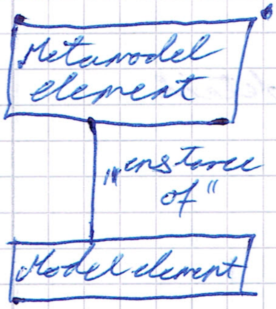
02 ① Domain specific language, metamodelling, EMF

a) Metamodell vs instance modell

→ Többszintű absztrakció, egymás példányosításai

; Precise specification of domain concepts!

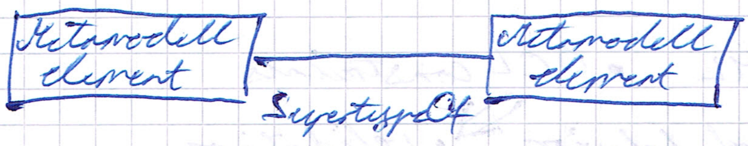
Elemei:
- alap koncepció
- attribútumok
- kapcsolatok
- absztrakció / finomítás
- aggregáció (kapcsolat számossága)
- „derived features"



direct type
↳ no other type lower in the hierarchy

indirect type
↳ any superclass of the direct type

It's not transitive!



It's transitive.

Multiple inheritance vs Multiple classification

„supertypeOf" — „instanceOf"

Célja: merge functions — Egy model elem több metamodellből örököl

Type conformance of references

Containment hierarchy

Multiplicity restrictions

Derived features

Enumeration

Bad Design / Smell

Use generalization instead of enumeration! E.g. States

- arrays in attributes
- explicit lists

DSM = Domain Specific Modeling

(5) MDA: CIM vs PIM vs PSM, role and benefit of PIM-PSM map

01 ↳ ~~Modelling levels, multi-level developing~~

CIM, PIM, PSM

Computation independent

- requirements and needs
- very abstract
- without any reference to implementation aspects

Platform independent

- behavior of the system
  - ↳ stored data
  - ↳ algorithms
- without technical or technological details

Platform-Specific

- define all the technological aspects in detail

Examples

CIM

- business process graph

PIM

- using e.g UML
- structure and behavior
  - data / functions
- OCL constraints ⇒ validation

PSM

- funcionality realized on certain platform

CIM → PIM → PSM → Code Transformation

Reverse Engineering (Code → PIM) and redeploy

~~Deep instantiation~~

PIM → PSM mapping

- OMG standard mappings
- needs some hand adjustments based on infrastructure decisions

Base level UML PIM of business functionality and behavior	→	Intermediate level UML PSM on selected platform

OMG = Object Management Group

03. ① DSL metamodelling, EMF

c) Concrete vs. abstract syntax

Pl. (1.) Concrete syntax: UML szekvencia diagramm (grafikus) ⟷ Abstract Syntax: UML diagram-mok listája

(2.) CS: Graphical syntax Editor → AS: Instance model hierarchy view ← Textual syntax: DSL instance

Abstract syntax	Concrete syntax
• taxonomy (rendszer/osztályozás)	• textual/visual notation (jelölés)
• elemek közti kapcsolat	• színek/formákkal is jelez különbséget elemek közt
→ Well-formedness rules	
• kötött forma	

Pl.: - kódrészlet
- automata

Textual syntax	Visual syntax
+ easy to write	+ easy to read
- difficult to read	+ safe to write (correct models)
	- difficult to write (slower)

Pl.: (3.) AS: UML hierarchy view — CS: UML Class Diagramm — CS: Textual notation

① b) OMG's MOF

M3 MOF Model
M2 UML Metamodel
M1 UML Model
M0 Application Data

EMF

M2 Ecore meta-metamodel
M1 Ecore Model (EPackage)
M0 Application Data (Resource)

VPM = Visual Precise Multilevel Metamodeling framework

03 (1) g) Operational vs denotational semantics (dynamic)

