26th annual **INCOSE**
international symposium

Edinburgh, UK
July 18 - 21, 2016

Delta Rhapsody

Oskar Berreteaga

ULMA Embedded Solutions

MONDRAGON
UNIBERTSITATEA

GOI ESKOLA
POLITEKNIKOA

ESCUELA
POLITÉCNICA
SUPERIOR

ULMA
Embedded Solutions

Agenda

26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

1 | Theoretical concepts

2 | Related work

3 | Architecture

4 | Example

5 | Conclusions & Future Work

Theoretical concepts

Variability

definition

Software variability is the ability of a software system or artefact to be changed, customized or configured for its use in a particular context.

A high degree of variability allows the usage of software in a broader range of contexts, i.e. the software is more reusable. Variability can be viewed as consisting of two dimensions, i.e. space and time. The space dimension is concerned with the utilization of software in multiple contexts, e.g. multiple products in a software product family. The time dimension is concerned with the ability of software to support evolution and changing requirements in its various contexts.

Software Variability Management
Jilles van Gurp, Jan Bosch
http://www.win.tue.nl/~sroubtso/svm2003-proceedings.pdf

Theoretical concepts

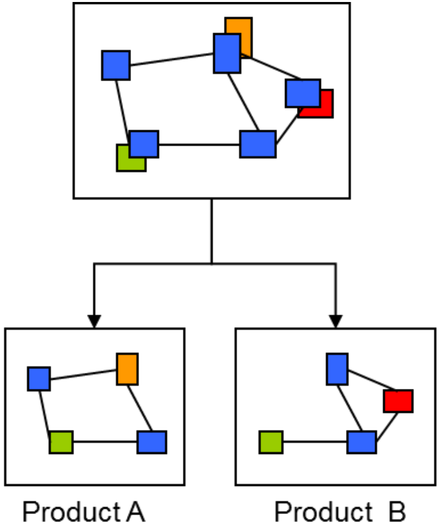
Software Product Lines

A **software product line** is "a set of software-intensive systems that share a common, managed set of features satisfying 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".

Two phases are distinguished:

– **Domain Engineering** refers to activities in which the core assets are created. An important part of domain engineering is domain analysis, during which a fundamental understanding of the domain and its commonalities and variability is established.

– **Application Engineering** is the phase in which the domain engineering artefacts are used to create products. Unless variation points use runtime binding, they are bound during this phase.



Theoretical concepts

26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

Software Product Line adoption approach

- **Proactive** approach, which develops a product line from scratch.
- **Extractive** approach, which starts with a collection of existing products and incrementally refactors them to form a product line.
- **Reactive** approach, which begins with a small, easy to handle product line (possibly consisting of a single product) and is extended incrementally with new features and implementation artefacts, thus extending the product line's scope.

Theoretical concepts

Feature Model

definition

A "feature" is defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system" (Kang 1990). A feature model is a compact representation of all the products of the Software Product Line (SPL) in terms of "features".

Feature models are visually represented by means of feature diagrams. Feature models are widely used during the whole product line development process and are commonly used as input to produce other assets such as documents, architecture definition, or pieces of code.

Kang, K.C.,Cohen, S.G., Hess, J.A.,Novak, W.E., Peterson, A.S.,
"Feature-oriented domain analysis (FODA) feasibility study",
Technical Report CMU/SEI-90-TR-021, SEI,
Carnegie Mellon University, November 1990

Theoretical concepts

Variability modelling approaches (Heidenreich et al. 2010)

- **Negative** Variability (annotative or 150% model). The negative variability approach relies on having a 150 % model, which means that all the features are allocated in the model, the model contains all the elements used for all variants of the software product line. During product derivation, the elements of the model that are not required according to the selected features are removed and the model of the variant is obtained.

- **Positive** Variability (compositional or 100% model). The positive variability approach consists of having a core model with the common or core elements for any product in the product line; and specifying which new elements must be added when a certain feature or set of features is selected. In this case, during product derivation, the core model is taken as starting point and elements are added to this model according to the selected features; and this way the model of the variant is obtained.

- **Modification** of Model Elements. In this approach, there is a reference or base model and the variant model is obtained modifying the existing model elements in this model according to the selected features.

Theoretical concepts

Variability modelling approaches: Advantages and disadvantages

Negative Variability (annotative or 150% model).

▲ All the elements are in the same model so getting an overview or spotting feature interactions is easier.

▼ It can increase model complexity especially when there are many variants, as it has to contain model elements for any product in the product line.

- Understanding or analysing this model could become quite complex.

Positive Variability (compositional or 100% model).

▲ It helps to cope with model complexity as model elements that are not common ones are in different models.

▼ There can potentially be a large number of model fragments describing the SPL as a whole, and this makes it more difficult to get an overview of the SPL.

- This makes it also more difficult to identify feature interactions.

Modification of Model Elements.

▲ It allows changes to be made inside the model elements.

▼ Variability not allowed at architectural level.

Theoretical concepts

Delta Modelling

definition

Delta modelling is a language-independent approach for modelling system variability.

In delta modelling, a set of systems is described by a designated core system and a set of system deltas that specify modifications of the core product to obtain other products. A particular system variant is obtained by applying the modifications of a selected subset of the deltas to the core product.

Delta Modeling for Software Architectures
Arne Haber1 , Holger Rendel1 , Bernhard Rumpe1 , Ina Schaefer2
http://www.se-rwth.de/publications/Delta-Modeling-for-Software-Architectures.pdf

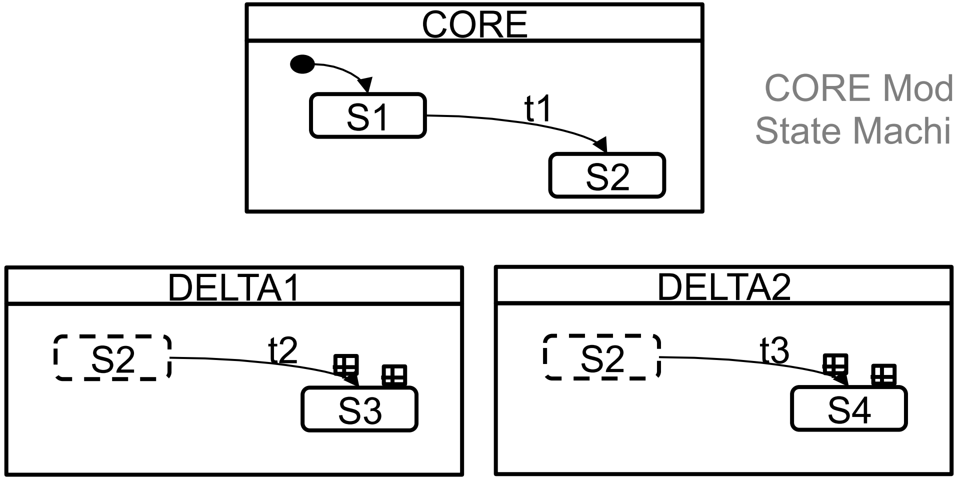
Delta-oriented Architectural Variability Using MontiCore
Arne Haber , Thomas Kutz, Holger Rendel, Bernhard Rumpe, Ina Schaefer
http://arxiv.org/ftp/arxiv/papers/1409/1409.2317.pdf

❑ Delta modelling supports variability modelling approaches:
- Negative
- Positive
- Modification

❑ Delta modelling supports SPLE adoption approaches:
- Proactive
- Extractive
- Reactive

Theoretical concepts

Delta Modelling conceptual sample



CORE Model
State Machine

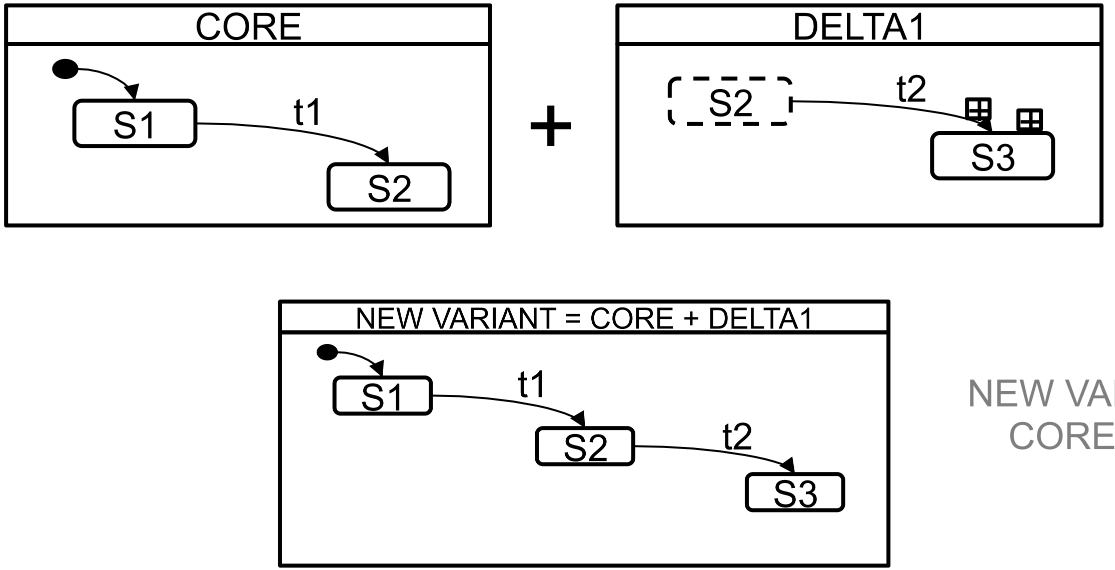
DELTA1 Model
State and transition addition

DELTA2 Model
State and transition addition

Based on Delta Modelling adaption
[Ina Schaefer 2010]

Theoretical concepts

Delta Modelling conceptual sample



NEW VARIANT Model
CORE + DELTA1

Theoretical concepts

Delta modelling: Advantages and disadvantages

Advantages

- ▲ It supports all **variability modelling approaches**: positive, negative and modification.
- ▲ It can handle **configuration** (variability in space) as well as **evolution** (variability in time) within a single notation. Variability in space is related to having several variants or products and variability in time is related to the evolution of the SPL.
- ▲ It can deal with an **open variant space** where not necessarily all configuration options are known in advance.
- ▲ It supports **proactive** (complete variability management is performed before starting with application engineering), **reactive** (the product line is updated when new variants appear) and **extractive** (existing products are taken as base for the core assets) SPLE.
- ▲ It is **language independent**.
- ▲ It supports modular and **flexible description of variability and change**. It is intuitively understandable and well-structured.

Disadvantages

- ▼ When there is a **large number of model deltas** describing the SPL as a whole, this makes it more **difficult to get an overview** of the SPL.
- ▼ It is more difficult to identify **feature interactions**.

Related work

Support for variability in Rhapsody

- ❑ **IBM Rational Rhapsody** gives some support for designing, reuse and envision product line variants (Scouler and Bakal 2013).

- ❑ The **AUTOSAR 4.0 profile** of Rhapsody provides support for AUTOSAR variation points in Rhapsody.

- ❑ There are also Product Line Engineering tools that integrate with Rhapsody thus enabling Model Driven Product Line Engineering:
  - **Pure::variants for Rhapsody by pure-systems**
  - **Rhapsody/Gears Bridge by BigLever Software**

Related work

Delta Modelling tools

- **Delta Simulink**, a tool for delta modelling for Simulink (Haber et al. 2013).
- **Delta-MontiArc** (MontiArc is an existing architecture description language) that offers an integrated modelling language for architectural variability (Haber et al. 2011).
- **Delta Ecore** provides delta modelling on basis of the structured data models conforming to meta models specified in EMF Ecore (Seidl et al. 2014).

Delta Programming tools

- **DeltaJava** programming language, a delta-oriented programming approach for Java (Schaefer et al. 2010).
- The **Abstract Behavioural Specification** language (ABS) (Clarke et al. 2010a) also proposes a delta approach for managing variability.

Architecture

Delta Rhapsody

- ❑ We have developed **Delta Rhapsody** solution, a tool for Delta modelling in IBM Rhapsody.

- ❑ A variability modelling approach is provided, which supports **positive** variability, **negative** variability and **modification** of model elements in IBM Rhapsody.

