

Software Product Lines with Model Driven Engineering

Prof. Jean-Marc Jézéquel

Triskell project-team

jezequel@irisa.fr

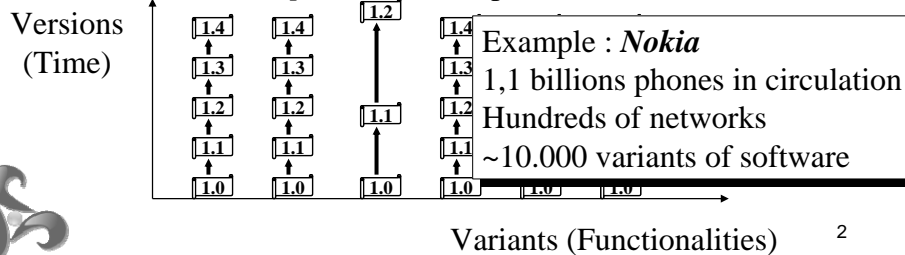
<http://www.irisa.fr/prive/jezequel>



Software Intensive Systems



- Importance of non-functional properties
 - distributed reactive systems, parallel & asynchronous
 - quality of service : security, reliability, latency, performance...
- Variations in functional aspects
 - notion of *product lines* (space, time)

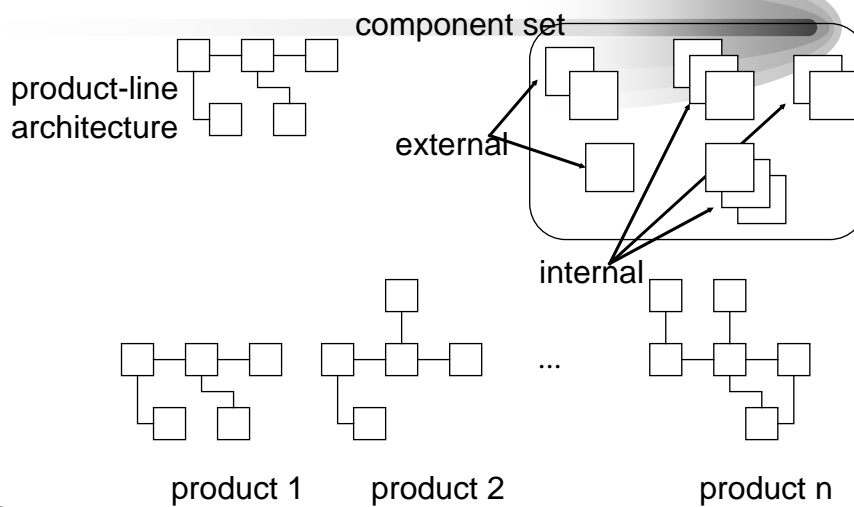


Software Product Lines

- A software product line consists of:
 - product line architecture
 - set of reusable components
 - set of products, where each product has
 - product architecture derived from PLA
 - instantiated and configured components
 - product-specific code

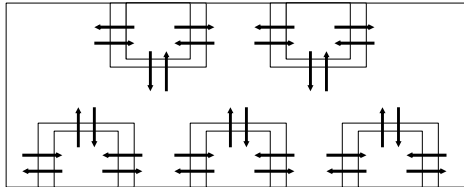


Software Product Lines



From OO to Components to Models

- frameworks



- Changeable software, from distributed / unconnected sources, even after delivery, or by the end user

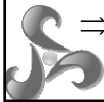
- Guarantees ?

Functional , synchronization, performance, QoS

- Handling variations in cross-cutting concerns?

⇒ Models to represent such assemblies of components

⇒ Model Driven Engineering to exploit them



5

Many Issues Around SPLs

- Assets [Jacobsen]:

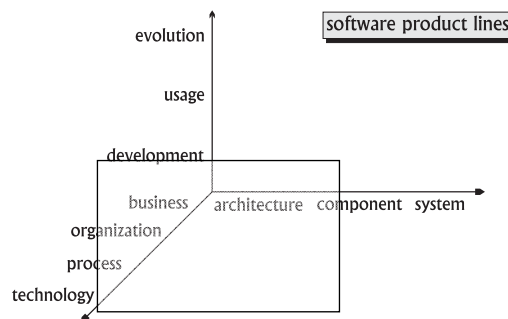
- architecture
- components
- systems

- Views [SEI]:


- business
- organization
- process
- technology

- Lifecycle [Bosch]:


- development
- usage
- evolution



6



*Technical Issues in
SPLs:
Managing variability*

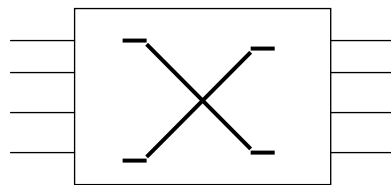


7

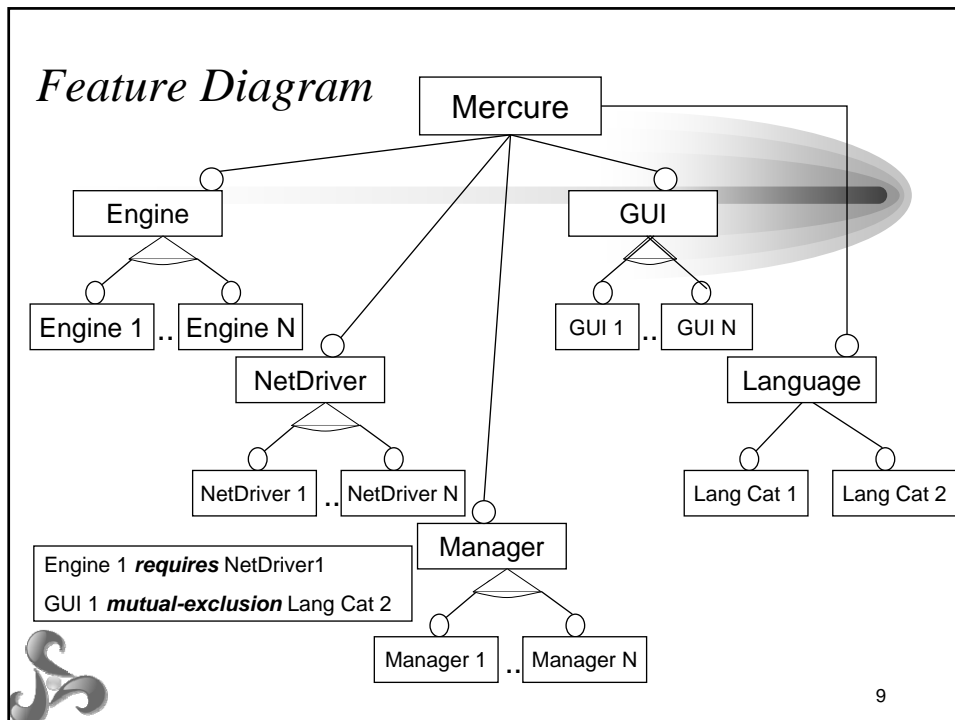
Example: The Mercure PL



- A family of telecom servers
- Delivering, forwarding, and relaying messages from and to a set of network interfaces.



