]The term system encompasses individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest.

[2]*National Institute of Standards and Technology, U.S. Dept. of Commerce, USA*

Multimodel viewpoints

- Represent the different viewpoints of a ***set of products*** that can be derived from the SPL.
- The multimodel comprises (at least) ***4 viewpoints*** of the SPL and the relationships among them:
  – **Variability:** expressing the commonalities and variations within the SPL.
  – **Architectural:** expressing the architectural variability of the PL architecture. It can be defined using different styles (e.g., component-and-connector, module, allocation).
  – **Quality:** expressing the different quality characteristics and attributes. It can be represented by a Quality Model (ISO 25010).
  – **Transformations**: expressing the possible architectural transformations (e.g., design decisions)

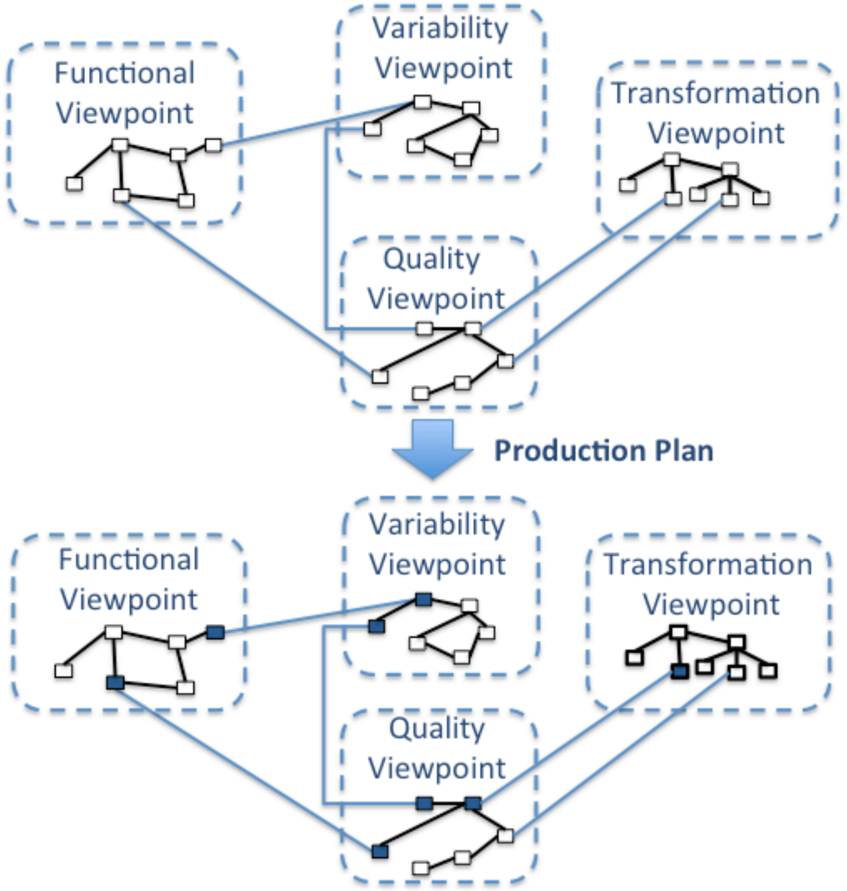
Multimodel Viewpoints

Domain Engineering

- The multimodel represents the impacts and constraints among variations, architectural viewpoints, quality attributes and architectural transformations.

Application Engineering

- The multimodel represents the selected and mandatory features from the Variability Model + the elements of the Architecture/ Functional Model + the elements of the Quality Model and the transformations affected by them

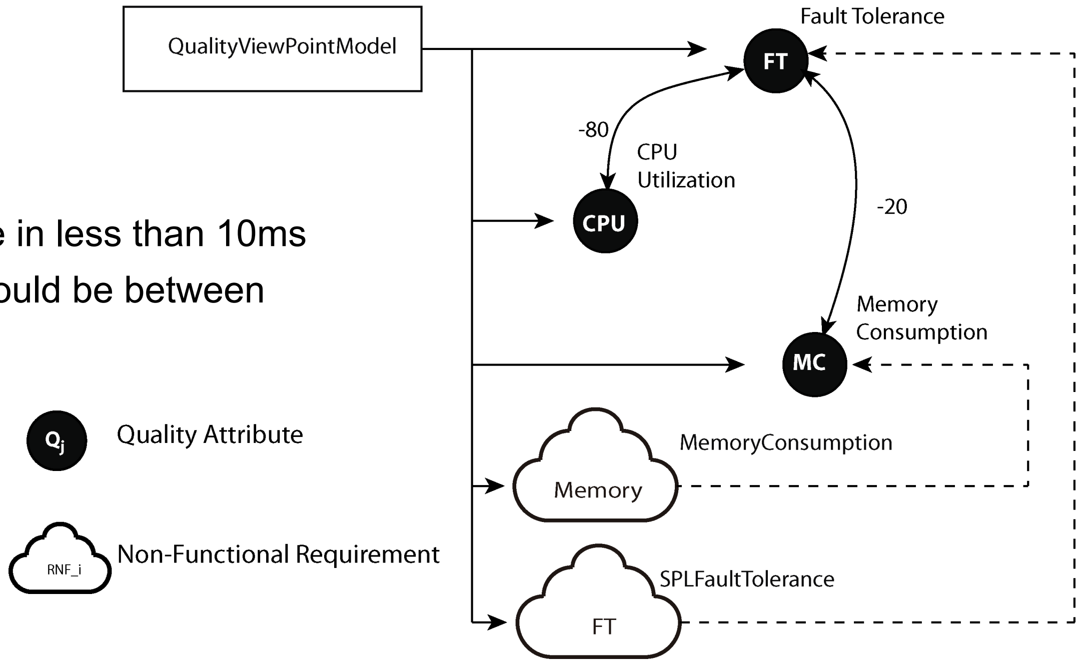


Quality Viewpoint

- Represented by a **Quality Model for SPLs** where we can:
  - Define the **impact relationships** among the quality attributes
  - Define the **NFRs** for both the SPL and the specific products (***as constraints over the Quality Model***). NFRs can be specified for specific features, core assets, etc.
  - Select the NFRs and prioritize the quality attributes for a given product (during the configuration).

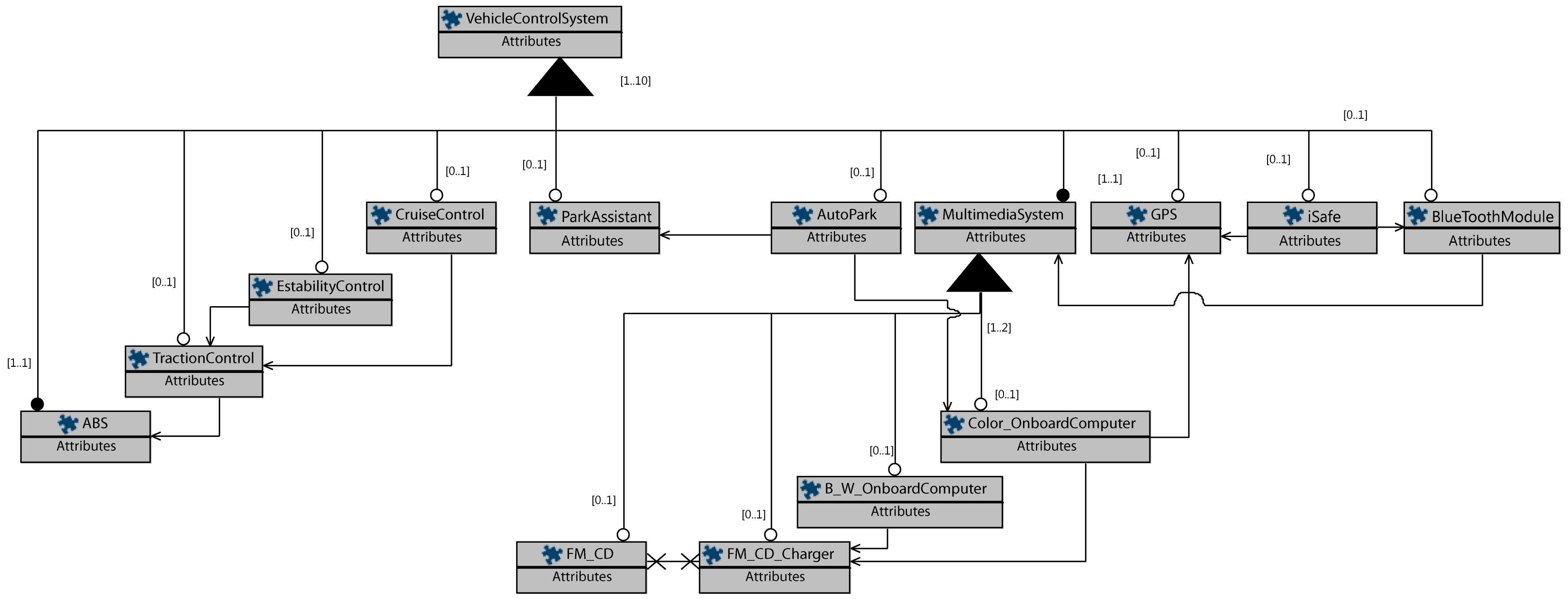
NFR000: System should give a response in less than 10ms

NFR001: The reliability of the system should be between 0.995 and 0.999



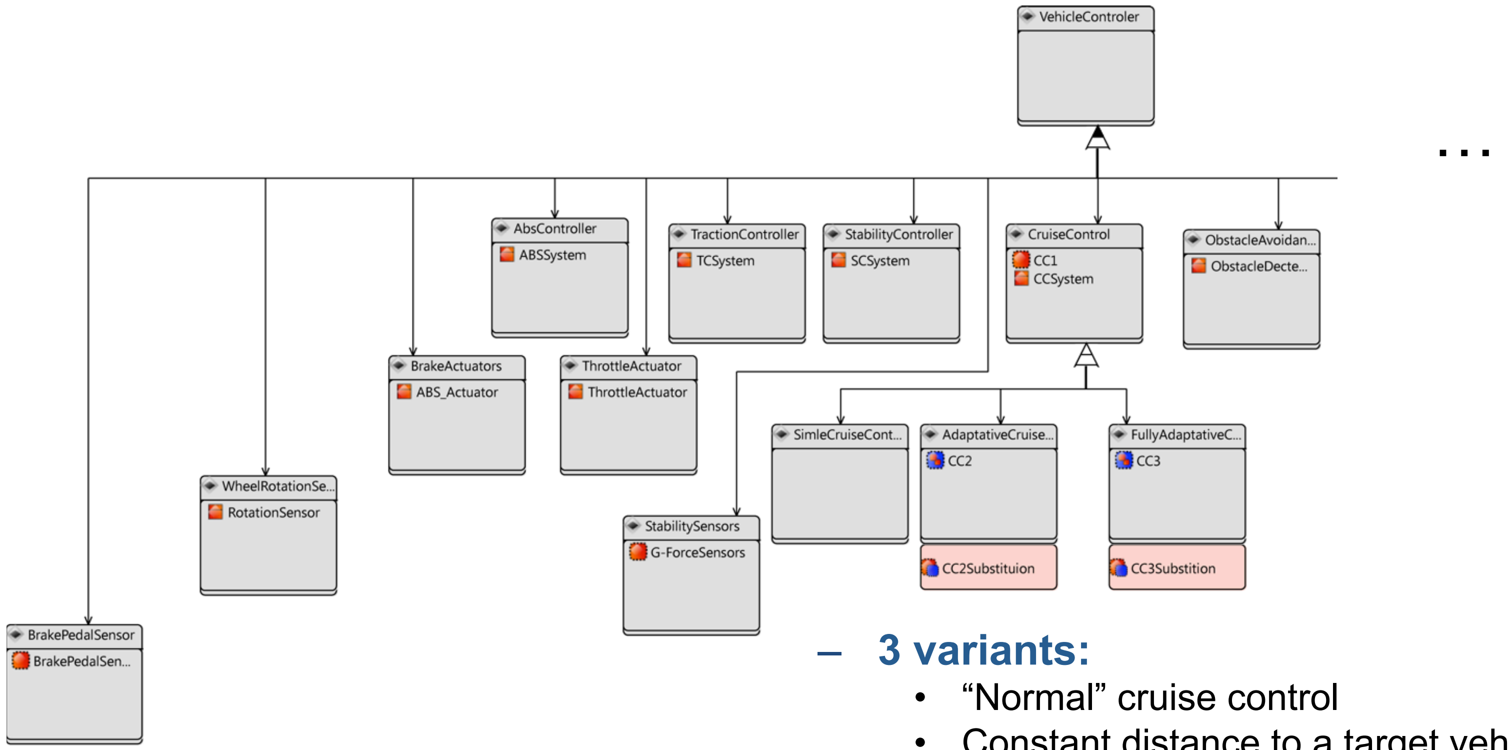
Variability Viewpoint

- Represented by a **Cardinality-based feature model** [Czarnecki, 2005] [Gómez et al, 2011] .