Architecture

Tools & Frameworks

IBM Rational Rhapsody

> To model system CORE & DELTAS
> To place generated New Variant

eclipse

Java

FeatureIDE

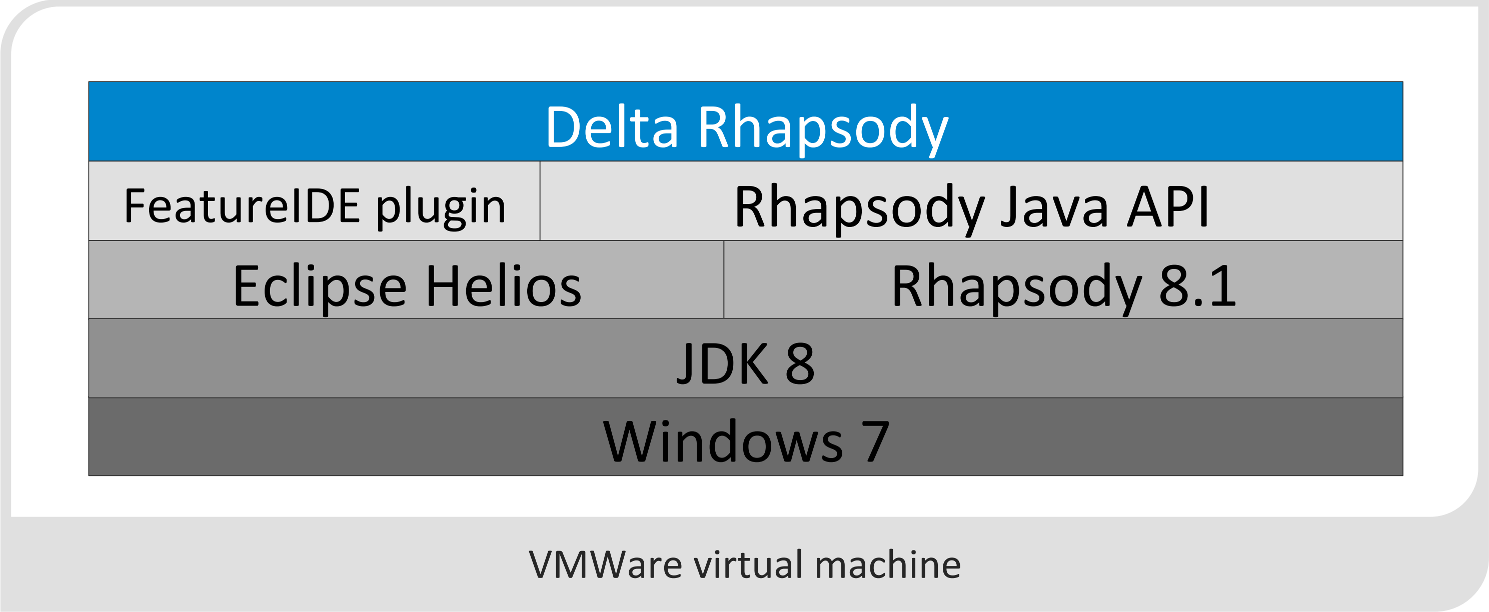
> To model & configure variability

Java API

> To generate New Variant into Rhapsody

Architecture

Tools & Frameworks



Delta Rhapsody

FeatureIDE plugin | Rhapsody Java API

Eclipse Helios | Rhapsody 8.1

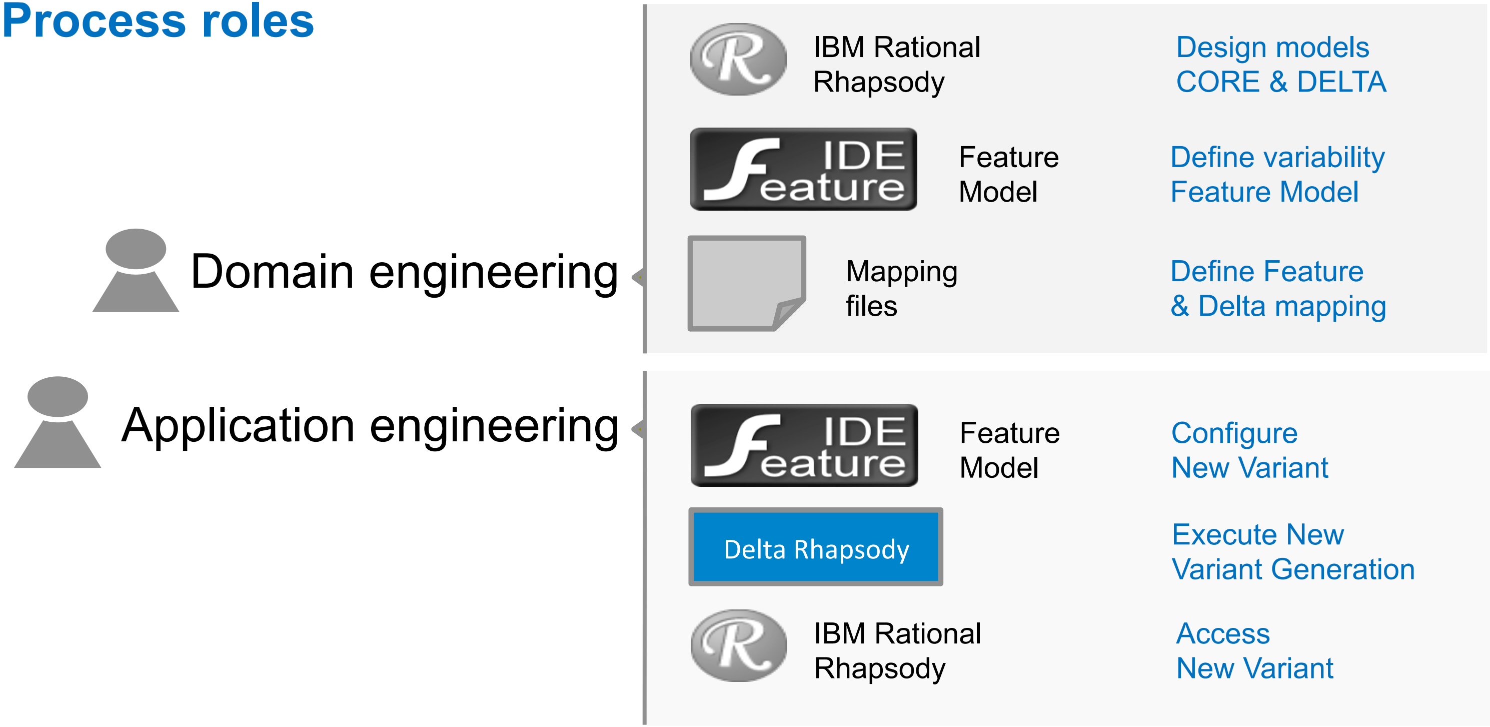
JDK 8

Windows 7

VMWare virtual machine

Architecture

Process roles



Domain engineering

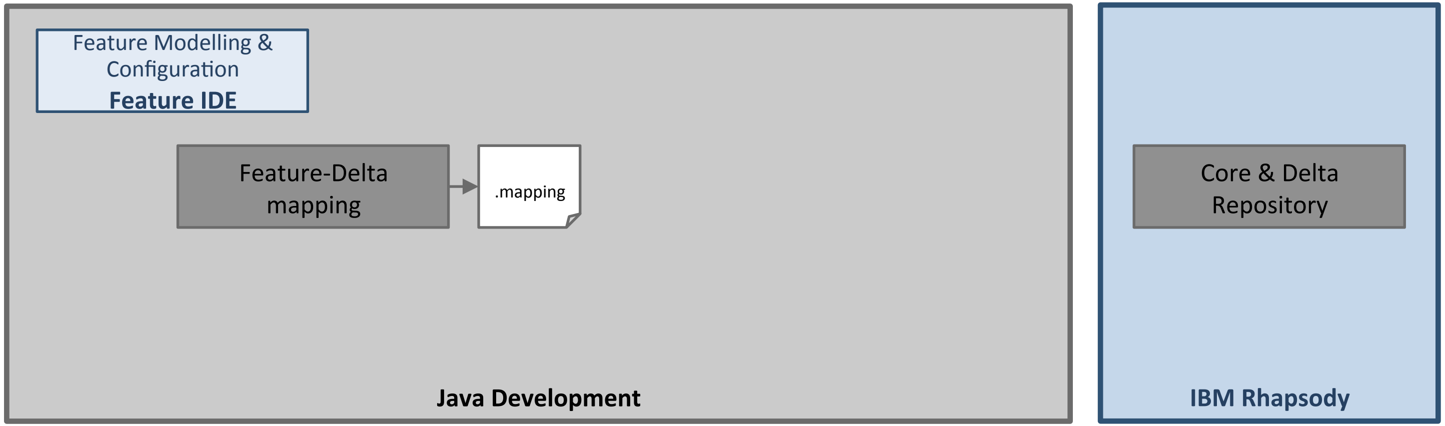
- IBM Rational Rhapsody — Design models CORE & DELTA
- Feature IDE — Feature Model — Define variability Feature Model
- Mapping files — Define Feature & Delta mapping

Application engineering

- Feature IDE — Feature Model — Configure New Variant
- Delta Rhapsody — Execute New Variant Generation
- IBM Rational Rhapsody — Access New Variant

Architecture

Process Flow for Domain Engineer



Feature Modelling & Configuration
Feature IDE

Feature-Delta mapping

.mapping

Java Development

Core & Delta Repository

IBM Rhapsody

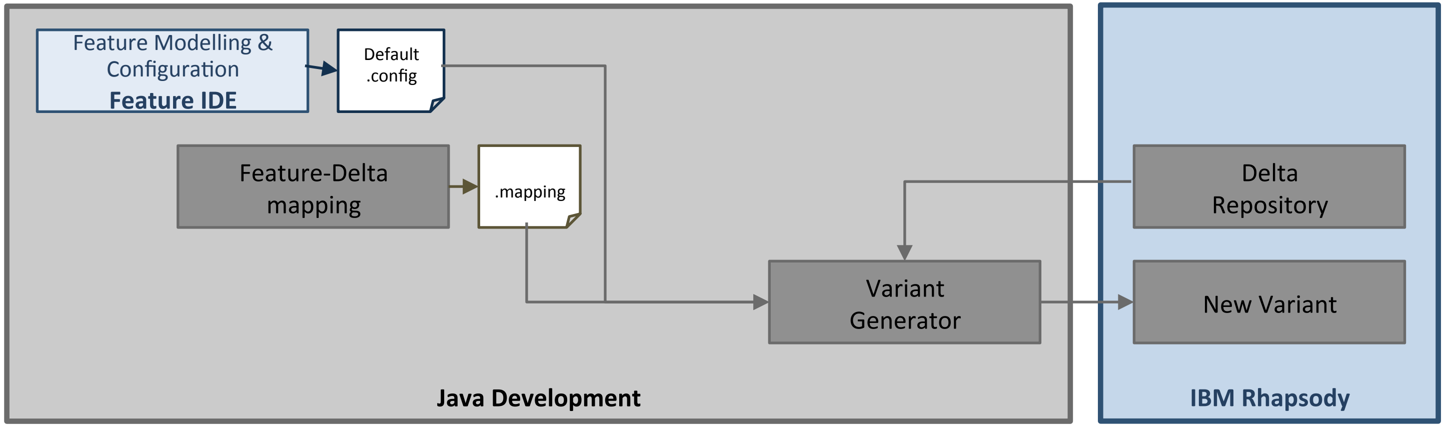
```
> 1 Core & Delta models visual generation
> 2 Feature Model Configuration visual edition
> 3 Mapping files generation (config file)
```

▨ Developed Application/Module

🟦 Standard Application/Module

Architecture

Process Flow for Application Engineer



```
> 1 Feature selection and save into default.config
> 2 Read required Features, Mapping and deltas (model)
> 3 Generate new variant (model) into Rhapsody
```