8



Variants in Software Systems

- Hardware Variants
 - Heterogeneous processors
 - Peculiarities in Target Operating System
 - Compiler Differences
 - Range of Functionalities (Engines)
 - User Interface
 - Internationalization.
- $V_i = 16$
 - $V_p = 4$
 - $V_n = 8$
 - $V_g = 5$
 - $V_l = 24$

- Number of Variants = $V_p * V_n * V_g * 2^{V_i + V_l - 2}$
 – That's 43,980,465,111,040 possible variants

Traditional Approaches

- Patch the executable
- Device Drivers
 - source level, link time, boot time, on demand at runtime
- Static Configuration Table
- Conditional Compilation / Runtime Tests

```
if (language == french) {  
#ifdef MSW  
    io_puts(0,"Bonjour",7);  
#elif TEXT  
    printf("Bonjour\n");  
#endif  
} else {  
#ifdef MSW  
    io_puts(0,"Hello",5);  
#elif TEXT  
    printf("Hello\n");  
#endif
```

- Static and Dynamic configuration information intermingled
- Hard to change your mind on what should be static or dynamic...

11

Modeling for managing complexity

12

Modeling in Science & Engineering

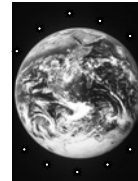
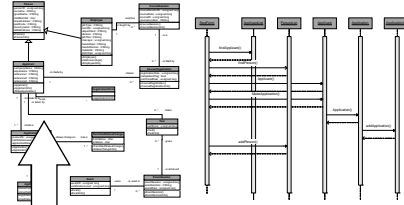
- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*

Specificity of Engineering:
Model something not yet existing (in order to build it)

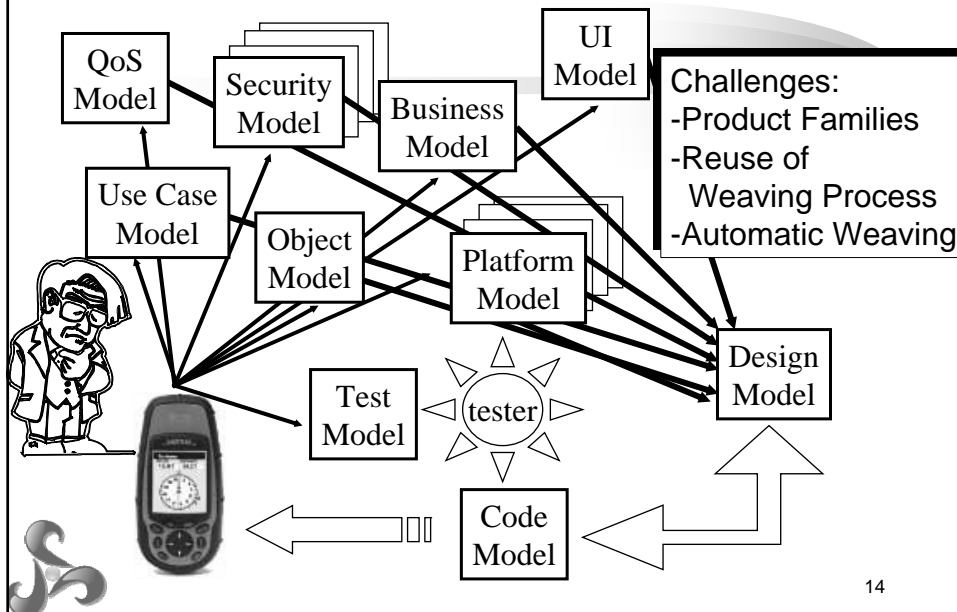
M_1
(modeling space)

Is represented by

M_0
(the world)

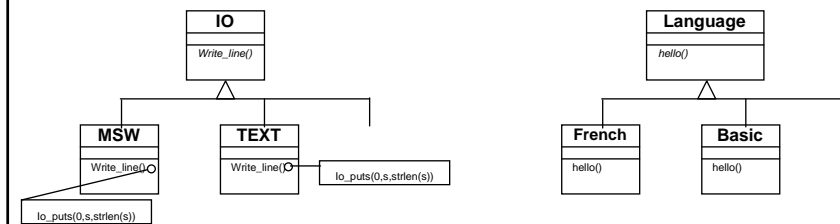


Modeling and Weaving



Basic Idea: model variability

- Abstract the Intent
 - io.write_line(language.hello)
- Rely on Dynamic Binding for the Details
 - Don't care now for static/dynamic distinction

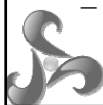


- Uncouple the variations from the selection process
 - Automatically derive a product using executable meta-modeling (aka *model transformations*)

15

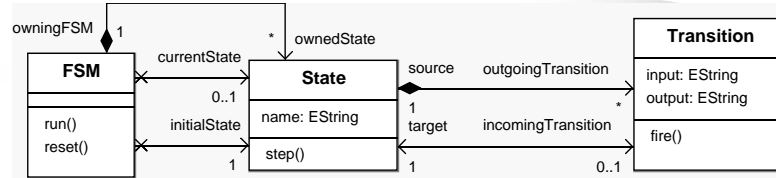
Kernel Meta-modeling Language: Kermeta

- Kermeta is a Model-Oriented Language
 - based on an AO/OO executable meta-modeling paradigm
 - Static typing, generics, functions objects, reflection...
 - First language where models are first class entities
 - Allows interesting questions to be asked: e.g.; what is the type of a model?
- Executable meta-modeling allows:
 - specification of abstract syntax, static semantic (OCL) and dynamic semantics, connection to the concrete syntax.
 - model and meta-model simulation and prototyping
 - model transformation, design level aspect weaving



