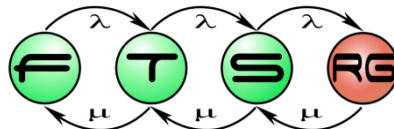


# Futásidejű modellezés

Models at runtime



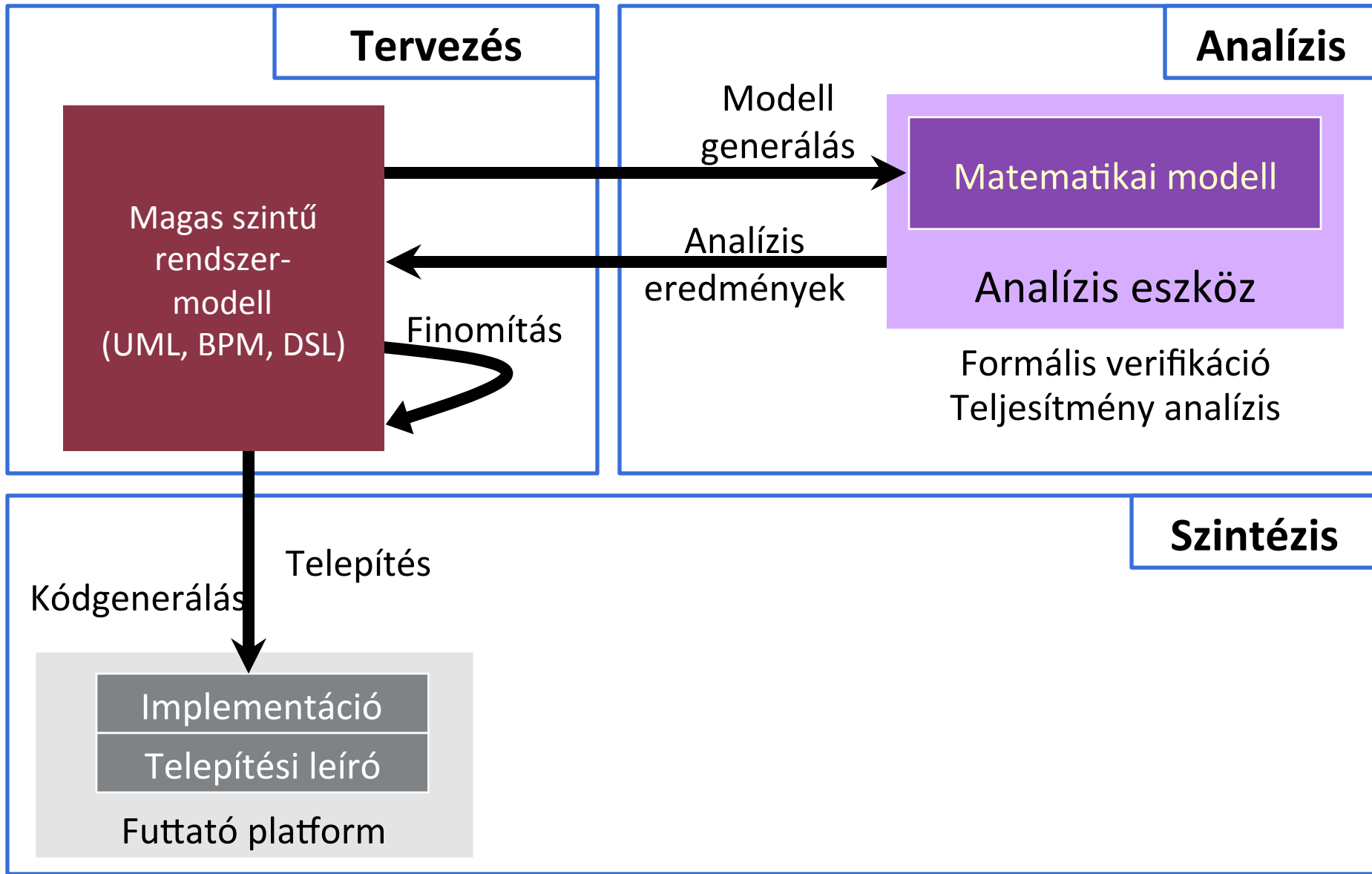
# Tartalom

- Koncepció: tervezési vs. futási idő
- Példák
- Összefoglalás

# TERVEZÉSI IDEJŰ MDE

Visszatekintés

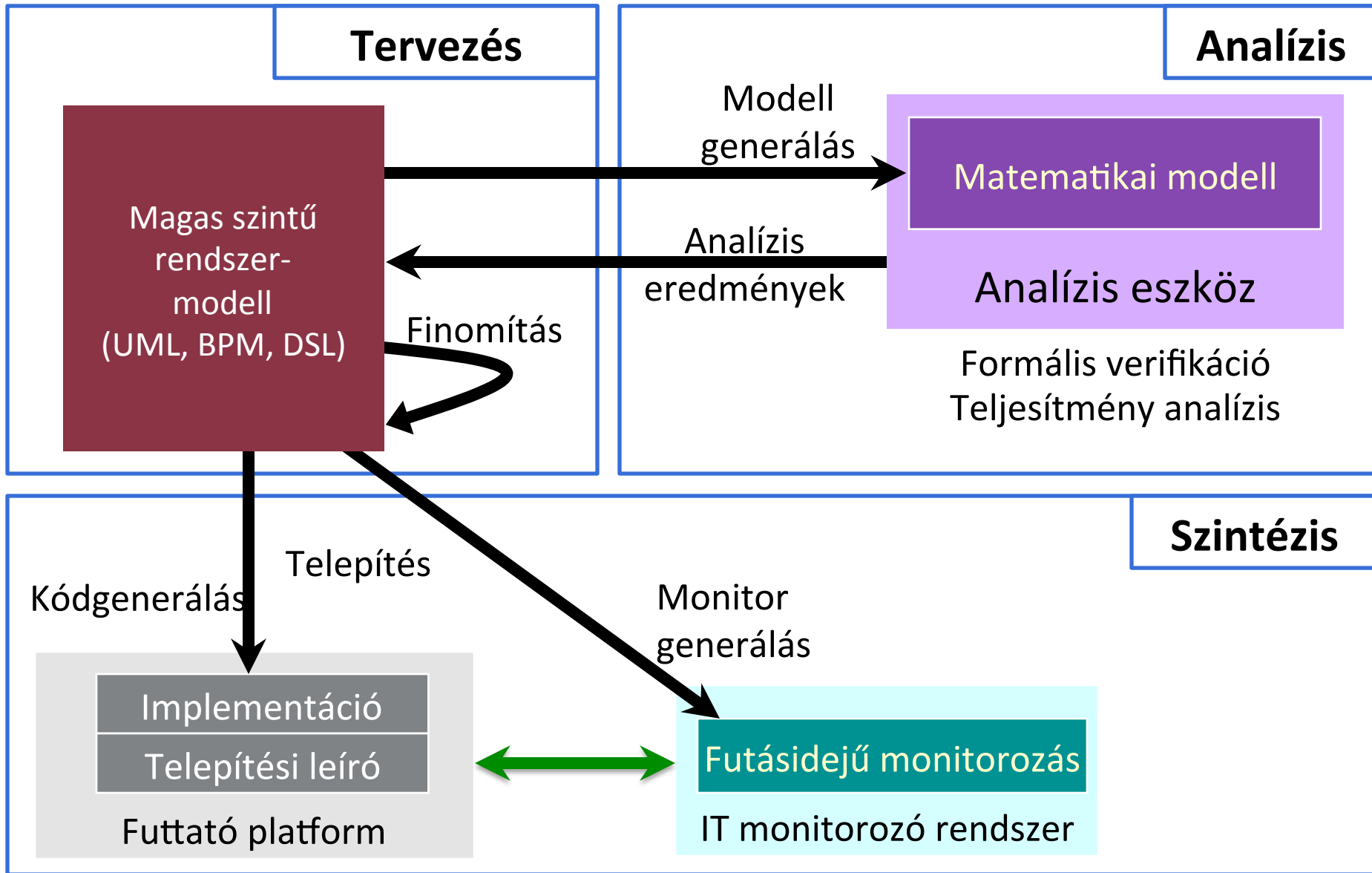
# Tervezési idejű MDE



# Modelltranzformációk, analízis technikák

- Tervezési idejű alaptermék
  - Modelltranzformációk: modellek automatikus
    - Lekérdezése, módosítása, leképzése, származtatása
  - Analízis technikák
    - Szimuláció
    - Modellellenőrzés
- Tervezési idejű követelmények
  - Futási idő (lehet nagyon hosszú is)
  - Erőforráshasználat (“végtelen”)
  - Technológia háttér (független az alkalmazástól)