Architectural Viewpoint

- Express the built-in variation mechanisms of the PL architecture **regardless the ADL or the domain.** Represented by the **Common Variability Language**.

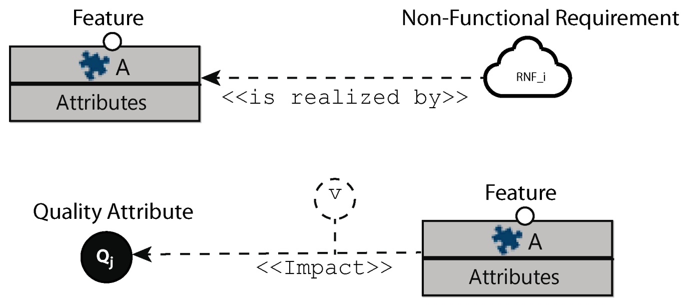


- **3 variants:**
  - "Normal" cruise control
  - Constant distance to a target vehicle
  - Full speed CC with image sensors

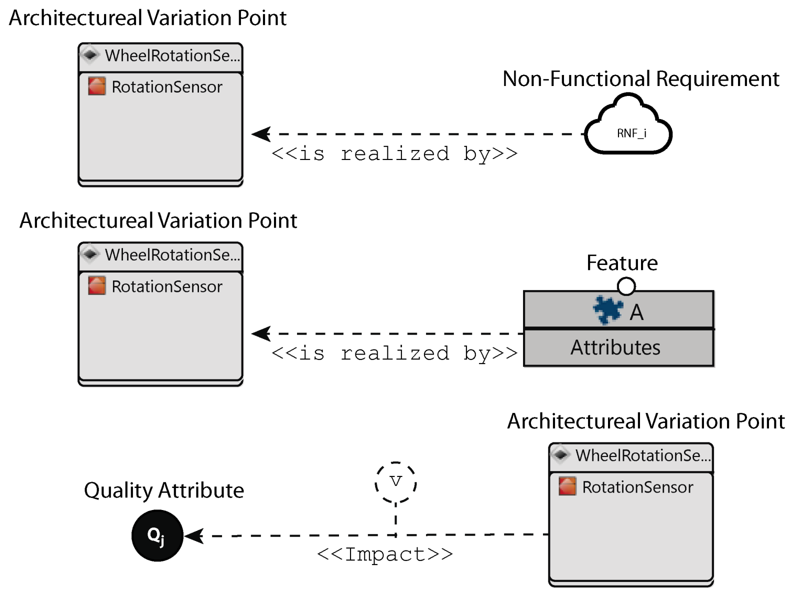
Relationships among views

- The multimodel can be used to define relationships among the elements on different ***viewpoint models or views***. **This will allow us to analyze properties over the SPL as a whole.**
- These relationships are used during the different tasks that integrate the QuaDAI derivation process.

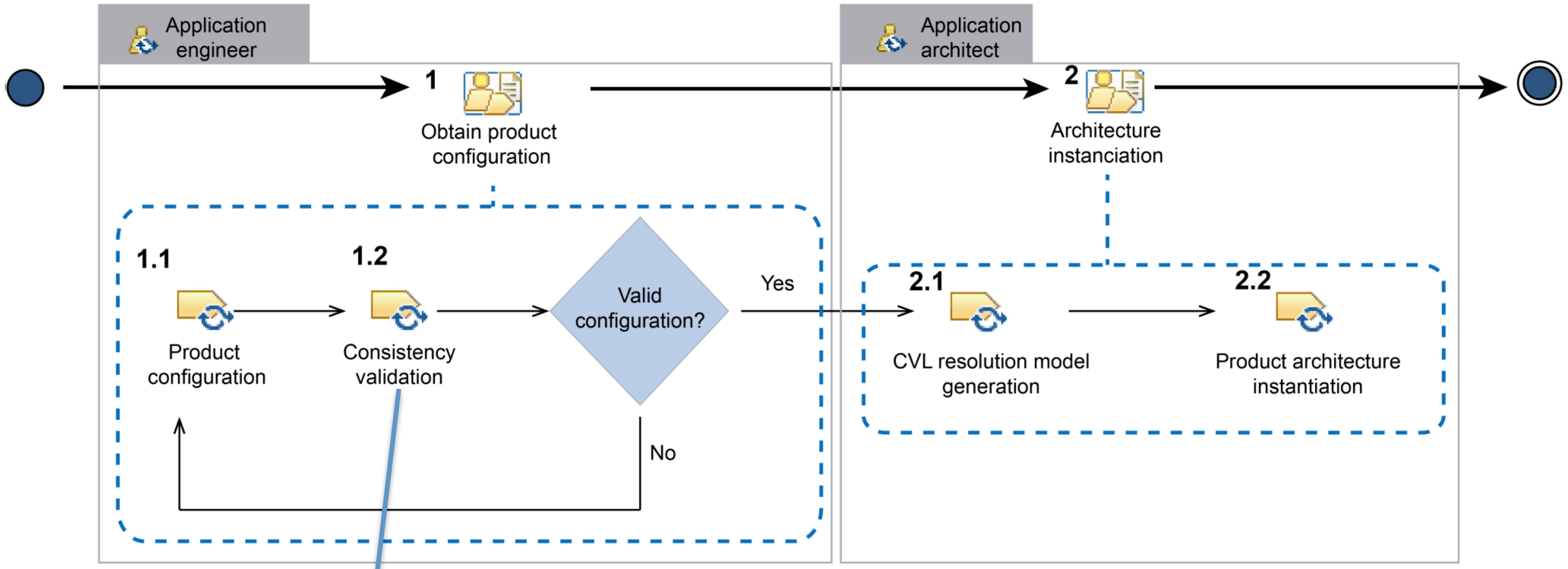
Relationships Used during Configuration



Relationships Used during Derivation



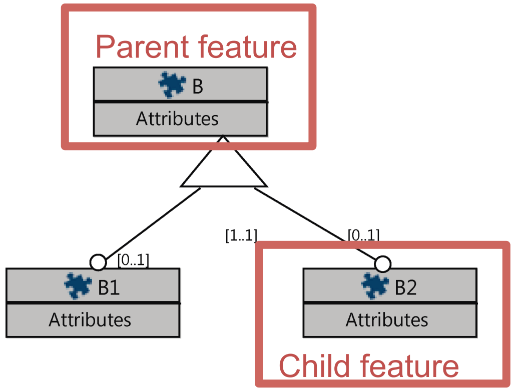
QuaDAI: Derivation Process

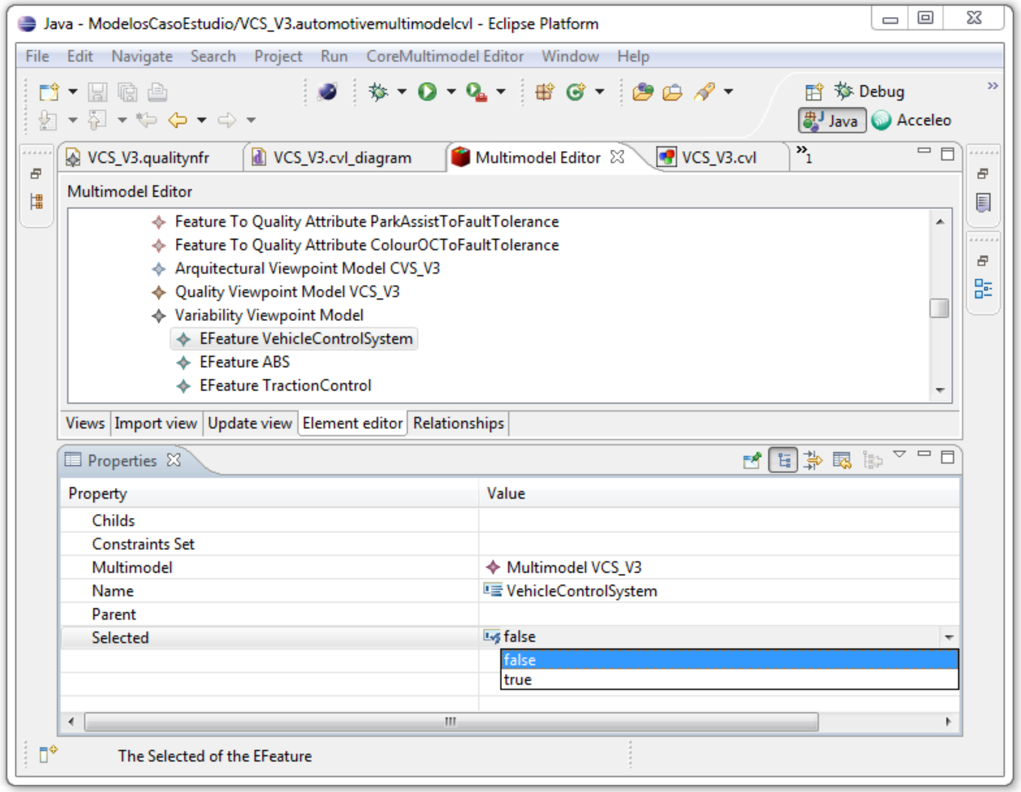


FaMa Framework
http://www.isa.us.es/fama/

Product Configuration

- **1. Select the features** that are required for the product.
  - Select the root feature.
  - if a *child feature* is selected, then its parent feature must be selected.




Java - ModelosCasoEstudio/VCS_V3.automotivemultimodelcvl - Eclipse Platform
File Edit Navigate Search Project Run CoreMultimodel Editor Window Help
Debug
Java
Acceleo
VCS_V3.qualitynfr
VCS_V3.cvl_diagram
Multimodel Editor
VCS_V3.cvl
Views
Import view
Update view
Element editor
Relationships
Properties

Multimodel Editor

- Feature To Quality Attribute ParkAssistToFaultTolerance
- Feature To Quality Attribute ColourOCToFaultTolerance
- Arquitectural Viewpoint Model CVS_V3
- Quality Viewpoint Model VCS_V3
- Variability Viewpoint Model
  - EFeature VehicleControlSystem
  - EFeature ABS
  - EFeature TractionControl

Property	Value
Childs	
Constraints Set	
Multimodel	Multimodel VCS_V3
Name	VehicleControlSystem
Parent	
Selected	false

