Compositional Model Based Software Development

Prof. Dr. Bernhard Rumpe
Software Engineering
RWTH Aachen

http://www.se-rwth.de/

Our Working Groups and Topics

SOFTWARE ENGINEERING

Automotive / Robotics

- Autonomous driving
- Functional architecture
- Variability & product lines
- Requirements engineering
- Simulation
- Robotics

Energy

- Modeling of facilities and buildings
- Formal planning of functions
- Data management
- Automated analyses
- Quality assurance
- Monitoring

Cloud Services

- Service platforms
- Migration into the cloud
- Evolution of services
- Internet of Services
- Internet of Things

Model-based Software Development

- Tool development
- Tool-Framework MontiCore
- UML, SysML, Architecture DL
- Domain-specific languages (DSL)
- Generation, synthesis
- Testing, Analysis, verification
- Software architecture, evolution
- Agile methods

Generative Software Engineering

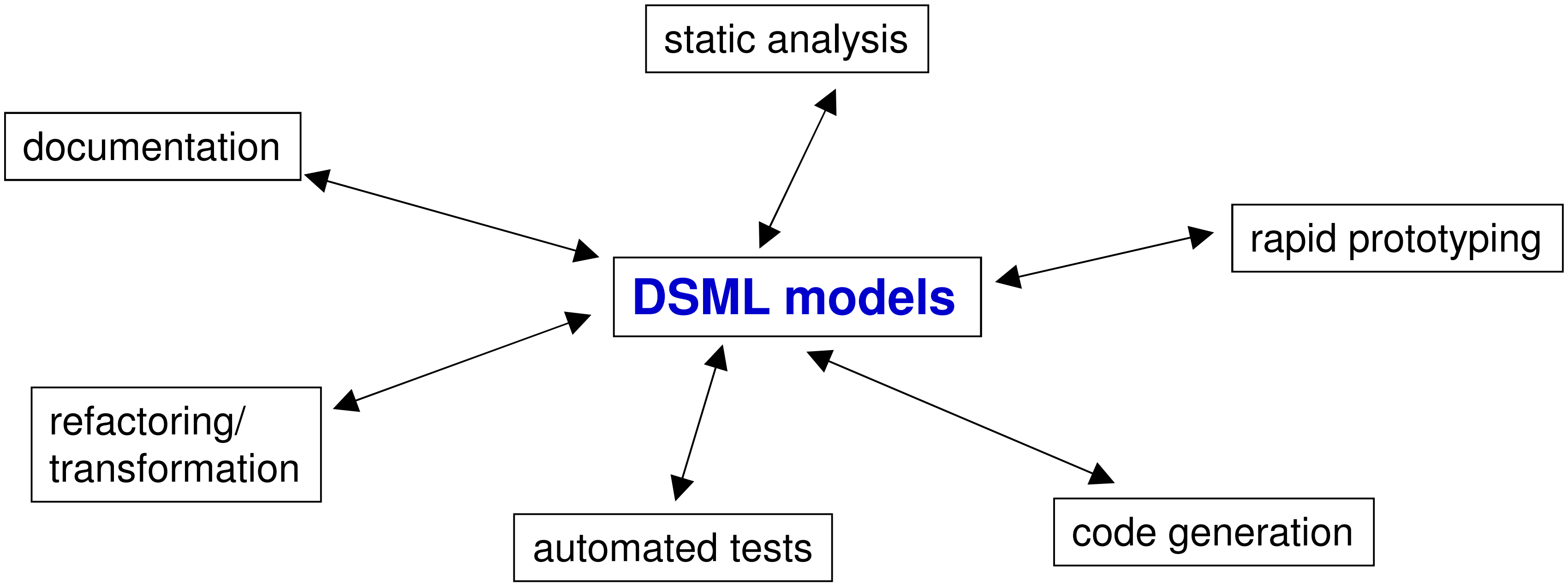
- Generative software engineering (GSE) is a

> • Method that uses generators to efficiently generate software systems or parts of software systems from models written in UML or a DSL in order to increase quality and decrease development time.

- If DSLs are used, domain experts can model, understand, validate, and optimize the software system directly.
- UML models or DSLs are used to model certain aspects of a software system in an intuitive and concise manner.
- Of-the-shelf or hand-made generators process the models to generate production and test code.

DSL-driven Development

- Domain Specific Modeling Languages (DSML) as a central notation in the development process



- DSMLs serve as central notation for development of software
- a DSML can be programming, test, or modeling language

Core Elements of an Agile Modeling Method

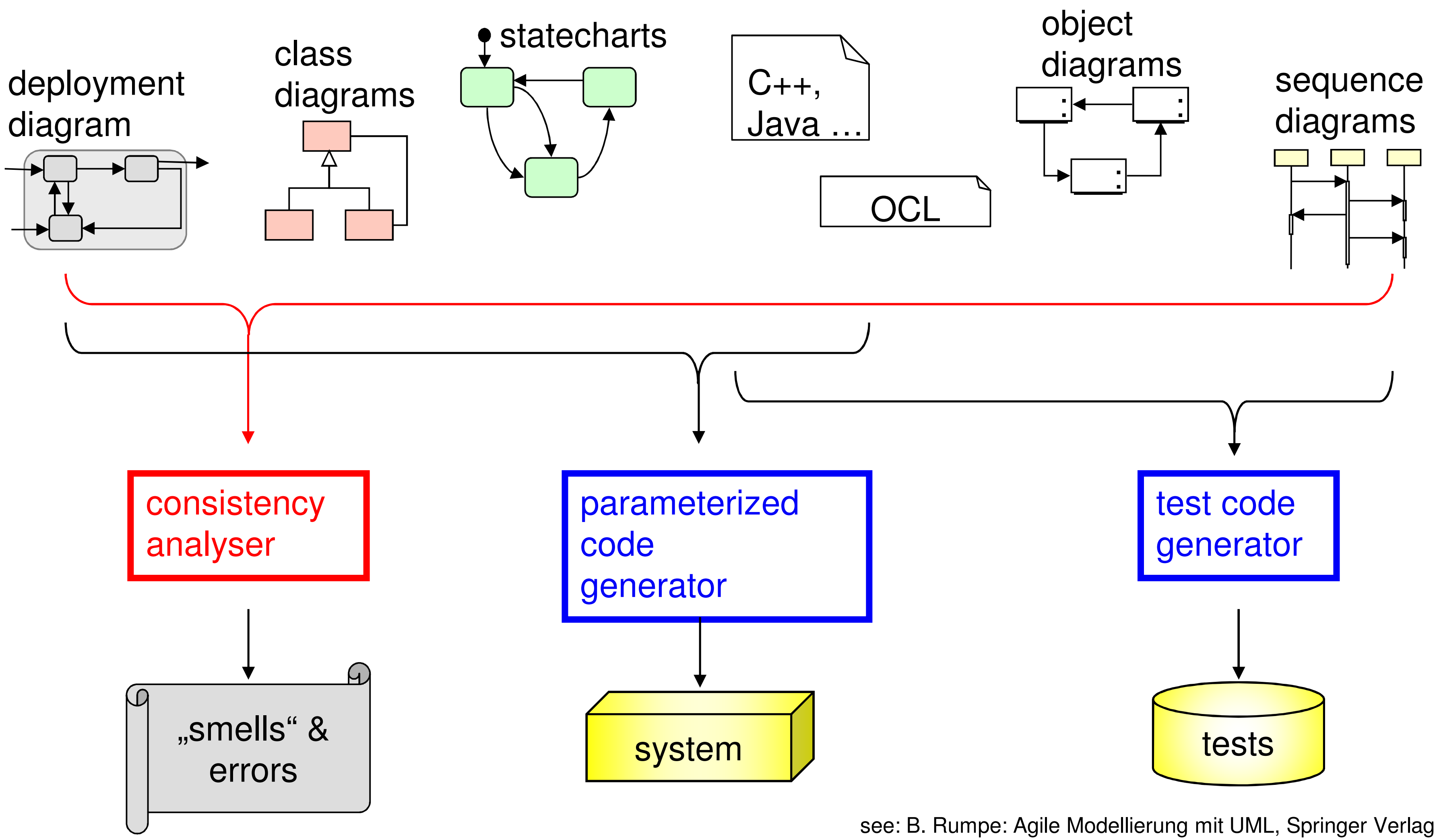
- Incremental modeling
- Modeling tests
- Automatic analysis: Types, dataflow, control flow, ...
- Code generation for system and tests from compact models