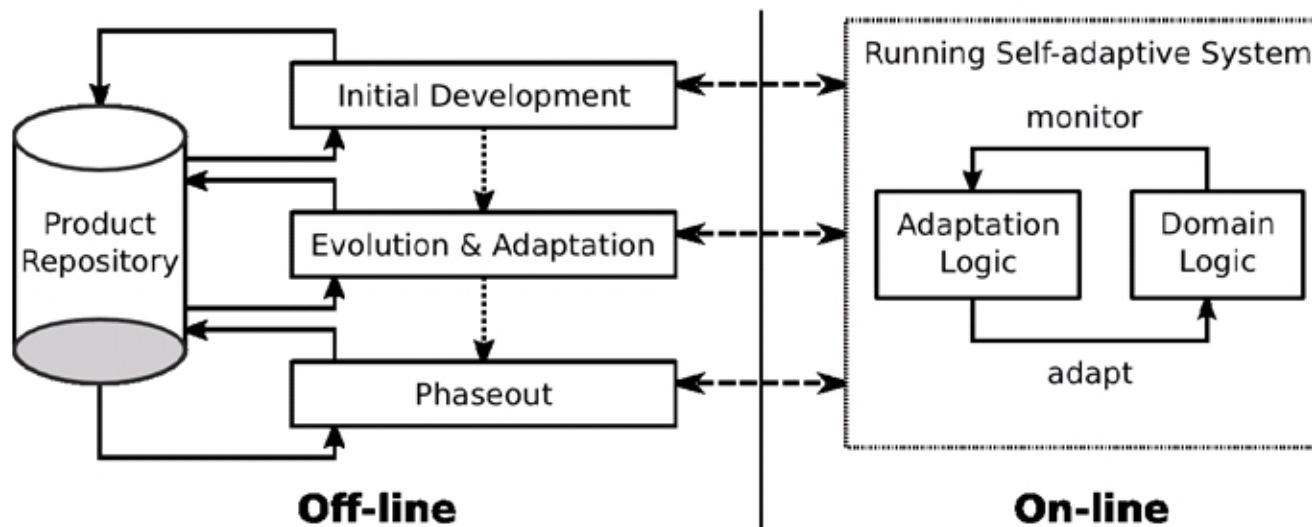
# Futási idejű komponensek



# FUTÁSI IDEJŰ MODELLEK

# Our “definition” for Models at Runtime Overview

- 2
- Any model (abstraction) used on-line, i.e., internal to the running self-adaptive software system, by the adaptation logic to represent running software (domain logic or adaptation logic of other subsystems) and its environment. Such models are used for monitoring or analyzing the running software and its environment, or for adapting the running software.



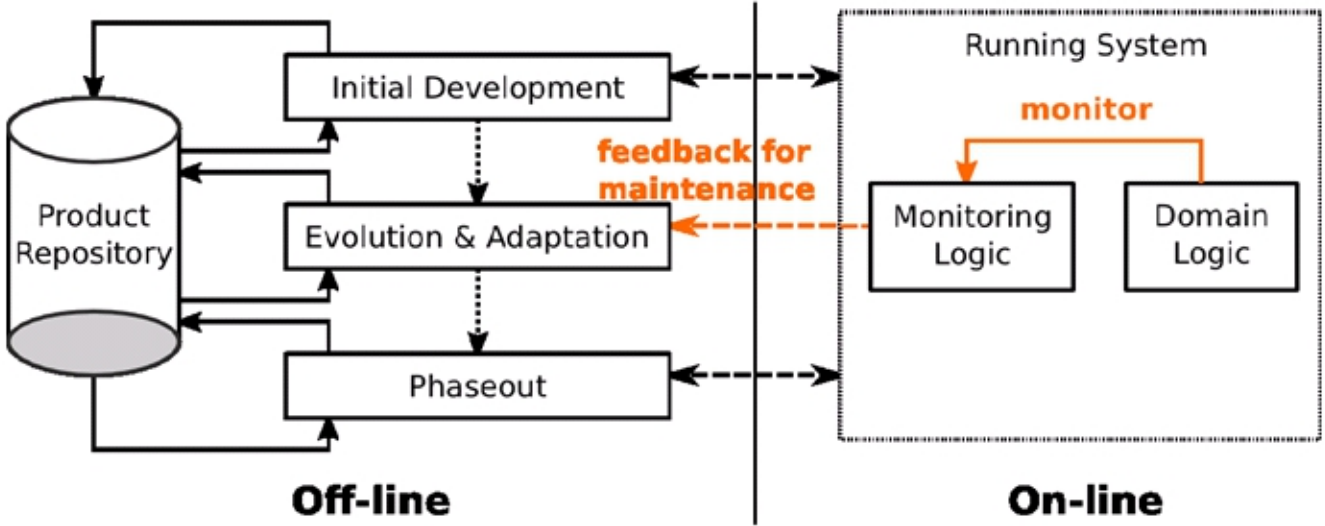
- On-line usage as the exclusive characteristic of “Models at Runtime”?
  - Application data (model) processed by a running system as “Models at runtime”?



# Our “definition” for Models at Runtime Monitoring

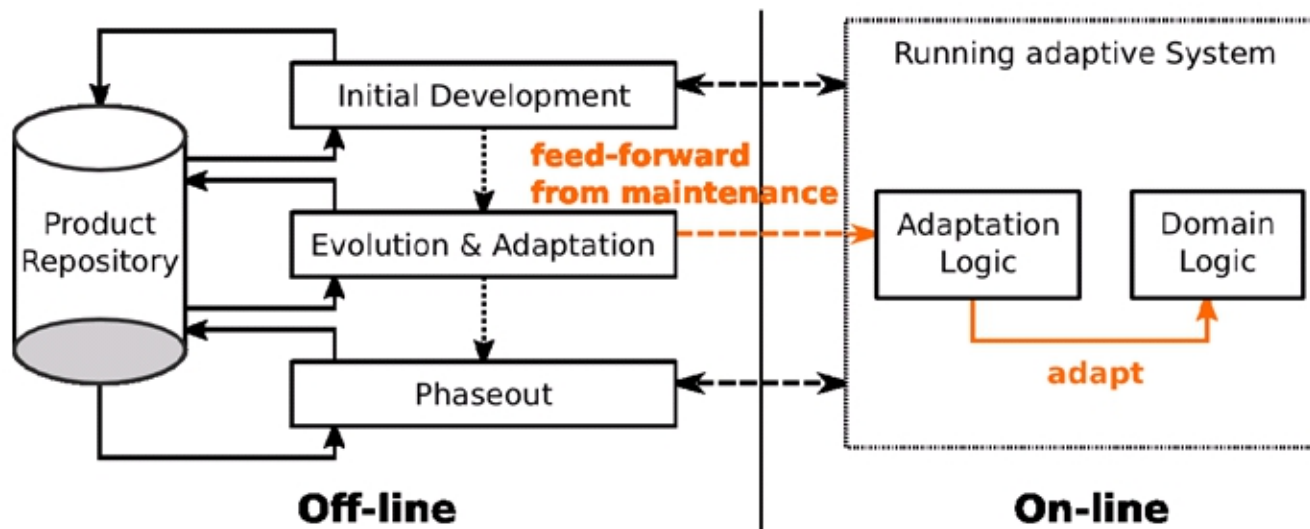
3

- Carrier of knowledge for typical (off-line) maintenance



# Our “definition” for Models at Runtime Adaptation

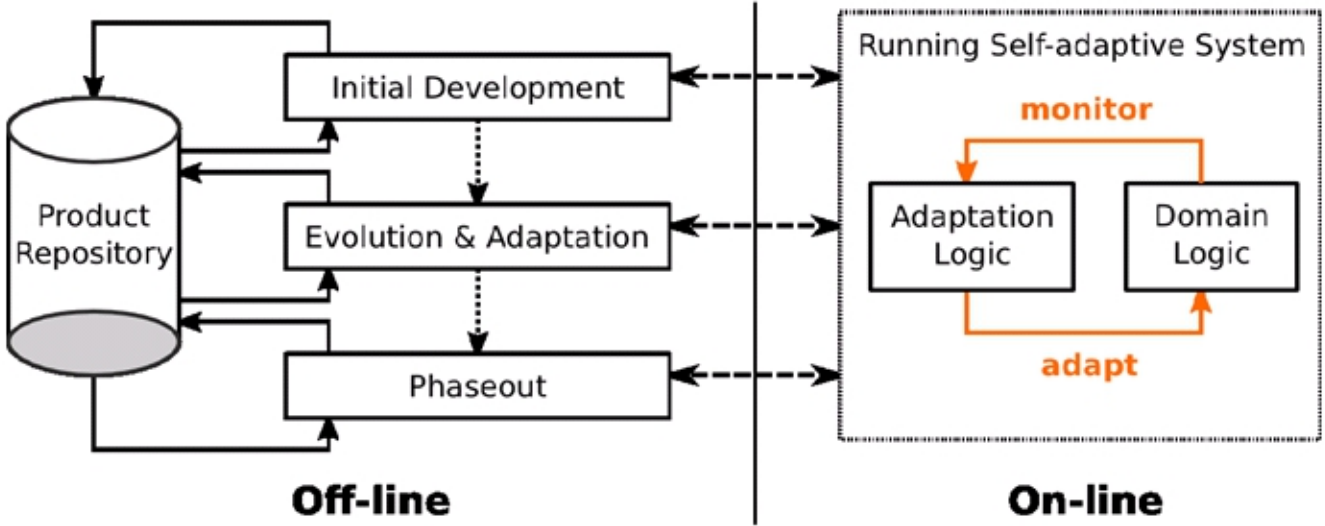
- Interface for (off-line) maintenance to change the running system