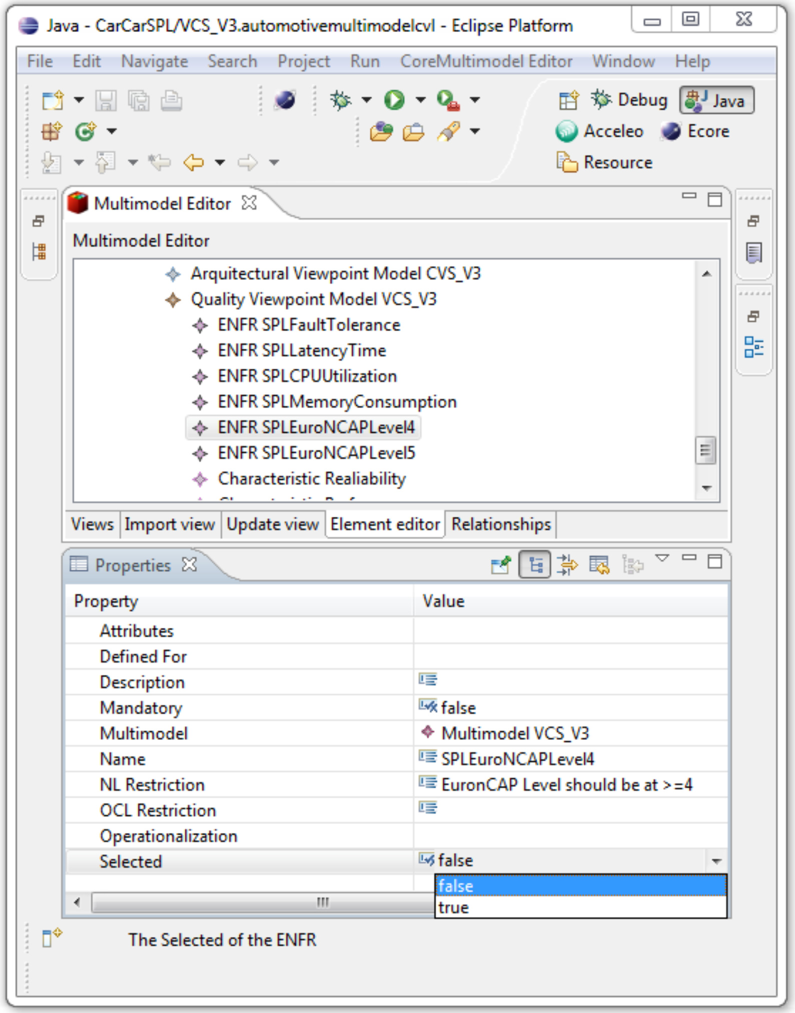
false
true

The Selected of the EFeature

Product Configuration

- **2. Select the SPL and the product specific NFRs** that the product has to fulfill.
- If a ***product specific NFR restricts a SPL's NFR***, both should be selected:
  - The relationships NFR-features are defined by using the SPL's NFR.

The probability of failure of our systems usually is below 0.00006 but in this specific case the requirements state that the probability of failure should be below 0.00004.


Java - CarCarSPL/VCS_V3.automotivemultimodelcvl - Eclipse Platform

Multimodel Editor

- Arquitectural Viewpoint Model CVS_V3
- Quality Viewpoint Model VCS_V3
  - ENFR SPLFaultTolerance
  - ENFR SPLLatencyTime
  - ENFR SPLCPUUtilization
  - ENFR SPLMemoryConsumption
  - ENFR SPLEuroNCAPLevel4
  - ENFR SPLEuroNCAPLevel5
  - Characteristic Realiability

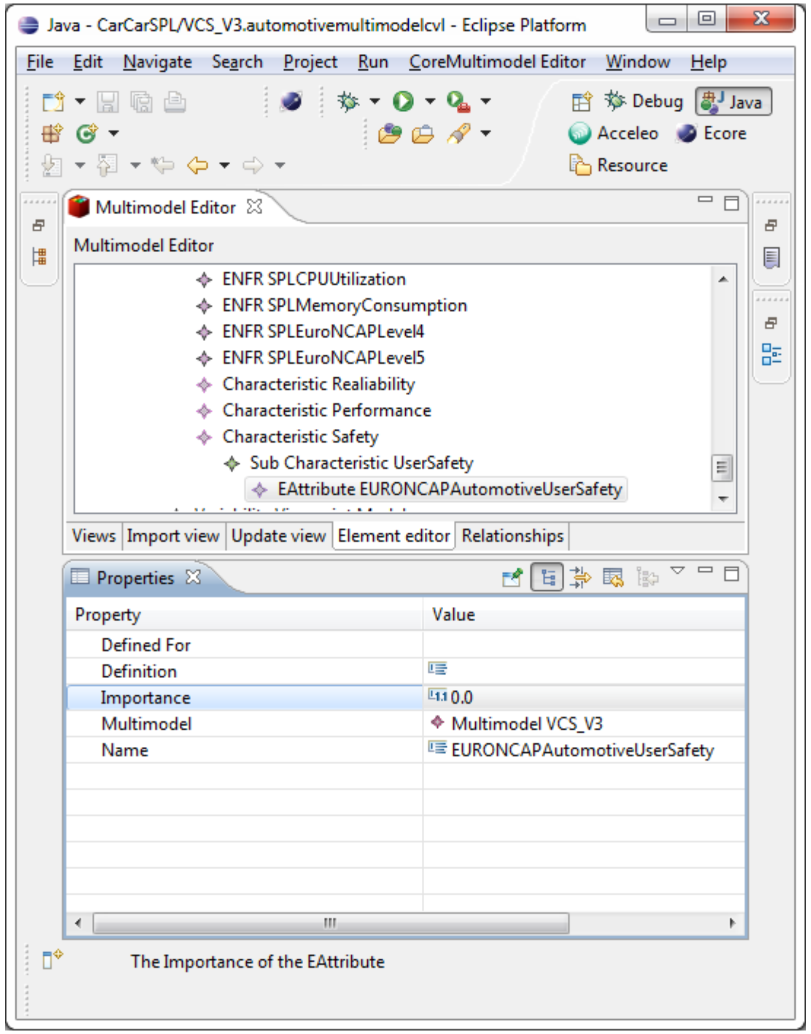
Properties

Property	Value
Attributes	
Defined For	
Description	
Mandatory	false
Multimodel	Multimodel VCS_V3
Name	SPLEuroNCAPLevel4
NL Restriction	EuronCAP Level should be at >=4
OCL Restriction	
Operationalization	
Selected	false

The Selected of the ENFR

Product configuration

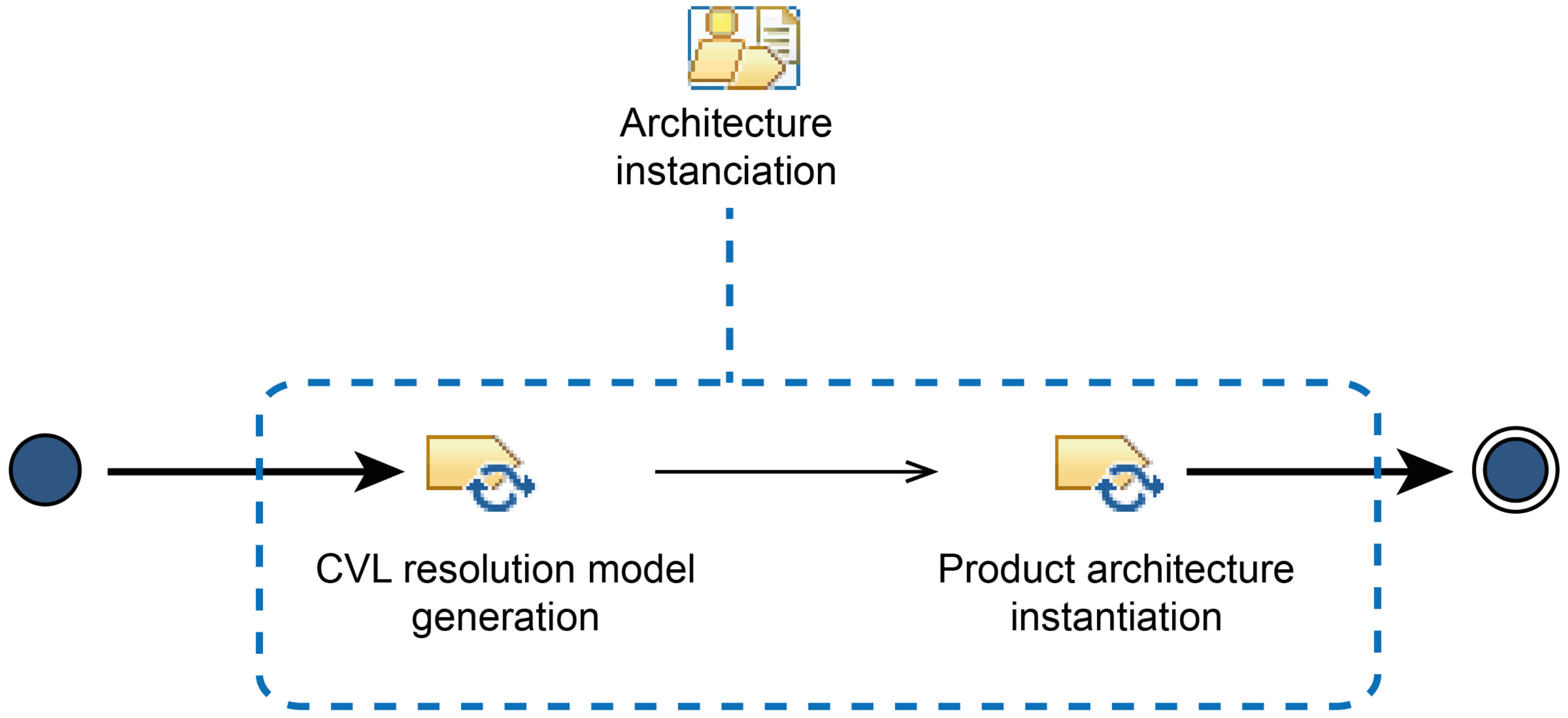
- **3. Prioritize quality attributes** (values ranging from 0 to 1)**.**
  - Relative importance of quality attributes (1 for critical 0 for trivial).
  - Leave some degrees of freedom:
    a. For quality attributes that are impacted negatively by other prioritized quality attributes.
    b. For quality attributes that, have certain importance, but have no constraints or requirements on the product.


Java - CarCarSPL/VCS_V3.automotivemultimodelcvl - Eclipse Platform
File Edit Navigate Search Project Run CoreMultimodel Editor Window Help
Debug
Java
Acceleo
Ecore
Resource
Multimodel Editor
Multimodel Editor
ENFR SPLCPUUtilization
ENFR SPLMemoryConsumption
ENFR SPLEuroNCAPLevel4
ENFR SPLEuroNCAPLevel5
Characteristic Realiability
Characteristic Performance
Characteristic Safety
Sub Characteristic UserSafety
EAttribute EURONCAPAutomotiveUserSafety
Views
Import view
Update view
Element editor
Relationships
Properties

Property	Value
Defined For	
Definition	
Importance	0.0
Multimodel	Multimodel VCS_V3
Name	EURONCAPAutomotiveUserSafety