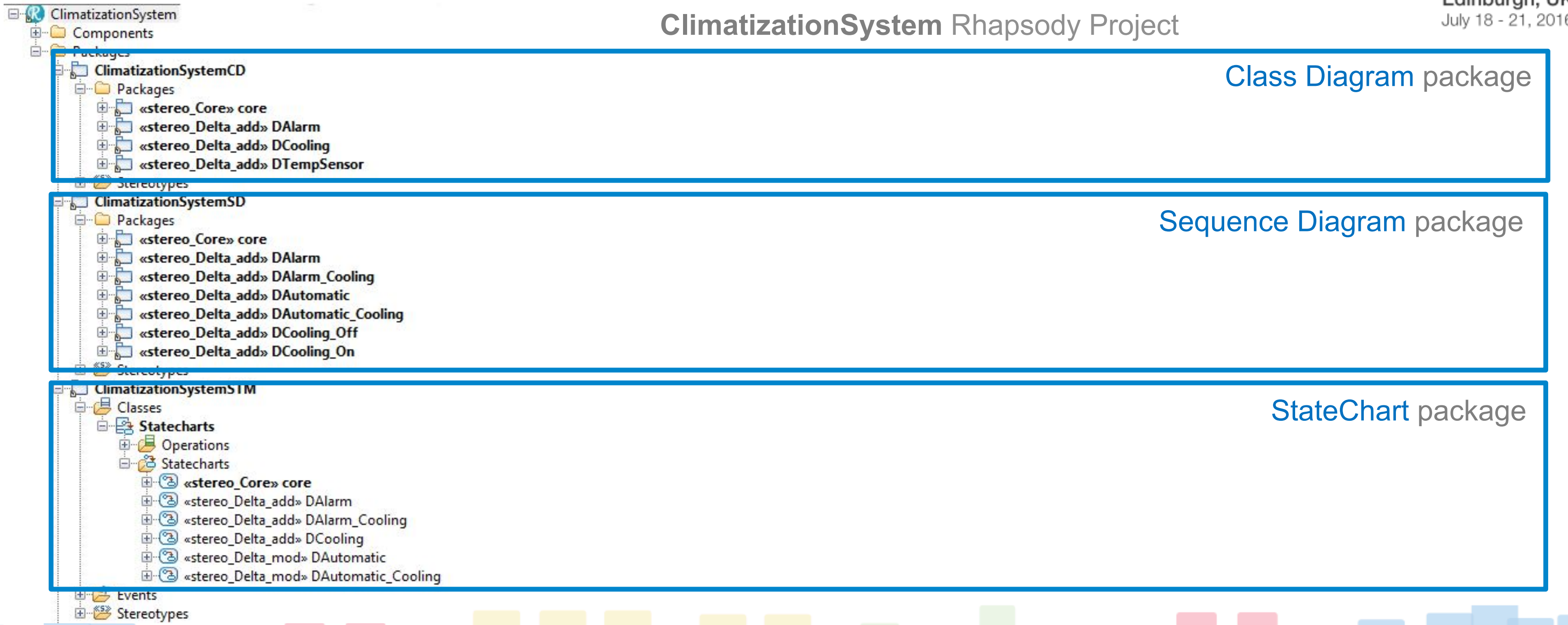
▨ Developed Application/Module

▨ Standard Application/Module

Architecture

Core & Delta Modelling

ClimatizationSystem Rhapsody Project



```
ClimatizationSystem
  Components
  Packages
    ClimatizationSystemCD
      Packages
        «stereo_Core» core
        «stereo_Delta_add» DAlarm
        «stereo_Delta_add» DCooling
        «stereo_Delta_add» DTempSensor
      Stereotypes
    ClimatizationSystemSD
      Packages
        «stereo_Core» core
        «stereo_Delta_add» DAlarm
        «stereo_Delta_add» DAlarm_Cooling
        «stereo_Delta_add» DAutomatic
        «stereo_Delta_add» DAutomatic_Cooling
        «stereo_Delta_add» DCooling_Off
        «stereo_Delta_add» DCooling_On
      Stereotypes
    ClimatizationSystemSTM
      Classes
        Statecharts
          Operations
          Statecharts
            «stereo_Core» core
            «stereo_Delta_add» DAlarm
            «stereo_Delta_add» DAlarm_Cooling
            «stereo_Delta_add» DCooling
            «stereo_Delta_mod» DAutomatic
            «stereo_Delta_mod» DAutomatic_Cooling
      Events
      Stereotypes
```

Class Diagram package

Sequence Diagram package

StateChart package

Architecture

Rhapsody Stereotypes

- stereo_Core
- stereo_Delta_add
- stereo_Delta_mod
- stereo_Delta_rem
- stereo_Delta_idle

STATECHART DIAGRAM

Stereotypes applicable to:

- Statechart Diagram
- States
- Transitions
- Events
- Action
- Operation

CLASS DIAGRAM

Stereotypes applicable to:

- Object Model Diagram
- Class
- Association
- Realization
- Dependency
- Interface
- Actor
- Package

SEQUENCE DIAGRAM

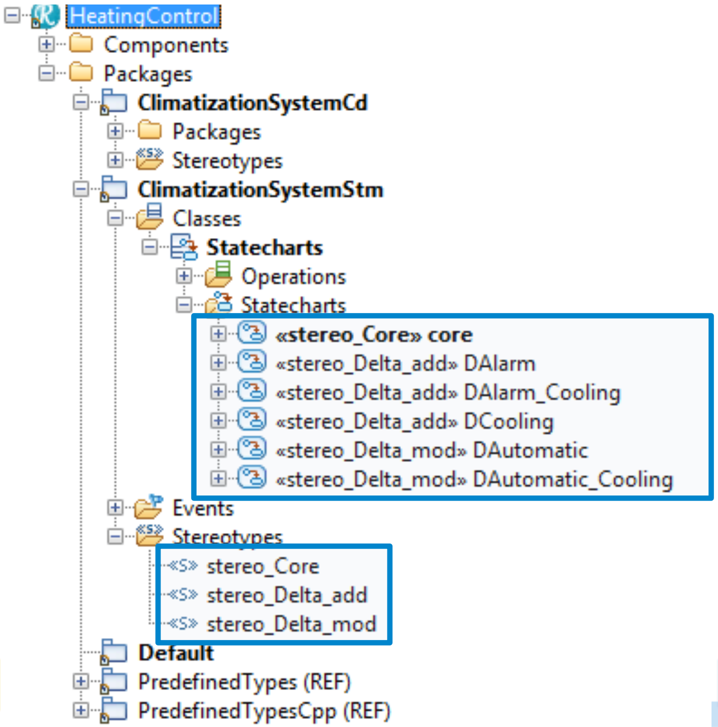
Stereotypes applicable to:

- Sequence Diagram
- Class
- Classifier Role
- Message
- Event
- Timeout
- Interaction Operator
- Interaction Occurrence
- System Border

Architecture

Rhapsody Stereotypes (Example)

Stereotypes are used to tag each model so the developed application will be able to process the model according to the function it has.

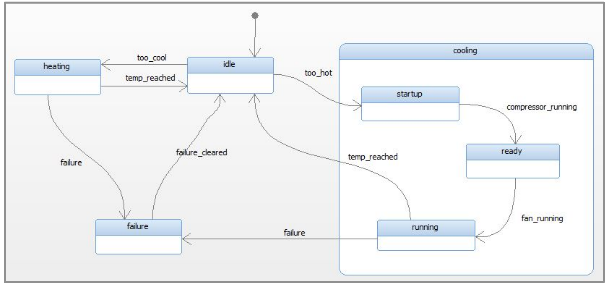


- HeatingControl
  - Components
  - Packages
    - ClimatizationSystemCd
      - Packages
      - Stereotypes
    - ClimatizationSystemStm
      - Classes
        - Statecharts
          - Operations
          - Statecharts
            - «stereo_Core» core
            - «stereo_Delta_add» DAlarm
            - «stereo_Delta_add» DAlarm_Cooling
            - «stereo_Delta_add» DCooling
            - «stereo_Delta_mod» DAutomatic
            - «stereo_Delta_mod» DAutomatic_Cooling
      - Events
      - Stereotypes
        - stereo_Core
        - stereo_Delta_add
        - stereo_Delta_mod
    - Default
    - PredefinedTypes (REF)
    - PredefinedTypesCpp (REF)

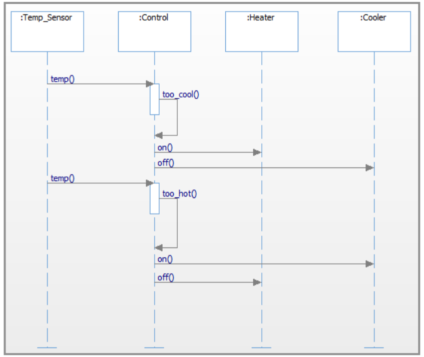
Architecture

Supported diagrams

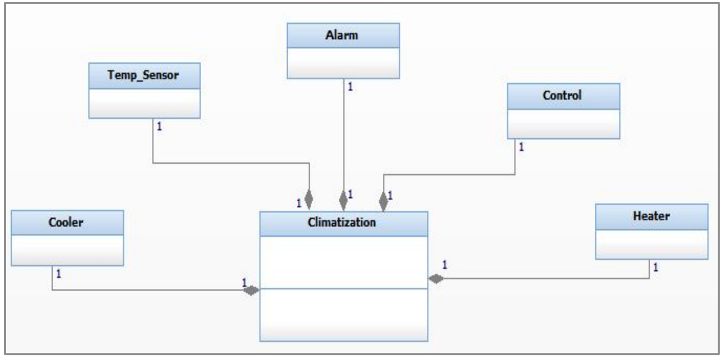
- Statechart Diagram
- Class Diagram
- Sequence Diagram



Statechart Diagram



Sequence Diagram



Class Diagram

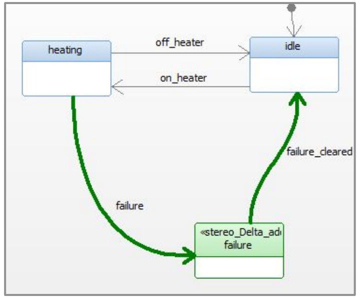
Architecture

Enhanced Delta visualization

DeltaAdd

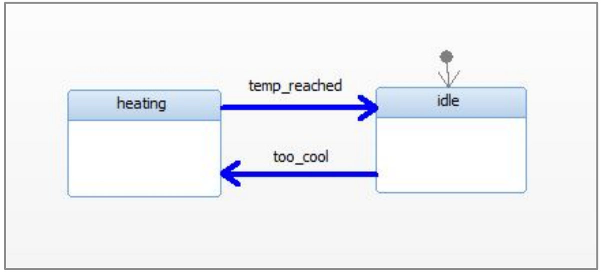
DeltaRem

DeltaMod

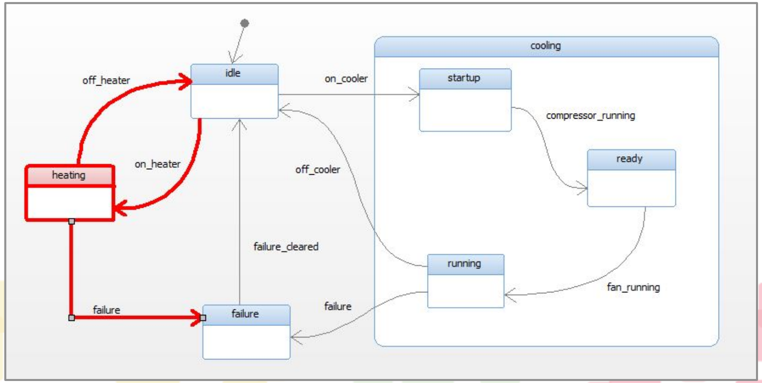


DeltaAdd

DeltaMod



DeltaRem



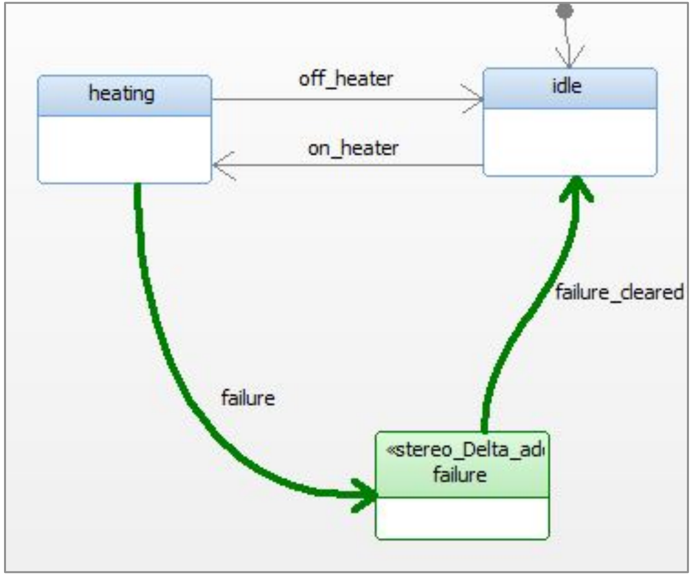
Architecture

Enhanced Delta visualization

DeltaAdd

DeltaRem

DeltaMod



Stereo_Delta_add stereotyped Delta sample

Blue elements belong to Core model.