16

Kermeta: Breathing life into Meta-Models



- // MyKermetaProgram.kmt
- // An E-MOF metamodel is an OO program that does nothing
- require "StateMachine.ecore" // to import it in Kermeta
- // Kermeta lets you weave in **aspects**
- // Contracts (OCL WFR)
- require "StaticSemantics.ocl"
- // Method bodies (Dynamic semantics)
- require "DynamicSemantics.kmt"
- // Transformations

```

class Minimizer {
    operation minimize (source: FSM):FSM {...}
}
    
```

```

Context FSM
inv: ownedState->forall(s1,s2|
s1.name=s2.name implies s1=s2)
    
```

```

aspect class FSM {
    operation reset() : Void {
        currentState := initialState
    }
}
    
```



17

Application Examples



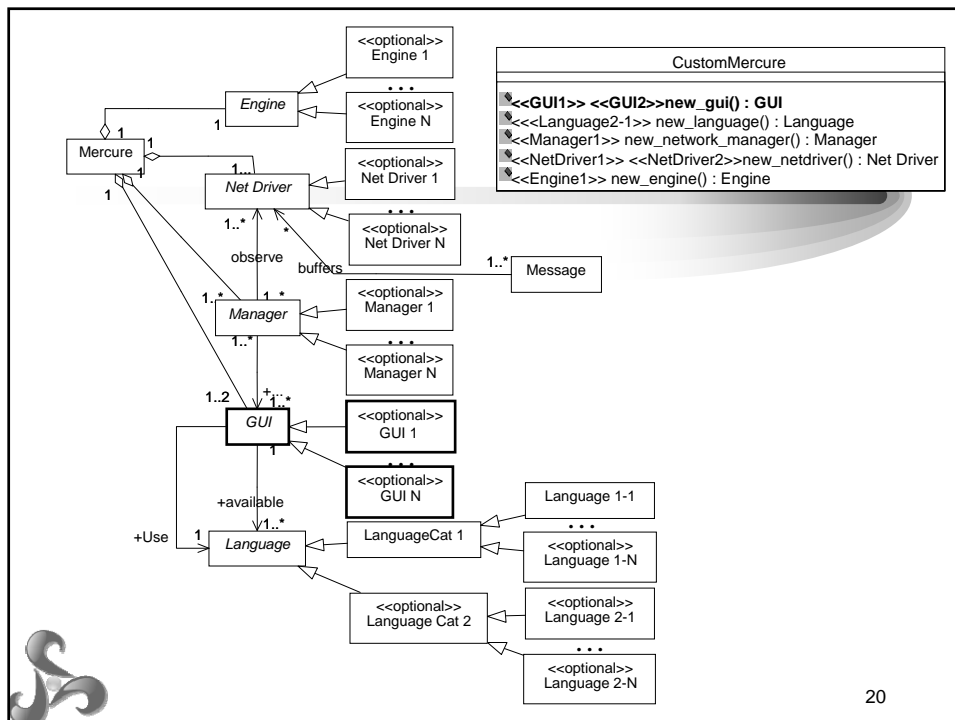
18

Expl. 1: Derivation of products from product-lines: structural issues

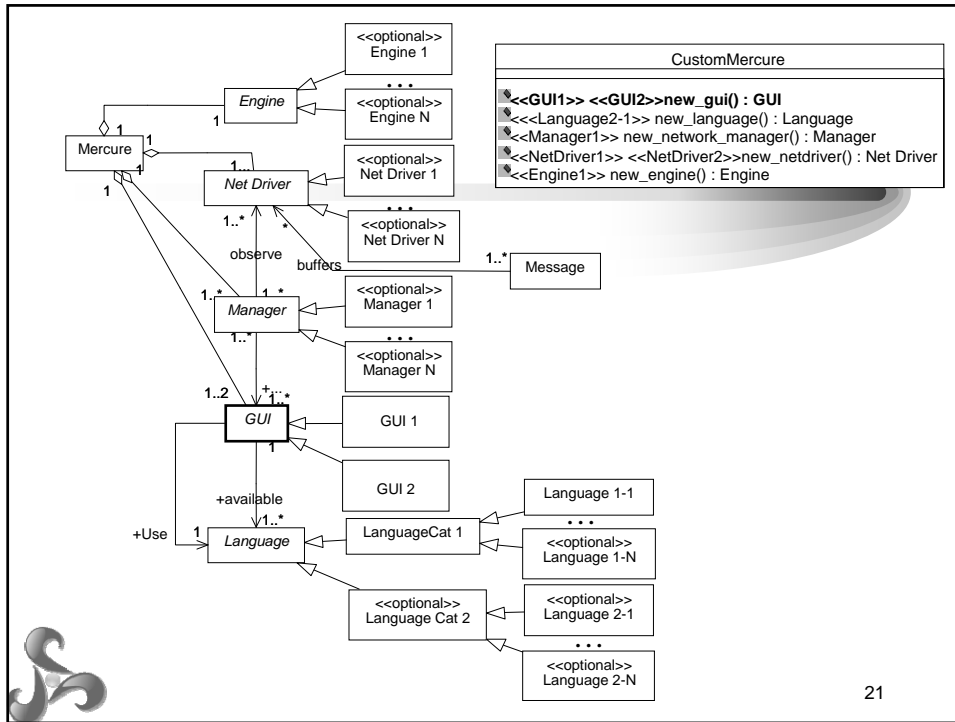
- Model a *full product-line* instead of a *single product*
 - Using OO inheritance & templates for modeling variability
 - Stereotypes for modeling optionality, mutual exclusion etc.
- The Variants selection:
 - Using operation factory stereotypes
- The Model specialization:
 - Removes all optional classes which have not been selected
- The Model optimization:
 - Deletes unused factories, Optimize inheritance