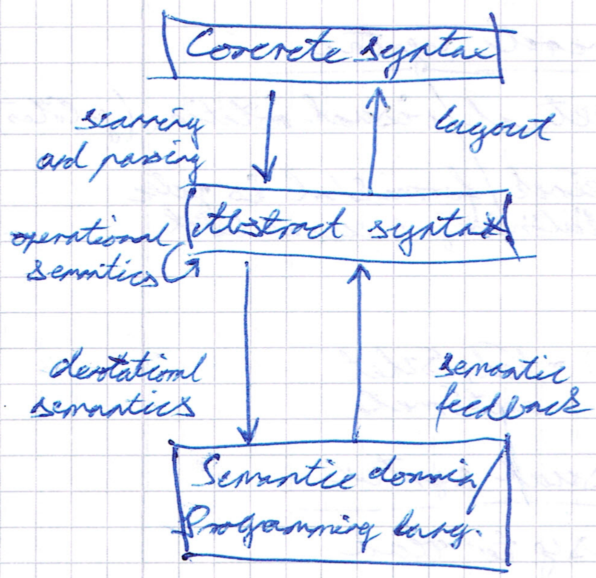
DEF Semantics is the meaning of the concepts in a language

↳ static: snapshot represents what?

↳ dynamic: changing/evolving/behaving of the model

Static

- meaning in the AS
- formal semantics (OCL)



Dynamic

i) Operational

- modeling operational behavior
- „interpreted"

ii) denotational (translational)

- „compiled"

e.g. explaining state machines as Petri-net

e.g. state machine current state dynamic feature (redirected along transition)

(1) h) Generative vs. interpreted modeling

Generative

3D shape modeling small tools → bigger tools

04a ① i) Core concepts in Ecore metamodels (EClass, EReference, EAttribute)

EClass

→ superclasses, associations, attributes

EDataType

EAttribute

→ typed attribute

EReference

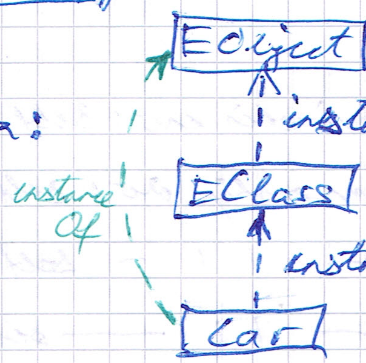
→ typed, optional inverse end

→ unidirectional relation

Complete Ecore hierarchy

EObject

↳ deep-instantiation:



EModelElement

↳ abstract class

04b ② Model Queries, transf, code generation

i) Model queries with OCL: core language concepts

OCL (Object Constraint Language)

Motivation: • special constraints in natural language

→ formal, precise, unique

Metamodels:

↳ invariants for meta model classes

↳ well-formedness

UML

↳ classes, interfaces, ...

↳ for describe guards, spec for messages and signals

↳ calculation of derived attributes

E.g.

context C inv: I	invariants
context C::op(): T pre: P post: Q	Pre- and Postconditions
context C::op(): T body: e	Query operations
context C::p: T init: e	Initial values
context C::p: T derive: e	Derived attributes
context C def: p: T = e	attribute/operation def.

! side effects are not allowed

↳ C::setAtt(arg) : T body: att = arg

Standard OCL

↳ OCL Types, OCL-Expressions, Queries → Constraints

+ Queries → Transformation

<u>Usage</u>

OCL-constraints defined on Model

evaluated on Snapshot

<u>Example</u>

context Championship inv: self.name <> ''

context Championship inv: self.minParticipants >= 0

context — " — self.maxParticipants >= 1

context — " — self.maxParticipants >= self.minParticip.

context Player inv: Player.allInstances()

→ forAll(p1, p2 | p1 <> p2 implies p1.userName <> p2.userName)

<u>Core language concepts:</u>

• predefined types / userdefined types

<u>OCL Expression</u>

• typed return value

<u>OCL Constraint</u>

• bool return value OCL Expr

~~No support on temporal~~

OE

② b) Applications of model queries

- early validation of design rules
- DSL + well-formedness constraints
  ↳ queries on model AUTOSAR