Green elements are added and highlighted for this Delta model.

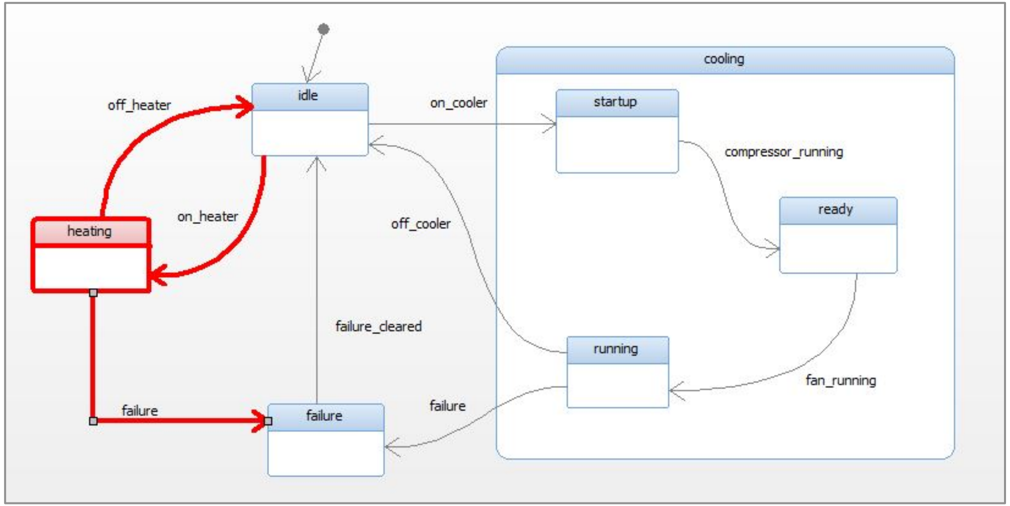
Architecture

Enhanced Delta visualization

DeltaAdd

DeltaRem

DeltaMod



Stereo_Delta_rem stereotyped Delta sample
Blue elements belong to Core model.
Red elements are removed
and highlighted for this Delta model.

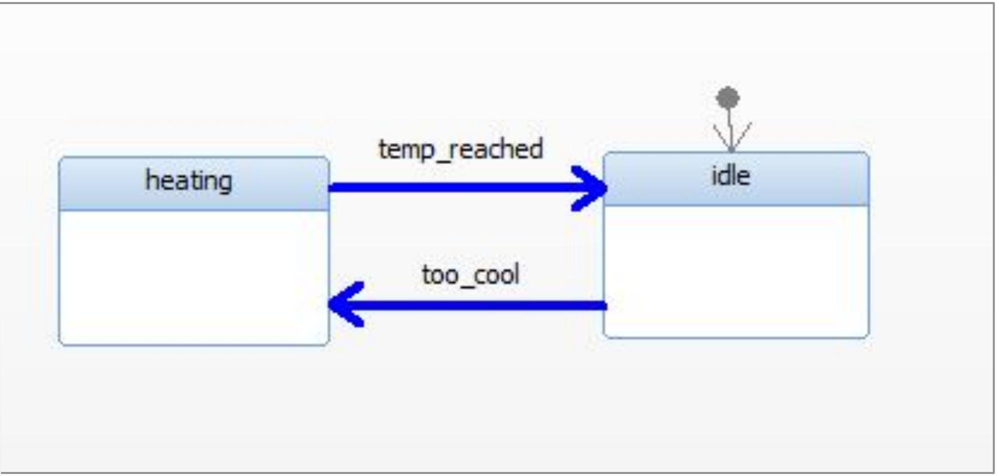
Architecture

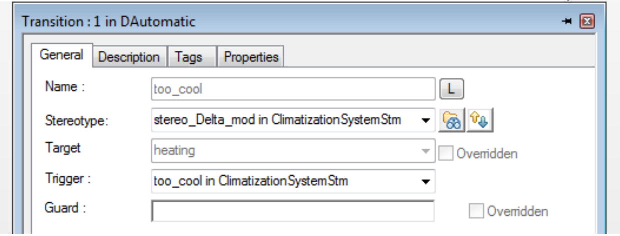
Enhanced Delta visualization

DeltaAdd

DeltaRem

DeltaMod




Transition : 1 in DAutomatic

General | Description | Tags | Properties

Name : too_cool

Stereotype: stereo_Delta_mod in ClimatizationSystemStm

Target heating □ Overridden

Trigger : too_cool in ClimatizationSystemStm

Guard : □ Overridden

Stereo_Delta_mod stereotyped Delta sample

Blue light elements belong to Core model.

Red elements are removed, blue bold elements are modified (attributes) and green elements are added.

Example

Case study overview

Climatization System is a toy example for showing the usage of Delta Rhapsody.

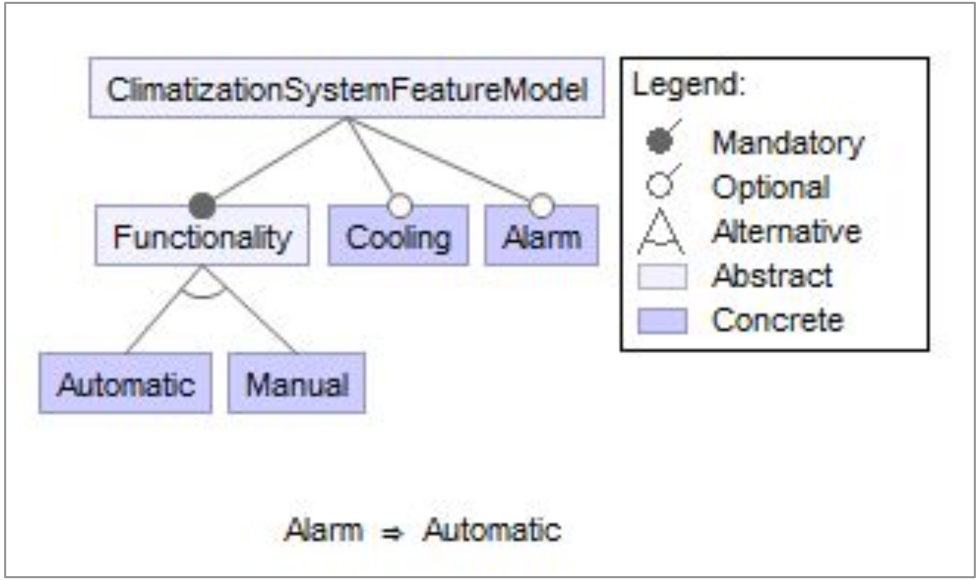
Having a Core as the most basic system, this can be transformed based on different Deltas, modifying the core with new features.

The manual heating system is **mandatory** and the new applicable features are:

- Add a **cooling** system
- **Automate** the system
- Add an **alarm** system

Example

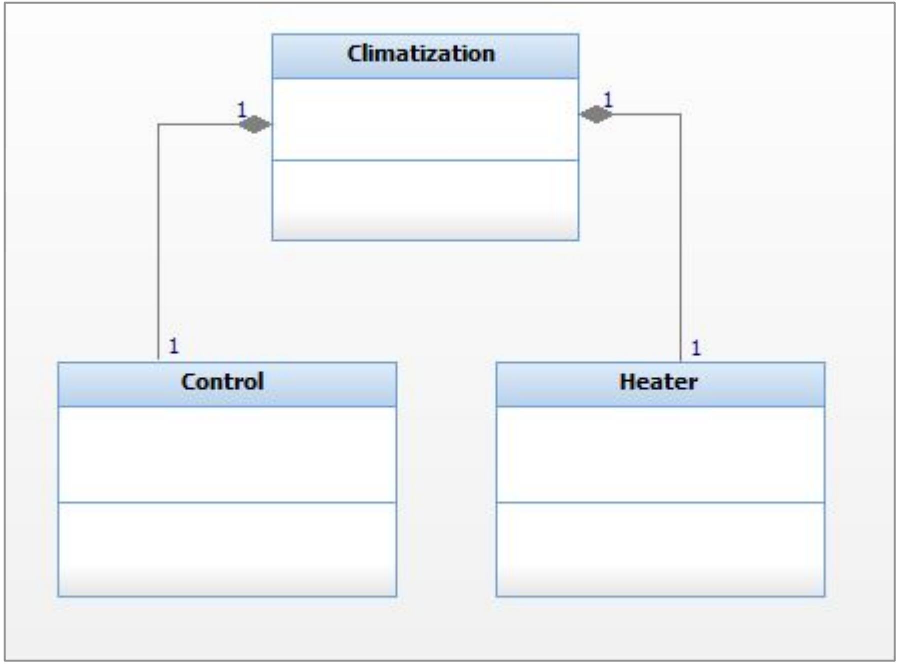
Feature Modelling & Configuration



Example

Core

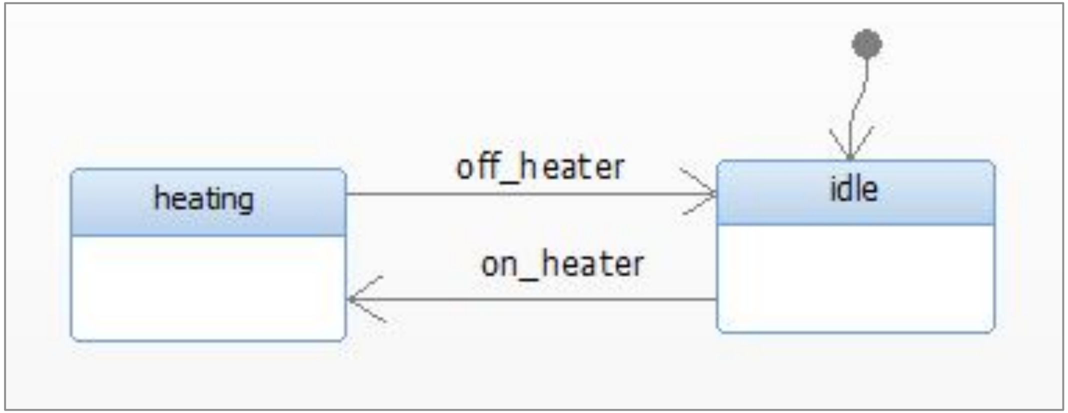
(class diagram)



Example

Core

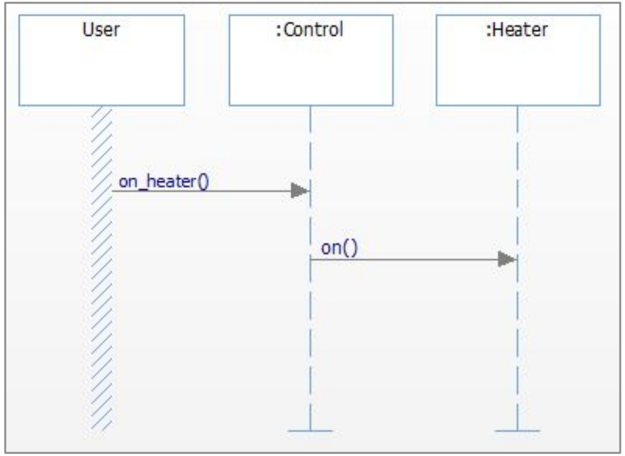
(state machine diagram)

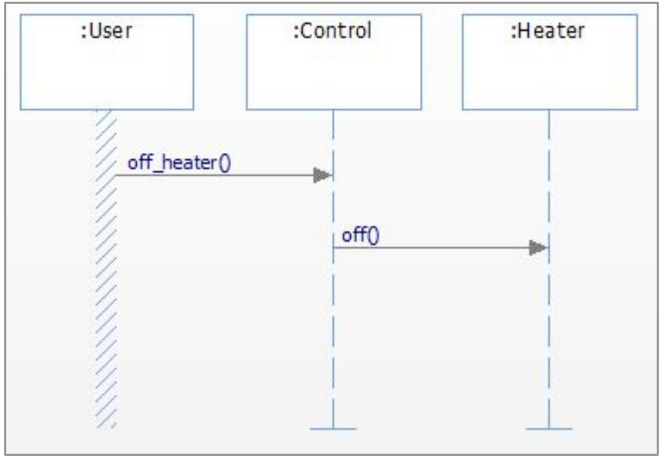


Example

Core

(sequence diagram)