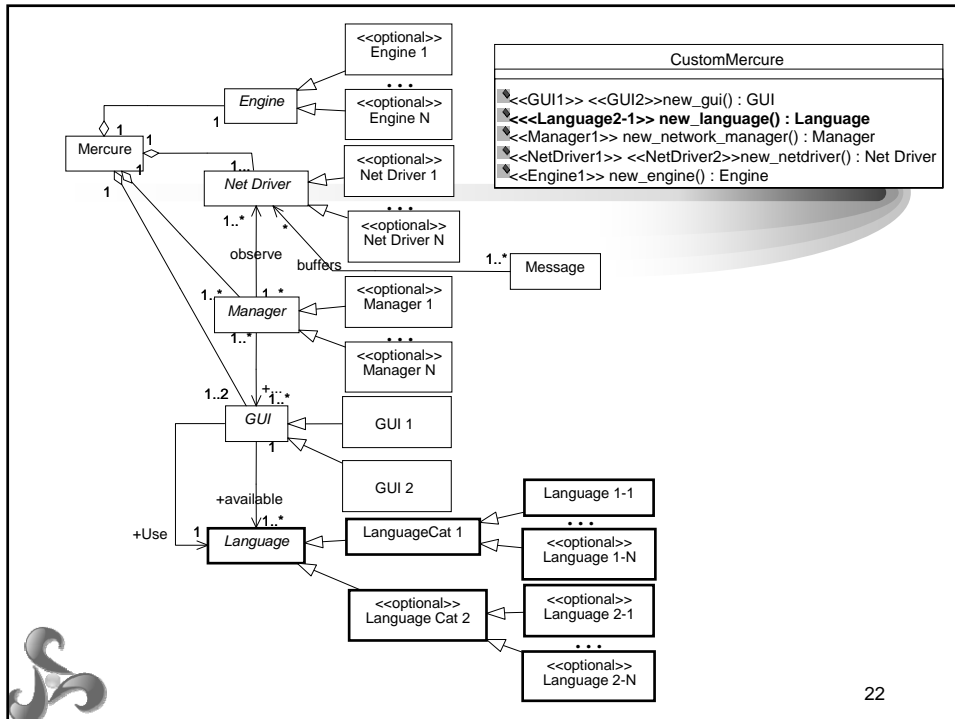
19



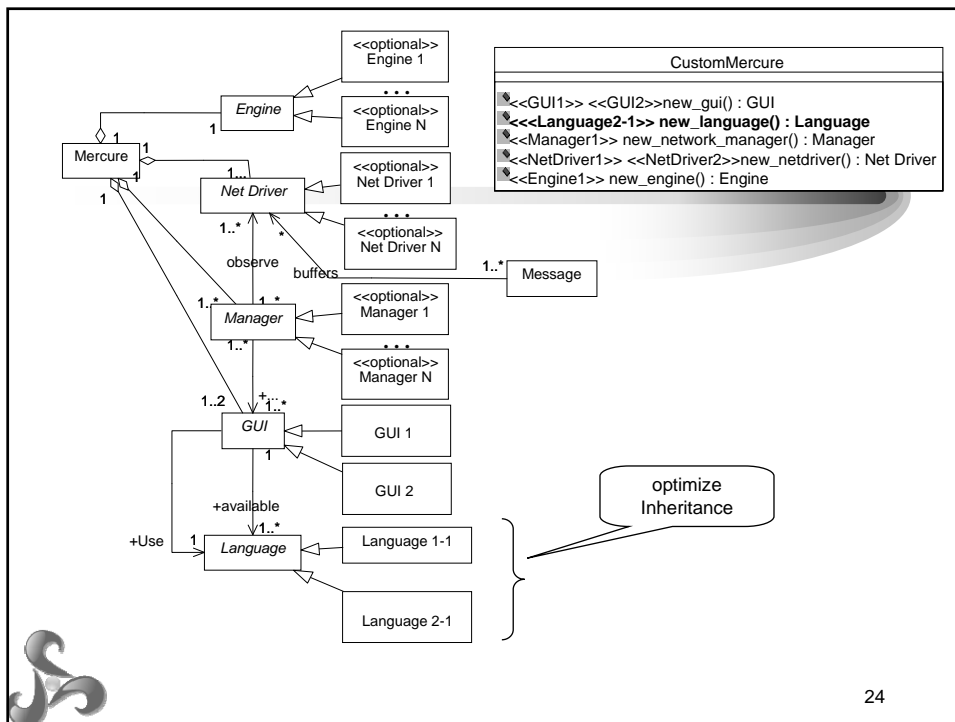
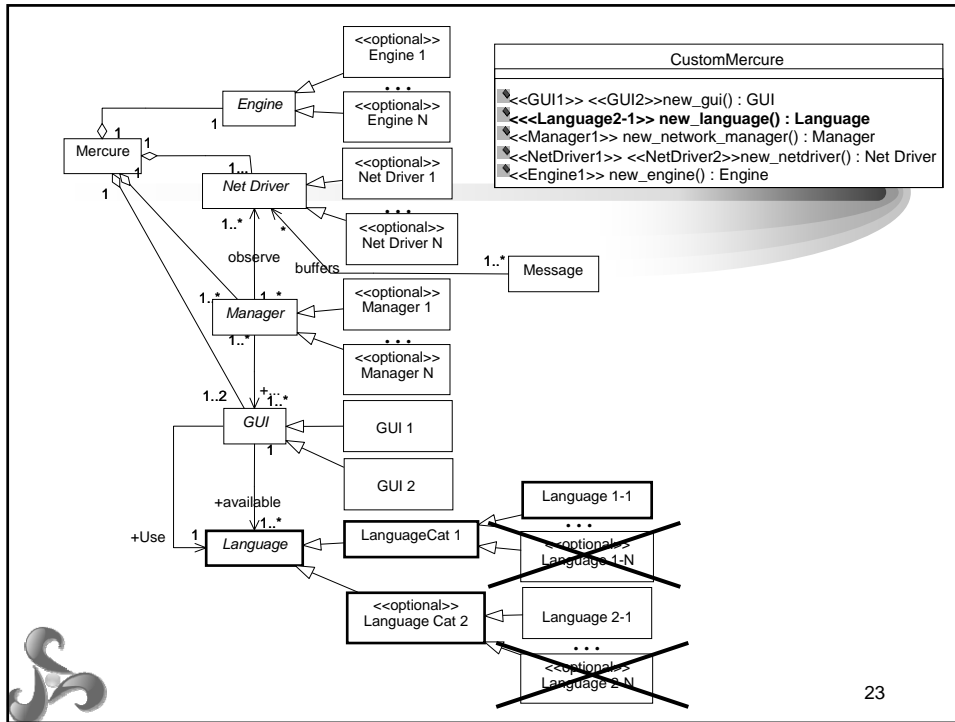
20