# Our “definition” for Models at Runtime Self-adaptation

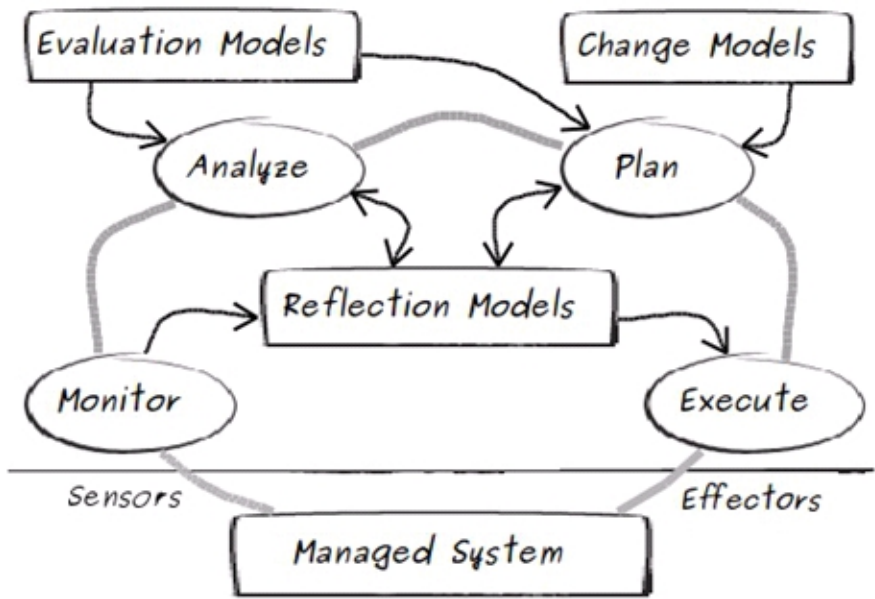
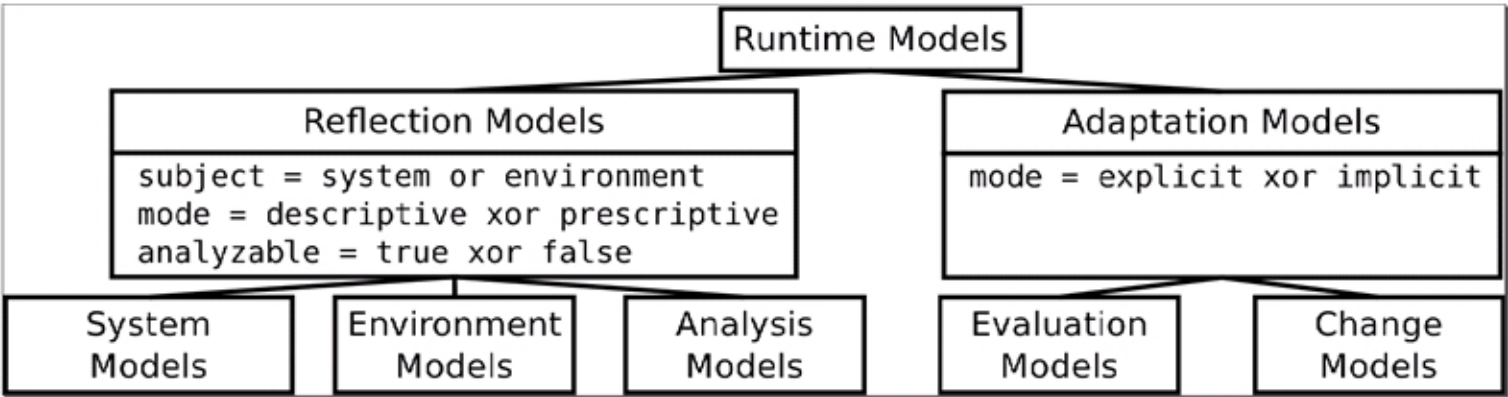
5

- Adaptation logic implements a feedback loop, like MAPE
- Co-existence of (off-line) maintenance & (on-line) self-adaptation



# Our “definition” for Models at Runtime

## Categories of Runtime Models



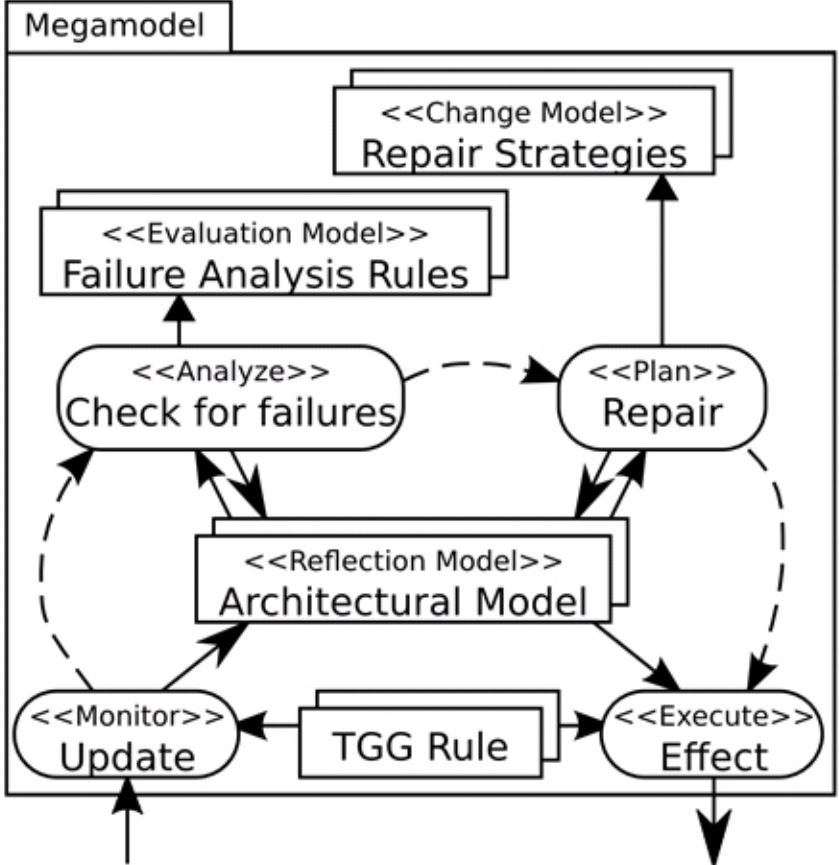
- APIs vs. Runtime Models
  - (Sensor & Effector)
- Not necessarily MDE (meta- and meta-metamodel levels) for all Runtime Models
- ... but promising!?
  - Abstraction
  - Relation to dev-time models

# Our research: Models at Runtime for Self-adaptive Software Systems

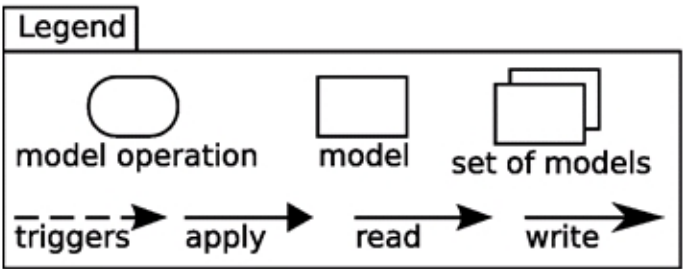
- Formal foundations based on graph transformations
- Multiple reflection models and adaptation loops (different concerns and abstractions)
- Incremental model synchronization techniques
  - Based on triple graph grammars (graph transformations)
- Adaptation models: executable, interpreted models
  - Based on Story Diagrams (graph transformations)
  - Based on Story Patterns (incremental graph transformations)
- **Megamodels: Specifying an Adaptation Loop**
- **Megamodels: Abstractions of Adaptation Loops in Systems with Multiple Loops**