The Importance of the EAttribute

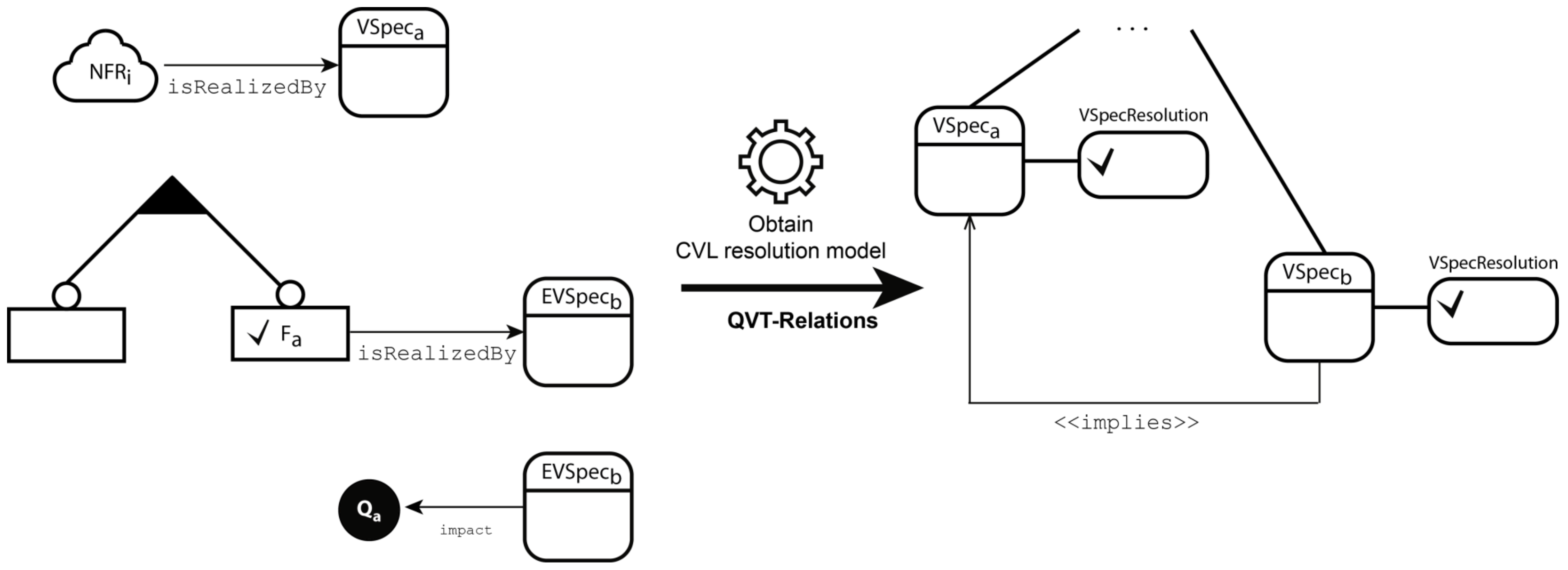
Architecture Instantiation



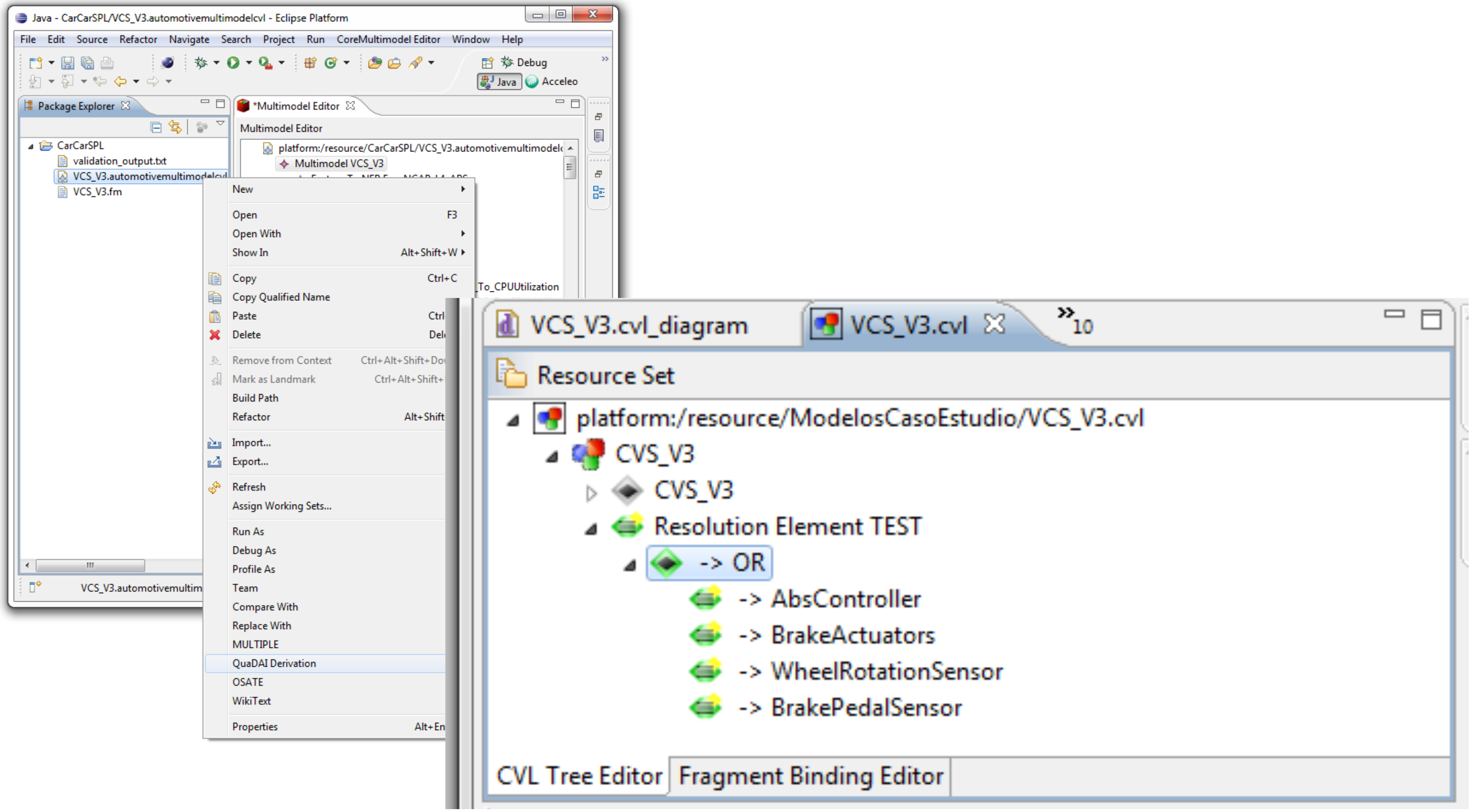
Architecture instanciation
CVL resolution model generation
Product architecture instantiation

CVL Resolution Model Generation

- The relationships among **architectural variation points**, **features, NFRs** and **quality attributes**, are used now to derive **the CVL resolution model** that will allow us to obtain the first version of the architecture.

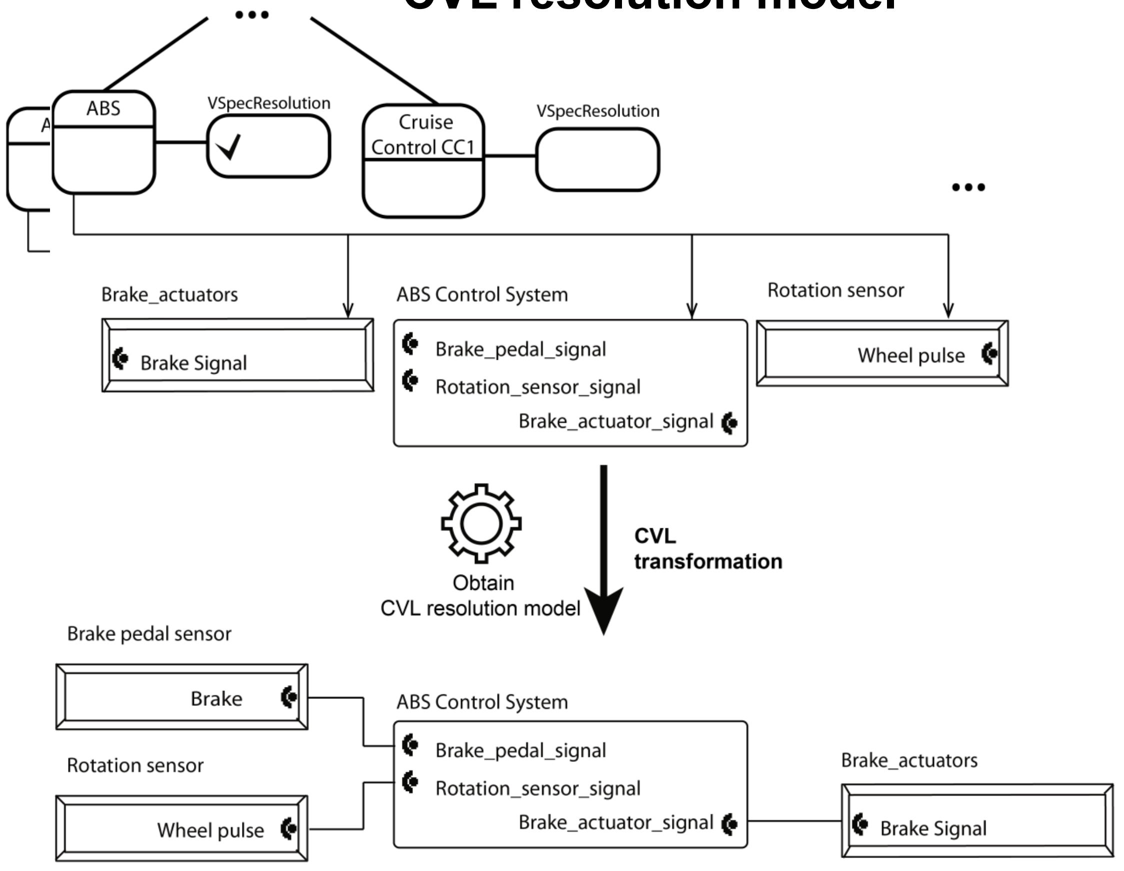


CVL Resolution Model Generation



Architecture Materialization

CVL resolution model



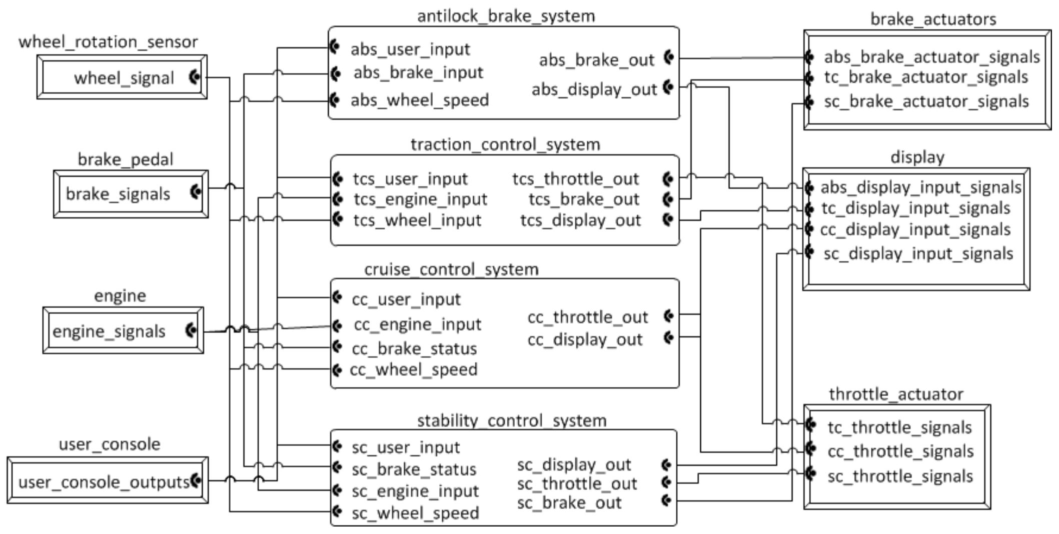
Architecture Evaluation

- The product architecture evaluation is carried out by applying a ***quality-driven model transformation*** process*
  - *Architectural patterns* are represented as ***architectural transformations***
  - The application of architectural transformations generates different product architectures that satisfies different quality attributes.
  - The domain expert should *establish* the impacts among architectural transformations and quality attributes. These impacts can be determined by using **empirical evidence or the domain expert's experience**.
  - A trade-off analysis among quality attributes and architectural transformations is performed using the ***Analytic Hierarchy Process (AHP).***
    - The result of the AHP is **a comparison matrix that shows the relative importance of each alternative with regard to each quality attribute.**
    - It is used in a quality-driven model transformation to select the appropriate architectural transformation to be applied.