! Model size

② c) Model queries with graph patterns

Query, Result, Match, Query engine

DEF Query: Set of constraints, have to satisfy by model

DEF Result: Model element tuples

DEF Match: Bind constraint variables from pattern to model elements

Categories:

→ Imperative → declarative
↳ exhaustive search → local search (search plan)
→ ~~iterative~~ incremental

Incremental query evaluation

\+ reuse computed results

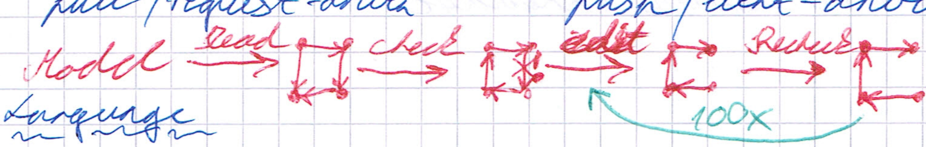
→ small usual changes & results needed all the time

→ More space to less time

- cache matches of patterns

① Batch query ② Live query

pull / request-driven push / event-driven



Language

pattern state(S:State, N) {
  State.name(S, N)
}

- find
- negfind
- or
- count find
- anonymous var
- transitive closure
- check

OE

② c, VQL

- EMF based
- generic and parameterized model queries
- reusability of patterns
- recursion
- negation
- bidirectional navigability
- immediate access to all instances of a type
- complex change detection

② d, Local search — Incremental

Local search:
+ ~~...~~ low memory usage
- slower
+ could be used on small PC

Incremental:
- memory usage
  ↳ f (model size, pattern complexity)
+ ~~...~~ fast

① d, Well-formedness constraints

By graph patterns

→ a pattern has constraints on model elements

→ defined for the „bad case"

→ Location/context: One selected element (root)

Can be used for validation (@Constraint)

along any DSM on Abstract Syntax

On abstract syntax

→ OCL

→ some declarative language

4) e) Derived features and Views

e.g. derived reference

→ calculated from other elements

→ defined declaratively as model queries

Views

It's a computed overlay. Abstract view of complex model driven by a query result.

→ query based view annotations

@Format(color=...)

@Item(item=S, label="N")

pattern redState(...) {...}

5) e) Design space exploration

→ find possible design options

Inputs:
- decl. req.
- initial design
- start state
- set of goals
- global constr.
- transformations
- define how to explore the design space

Output:
- design alternatives

Guidance of exploration:

→ hints from the designer

→ formal analysis

3, Concrete syntax and code generation

a, Textual vs. graphical syntax and editors

Textual languages	Graphical languages
+ quick and simple editing	~~more often~~ - more complicated editing
• references described as string identifiers	• references displayed visually
- inconsistent models during editing	+ models always syntactically correct
+ automatic formatting	+ automatic layout
+ content assist	+ tool list to add edges/nodes

Displaying validation errors, quick fixes

Both are supported with EMF-based technologies

For behaviour descr. ~~For~~ structural information

Combinable

E.g.: Sirius (GEF, GMF)

Sprey (Graphiti, GEF)

GEF
- low level editing framework
- MVC   JEMF

GMF
- based on GEF & EMF
- model-driven dev.

Graphiti
- high level editor · GEF & EMF
- newer

Sirius
- every diagram is a view of the model
  ↳ viewpoints
- interpreted expressions (Acceleo)

3.) c) Lexer vs. Parser, AST vs. DOM

	Lexer	Parser
Symbols	ASCII characters (atomic)	Tokens (terminal symbols, atomic)
	Try to match with the gramer	
Under-standing	regular grammar	context-free grammar
	finite state automata	can handle nested structures FSA with stack
	assign semantics to the language pieces	
Respons.	classifying lexemes e.g. {*, ==, <=, 1} → operators [1-9]+ → numbers	classifying string / sentences e.g. [id] [number] [id]
Output	produce tokens	builds parse tree, AST