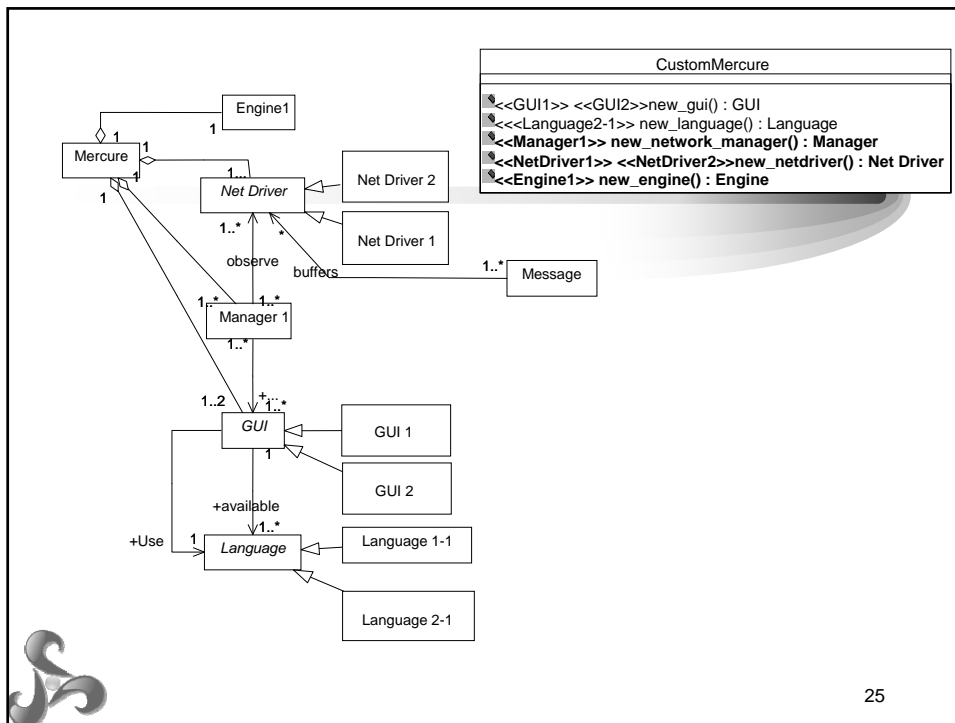


21



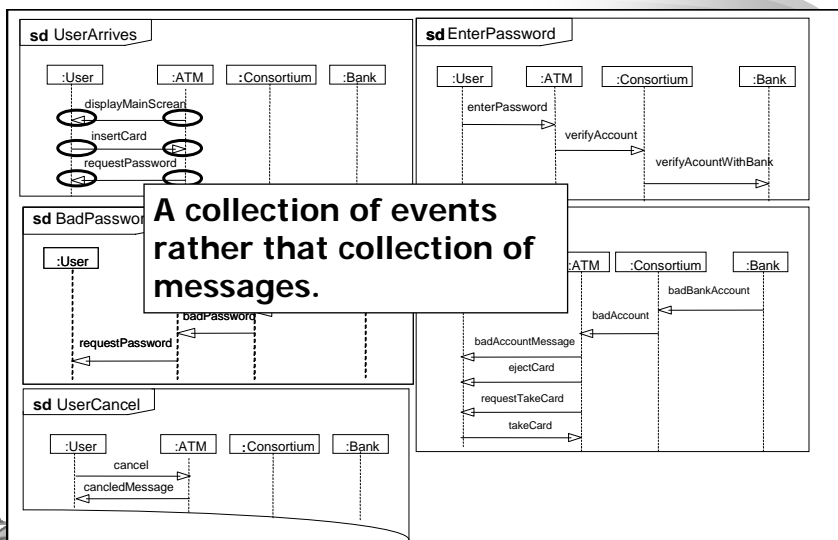
22





25

*Expl. 2: Derivation of products from product-lines:
behavioral issues based on Sequence Diagrams*

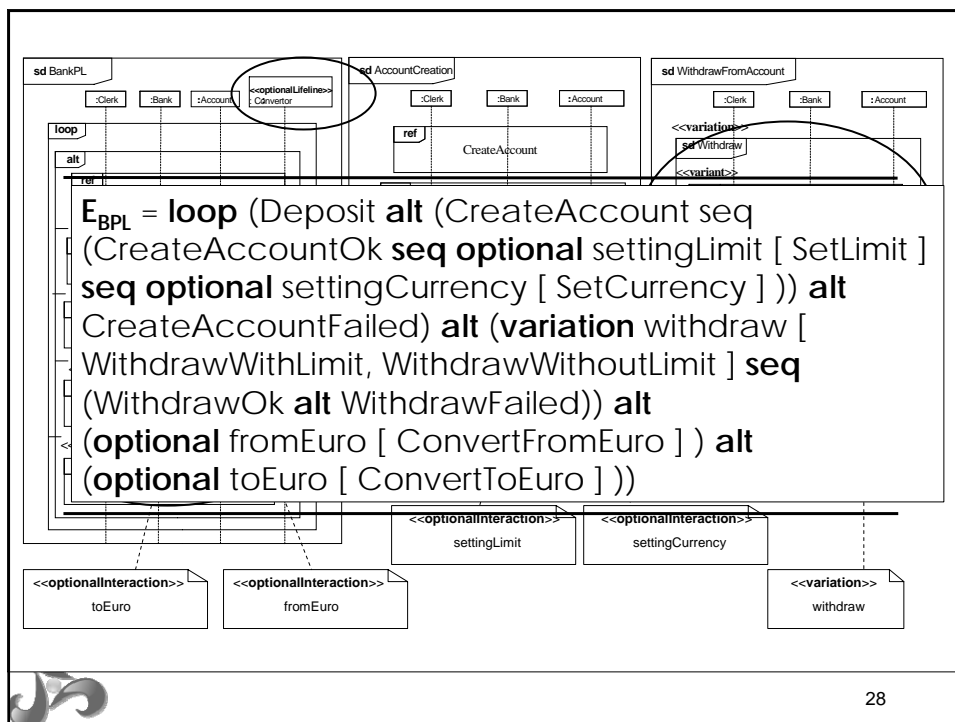


HMSCs & UML2.0 SDs: Combined SDs

- A combined SD: refers to a set of interactions and composes them by means of operators:
 - **Sequence (seq)**: weak sequential composition.
 - **Alternative (alt)**: choice between interaction operands.
 - **Loop (loop)**: iteration of an interaction.
- Extended with operators to model variability
 - **Optional, variation, virtual...**