- Small increments
- Intensive simulation with customer participation for feedback

- Refactoring for incremental extension and optimization
- Common ownership of models
- ...

⇨ This approach uses elements of agile methods based on the UML notation

Constructive use of Models for Coding and Testing: Usage of UML-Diagrams

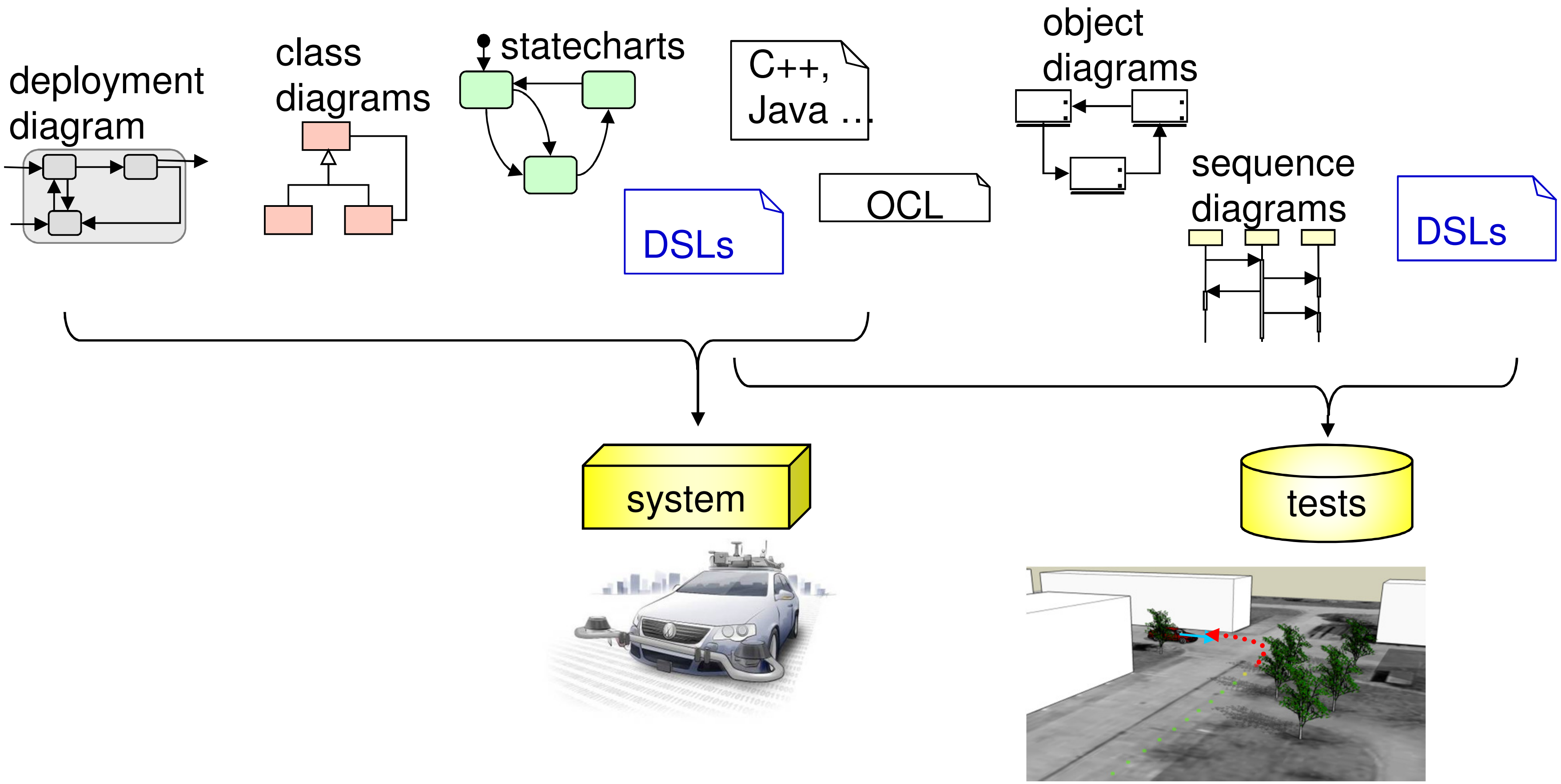


see: B. Rumpe: Agile Modellierung mit UML, Springer Verlag 2011

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

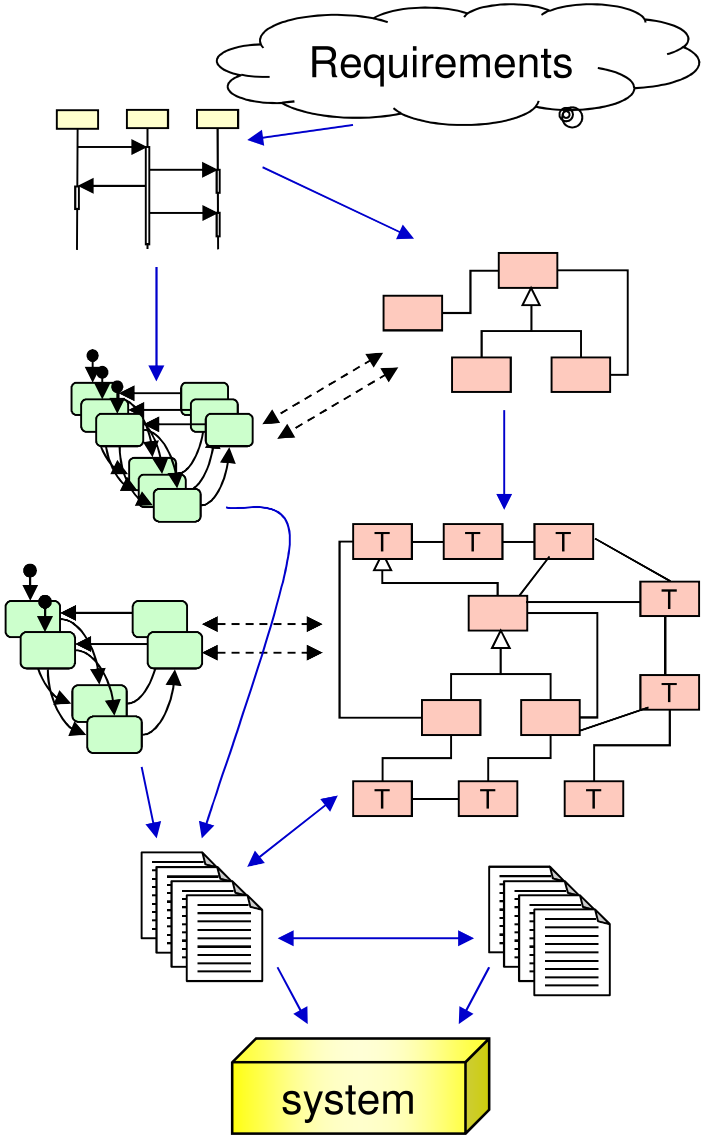
Model-based Simulation for SE

- Test-Infrastructure needs simulation of its context:
  - context can be: geographical, sociological, etc.
- Simulation helps to understand complexity



Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

View on Model Driven Architecture (MDA)



use cases and scenarios:
sequence diagrams describe users viewpoint

application classes define data structures

state machines describe
states and behavior