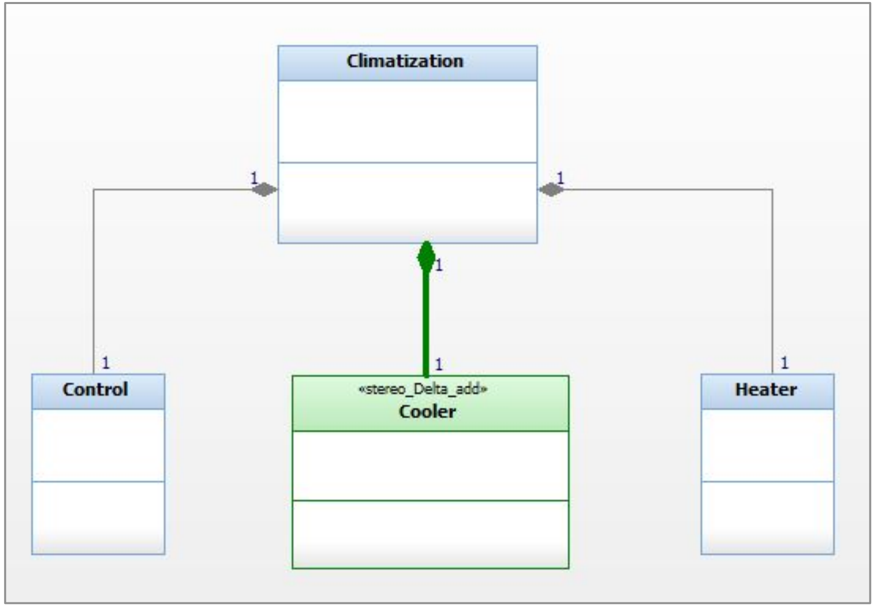


26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

Example

Delta Cooling

(class diagram)

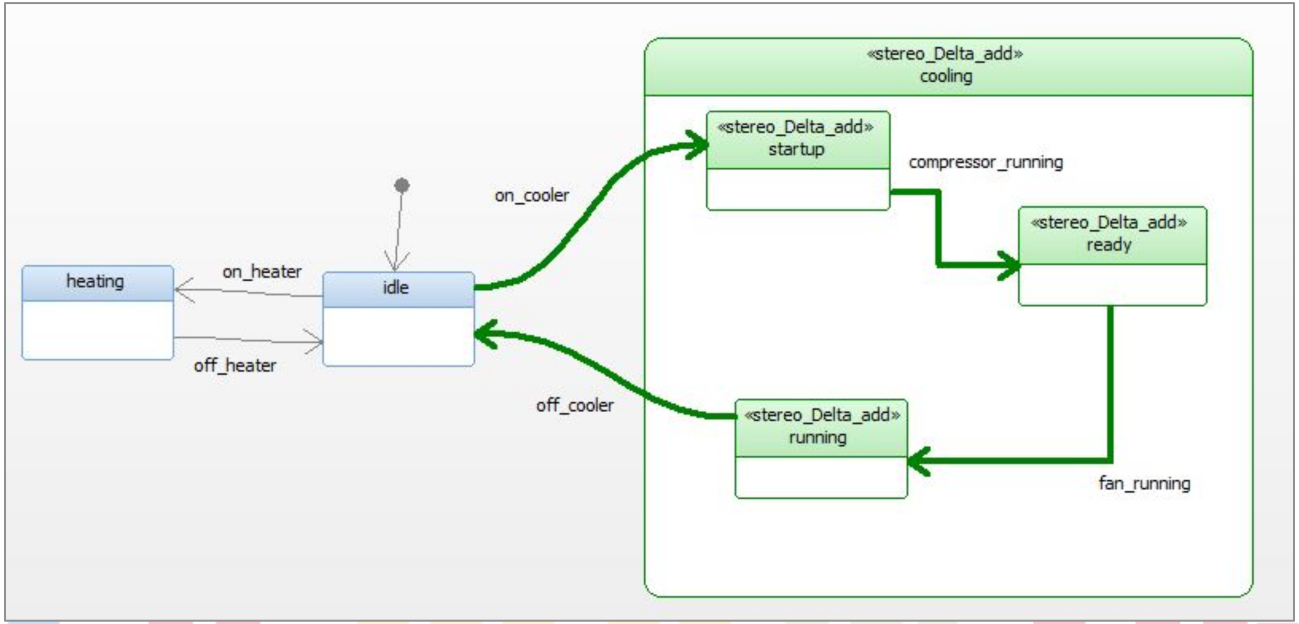


DeltaAdd

Example

Delta Cooling

(state machine diagram)

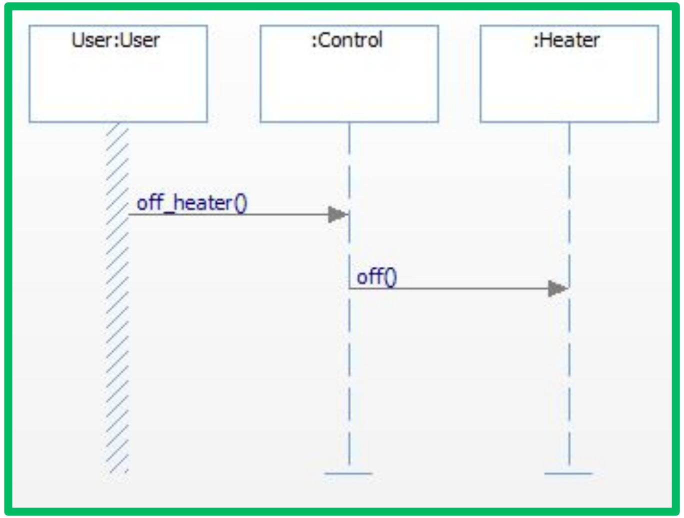


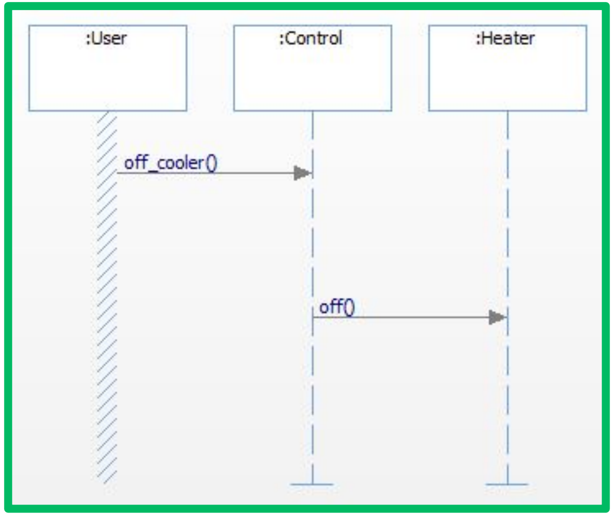
DeltaAdd

Example

Delta Cooling

(sequence diagram)



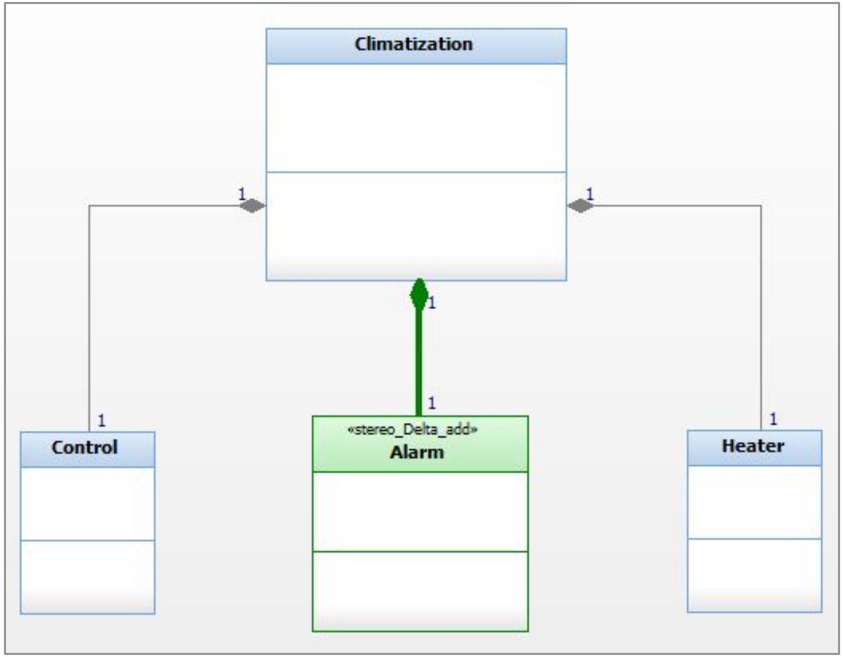


DeltaAdd

Example

Delta Alarm

(class diagram)

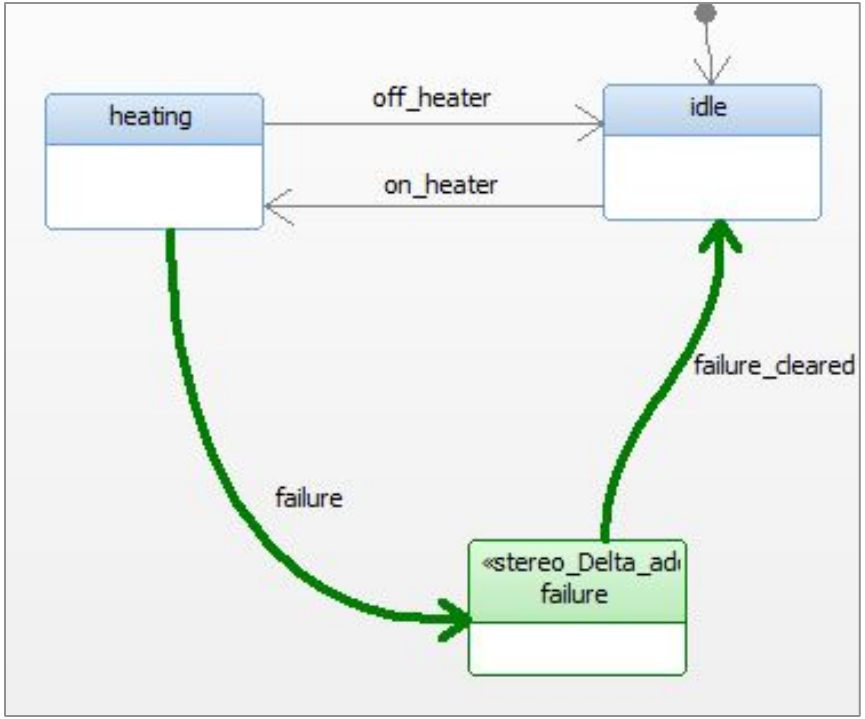


DeltaAdd

Example

Delta Alarm

(state machine diagram)



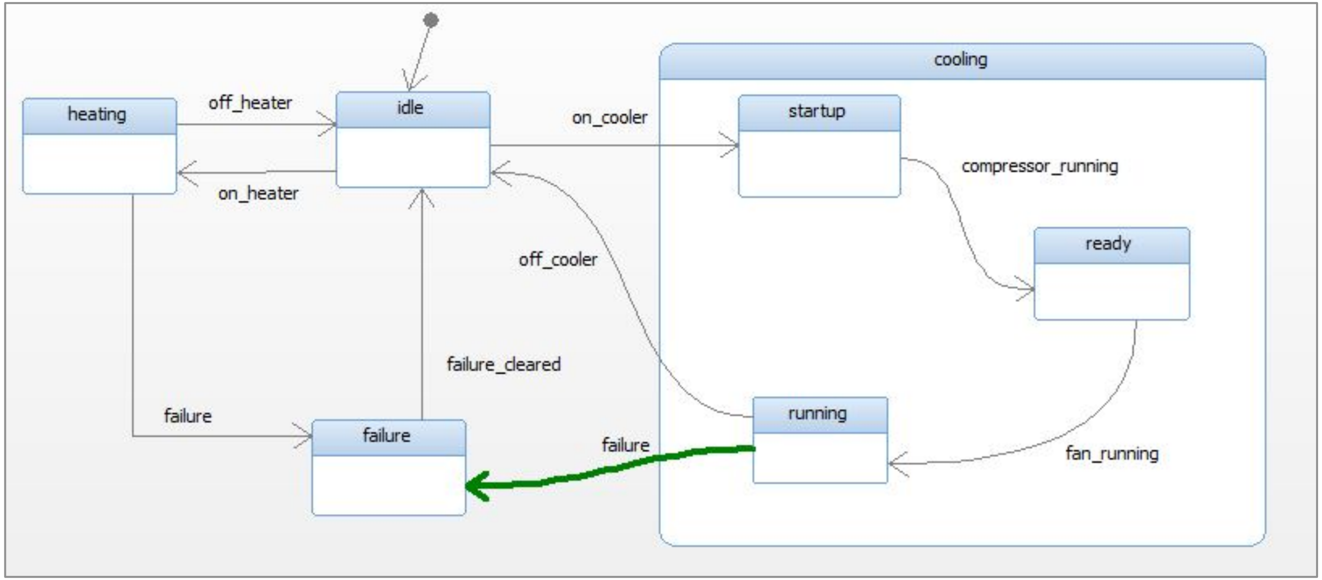
26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