08 3.) ad hoc, dedicated, template based code generators

ad hoc

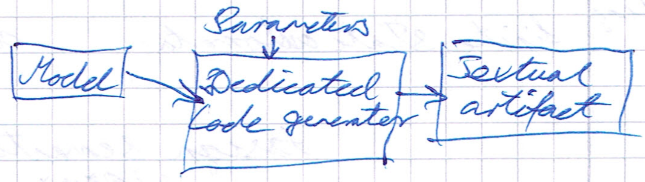
- designed for the specific problem

+ best performance
+ quick
- dirty
- long development
- hard maintainability
- ∅ reusability

Dedicated

- based on framework

+ faster development
- slower performance
+ better reusability



Template based

+ fastest development.
+ highest reusability
+ fast changing environments
+ complex changes

08 ③ h) Direct source code generation vs. AST generation

Direct source code gen.	AST
simple structure	represents program structure
low complexity	can be very complex
fast development	slower development
single pass	non-linear generation process
problematic formatting	support for M2C
probl. M2C synch.	incremental output gen.
	"pretty formatting"
	Model2Text sync.

③ i) Model ~~to~~ Code synchronization, manually written parts

In case of output text is changed!

Only AST based approach

Requires: • traceability • change localization

• model compare

Incremental model building for better performance

Manually written vs. parts in generation

Where?

Model	Template	AST	Directly to Code
• better reusability	• only for simple cases	• markings rest is generated	• Java => no support (generalization)
• more complex			• C# partial classes

előadás

2. Model queries

Deklaratív nyelvek előnyei:

1. DSL (könnyű olvasni, írni, magasszintű)
2. Deklaratív (végrehajtás nem kötött)
3. Platformfüggetlen

OCL: 1. ~ 2. ~ (inkább nem) 3. OK

Viatra: 1. OK 2. OK 3. X (Eclipse, JAVA)

2. 3. Graph transformation rules (structure + core semantics)

09 ~~30-36~~ 20-25

LHS, RHS, NAC
- deklarative, formal
- rule based

1. Pattern matching
2. NAC check
3. Non-determ. selection
4. Deletion
5. Creation

Typical problems
↳ Once the rule is applied we need to prevent an infinite cycle.

2. 4. Causal dependence vs. conflicts in gt

09 ~~39-42~~ 32-36

Conflict
- → two rule LHS wants to modify a model element
- → not parallel independent (e.g. deletion)
  - ↳ More analysation:
    - → sequential independence (can be swapped)
    - → causally dependent (not serial independent)

② g) Model T (M2T / M2M), Model T chains

→ traceability links (end to end)
(REQ ⇔ Model ⇔ Code)

<u>Chaining</u>

↳ Goal: „divide and conquer"

- intermediate model {.
- one or more source model (Statechart, Class, Sequence, ...)

② h) Incremental model transformations

09
44-46

<u>Forward</u>

~~bidirectional transf.~~

① first transformation
② source model changes
③ apply changes to target model

<u>Backward</u>

① – " –
② target model changes
③ apply to source

<u>Needed</u>

- bidirectional transformations

⇒ change driven

② i) Levels of incrementality

09
48

① Batch transformation
↳ re-execute from scratch for all source models

② Dirty incrementality
↳ re-execute from scratch only for changed models (+ large step + can be slow)

③ Incrementality by Traceability + small-step + better perf.
↳ detect missing trace links
re-execute only for untraced

④ Event driven

② i) 4. Event-driven (Viatra)

→ detect new activations
fire rule activations

+ refined context
+ chaining

− language restr.