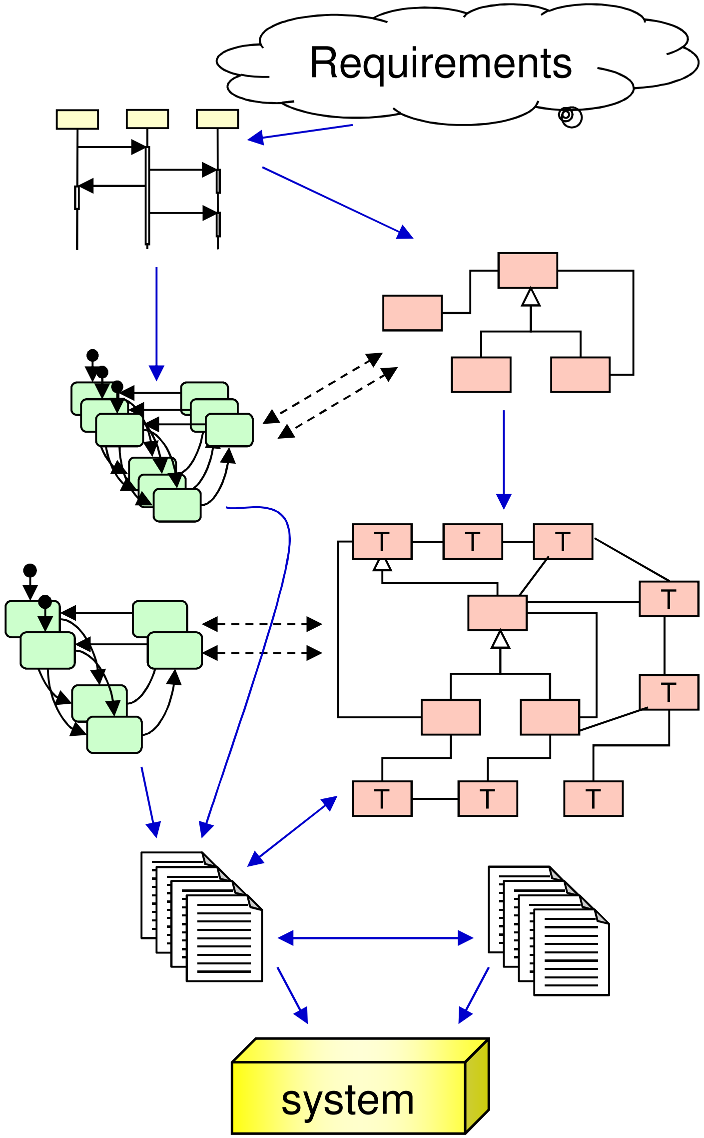
technical class diagram
adaptation, extension, technical design
+ behavior for technical classes

code generation +
integration with manually written code

complete and running system

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Problems of Model Driven Architecture



- No reuse

- Tool chain too deep
- No efficient tools
- Tracing problems
- Evolution is awkward

- Lot of information missing, e.g.,
  - design rationale
  - non-functional reqs.

- "Agile" development is not possible

- SE-Models are not integrated with other Engineering Models (spatial, biological, ...)

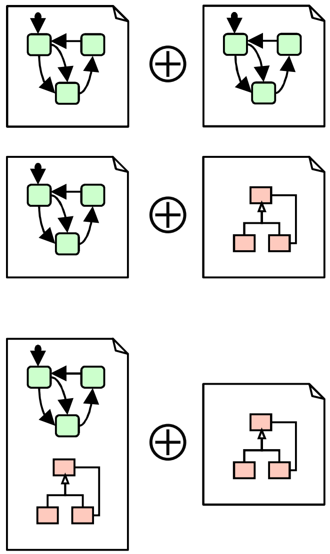
Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Model composition helps…

- Modularity and composition are essential for:
  - distributed development
  - reuse from libraries
  - Efficient tools (generation, analysis)

- The principle: independently developed artifacts A, B
  with explicit interface S