DeltaAdd

Example

Delta Alarm_Cooling

(state machine diagram)

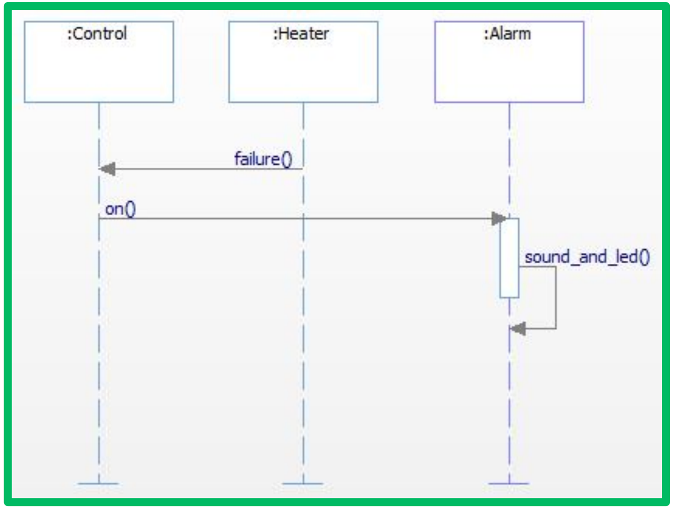


DeltaAdd

Example

Delta Alarm

(sequence diagram)



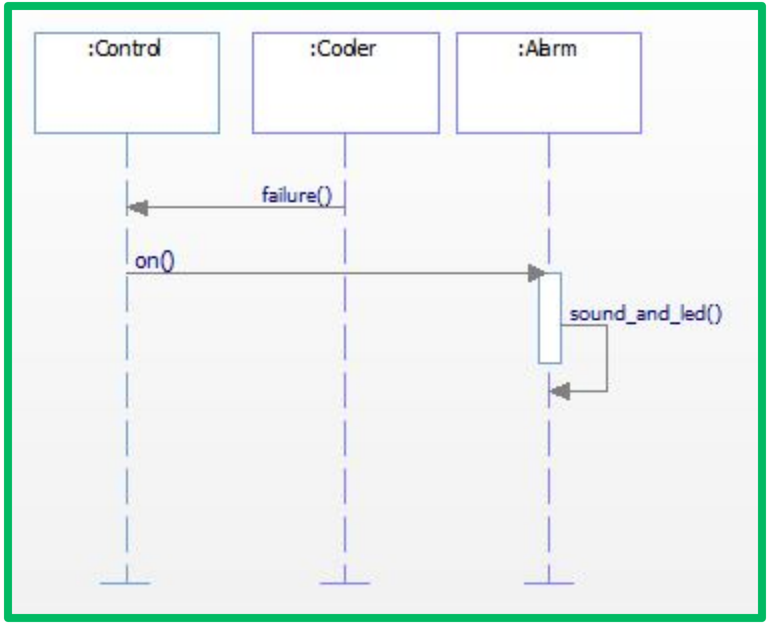
DeltaAdd

26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

Example

Delta Alarm_Cooling

(sequence diagram)



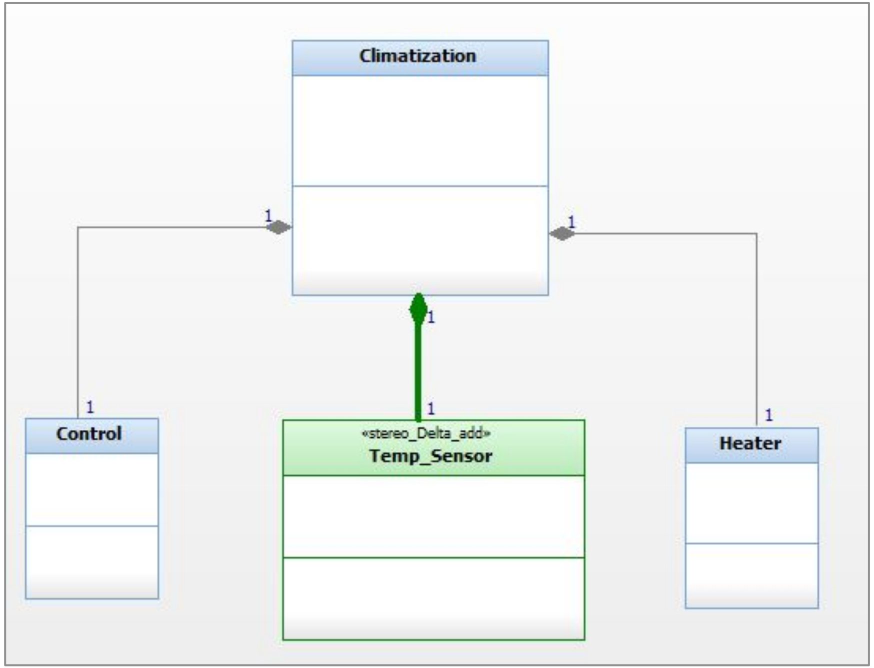
26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

DeltaAdd

Example

Delta Automatic

(class diagram)

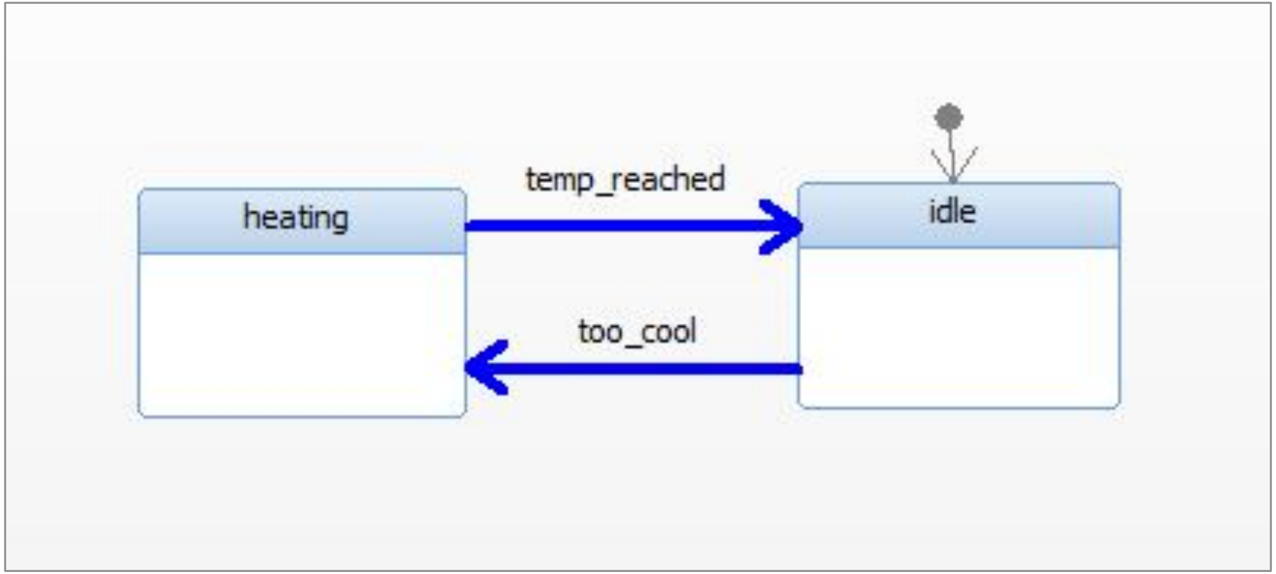


DeltaAdd

Example

Delta Automatic

(state machine diagram)

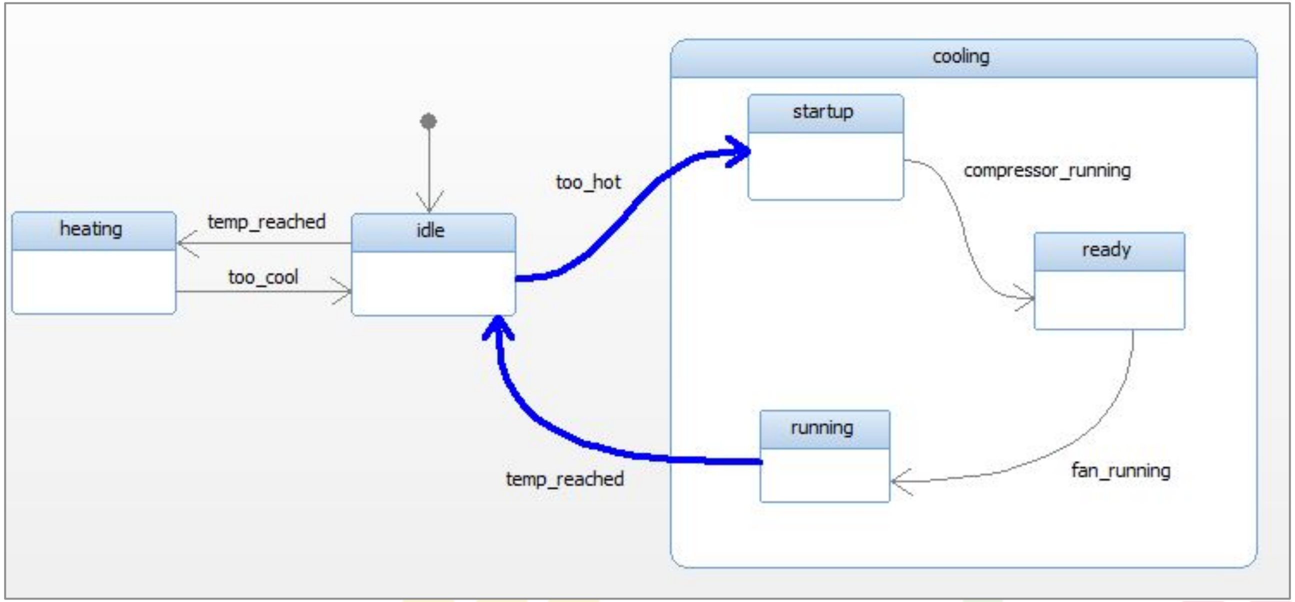


DeltaMod

Example

Delta Automatic_Cooling

(state machine diagram)

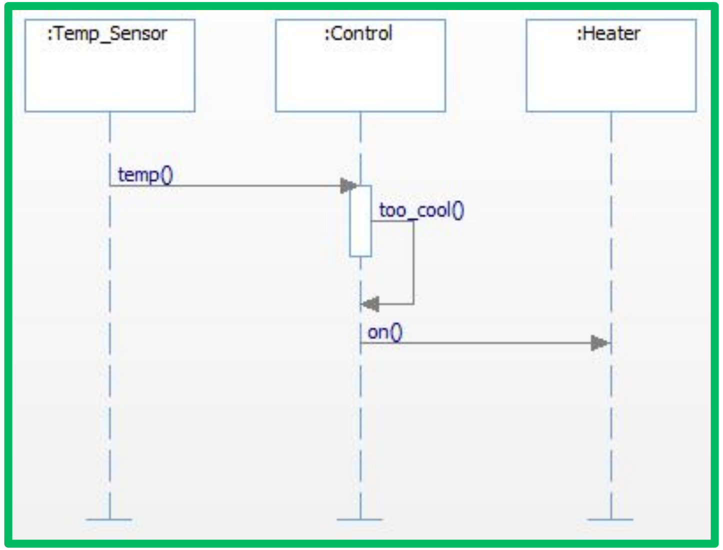


DeltaMod

Example

Delta Automatic

(sequence diagram)