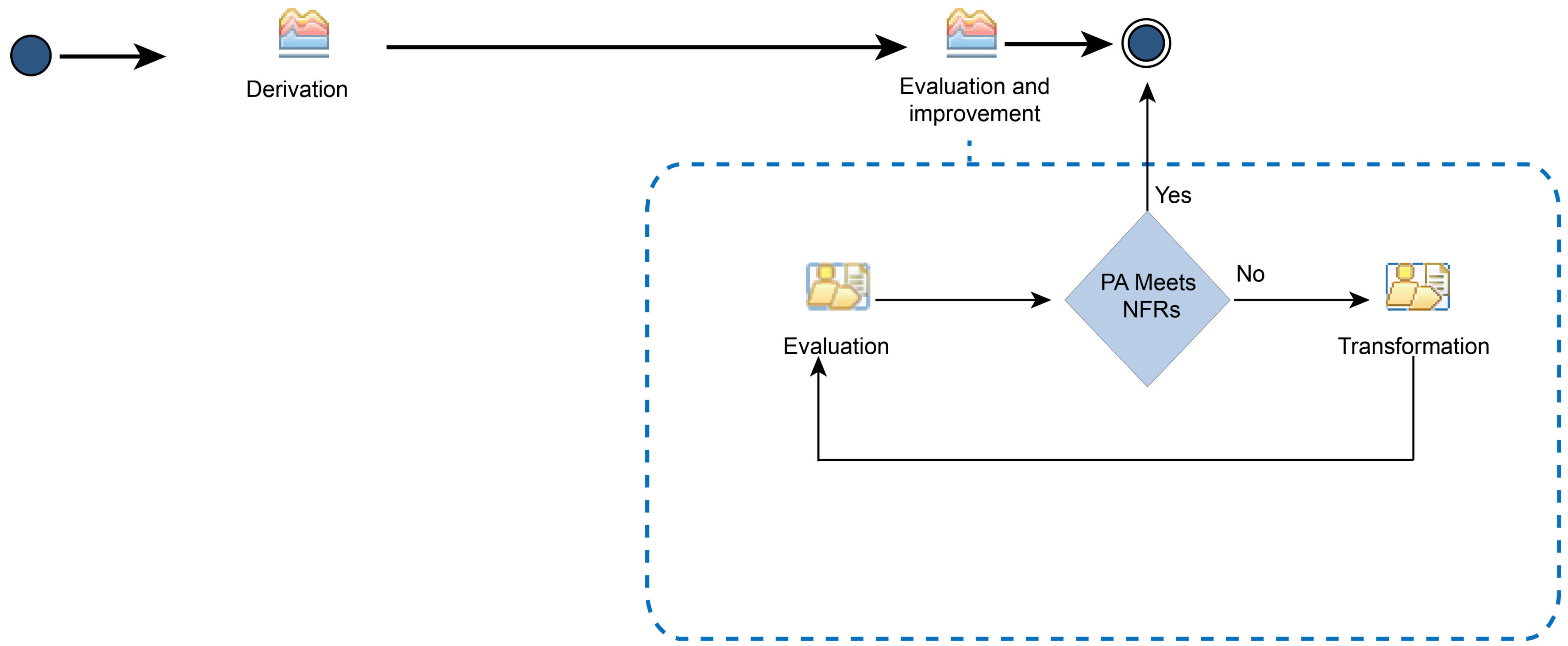
Emilio Insfrán, Javier Gonzalez-Huerta, Silvia Abrahão: Design Guidelines for the Development of Quality-Driven Model Transformations. MoDELS 2010: 288-302

Example of architecture evaluation

- The Vehicle Control System contains several subsystems (features):
  - **Antilock Braking System (ABS):** ensures that the maximum braking force is transmitted to all four wheels of the vehicle.
  - **Traction Control System (TCS):** prevents the wheels from slipping.
  - **Stability Control System (SCS):** keeps the vehicle going in the direction in which the driver is steering the car.
  - **Cruise Control System (CC):** attempts to maintain a constant driver determined.



Evaluation and Improvement

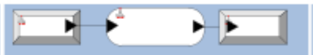
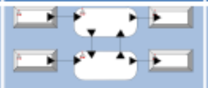
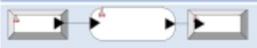
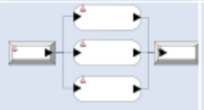


Example: Quality Attributes

- **Reliability:** the degree to which a system, product or component performs specified functions under specified conditions
  - ***Fault tolerance:*** the degree to which a system operates as intended despite the presence of hardware or software faults.
- **Performance:** characterized by the amount of resources used under stated condition for a stated period of time
  - ***Time-behavior:*** the degree to which the response and processing times and throughput rates of a product or system meet the requirements when performing its functions.
    - ***Latency time:*** time elapsed between firing an input event and obtaining the response from the system.

Example: Architectural Transformations

- The alternative architectural transformations considered here are:
  - **The Homogeneous Redundancy pattern (HR)**
    - **Improves reliability** offering two units of subsystem monitoring and performing the same operations on the input signals.
    - The primary channel runs as long as there are no problems detected.
    - When a failure in the primary channel is detected, the system switches to the backup channel and vice versa. *There is no concurrency at run-time, only replication*.
  - **The Triple Modular Redundancy pattern (TMR)**
    - **Improves reliability and safety** of a system by offering an odd *number of channels operating in parallel* (reducing the **performance**).
    - if there is a disagreement between channels, then the results with a two out of three majority win and are sent to the actuator.

Pattern	Left hand side	Right hand side	Quality Attributes Impacted
T1: Homogenous Redundancy Pattern			Reliability
T2: Triple Modular Redundancy			Reliability Safety

Example: Trade-Off Analysis

Domain Engineering:

- The domain expert ranks the *N* architectural patterns (2) with regard to the *Q* quality attributes (2) in a pairwise comparison:
  a) An AHP weight is assigned (e.g., TMR is strongly most important than HR = 5)
  b) The resulting matrix in (a) is normalized applying formula (1)
  c) The Impact is calculated applying formula (2)

$$NormQ_y[i,j] = \frac{Q_a[i,j]}{\sum_{k=1}^{n} Q[k,j]} \qquad (1)$$

$$I[i] = \frac{\sum_{k=1}^{n} NormQ_a[i,k]}{n} \qquad (2)$$

(a)	Fault Tolerance		Latency	
	TMR	HR	TMR	HR
TMR	1	5	1	1/3
HR	1/5	1	3	1
Sum	1.2	6	4	1.3

(b)	Fault Tolerance		Latency	
	TMR	HR	TMR	HR
TMR	1/1.2	5/6	1/4	0.3/1.3
HR	0.2/1.2	1/6	3/4	1/1.3

(c)	Impacts	
	Fault Tolerance	Latency
TMR	0.83	0.24
HR	0.17	0.76

Example: Transformation Result

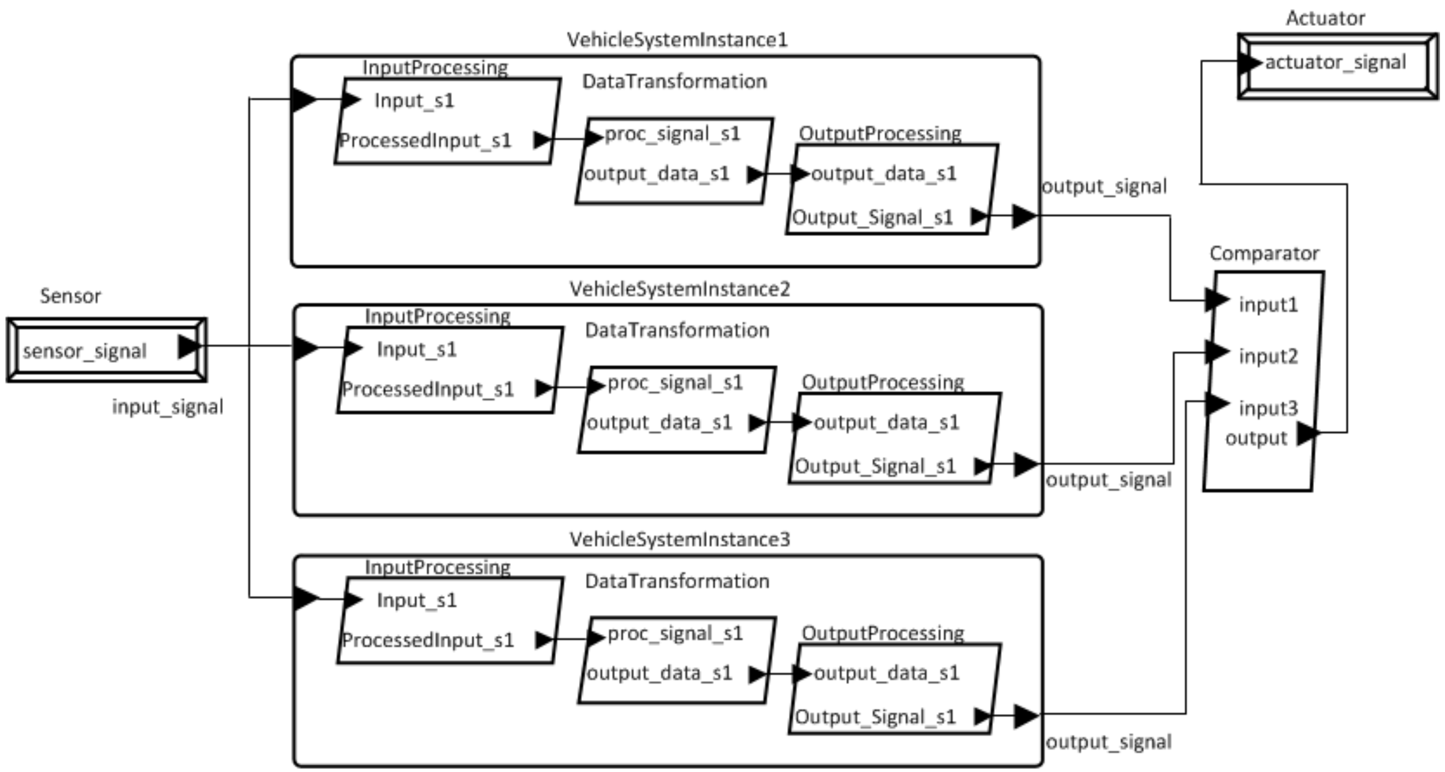
Application Engineering:

- The Application Engineer introduces the ***quality attribute levels Q that the specific product must fulfill as normalized weights ranging from 0 to 1.***
- For *k* quality attributes, the transformation process calculates the ranking *R* for each pattern *j* by applying the following equation.
  - For example, introducing a ***weight of 1 for fault tolerance and 0 for latency*** will make the transformation process to select the ***TMR pattern*** using the impact values in the Table (c) (TMR: 1*0.83+0*0.24 > HR: 1*0.17 + 0* 0.76).

$$R_j = \sum_{i=0}^{k-1} Qi * Iij$$

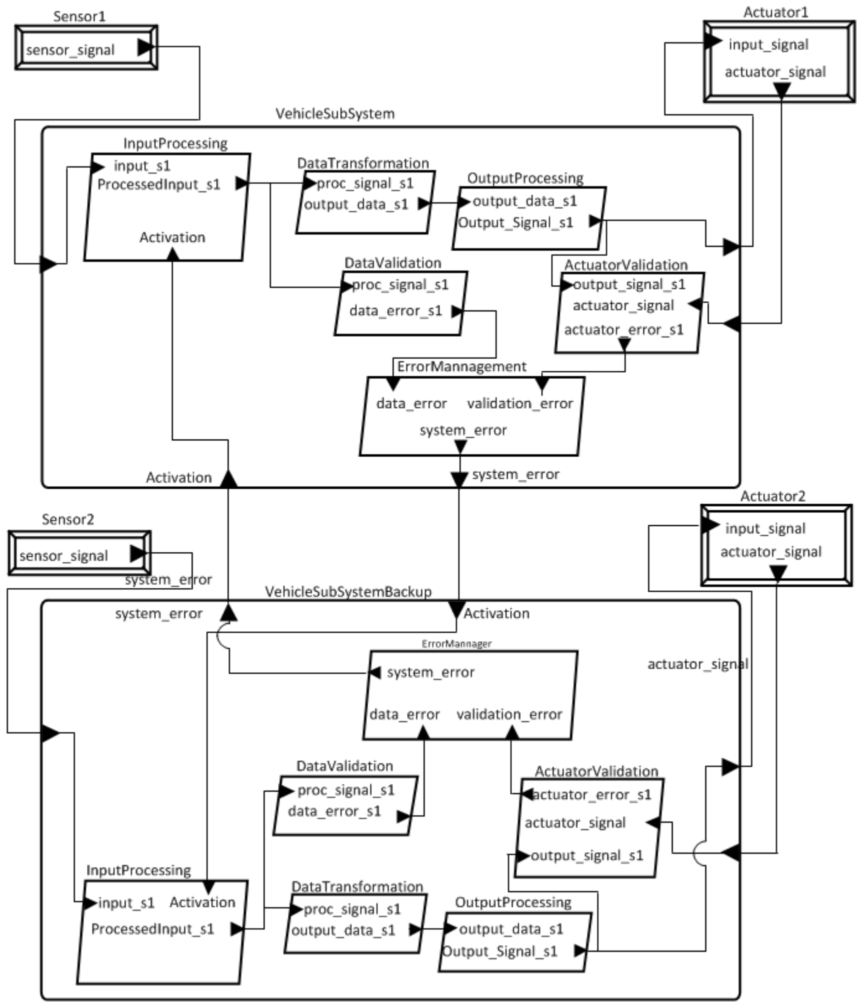
Example: Transformation Result

- According to table (c), if the quality attribute selected is ***fault tolerance*** the transformation will select and apply the ***triple modular redundancy*** *pattern*



Example: Transformation Result

- If the quality attribute selected ***latency*** the transformation will select and apply the ***homogenous redundancy*** *pattern.*
- The approach supports multi-criteria quality attributes selection.