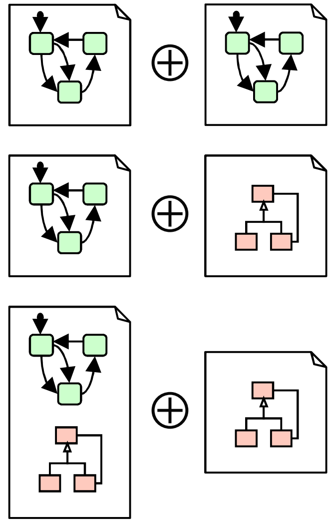
- composition:
  $C = A \oplus B$
  connects A with B at interface S
  and encapsulates internals



- The principle is well known
  - e.g. classes in object orientation

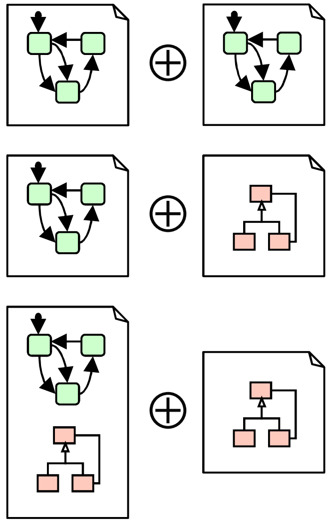
- But: How does composition of models look like?

Model composition



- Dimensions of composition :
  - Syntactic: How does A ⊕ B look like?
  - Semantical: What does A ⊕ B mean?
  - Methodical: How to develop A as well as B?
  - Organisational: Can we develop A and B in parallel?
  - Technical: Can I compile incrementally & individually: means: is there a binding technique for Code(A) ⊕ Code(B)?

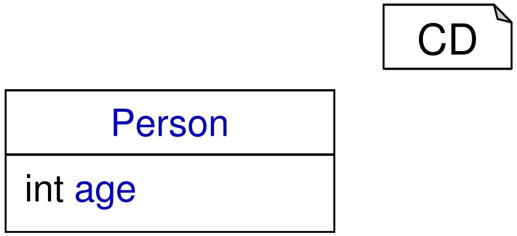
Model composition



- Model composition needs
  - a notion of interfaces for models
  - organization of models in artifacts (files)
  - incremental, individual analyses and generation
- but not really a syntactically executed composition.
- Hypothesis:
  Compositional modularity for models is essential
  for the success of model based software development.

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Example: class diagram + OCL



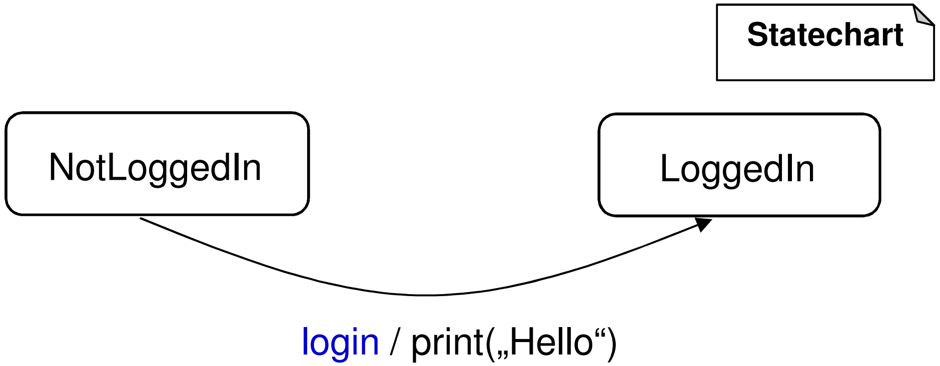
OCL

```
context Person p inv:
    p.age >= 0
```

- OCL relies on CD
- Interface is:
  - Person → Kind: class + Signature
  - age → Kind: attribute + Type
- Checking correctness early is desirable!
- OCL can also be combined with :
  - Java, Object diagrams, Statecharts, …

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Example: Statechart & Java



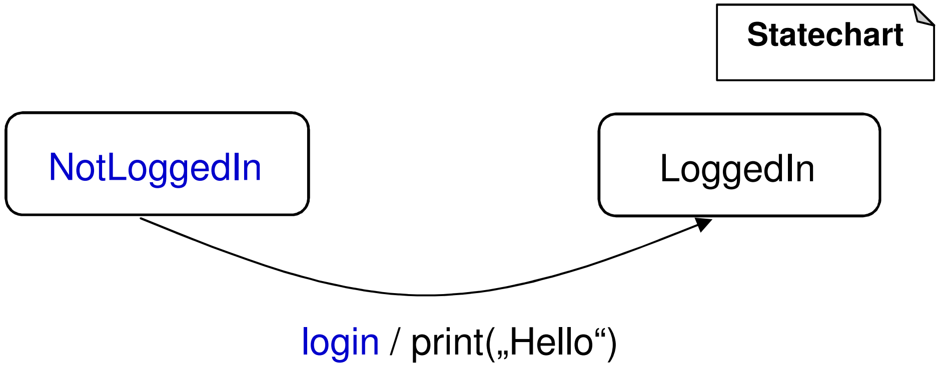
Java

```
public class Person{

 public void login() {
     …
 }
}
```

- Statechart uses Java
- Interface:
  - login → in Statechart: Kind: Message
  - in Java: Kind: Methodname + Signature „ ()“

- Languages have different interpretations of shared elements!
- → translation is necessary!

Example: Statechart & Java



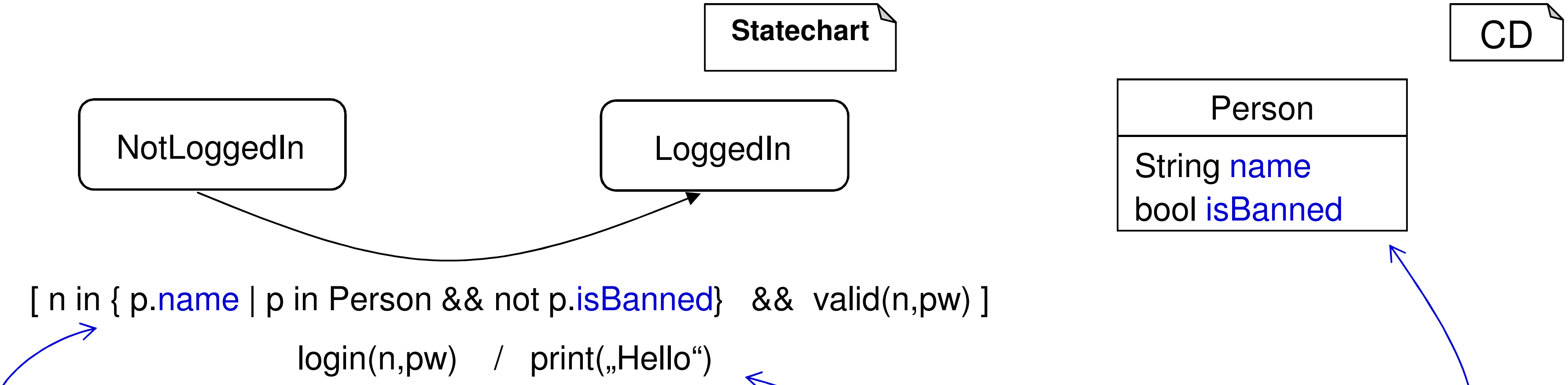
Java

```
public class Person{

 public
 handle(LoginMessage m)  { … }

 enum State { NotLoggedIn … }

}
```