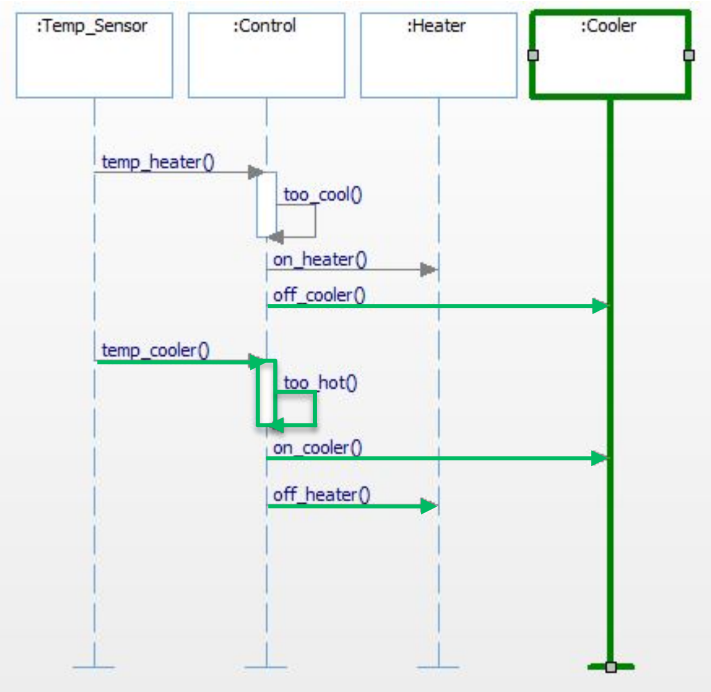


DeltaAdd

Example

Delta Automatic_Cooling

(sequence diagram)



DeltaAdd

Example

Mapping files

sd.mapping

```
productline ClimatizationSystem;
features Automatic, Cooling, Alarm;

delta DAlarm when Alarm;
delta DAutomatic when Automatic;
delta DCooling when Cooling;
delta DAlarm_Cooling after DAlarm && DCooling when Alarm && Cooling;
delta DAutomatic_Cooling after DAutomatic && DCooling when Automatic && Cooling;
```

```
productline ClimatizationSystem;
features Automatic, Cooling, Alarm;

delta DAlarm when Alarm;
delta DAutomatic when Automatic;
delta DCooling when Cooling;
delta DAlarm_Cooling after DAlarm && DCooling when Alarm && Cooling;
delta DAutomatic_Cooling after DAutomatic && DCooling when Automatic && Cooling;
```

stm.mapping

cd.mapping

```
productline ClimatizationSystem;
features Automatic, Cooling, Alarm;

delta DAlarm when Alarm;
delta DAutomatic when Automatic;
delta DCooling when Cooling;
```

Based on ABS language

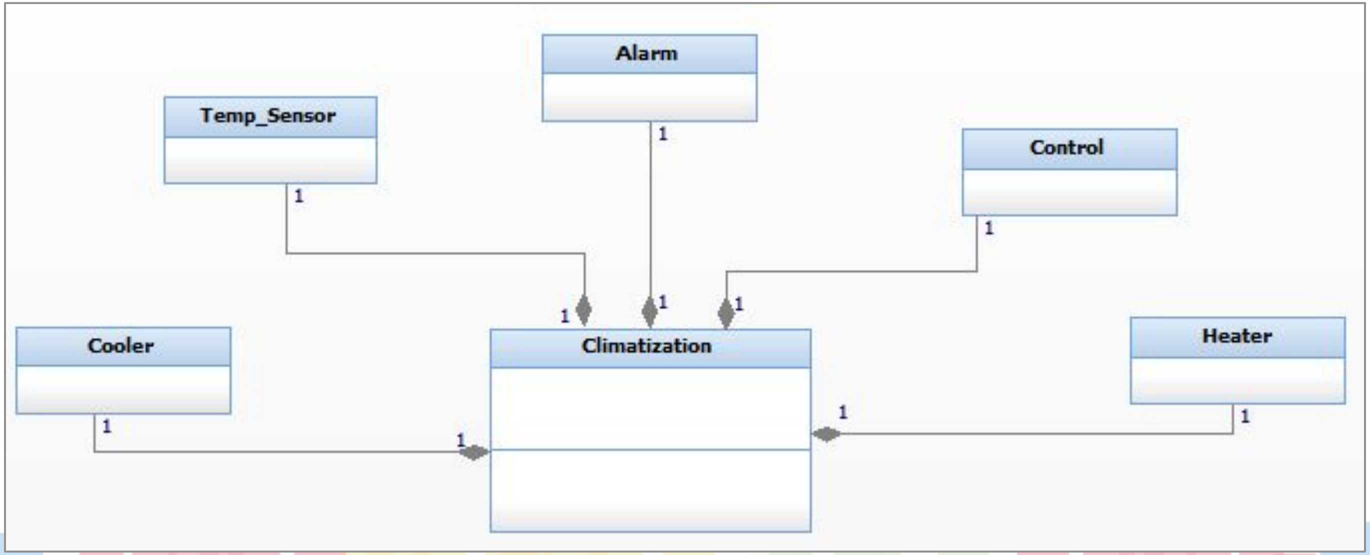
Example

New Variant

Features: Cooling + Alarm + Automatic

Deltas: DAlarm + DAutomatic + DCooling

Class diagram



Example

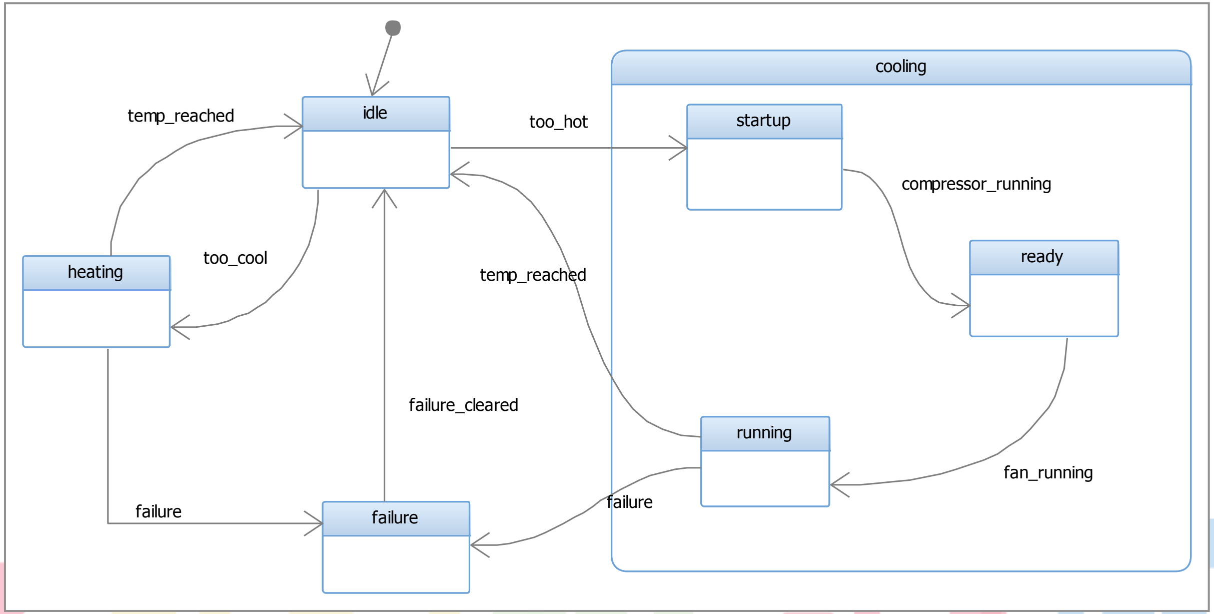
26th annual INCOSE international symposium
Edinburgh, UK
July 18 - 21, 2016

New Variant

Features: Cooling + Alarm + Automatic

Deltas: DAlarm + DAutomatic + DCooling + DAlarm_Cooling + DAutomatic_Cooling

statechart diagram

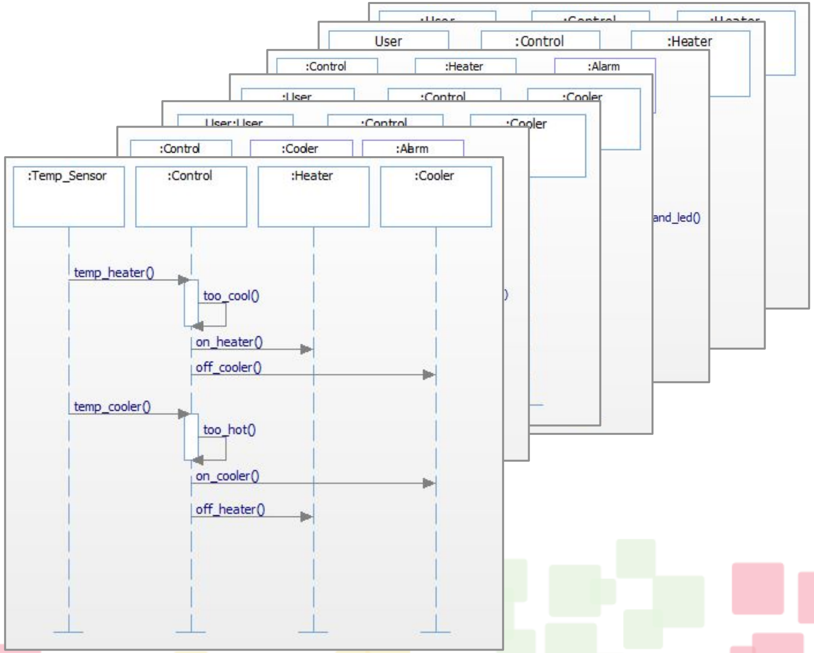


Example

New Variant

Features: Cooling + Alarm + Automatic

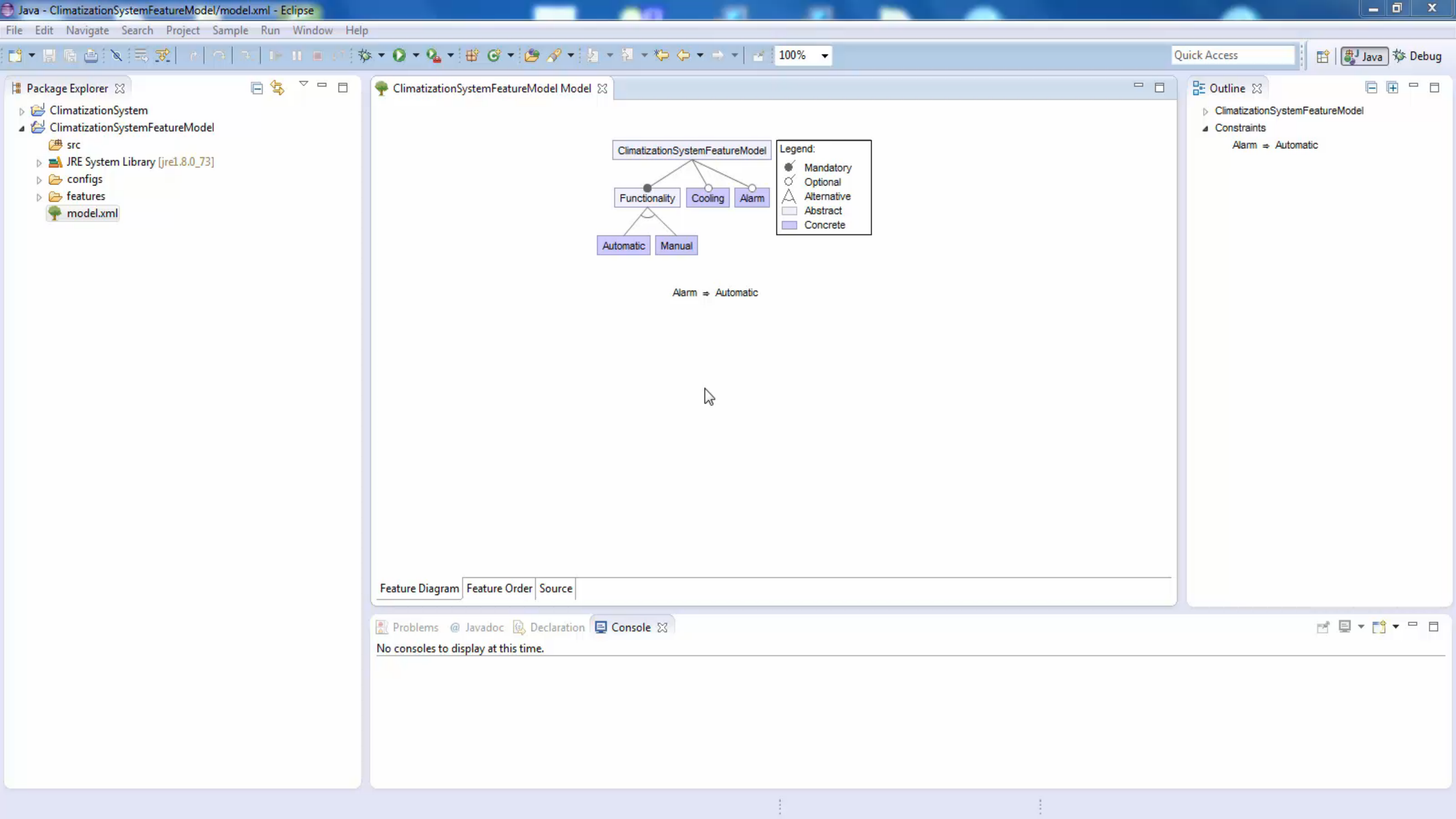
Deltas: DAlarm + DAutomatic + DCooling + DAlarm_Cooling + DAutomatic_Cooling