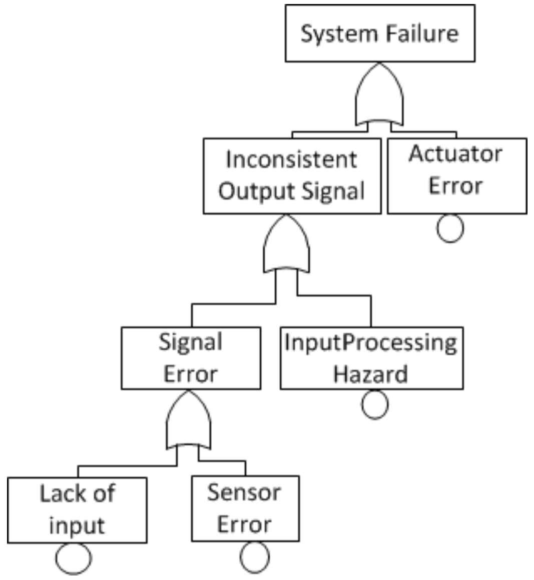


Example: Architecture Evaluation

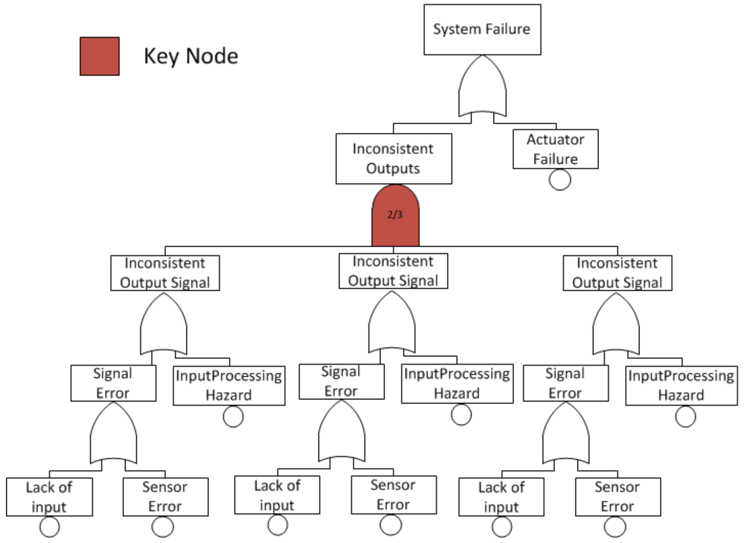
- After applying the architectural transformation, we evaluate the derived product architecture to assess if the application of the architectural ***transformation pattern resulted in an improvement*** of the product architecture quality.
- We ***compare*** the measures values obtained over the product architectures derived ***with* and *without* applying the architectural pattern**.
- As an example, we use the the fault tolerance quality attribute to illustrate the product architecture evaluation:
  - The ***fault tolerance*** attribute is measured by applying the ***Key Node Safety (KNS) metric*** on a fault tree for the product architecture.
  - The value of the KNS metric expresses how a mutation of a system improves its fault tolerance; ***the higher value of the metric is the better the fault tolerance the system has*.**

Example: Key Node Identification

Fault tree of the original architecture



Fault tree after applying TMR Pattern



Example: Metric Operationalization

- The following formula calculates the *key node safety* (KNS) metric:

$$S = \frac{kh'}{n^2} \sum_{i=0}^{k-1} \frac{c_i}{d'_i}$$

	Original	TMR
k: Number of key nodes in the fault tree	0	1
h': Total height of the fault tree +1	5	6
n: Total number of nodes in the fault tree	7	18
c_i: Number of nodes in the sub-tree rooted at key node k_i	0	15
d'_i: Depth of the sub-tree rooted at key node *ki* +1	0	4
***S:* Key Node Safety Metric**	**0**	**0.069**

*The metric results indicates that the **TMR pattern** slightly **improves the fault tolerance** of the product when compared to the values of the original product architecture.*

Conclusions

- The use of MDE principles as a way to provide a ***richer semantic representation*** of a software product line (the multimodel).
- The approach explore MDE concepts and techniques to ***make explicit the knowledge and rationale*** used for architectural design.
  - Capturing and representing architectural design decisions during the architecting process is necessary for ***reducing architectural knowledge evaporation***
  - The multimodel is a solution for ***documenting design decisions*** and their impact on the product quality attributes.
  - The multimodel can be used to ***analyze the cost/benefit*** of having core assets with certain qualities (impact on quality and cost)
  - The evaluation process was found to be ***useful to novice software architects*** (empirical validations with practitioners)

Conclusions

- Model-driven development (MDD) helps to meet time-to-market and other business goals.
- The multimodel provides a sufficiently ***formal interrelated model*** that can be supported by ***tools capable of automating*** portions of the Product Line Production Planning.
- MDD relies on industry standards: part of the ***production strategy*** and production methods ***could easily be reused across SPLs***.
- The approach improves traditional MDE practices
  - Flexible mechanism for modeling the relationships among elements of different viewpoint models, rather than introducing this information directly into the model transformation definitions.

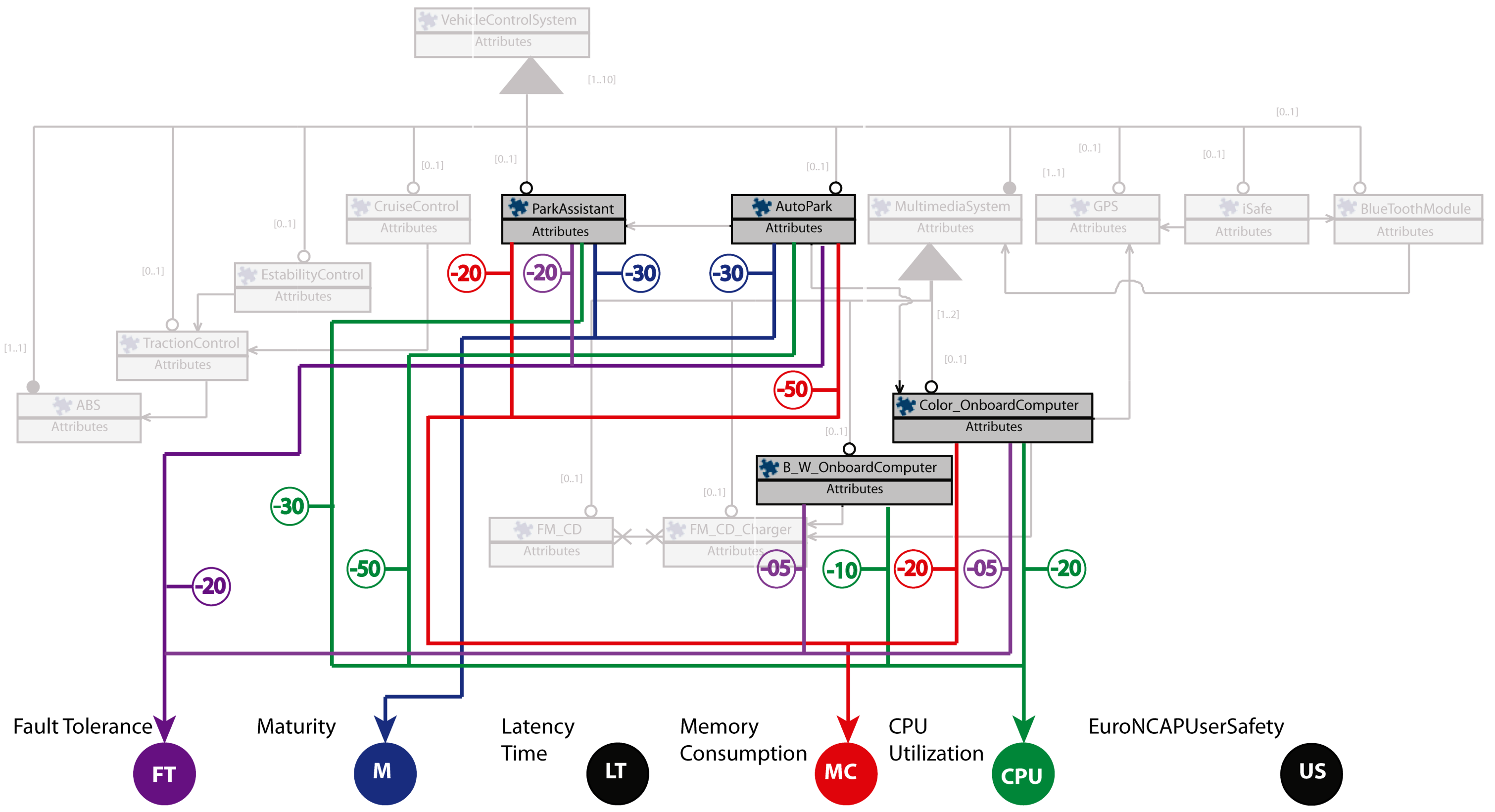
Thank you!