# Megamodels: Specifying an Adaptation Loop

8



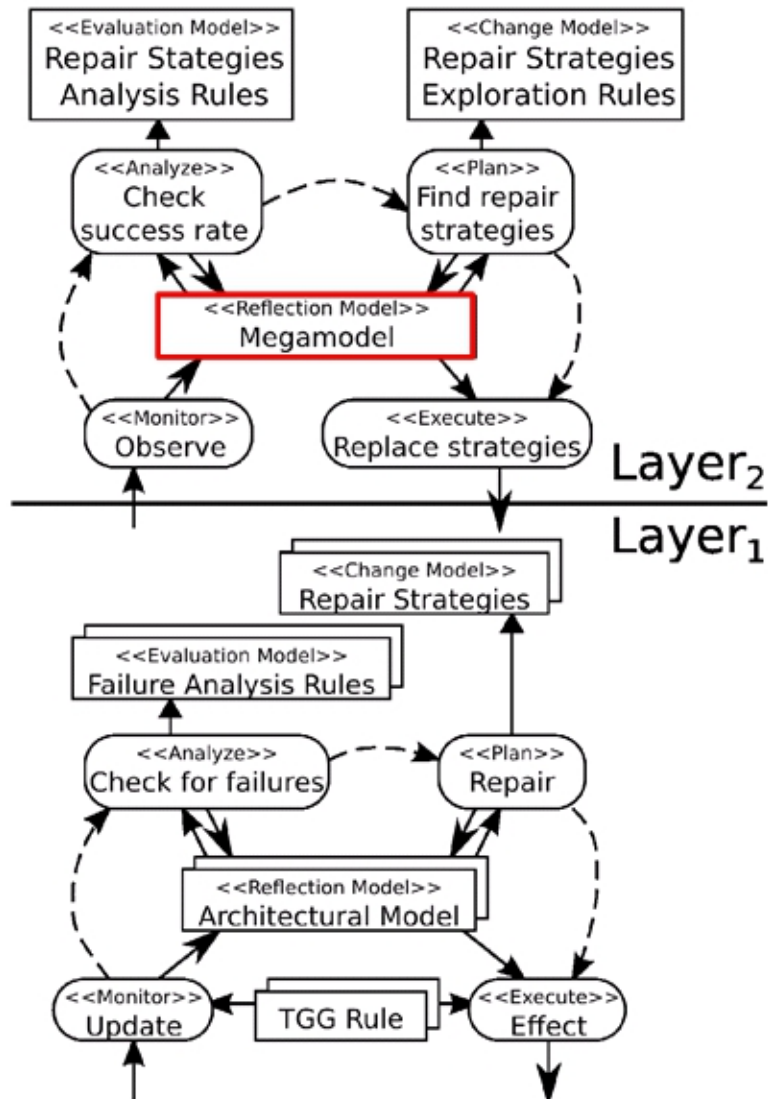
- MAPE as model operations on runtime models
- Megamodels
  - Specification of a loop
  - Structuring models and operations
  - Operationalization for MAPE
- Story Diagrams/Patterns + OCL for Evaluation and Change Models



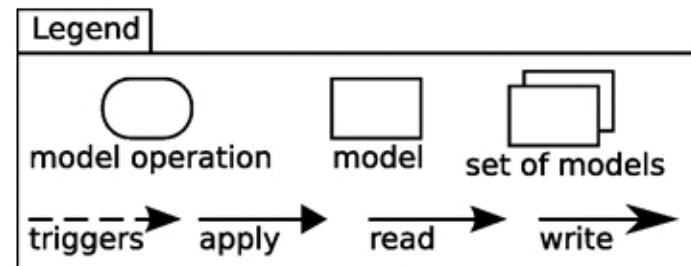
# Megamodels:

## Abstractions of Adaptation Loops in Systems with Multiple Loops

9



- Layered architectures for self-managing software systems (cf. Kramer & Magee, 2007)
- Each layer as a loop
- Timely decoupled layers/loops
- Megamodel as the abstraction of Layer<sub>1</sub> for Layer<sub>2</sub>
- Layer<sub>1</sub> - level of model elements
- Layer<sub>2</sub> - level of models contained in the megamodel



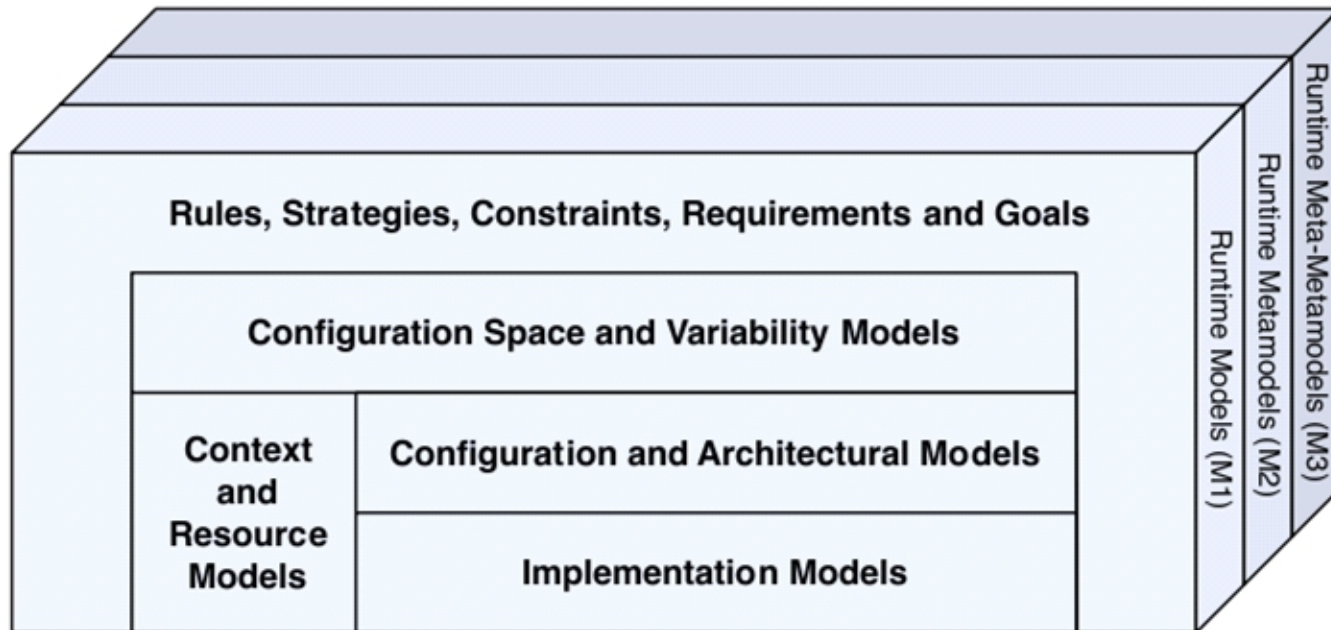
# Topics and Challenges

- Specialized vs. generic MDE/SE techniques for the runtime, and their trade-offs, e.g., concerning efficiency and development costs
- **Uncertainty** concerning the running system and its environment
  - Incomplete or imprecise information due to
    - Abstraction
    - Adaptive (not continuously) monitoring
    - Unobservable phenomena
    - Measurement errors
    - ...
  - How models at runtime may help?
- **Assurance** for self-adaptive software concerning
  - the core functionality (domain logic)
  - the adaptation loops (adaptation logic) based on models at runtime



# CLASSIFICATION

# Starting point...



# Purposes of Models@Run.time

- Reasoning (Analysis)
  - Defect Detection
    - Also Detect Defects in the Runtime Adaptation
  - Model Checking
- Decision
  - Changes to Running Systems
  - Runtime Adaptation
  - Reports, etc.

# Classification of Models@Run.time

Type of Model	Purpose	
	Type of Reasoning	Type of Decision
Context Model, Architectural System Model, Decision Model, Variability Model	Reasoning on context, environment, and current state of the system	Adapt the system or not (Architectural Level Adaption)
Context Model, Behavioral System Model (Sequence Diagram), Variability Model	Reasoning on context, environment, and current state of the system	Adapt the system or not (Behavioral/Trace Level Adaption, e.g., Aspect-oriented)
Context Model, Architectural Model, Decision Model, Variability Model	Defect Detection on the Decision Model	Identify possible invalid system states
PSM	Reasoning on applicability of reconfiguration at the platform specific level	Perform, delay or abort reconfiguration
Code	Reasoning on defects in code	Fix defects in code
...	...	...

# Resubic Systems (Human-CPS, Res Ubique)

Cyber-physikalische Systeme: Eine strategische Chance für Sachsen

Mixed-Reality  
Interaction

Cloud

Embedded

Ubiquitous



Cyber-Physical  
Systems (CPS)

Embedded systems  
meshed with clouds

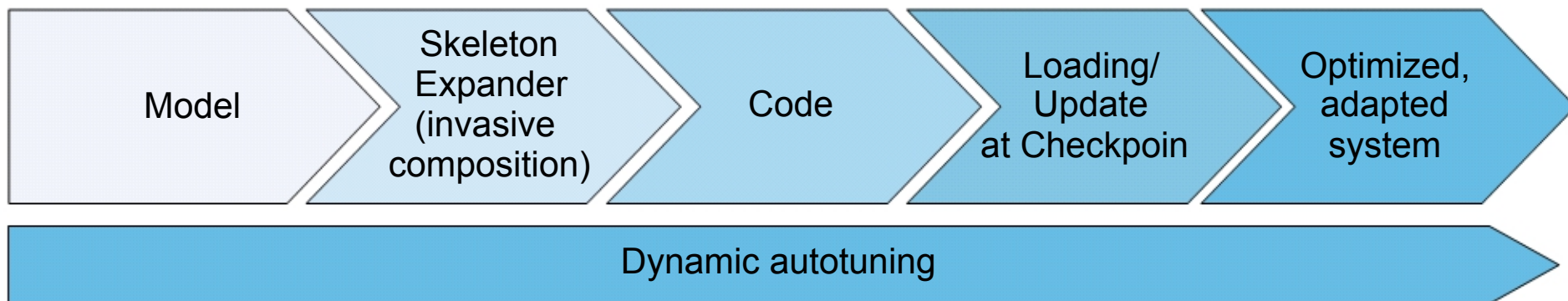


„res ubique“  
Resubic Systems  
Human CPS



## Cyber-physikalische Systeme: Eine strategische Chance für Sachsen

- Models are partial code (skeletons, non-executable)
- Dynamic auto-tuning means to complement the code to become executable
  - Invasive composition
  - Adaptive to parameter changes
- Dynamic code updating at checkpoint