27



28

STEP 1: Behavioral derivation

Product	Decision model instance (DMI)
BS1	DM1 = {(settingLimit, TRUE), (settingCurrency, FALSE), (withdrawAccount, 1), (fromEuro, FALSE), (toEuro, FALSE)}
BS2	DM2 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, FALSE), (toEuro, FALSE)}
BS3	DM3 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, TRUE), (toEuro, TRUE)}
BS4	DM4 = {(settingLimit, TRUE), (settingCurrency, TRUE), (withdrawAccount, 1), (fromEuro, TRUE), (toEuro, TRUE)}

$$\text{RESD} = \text{[[PL-RESD]]}_{\text{DMi}}$$



29

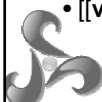
Behavioral derivation rules

$$\bullet \text{ [[optional name [E]]]}_{\text{DMi}} = \begin{cases} E \text{ IF } (name, TRUE) \in \text{DMi} \\ E_{\emptyset} \text{ IF } (name, FALSE) \in \text{DMi} \end{cases}$$

E_{\emptyset} is the *empty SD*: neutral element for **seq**, **alt** ; idempotent for **loop**

$$\bullet \text{ [[variation name [E1, E2,..]]]}_{\text{DMi}} = E_i \text{ IF } (name, i) \in \text{DMi}$$

$$\bullet \text{ [[virtual name [E]]]}_{\text{DMi}} = E_{\text{reff}} \text{ IF } (name, E_{\text{reff}}) \in \text{DMi}, E \text{ ELSE}$$



30

DM2 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, FALSE), (toEuro, FALSE)}

$E_{BS2} = [[E_{BPL}]]_{DM2}$

$E_{BS2} = \text{loop (Deposit alt (CreateAccount seq (CreateAccountOk seq } E_{\emptyset} \text{ seq } E_{\emptyset} \text{)) alt CreateAccountFailed) alt (WithdrawWithoutLimit seq (WithdrawOk alt WithdrawFailed)) alt (} E_{\emptyset} \text{) alt (} E_{\emptyset} \text{))}$

- E_{\emptyset} is the *empty* SD: neutral element for **seq**, **alt** ; idempotent for **loop**
 → expression reduction



DM2 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, FALSE), (toEuro, FALSE)}

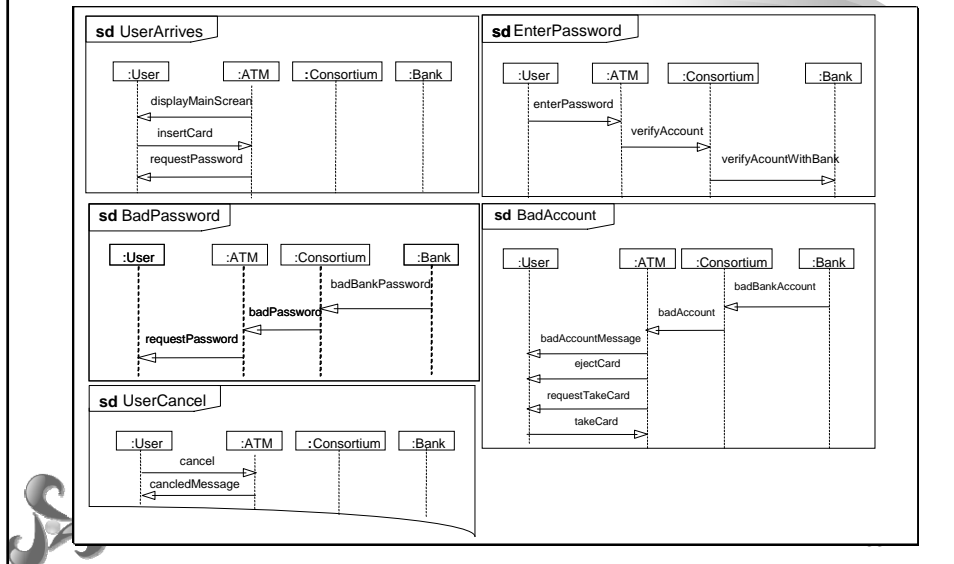
$E_{BS2} = [[E_{BPL}]]_{DM2}$

$E_{BS2} = \text{loop (Deposit alt (CreateAccount seq (CreateAccountOk)) alt CreateAccountFailed) alt (WithdrawWithoutLimit seq (WithdrawOk alt WithdrawFailed))}$

Step 1 result : One expression (RESA) for each product

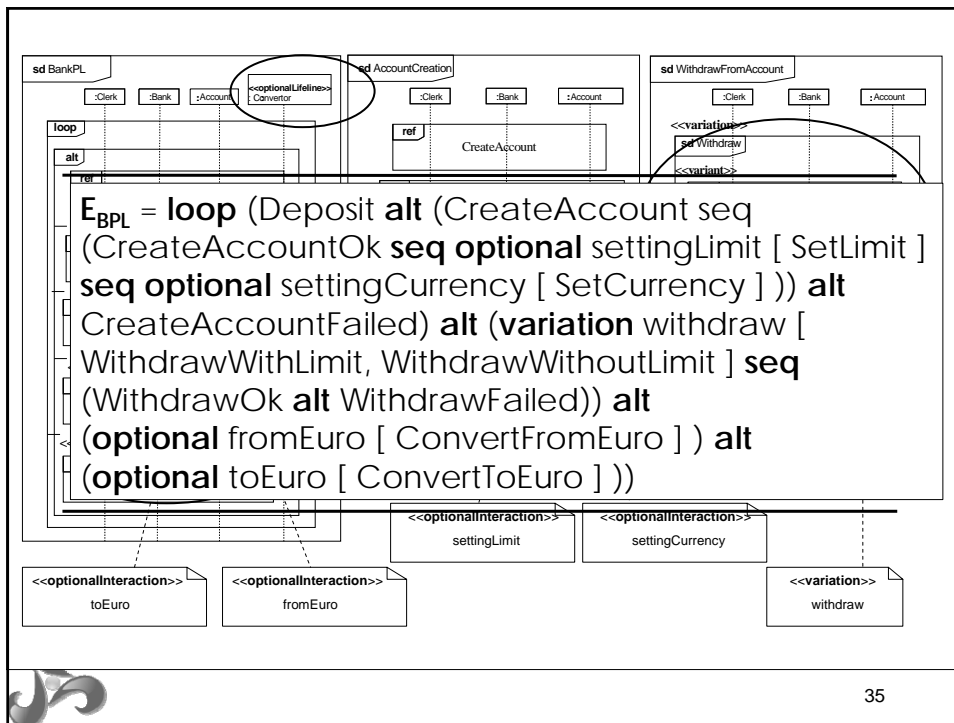


*Expl. 1: Derivation of products from product-lines:
behavioral issues based on Sequence Diagrams*