Sequence Diagram

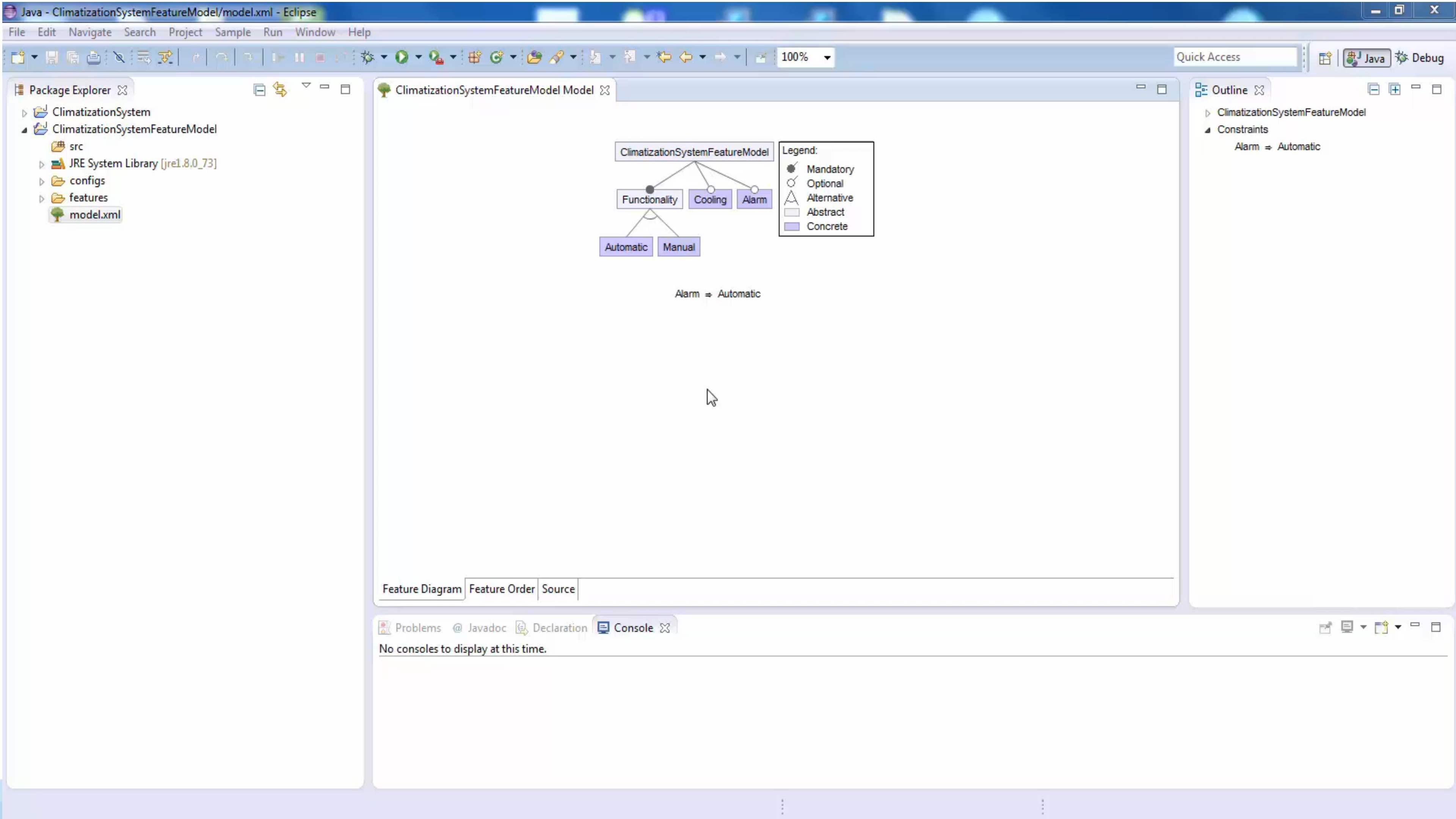
RECORDED DEMO

New Variant generation demonstration (Application Engineer role)



Alarm ⇒ Automatic

```
No consoles to display at this time.
```



Alarm ⇒ Automatic

No consoles to display at this time.

Conclusions

Support for variability: Delta Rhapsody

- ▲ Rhapsody based Delta Modelling solution provided.
- ▲ Application Engineering workload simplified.

The prototype has been applied in a real case study of the transportation sector with good results:

- ▲ The current models (core and deltas) are easier to understand than the complete model that was previously used.
- ▲ The required time to develop variant models is reduced.
- ▲ The inclusion of future new products is easier.
- ▼ The extra cost required must be also considered: especially the time required to model the core and deltas at domain engineering.

Future Work

- ❑ **Validate the tool in other domains.**
- ❑ **Empirical validation to measure the benefits of using the tool taking into account the learning curve of the domain engineer.**
- ❑ **Define appropriate workflows for:**
  - Maintenance when core is changed:
    - Integration of deltas in the core.
    - Maintenance of released variants.
  - Modification of Deltas during Application Engineering:
    - Process for deciding if it should be reused in the Domain Engineering scope.
- ❑ **Delta Rhapsody**
  - Generate **Rhapsody checkers** to ensure design coherence through Domain Engineering phase.
  - **Mapping** file generation GUI development.
  - Automatic **Test** Generation.
  - Use **Design Manager** as repository for core and delta models.
  - Allow more than one state machine diagram per project.

References

- Batory, D., Sarvela, J., Rauschmayer, A. 2004. “Scaling Step-Wise Refinement.” *IEEE* Trans. Software Eng. 30(6).
- Clarke, X. D., Muschevici, R., Proença, J., Schaefer, I., Schlatte, R. 2010a. “Variability Modelling in the ABS Language, Chapterin Formal Methods for Components and Objects.” *Volume 6957 of the series Lecture Notes in Computer Science*: 204-224
- Clarke, D., Helvensteijn, M., Schaefer, I. 2010b. “Abstract Delta Modelling.” *ACM Sigplan Notices*: 13-22.
- Clements, P. and Northrop, L. 2010. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley.
- Estefan, J. A. 2008. “Survey of Model-Based Systems Engineering (MBSE) Methodologies.” *Rev. B, INCOSE Technical Publication, Document No.: INCOSE-TD-2007-003-01* (paper presented at the International Council on Systems Engineering, San Diego, CA, June 10, 2008).
- Haber, A., Hölldobler, K., Kolassa, C., Look, M., Müller, K., Rumpe, B., Schaefer,I. 2013. ”Engineering Delta Modelling Languages.”(paper presented at the Proceedings of the 17th International Software Product Line Conference (SPLC), Tokyo, September, 2013): 22–31.
- Haber, A., Hölldobler, K., Kolassa, C., Look, M., Rumpe, B., Müller, K., Schaefer, I. 2013. “Engineering Delta Modelling Languages.*” Proceedings of the 17th International Software Product Line Conference*: 22-31.
- Haber, A., Kolassa, C., Manhart, P., Nazari, P. M. S., Rumpe, B., Schaefer,I. 2013.“First-class variability modelling in matlab/ simulink.” (paper presented at the International Workshop on Variability Modelling of Software-intensive Systems (VaMoS).
- Haber, A., Kutz, T., Rendel, H., Rumpe, B., Schaefer, I. 2011. “Delta-Oriented Architectural Variability Using Monticore.” *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*: 6.
- Haber, A., Rendel, H., Rumpe, B., Schaefer, I., van der Linden, F. 2011. “Hierarchical Variability Modelling for Software Architectures” (paper presented at the Proceedings of International Software Product Lines Conference, August 2011).
- Halmans, G., Pohl, K. 2004. “Communicating the variability of a software-product family to customers. Inform. “*Forsch. Entwickl. 18*: 113–131.
- Heidenreich, F., Sánchez, P., Santos, J., Zschaler, S., Alférez, M., Araújo, J., Fuentes, L., Kulesza, U., Moreira, A., Rashid, A. 2010. “Relating feature models to other models of a software product line: a comparative study of feature mapper and VML.” *Transactions on aspect-oriented software development VII*: 69-114.

References

- Kang, K.C.,Cohen, S.G., Hess, J.A.,Novak, W.E., Peterson, A.S., "Feature-oriented domain analysis (FODA) feasibility study", Technical Report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University, November 1990.
- Katz, S., and Mezini, M. 2011. “A Common Case Study for Aspect-Oriented Modelling*”.* Berlin, Heidelberg: Springer-Verlag.
- Lochau, M., Lity, S., Lachmann, R., Schaefer, I., Goltz, U. 2014. “Delta-oriented model-based integration testing of large-scale systems.”*J. Syst. Softw.* 91 63-84
- Mhenni, F., Nguyen, N., Choley, J.Y. 2013. “Towards the Integration of Safety Analysis in a Model-Based System Engineering Approach with SysML, Design and Modelling of Mechanical Systems.” *Lecture Notes in Mechanical Engineering* (2013): 61-68 (paper presented at Fifth International Conference Design and Modelling of Mechanical Systems, Djerba, Tunisia, March 2013).
- Murray, J. 2012. “Model Based Systems Engineering (MBSE)”, *Media Study*.
- Schaefer, I. 2010. “Variability modelling for model-driven development of software product lines.” *VaMoS*: 85-92
- Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N. 2010. ”Delta-oriented programming of software product lines.” (paper presented at Proc. of 15th Software Product Line Conference, Sep 2010)).
- Schaefer, I., Bettini, L., Damiani, F. 2011. “Compositional type-checking for delta oriented programming.” (paper presented at the 10th International Conference on Aspect-Oriented Software Development, AOSD 2011): 43–56.
- Schaefer, I., Bettini, L., Damiani, F., Tanzarella, N. 2010.”Delta-Oriented Programming of Software Product Lines.” *Proceedings of the 14th international conference on Software product lines: going beyond*: 77-91.
- Schaefer, I., Damiani, F. 2010. “Pure Delta-oriented Programming”. *FOSD 2010.*
- Schaefer, I., 2012, “Efficient Incremental Testing of Variant-Rich Software Systems”, *The 23rd CREST Open Workshop Change Impact Analysis and Testing of Software Product Lines.*
- Scouler, J.L., Bakal, M.R. 2013. “Product Design for Variants: Considerations, incentives, and best practices.” http://www.ibm.com/developerworks/rational/library/product-design-variants/.
- Seidl, C., Schaefer, I., Aßmann, U. 2014. “DeltaEcore - A Model-Based Delta Language Generation Framework.” *Modellierung*: 81-96
- Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T. 2014. “FeatureIDE: An Extensible Framework for Feature-Oriented Software Development.” Science of Computer Programming, 79(0): 70-85.
- Trigaux, J. C. 2010. “Modelling variability requirements in software product lines: a comparative survey,” *Tech. rep., FUNDP Namur*.
- Van Gurp, J., Bosch, J., Svahnberg, M. 2001. “On the notion of variability in software product lines.” *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*: 45

Q&A

Thank you