## Cyber-physikalische Systeme: Eine strategische Chance für Sachsen

- State migration during code updating
- Lean deciders (incremental)
- Influence of quality contracts (rich component models)

Quality  
contract  
checking

Quality  
contract  
negotiation

Static quality  
autotuning

Dynamic  
quality  
autotuning

Multi-quality  
dynamic  
autotuning

Dynamic product lines  
(FLEXEPLÉ)

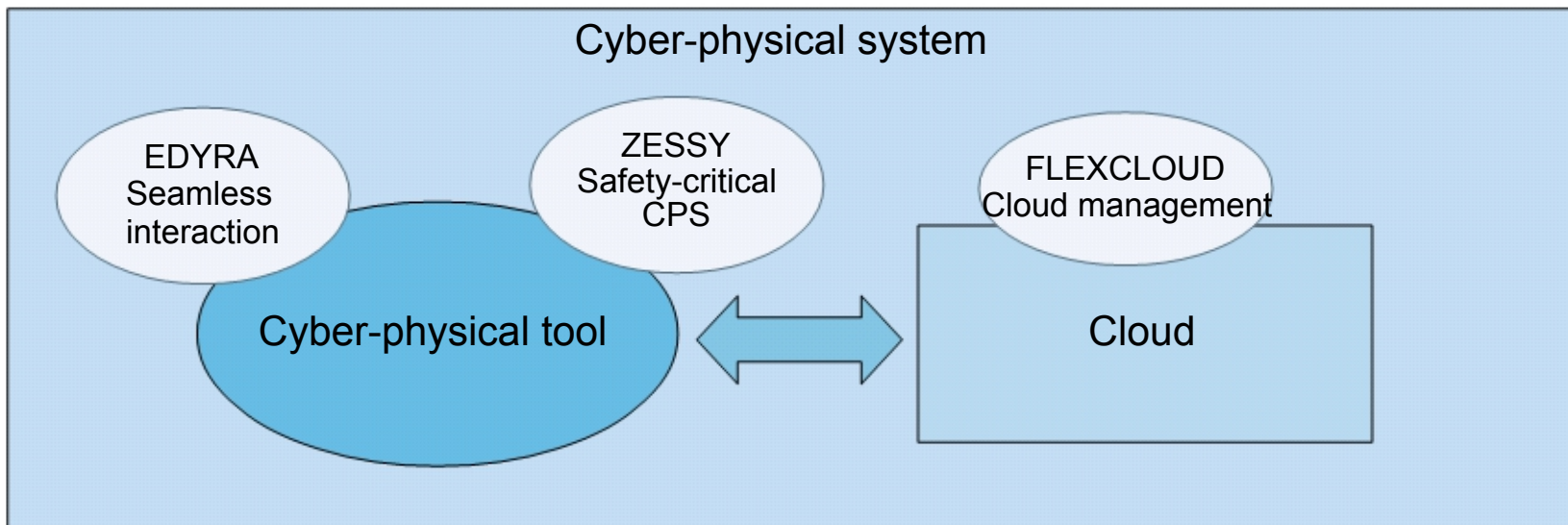
Energy-adaptivity with rich component models and  
dynamic autotuning (HAEC project)

Rich component models for adaptive systems  
(ZESSY-Qualitune project)



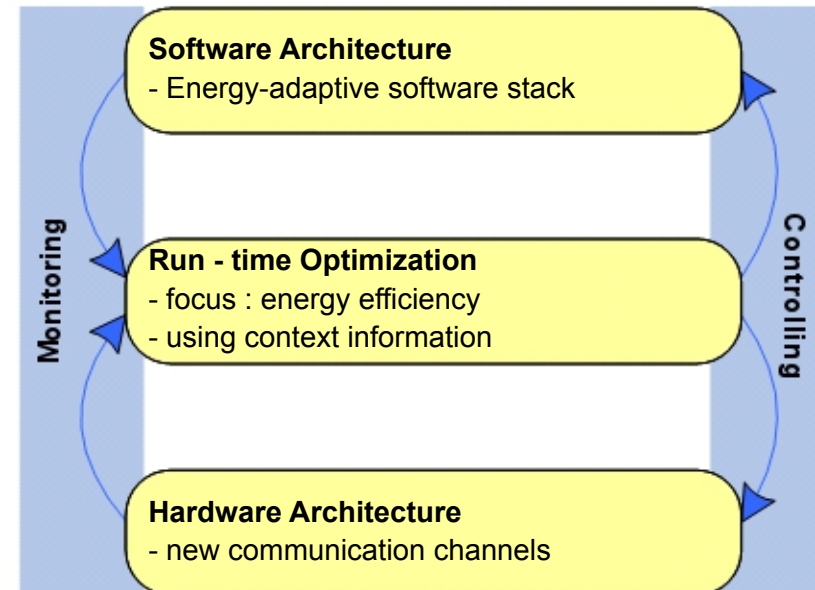
## Cyber-physikalische Systeme: Eine strategische Chance für Sachsen

- .. exploring human cyber-physical systems
- ESF Research Groups
  - ZESSY: safety-critical cyber-physical systems
  - EDYRA: seamless interaction, personal info services
  - FLEXCLOUD: cloud management
- 4,5 Mio € 2011-13, 19 researchers, ESF, SMWK
- Focus „Smart Office“



## Energy closed-loop optimization needs:

- Energy Modeling
  - Quality specifications (utility/resource)
  - Energy models
  - Context models
  - Preplanning
- Energy Measuring, Monitoring and Analysis
  - Stream analytics
  - Quantitative models
  - Semantic technologies
  - Security
- Energy Control and Reconfiguration
  - Decision algorithms and energy-utility functions
  - Energy-adaptive software architectures
  - Energy-adaptive OS
  - Energy-adaptive network



# Idea

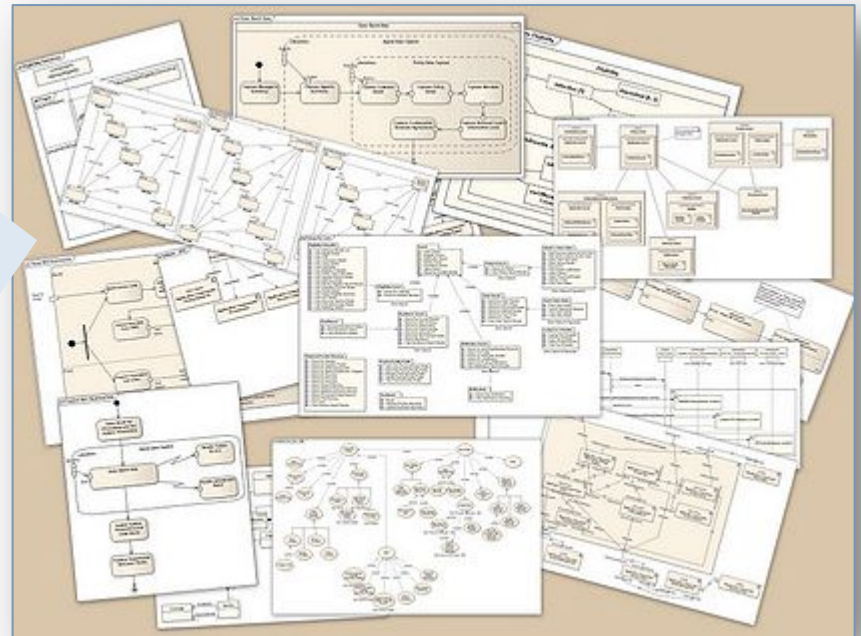
Extracting relevant information from a linux box («snapshot») at a given moment of time

Analysis and upgrade simulation on the «snapshot»



*Linux box*

*injectors*



*«snapshot»*

# Ingredients 1

**Modeling languages** for describing the several aspects of a linux distribution

- Packages
  - including maintainer scripts
- System Configuration
  - Installed packages
  - Configuration files
  - MIME-type handlers
  - Alternatives
  - etc

**Injectors** for harvesting the system and building the models

- collection of injectors

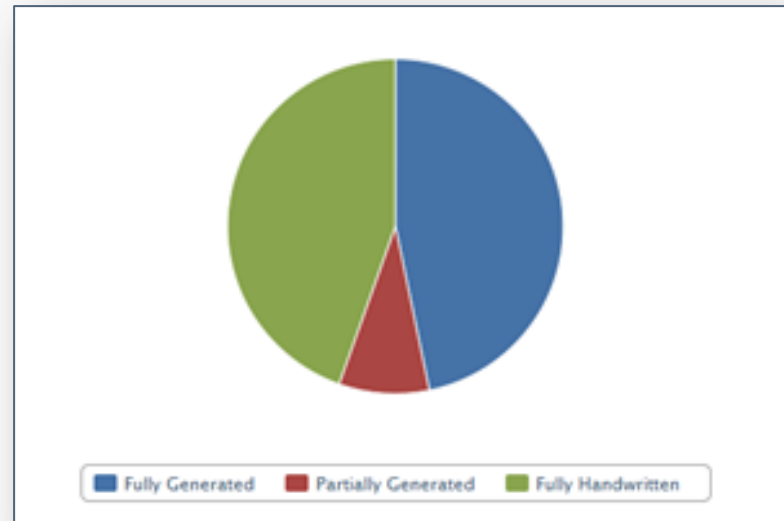
# Ingredients 2

## Modeling language for the maintainer scripts

- scripts are written in POSIX languages whose semantics is far from being simple, although
- maintainer scripts **does not** harness the full expressivity of such languages (template-based “macro-language”)

## Maintainer scripts as models

- which semantics ?