Silvia Abrahão

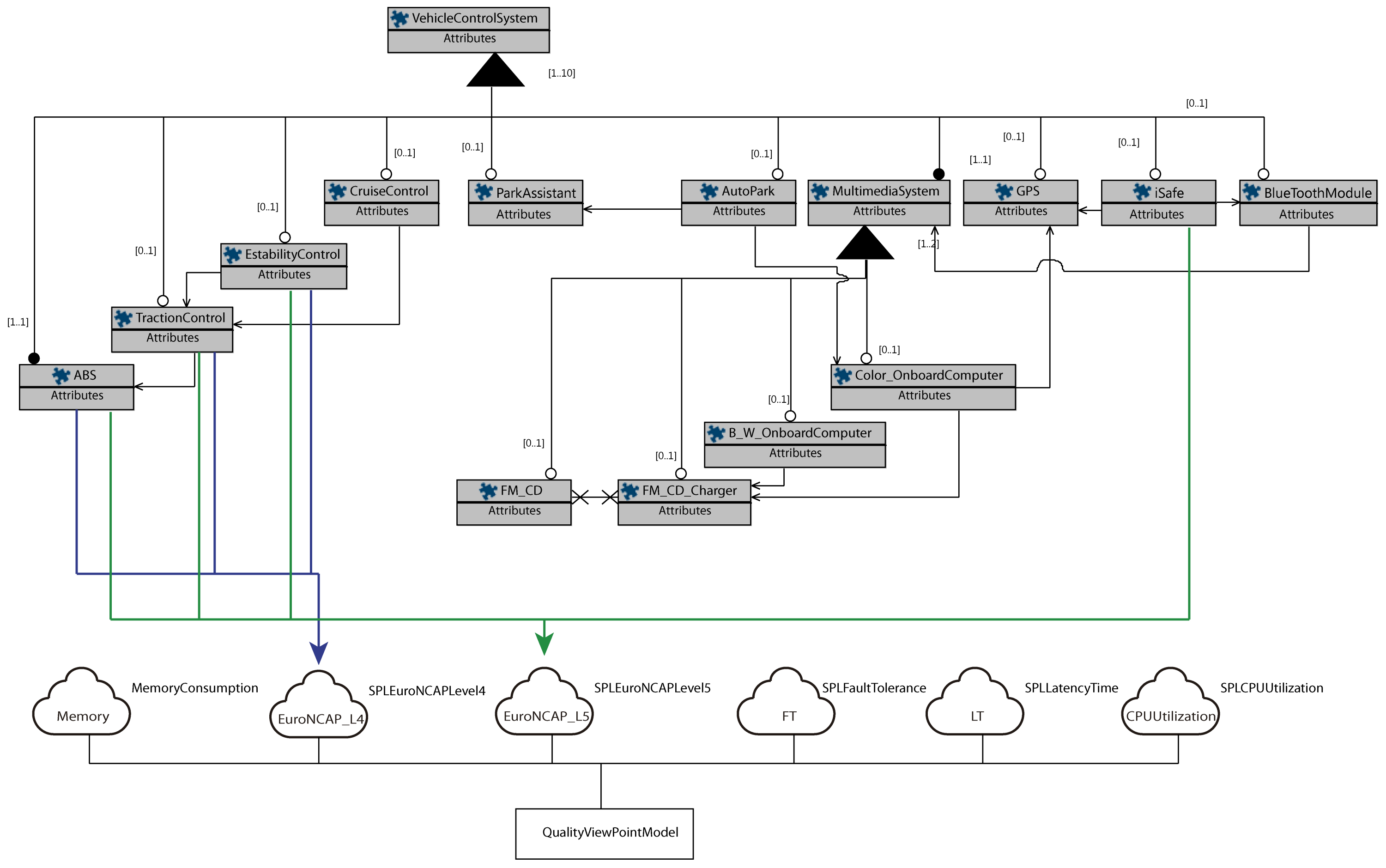
Universitat Politècnica de València

sabrahao@dsic.upv.es

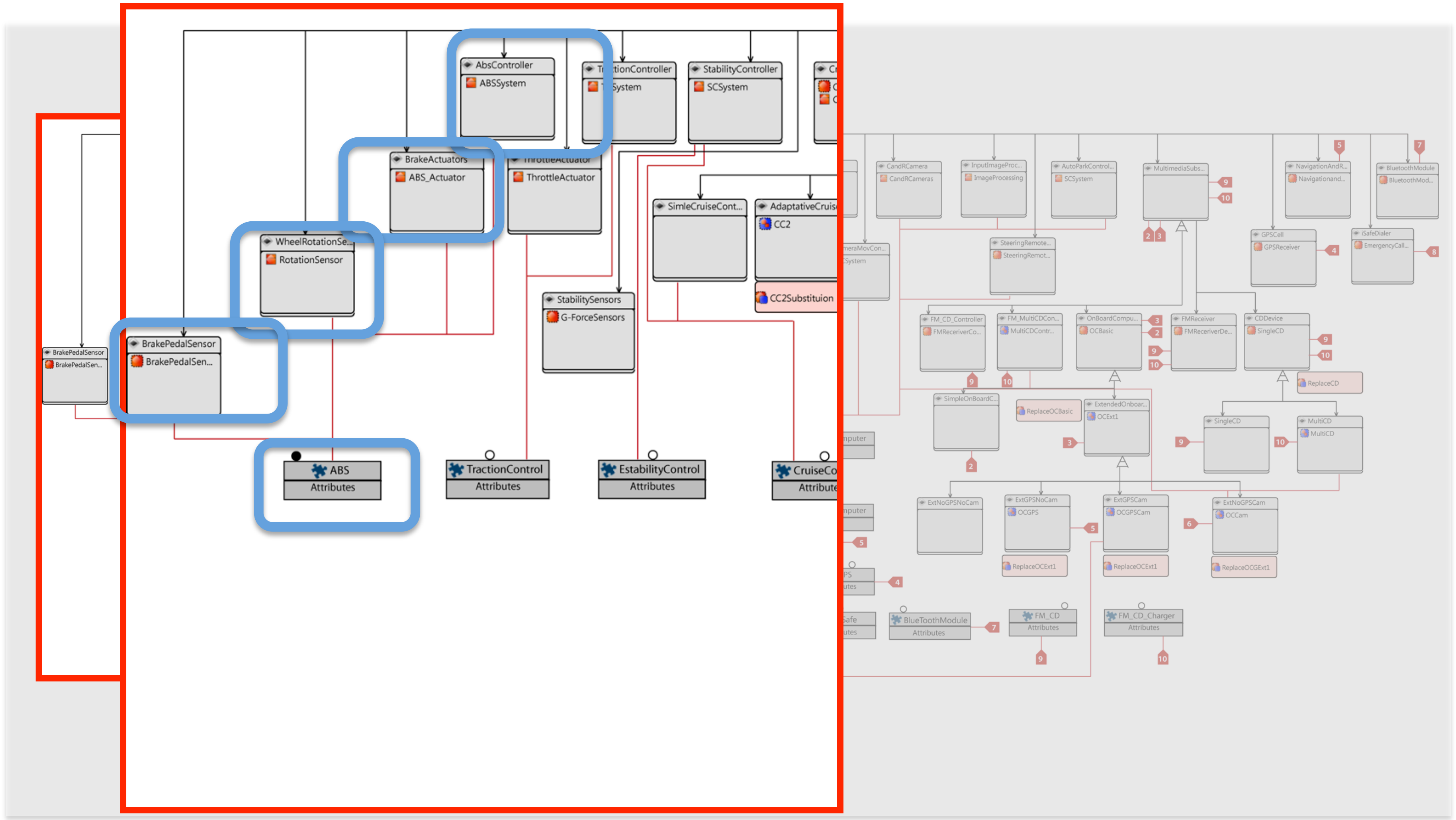
Relationships Features-Quality Attributes



Relationships Features-NFRs

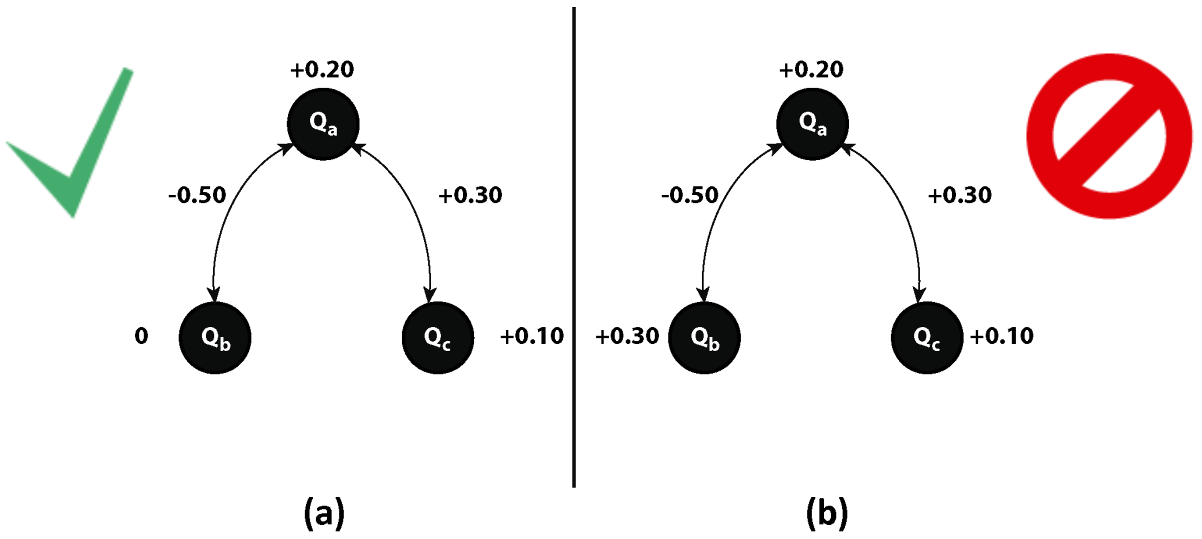


Relationships Features-ArchVariability points (CVL)

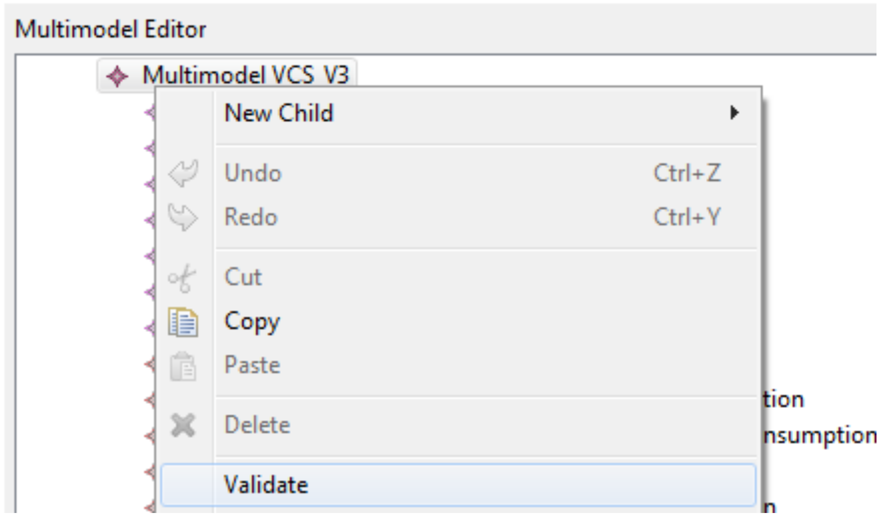


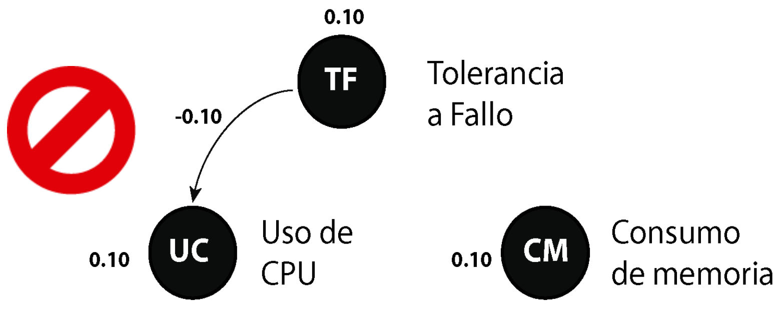
Quality Viewpoint Consistency

- For validating the **consistency of the quality viewpoint** we analyze that the prioritized quality attributes (QA) do not have negative impact relationships among them:
  - **(a)**: The configuration there are no prioritized QAs that have negative impacts among them (Qb has no priority)
  - **(b)**: The configuration includes a pair of QAs which impact negatively on the other (Qa and Qb)

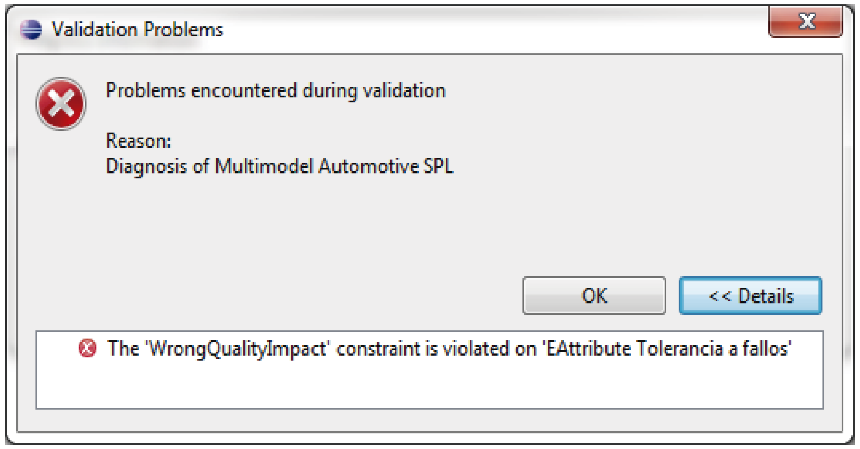


Quality Viewpoint Consistency Validation





(a)