- Interface:
  - login → in Statechart: Kind: message
  - in Java: Kind: class + (adapted name)
  - NotLoggedIn → in Statechart: Kind: state
  - in Java: Kind: constant
- Transformation necessary and dependent on the context

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Example: Statechart & Java



- Statechart contains foreign languages
  - OCL for the preconditions
  - Java-statements for actions
  - Import-interface from CD, Java, …
    (while the actual source should be transparent)

- Combined use of models typically also means language embedding

Interfaces/Namespaces

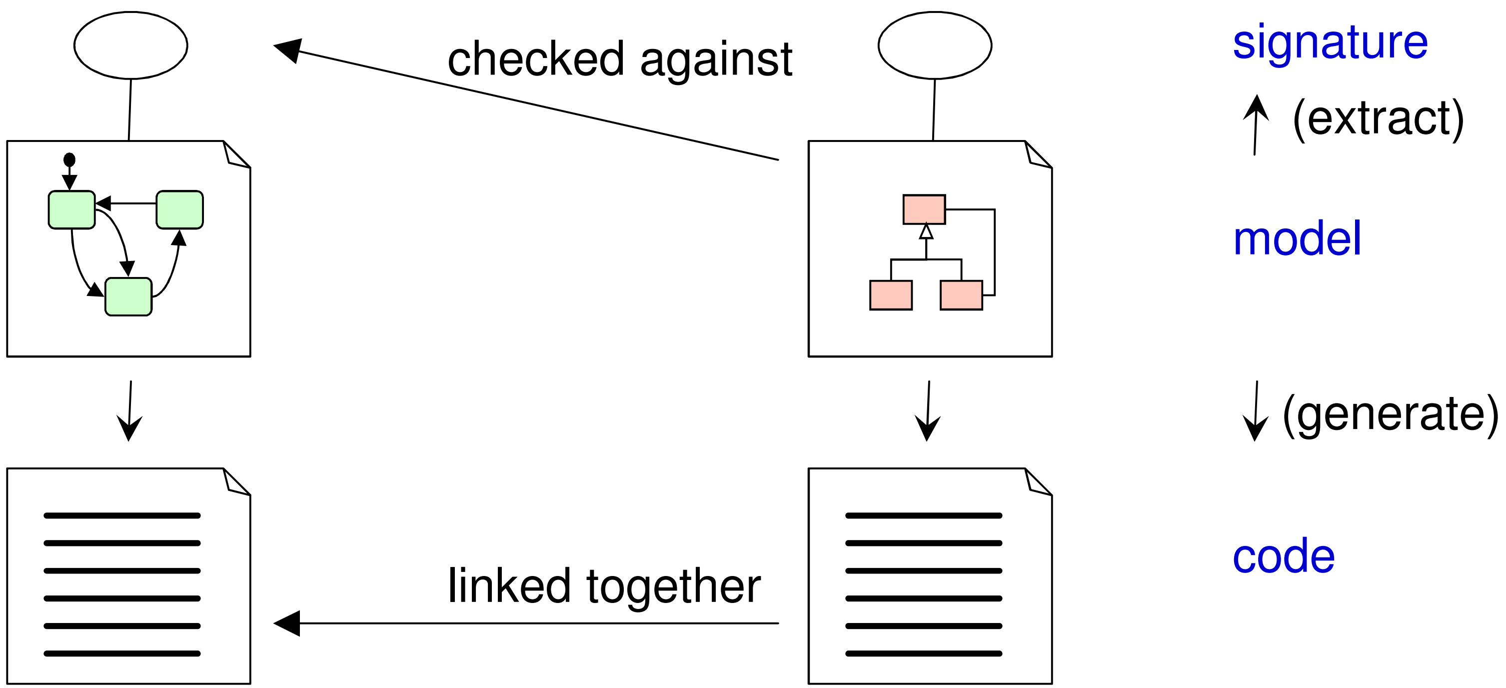
- Hypothesis:
  - Interfaces between models are defined using names
- Interfaces are imported, exported, passed-through (and local)
- There are variants of exports,
  - e.g. for subclasses, global (see e.g. Java)
- „Kinds“ of named elements:
  - state, message, method, class, activity, etc.
  - Each kind has its own “form” of interface
    - e.g. state has a name
    - e.g. method has parameters
    - e.g. class has methods + attributes, …

Interfaces/Namespaces

- Composition of heterogeneous languages:
    - E.g. Statemachines know “state”; CD’s or Java’ doesn’t
- Transformation between interfaces adapts
    - kind & signature; sometimes also name
        - E.g. mapping states to constants
- Variants of transformations are possible
    - E.g. mapping states to classes (see GOF’s state pattern)
- Special cases may be complex, e.g.
    - Messages may map to action sequences
    - Timing and computations models come into play, …

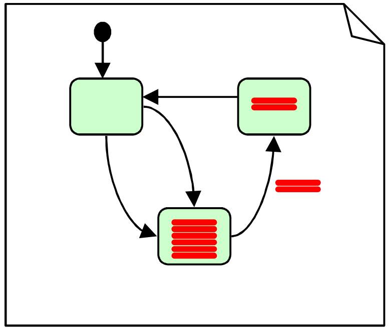
Signatures (interfaces) for models

- A signature for a model, allows us to
  - check compatibility against signatures
  - and ensure the composition of derived code to be correct.
- This allows to delay the composition: „Late Binding“



Language composition vs. model composition

- In agile DSL development we reuse sub-languages and combine languages.
- Consequence:
  - We do not only compose artifacts (files), but also sub-artifacts
  - E.g. a Statemachine embodies Java statements & OCL conditions within the same artifact. They share e.g. local variables.



- Can we apply composition here as well?
  - Can we reuse independently developed code generation within the same artifact?
- Hypothesis:
  - Model composition and language composition are pretty related.