# Ingredients 3

**Transformational semantics** for simulating the behavior of the maintainer scripts on the system «snapshot»

- M2M transformations obtained by «compiling» the maintainer scripts into ATL

**Fault detector**, a general mechanism for performing queries over the «snapshot» for digging the model and search for inconsistencies



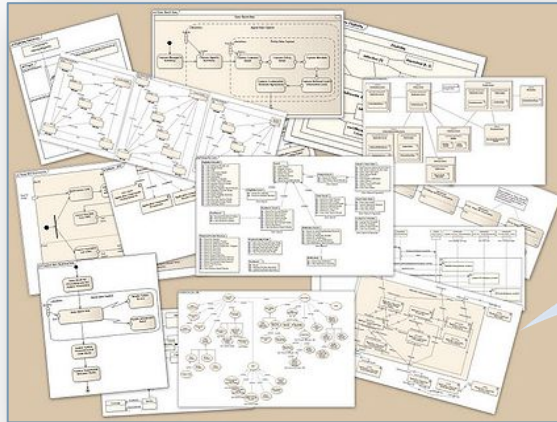
Configuration 1



Configuration 2

system injection

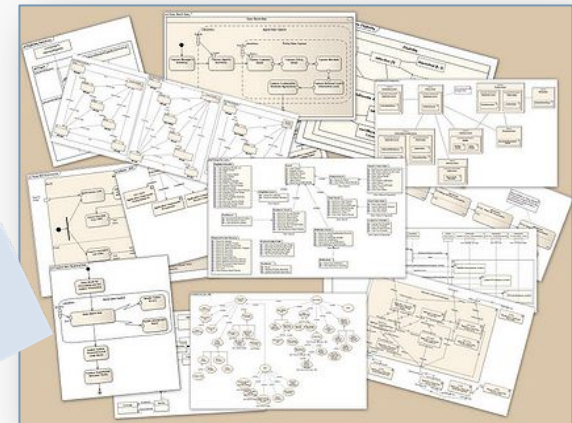
package injection



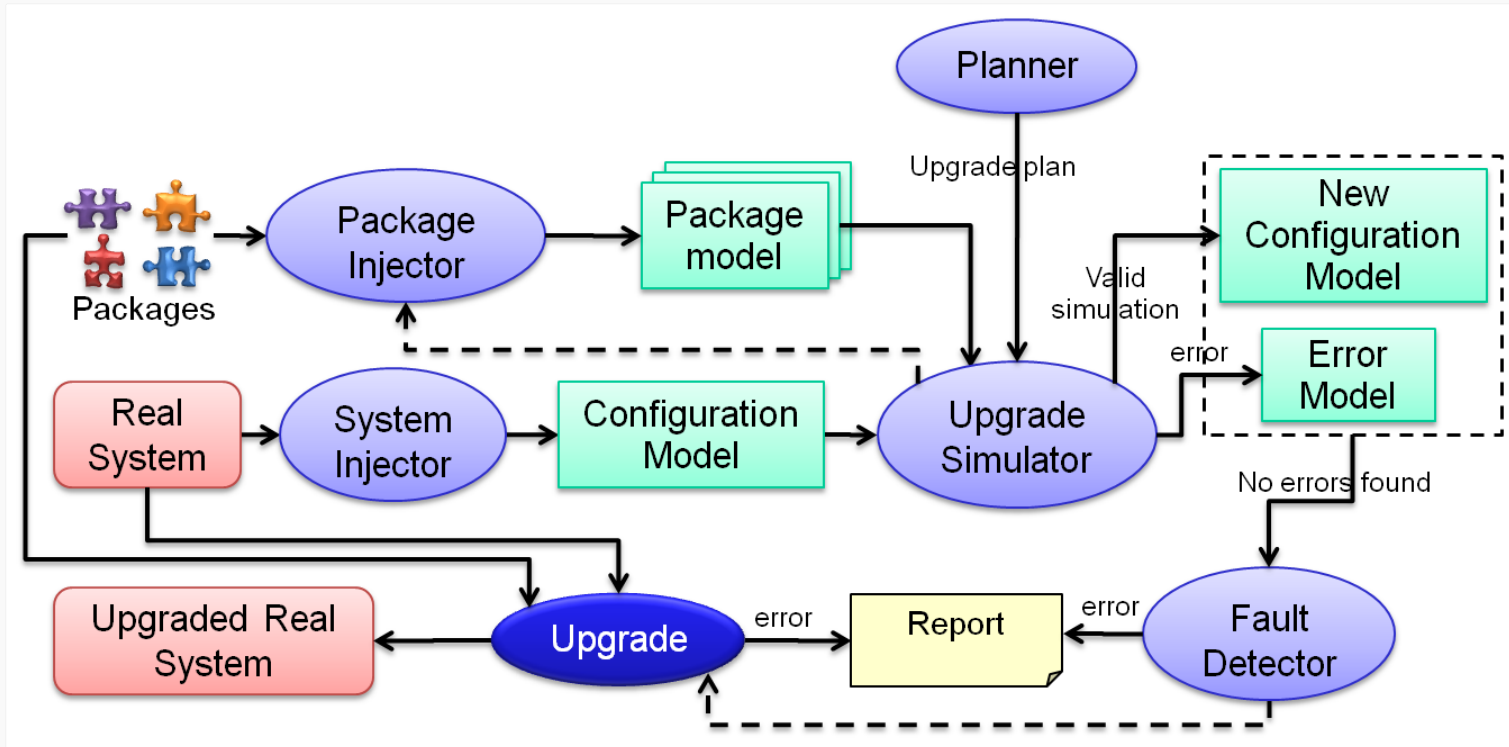
«snapshot» 1



M2M transformation



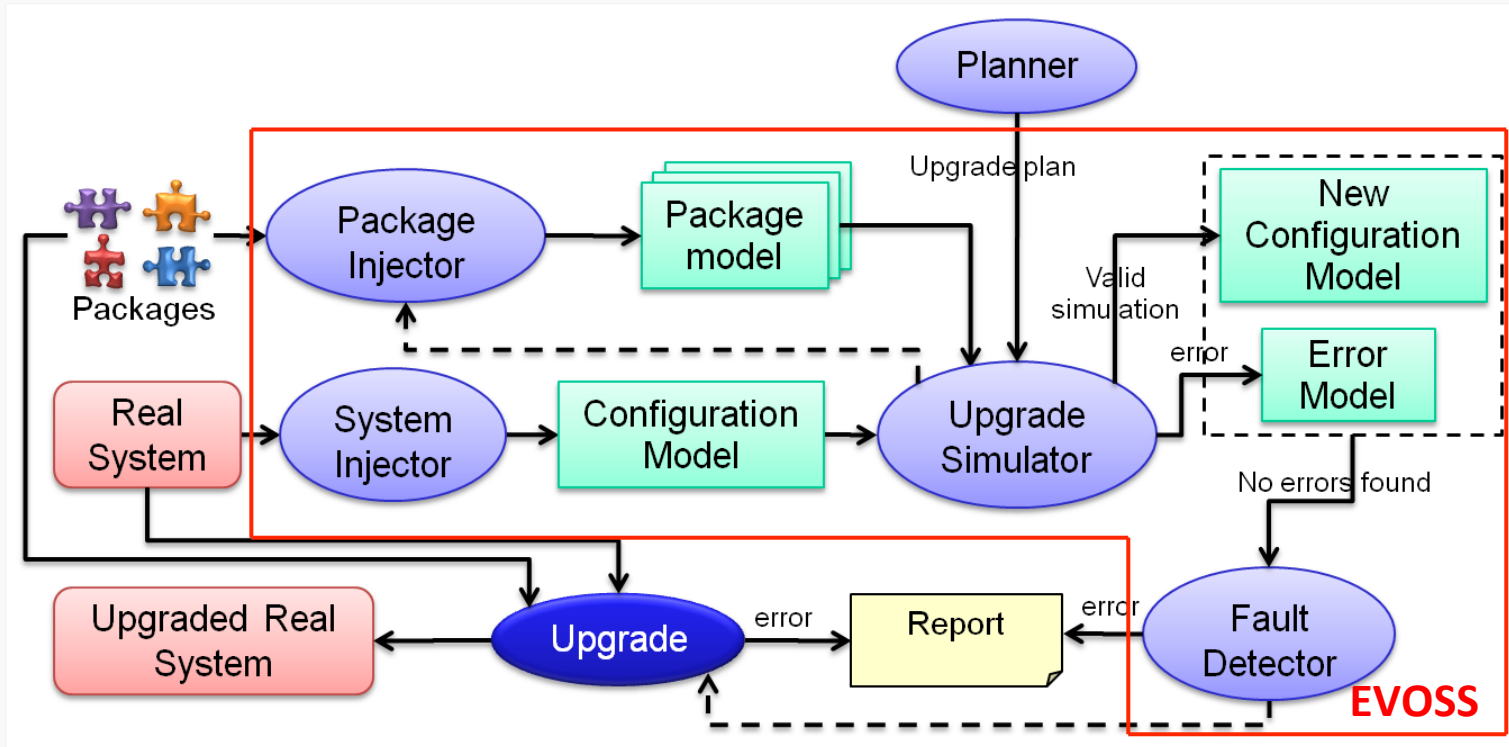
«snapshot» 2



A model-based approach is introduced to support the package upgrades and enhance the failure detection possibilities:

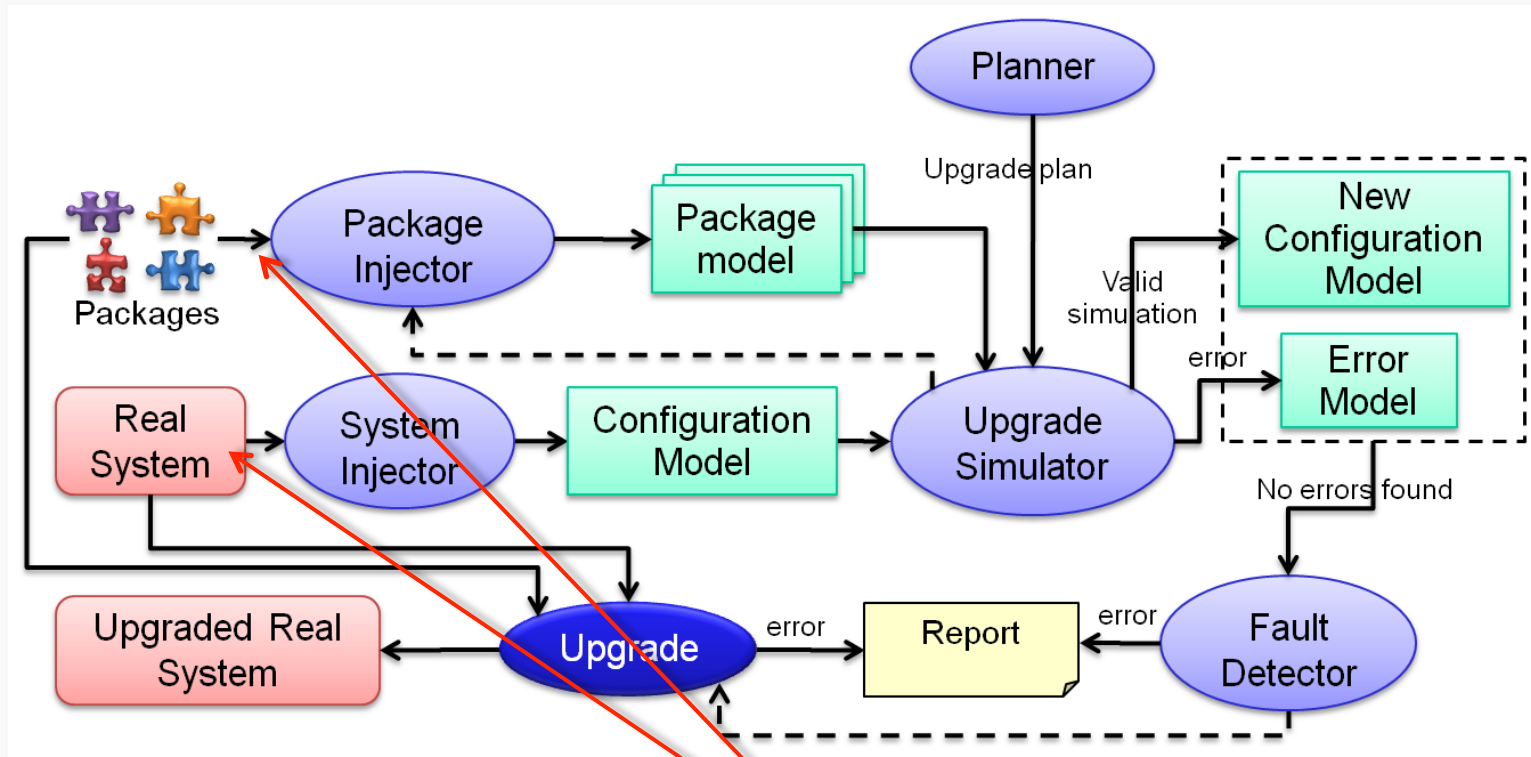
- the **simulator** is used to predict the effect of maintainer script executions (deploy-time failures)
- the **fault detector** is used to deal with undetected failures





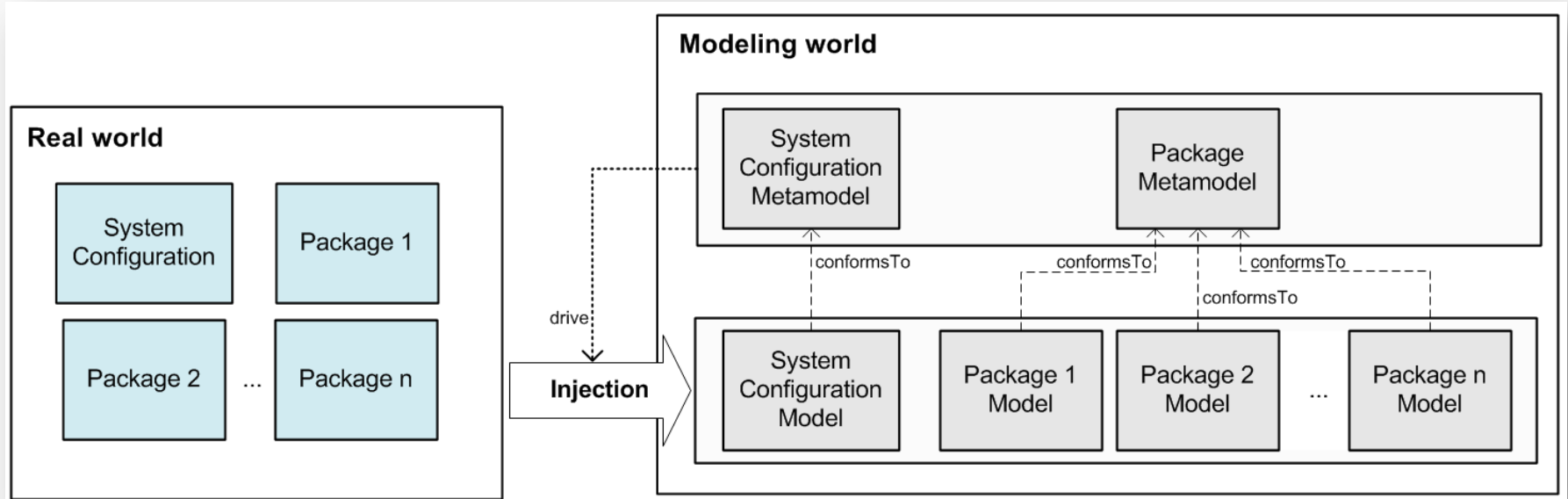
A model-based approach is introduced to support the package upgrades and enhance the failure detection possibilities:

- the **simulator** is used to predict the effect of maintainer script executions (deploy-time failures)
- the **fault detector** is used to deal with undetected failures