Validation Problems

Problems encountered during validation

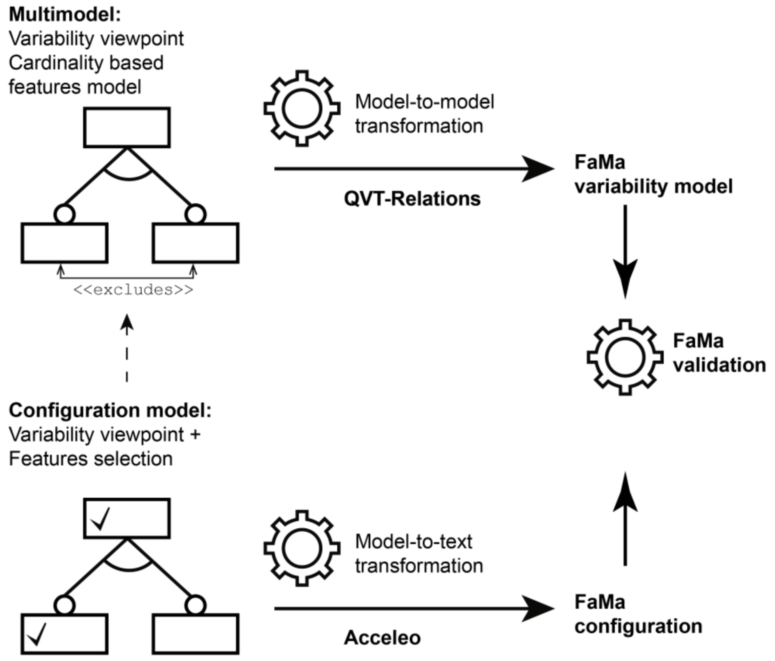
Reason:
Diagnosis of Multimodel Automotive SPL

OK | << Details

The 'WrongQualityImpact' constraint is violated on 'EAttribute Tolerancia a fallos'

(b)

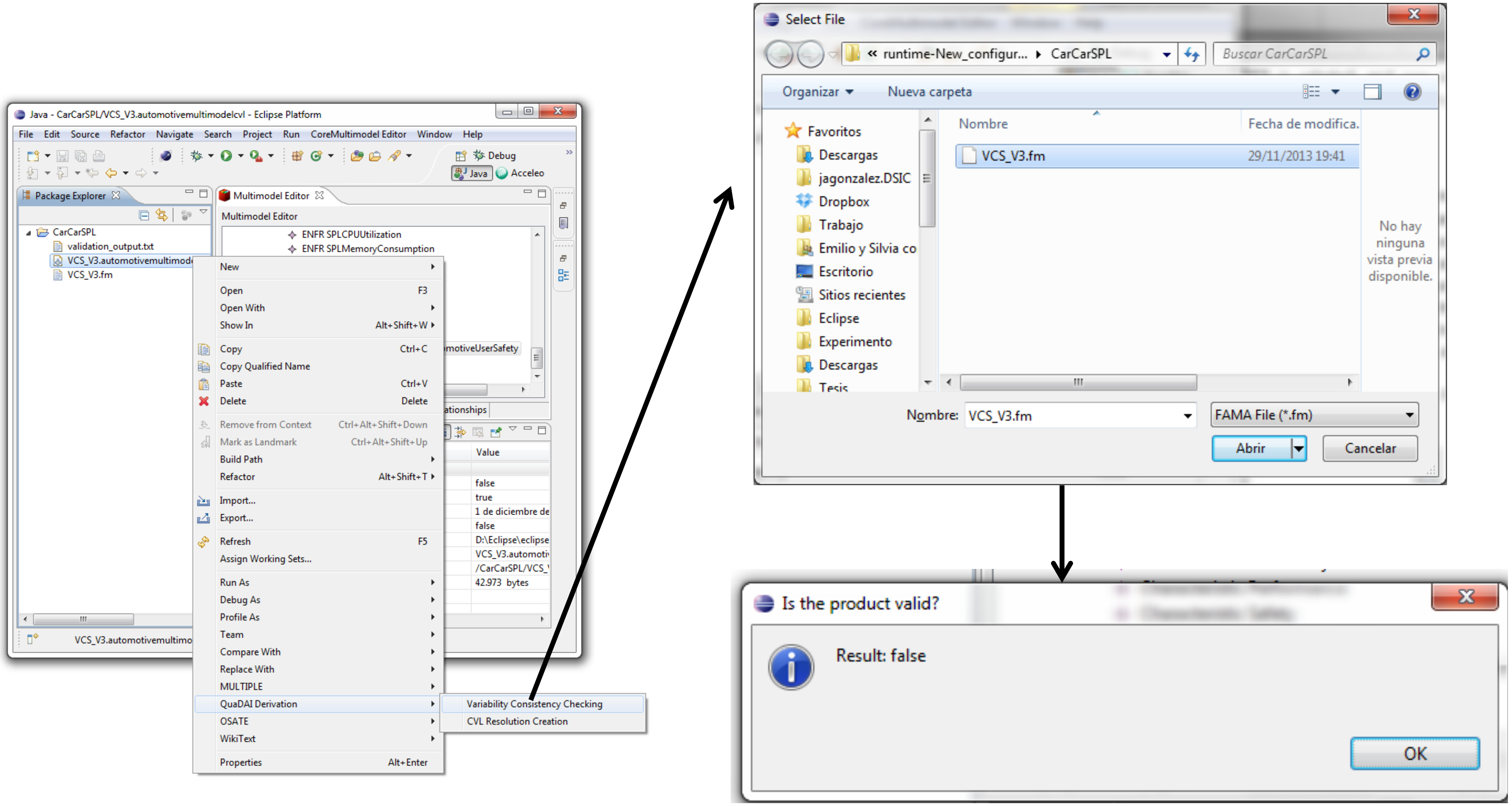
Variability Consistency Validation



- The **variability consistency validation** checks the conformance of the selected set of features with the constraints and restrictions defined in the **Feature Model:**
  - We translate the Feature Model to the FaMa Tool[1] representation.
  - We inject the selected features to the FaMa validator and obtain whether the set of features is a valid configuration or not.

[1] FaMa Framework© ISA research group
http://www.isa.us.es/fama/

Variability Consistency Checking

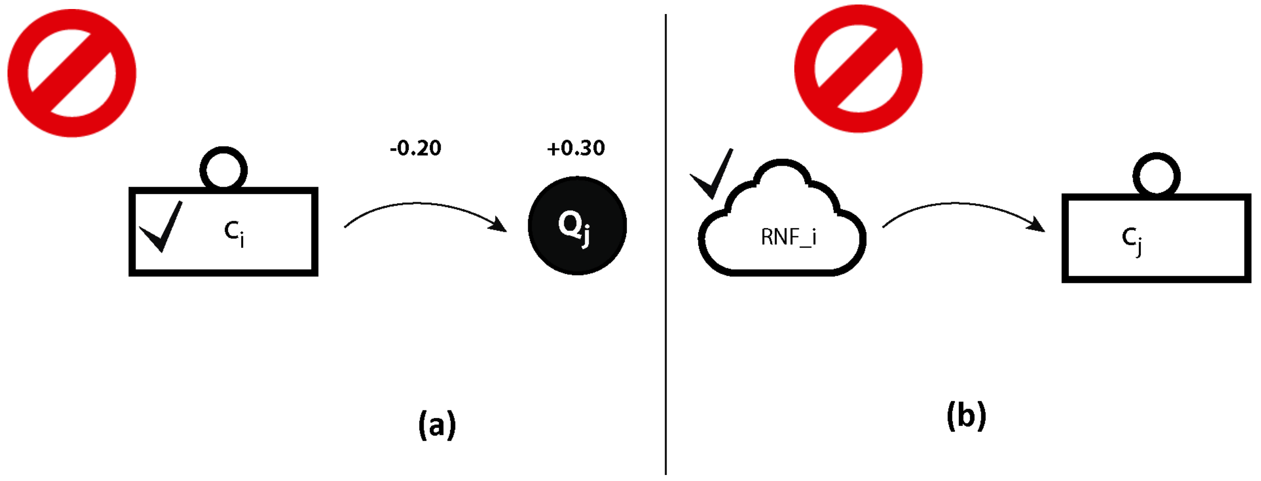


Is the product valid?

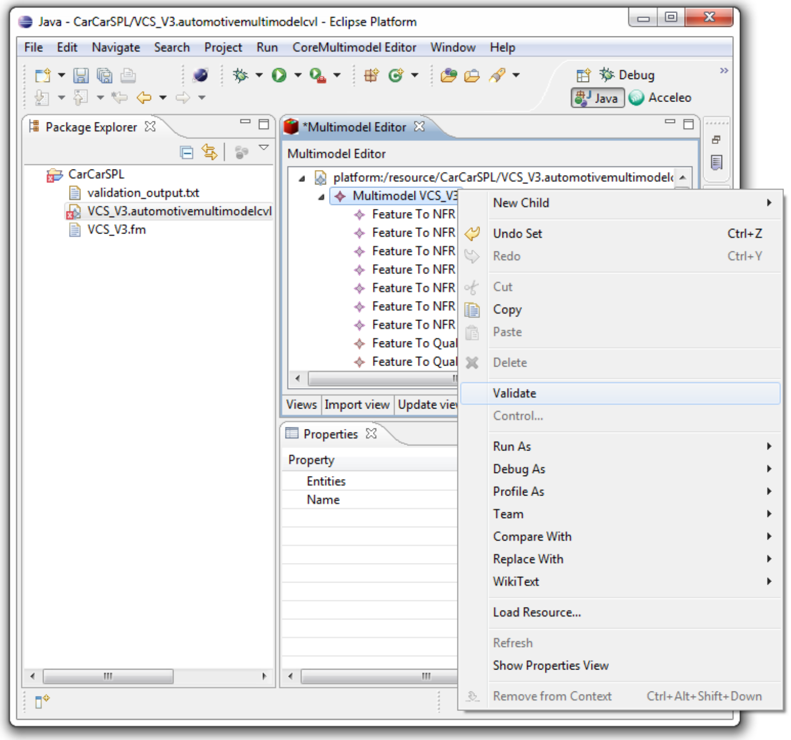
Result: false

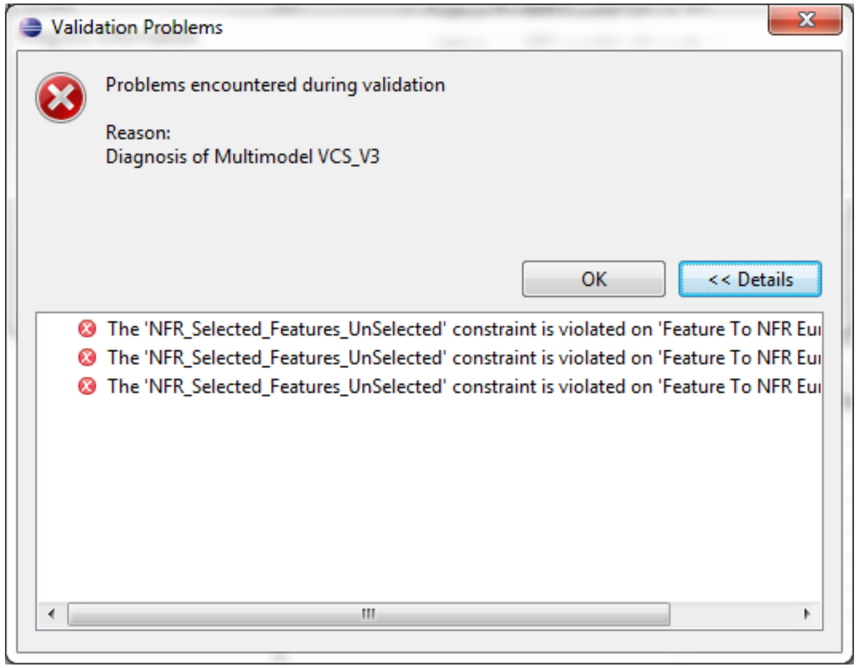
Inter-Viewpoint Consistency Checking

- The **consistency** among the viewpoints should be checked to assure that the selected features, NFRs and priorities of quality attributes meet the constraints we have defined in the multimodel by means of the multimodel relationships.
- We can check two main issues:
  - That there is no feature selected which impacts negatively on a prioritized quality attribute.
  - That all the features that realize the selected NFRs had been selected.



Inter-Viewpoint Consistency Checking




Validation Problems

Problems encountered during validation

Reason:
Diagnosis of Multimodel VCS_V3

The 'NFR_Selected_Features_UnSelected' constraint is violated on 'Feature To NFR Eu
The 'NFR_Selected_Features_UnSelected' constraint is violated on 'Feature To NFR Eu
The 'NFR_Selected_Features_UnSelected' constraint is violated on 'Feature To NFR Eu