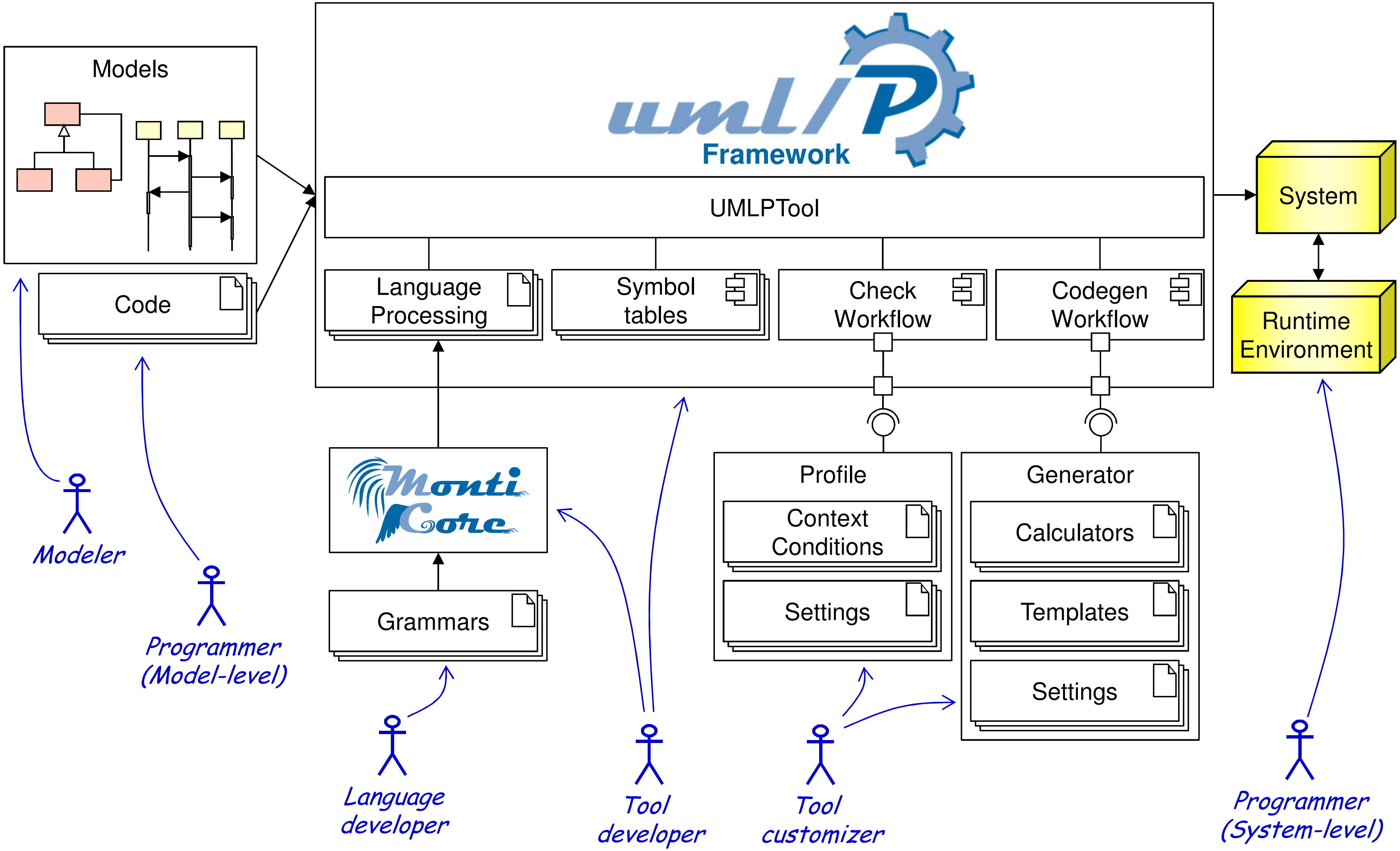
Language & tooling workbench MontiCore

- Definition of modular language fragments
- Interfaces between models/language fragments
  - Name spaces, typing (~ Java, UML)
  - „kinds“ + signatures
- Assistance for analysis
- Assistance for transformations
- Pretty printing, editors (graphical + textual)

- Composition of languages:
  - independent language development
  - composition of languages and tools
  - Language extension
  - Language inheritance (allows replacement)

- Quick definition of domain specific languages (DSLs)
  - by reusing existing languages
  - variability in syntax, context conditions, generation, semantics

UML/P language tooling @ MontiCore

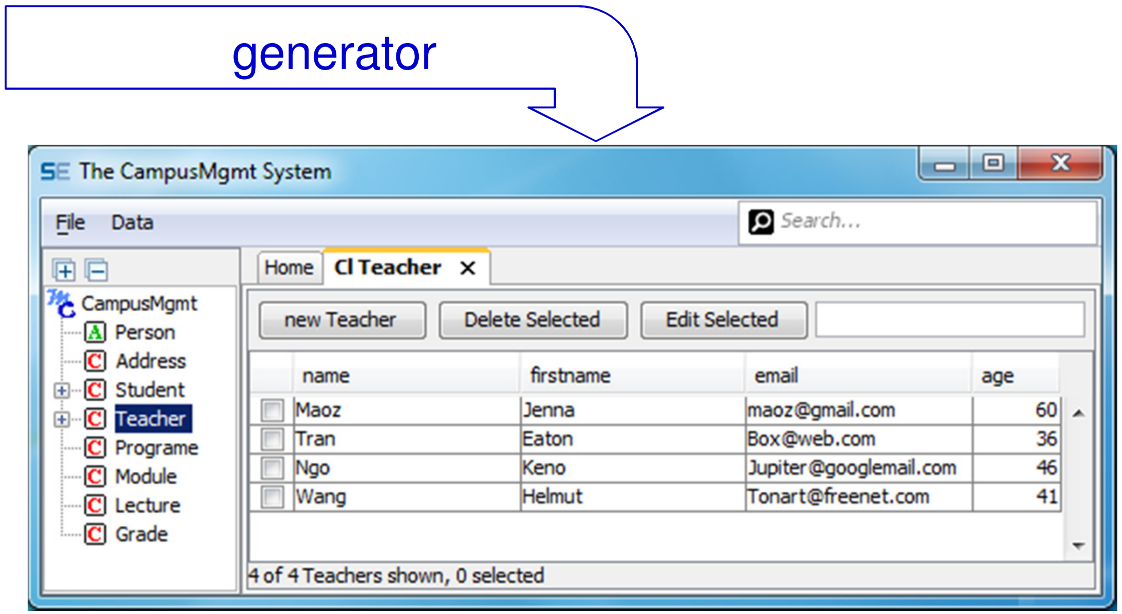


Prof. Dr. B. Rumpe
Software Engineering
Department
RWTH Aachen

Application: Data-Explorer (Dex)

- Goal: generate a complete application
  - basically from a single class diagram
- using an intelligent generator
- GWT-based GUI, search functionality, cloud-based persistence, authentication, roles, rights, …
- easy extensibility for functionality, GUI, etc.