*HMSCs & UML2.0 SDs:
Combined SDs*

- A combined SD: refers to a set of interactions and composes them by means of operators:
 - **Sequence (seq):** weak sequential composition.
 - **Alternative (alt):** choice between interaction operands.
 - **Loop (loop):** iteration of an interaction.
- Extended with operators to model variability
 - **Optional, variation, virtual...**



STEP 1: Behavioral derivation

Product	Decision model instance (DMI)
BS1	DM1 = {(settingLimit, TRUE), (settingCurrency, FALSE), (withdrawAccount, 1), (fromEuro, FALSE), (toEuro, FALSE)}
BS2	DM2 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, FALSE), (toEuro, FALSE)}
BS3	DM3 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, TRUE), (toEuro, TRUE)}
BS4	DM4 = {(settingLimit, TRUE), (settingCurrency, TRUE), (withdrawAccount, 1), (fromEuro, TRUE), (toEuro, TRUE)}

RESD = [[PL-RESD]]_{DMi}

Behavioral derivation rules

- $[[\text{optional name } [E]]]_{DMi} = \begin{cases} E \text{ IF } (name, TRUE) \in DMi \\ E_{\emptyset} \text{ IF } (name, FALSE) \in DMi \end{cases}$

E_{\emptyset} is the *empty* SD: neutral element for **seq**, **alt** ; idempotent for **loop**

- $[[\text{variation name } [E1, E2, \dots]]]_{DMi} = \bigvee_i E_i \text{ IF } (name, i) \in DMi$

- $[[\text{virtual name } [E]]]_{DMi} = E_{\text{reff}} \text{ IF } (name, E_{\text{reff}}) \in DMi, E \text{ ELSE}$



37

DM2 ={(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, FALSE), (toEuro, FALSE)}

$E_{BS2} = [[E_{BPL}]]_{DM2}$

$E_{BS2} = \text{loop (Deposit alt (CreateAccount seq (CreateAccountOk seq } E_{\emptyset} \text{ seq } E_{\emptyset} \text{)) alt CreateAccountFailed) alt (WithdrawWithoutLimit seq (WithdrawOk alt WithdrawFailed)) alt (} E_{\emptyset} \text{) alt (} E_{\emptyset} \text{))}$

- E_{\emptyset} is the *empty* SD: neutral element for **seq**, **alt** ; idempotent for **loop**
 → expression reduction



38

DM2 = {(settingLimit, FALSE), (settingCurrency, FALSE), (withdrawAccount, 2), (fromEuro, FALSE), (toEuro, FALSE)}

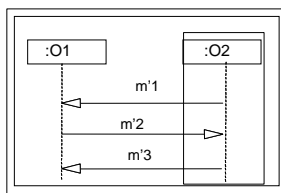
$E_{BS2} = [[E_{BPL}]]_{DM2}$

$E_{BS2} = \text{loop (Deposit alt (CreateAccount seq (CreateAccountOk)) alt CreateAccountFailed) alt (WithdrawWithoutLimit seq (WithdrawOk alt WithdrawFailed))}$

Step 1 result : One expression (RES D) for each product

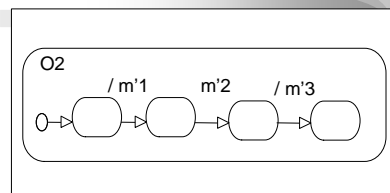


STEP2: UML Sequence Diagrams (SDs): → Statecharts



Inter-object view: Many objects, one example.

Synthesis



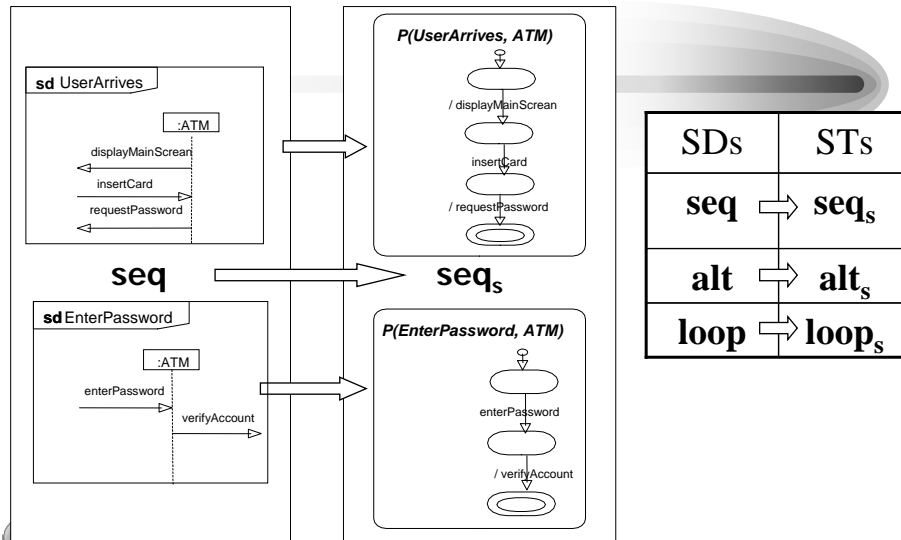
Intra-object view: Single object, a complete behavior.