Used to represent in terms of models packages and system configurations

# Configuration and package injectors

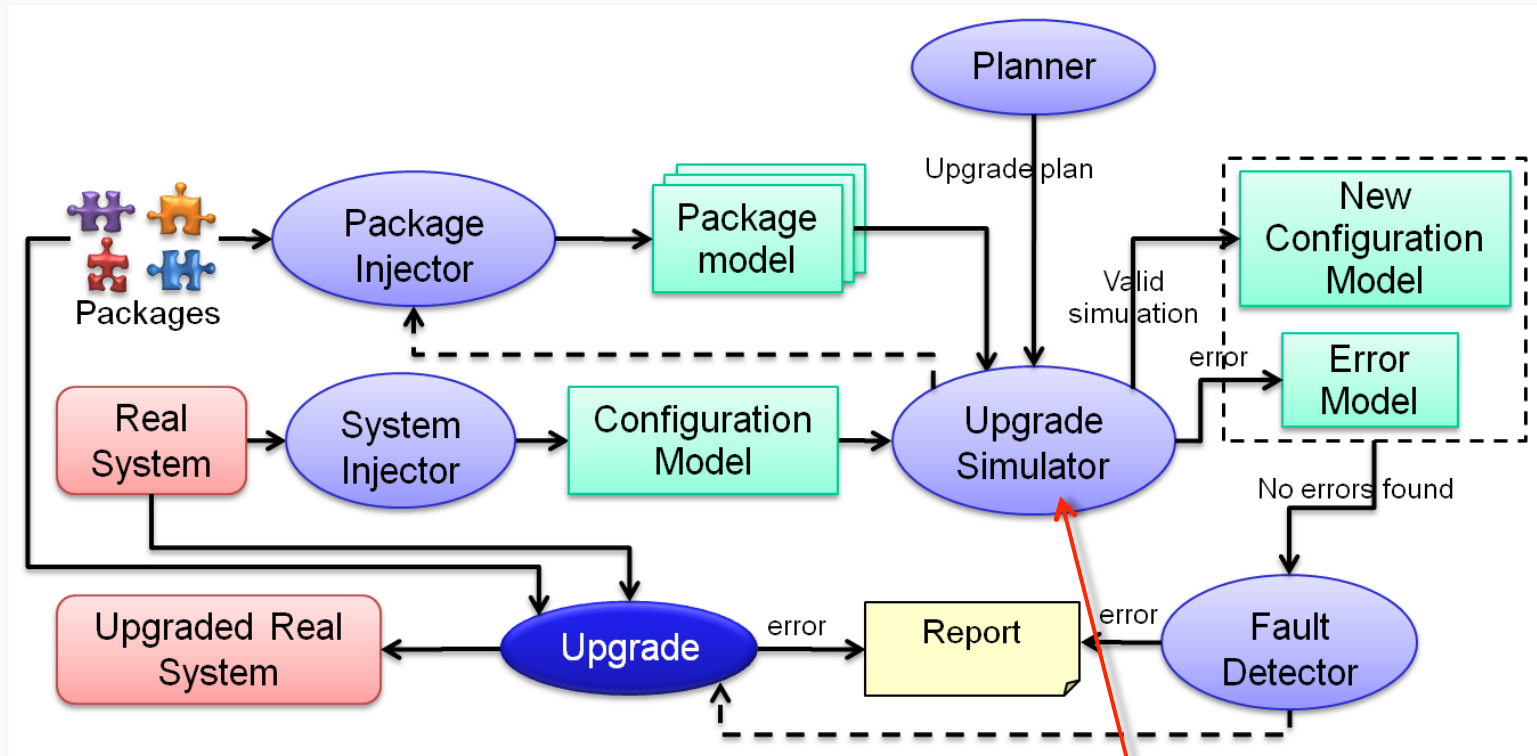


By means of the model injection, given a linux system a corresponding model is obtained

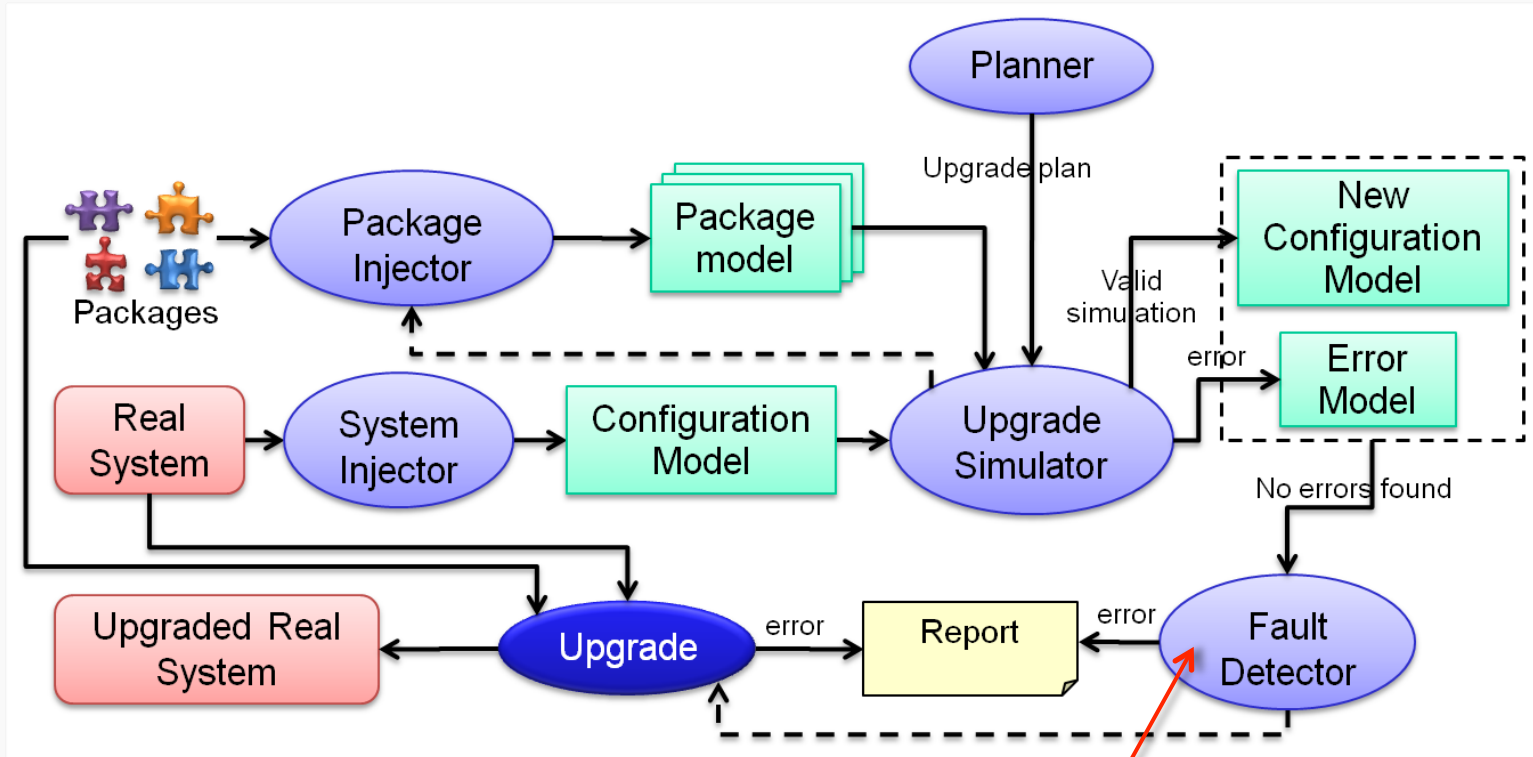
Injectors inspect software artifacts and inject relevant information into corresponding models

The process is driven by the metamodels

# Upgrade Simulator



Used to simulate system configuration upgrades



Used to check system configurations for faults that might give place to failures in the real system

# Conclusions 1

The **prediction** of the package upgrade impact has been enhanced: a **larger** number of cases which lead to faulty behaviors can be detected

The core ingredients have been defined

- **Modeling languages** for the different aspects
- **Injectors**: system configuration and packages
- **Transformational Script Semantics**
  - based on ATL
- **Fault detection**: query-based global knowledge base
  - OCL and JAR queries to detect faults
  - Contribution from the community

# Conclusions 2

The main difficulty is in building the model injectors

- labor intensive, deep platform knowledge required, ad-hoc techniques
- Interesting (academic) tools available
  - GRA2MOL – grammarware / modelware bridging
  - WIRES\* – Model transformation orchestration

## Monolithic real scale metamodels

- Maintenance of the developed metamodels has been an issue: interdependencies, ripple effects
- Difficult stabilization of the support tools

## Model comparison for validating the simulation

- EMF Compare

# Conclusions 3

## Project

This work has been done within the

- EU FP7 ICT STREP MANCOOSI  
<http://www.mancoosi.org>



(terminated on June 2011)

## Industrial partners

- Mandriva
- Caixa Magica
- IBM Ilog



```
#!/bin/bash
MAXLENGTH=4 (cut
MINLENGTH=1MAX
SMALLESTNAME=
do if [ $(cut
then echo #NAME)
echo #NAME
fi
```

## Fault Detector Web Portal

**Instructions**  
This is the WebPortal component of the Fault Detector. By means of the the right menu, you can edit:

- THE LIST OF OCL QUERIES
- THE LIST OF JAR QUERIES
- THE LIST OF KNOWN FAULTS
- THE LIST OF POSSIBLE SOLUTIONS
- THE LIST OF USERS THAT CAN ACCESS TO THE PANEL

**Press:**  
[+] to add an element to the associate list  
[M/C] to start the search for an existing element. Once found, you can then modify or delete it.

**Manage Elements**

- > fault [+] [M/D]
- > jarquery [+] [M/D]
- > oclquery [+] [M/D]
- > solution [+] [M/D]
- > users [+] [M/D]

**Manage Associations**

- > jarquery <> fault  
[+] [M/D]
- > oclquery <> fault  
[+] [M/D]
- > solution <> fault  
[+] [M/D]

Contact Us | Logout  
Copyright © 2011 - All Rights Reserved.

# ÖSSZEFOGLALÁS