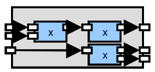
```
classdiagram CampusMgmt {                          CD

  abstract class Person {
    + String name;
    + String firstname;
    + String email;
    + int age;
  }

  class Teacher extends Person;

  association Person -> Address [*];

  // ...
}
```



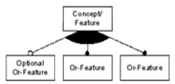
Prof. Dr. B. Rumpe
Software Engineering
Department
RWTH Aachen

MontiCore: Selected languages

- MontiCore
  - Bootstrapping
- UML
  - Class diagrams
  - Object diagrams
  - Statecharts
  - Activity diagrams
  - Sequence diagrams
  - OCL
- MontiArc



  - Architectural models / ADL, function nets
  - + automata + Java + views
- Java
  - Java 5.0 grammar
- C++
  - Ansi-C++ grammar
- MontiCore transformations
  - Pattern matching
  - Extended by Java
- FeatureDSL



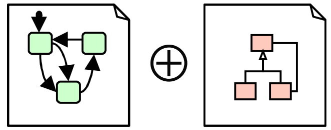
  - Feature diagram & config.
- AutosarDSL
  - Components, deployment, interfaces
- Flight control: constraint language
- Building facility specification
- Curriculum

- Cloud Service Configurator
  - Management of Services

Prof. Dr. B. Rumpe
Software Engineering
Department
RWTH Aachen

Status of compositional MBSE

Model- and language composition is key to successful use of MBSE



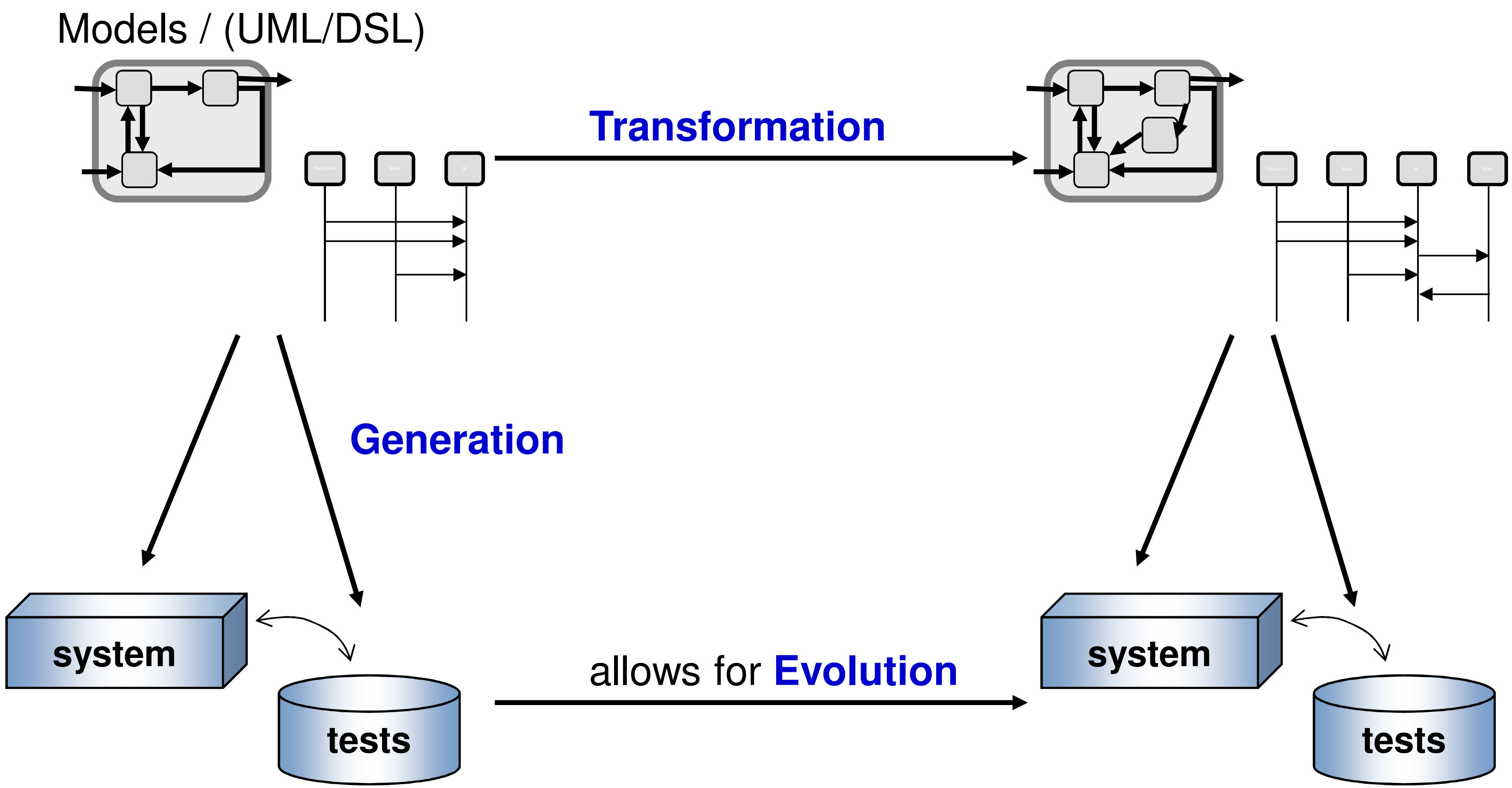
▪ Model composition	++
▪ Language composition	++
▪ Variability for languages & usages	++
▪ Modular language definition	++
▪ Modular analysis	++
▪ Modular generation	open
▪ Modular verification	open
▪ Tooling	+
▪ Model evolution / transformation	(+)
▪ Language library	(+)
▪ Transfer to industry	(+/-)

Thanks for listening.

Questions?