- **Related work:**

- Kriss et al [UML 98],
- Koskimies et al [IEEE Software 98]
- Whittle et al [ICSE 00],
- Mäkinen et al [ICSE 01]..etc



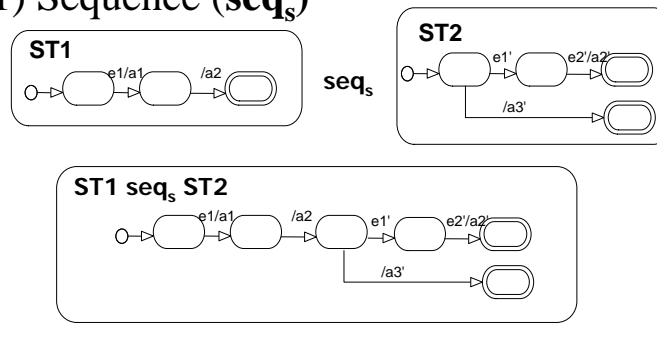
From combined SDs



41

Statecharts operators

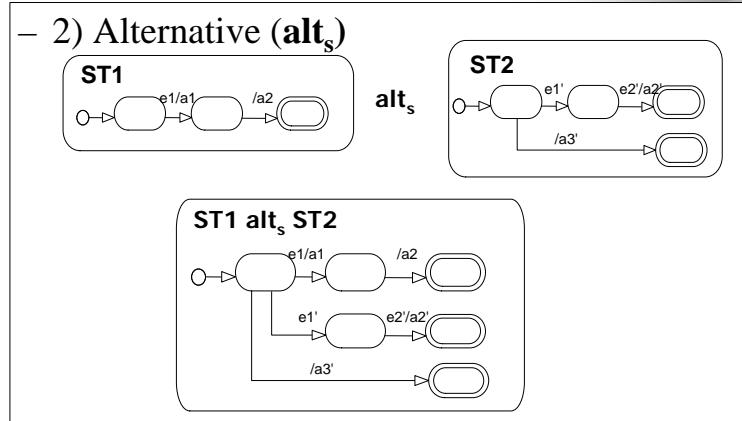
- 1) Sequence (**seq_s**)



42

Statecharts operators(contd.)

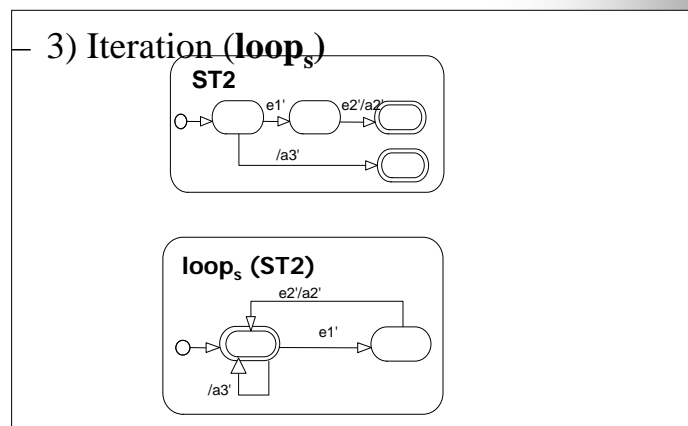
– 2) Alternative (alt_s)



43

Statecharts operators(contd.)

– 3) Iteration (loop_s)



44

From combined SDs (contd.)

E = loop (UserArrives seq (loop (EnterPassword seq BadPassword) seq EnterPassword seq (BadAccount alt UserCancel) alt UserCancel))

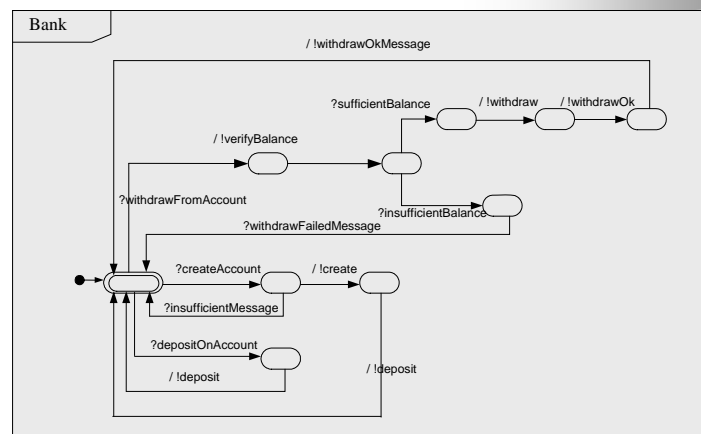


E = loop_s (P(UserArrives, ATM) seq_s (loop_s (P(EnterPassword,ATM) seq_s P(BadPassword, ATM)) seq_s P(EnterPassword,ATM) seq_s (P(BadAccount,ATM) alt_s P(UserCancel,ATM)) alt_s P(UserCancel,ATM)))



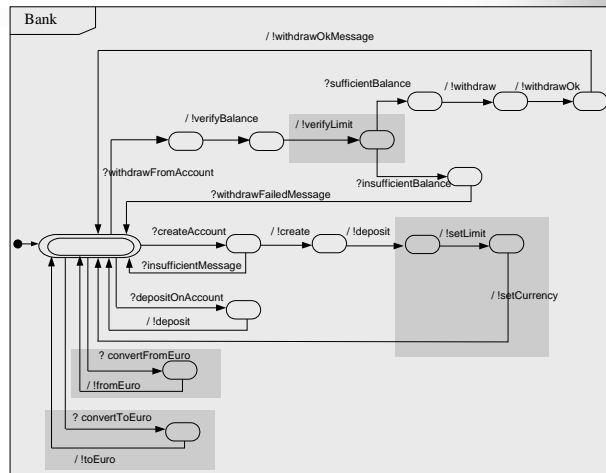
45

Result: Bank StateChart for BS2



46

Result: Bank StateChart for BS4



47

Conclusion & Perspectives

48

Conclusion

- Handling variability in SPL is complex
 - Exponential number of configurations
- Use models (& aspects) to manage complexity
 - Abstract away from #IFDEF & diff
- Executable Meta-Modeling to Automate Product Derivation from SPL Models
 - Both from Static & Dynamic aspects



49

Perspectives on SPL Research

- Composition of models
 - New ways of composing software from modeling elements
 - at both model and meta-model levels
 - Unifying MDE, AOSD, SPL, Generative Programming...
 - Composing models at runtime: dynamic adaptation
 - FP7 STREP: DiVA (03/2008-03-2011)



50

IRISA

Thank you!



INRIA

UNIVERSITÉ DE
RENNES 1