② j) Reactive transformations

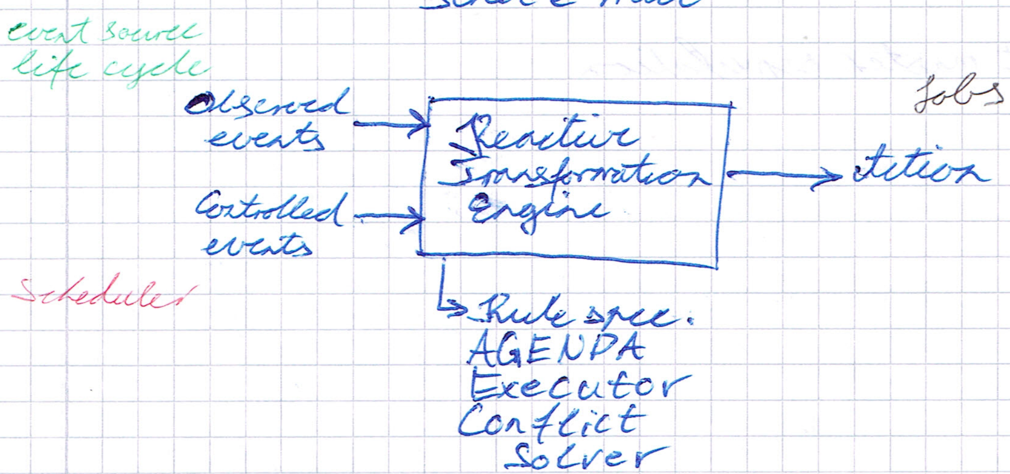
09

64–66

Changed
↳ model modified
match appeared
event sequence identified

When to react?
↳ button pushed
consistent state reached
transaction comitted

What to act?
↳ modify model
add error marker
update view
send e-mail



④ e) Back-annotation

[Model Based Analysis

Target → Source between Traces

??? correctness and minimality

⑤ d) Simulations, functional mock-up

Efficient model integration and simulation

↳ openSource tool for import, connection and efficient co-simulation of functional Mock-Up Units

FMI – Functional Mock-Up Interface

FMI master simulation

④ Model Management, advanced modeling topics

13 a) Standard model serialization in XMI

→ need for exchange models

XMI (XML Metadata Interchange)

↳ OMG standard for UML and MOF

→ also Diagram Definition to exchange graphical layout

• Scalability issue (stored in plain files)

↳ CDO (Connected Data Objects) Model Repository

b) Model comparison/model differencing, model merge

Model Comparison

Application: • model versioning

Difference model ① common elements identification

② matched elements are searched for differences (atomic operation: - add - delete - update - move

↳ EMF compare

↳ SiDiff

↳ Epsilon Comparison Language

Model Versioning, merge

Version Control System

↳ text-based

↳ inconsistent merge

⇑

Dedicated model-based VCSs are needed

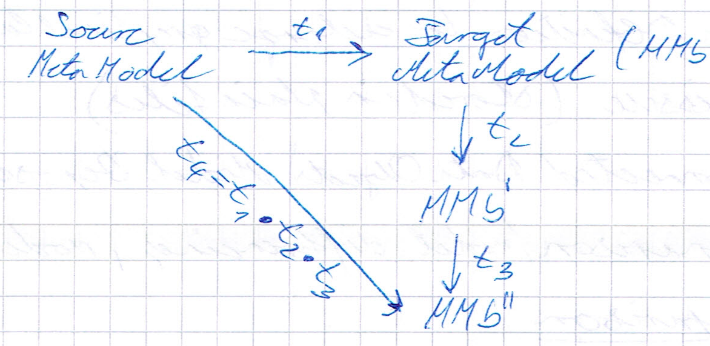
E.g. EMF Store
AMOR

④ c, Evolution: model, metamodell, transformation

13 Model - Metamodell

Changes: · non-breaking
· breaking and resolvable
· breaking and unresolvable

Metamodel - Transformation



d, ~~Mode~~ Megamodels, global model management

Megamodels: all the models, configuration files, relationships, artifacts

Metamodel of a megamodel

Own DSL to write model management scripts

f, Offline vs. online collaborative modeling

Modeling = team activity

Offline
↳ VCSs

Online
↳ several users updating the same model at the same time
↳ Short transactions model
↳ changes propagated to everybody immediately
↳ lightweight conflict management

Eg. EMF Collab
Sync Eclipse - CGMF
Dawn