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Transformations in MBSE

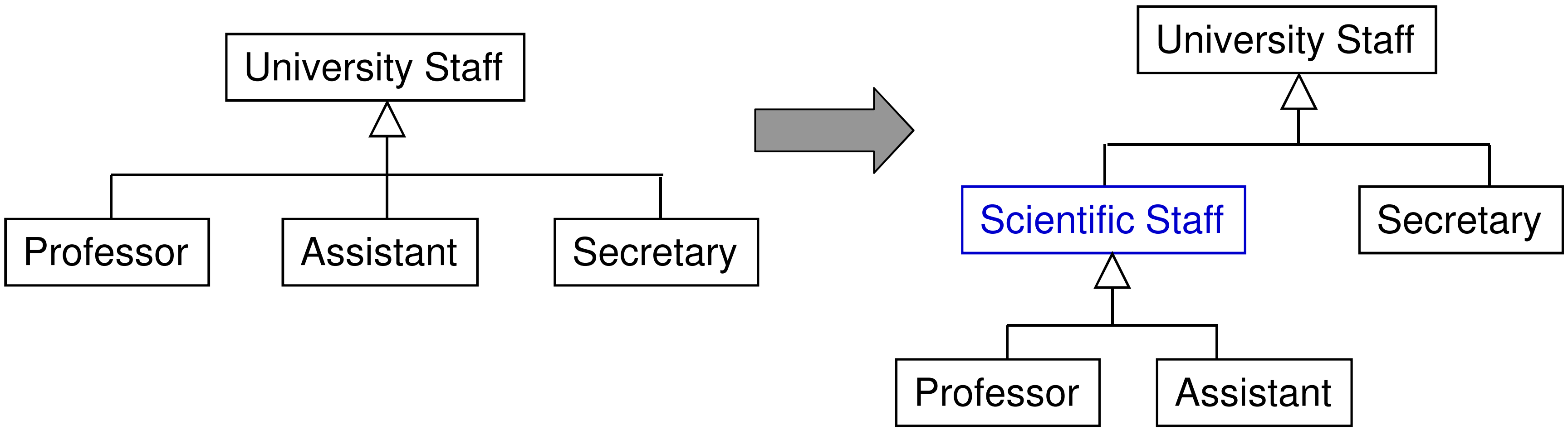


- Repeatable generation is necessary
- (no one-shots, no manual adaptation of generated code)

Prof. Dr. B. Rumpe
Software Engineering Department
RWTH Aachen

Transformations

- ... strongly depend on the language
  - primitive transformations (add, remove, rename) don‘t help
  - Semantically relevant transformations needed
- Examples:
  - Split a state in Statecharts
  - Extend an interfaces in an architecture
  - Move an attribute between classes
  - Introduce new class in hierarchy



Prof. Dr. B. Rumpe
Software Engineering
Department
RWTH Aachen

Transformations using concrete Textual Syntax

Given a language L we derive (:

- transformation language for T(L)
- transformation engine for T(L)

T(L) understandable for modelers

- It uses concrete syntax!

Explaning the transformation rule:

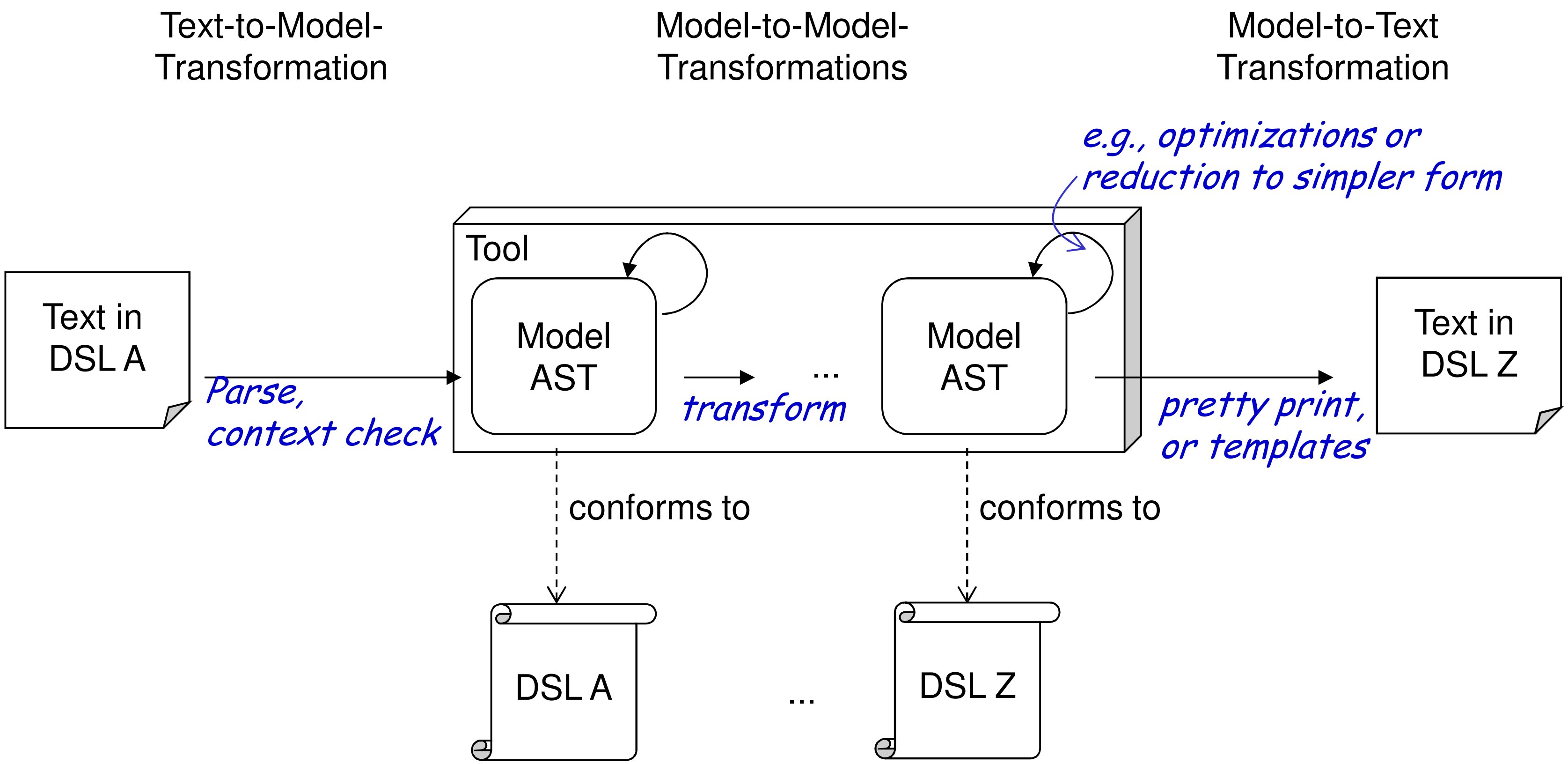
- pattern to be matched
- and replacement parts: [[old :- new]] (where “old” is matched and then replaced by “new”)
- $outer, $inner are matching variables (here bound to state names, but could be any nonterminal)
- Control language for composing transformations
- Negative patterns allowed
- Java for calculations embedded
- …

```
statechart S {
  state A;
  state B { state Sub1, Sub2 <<initial>>;
            state Sub3;                          }
  A -> B;                // transition
}
```

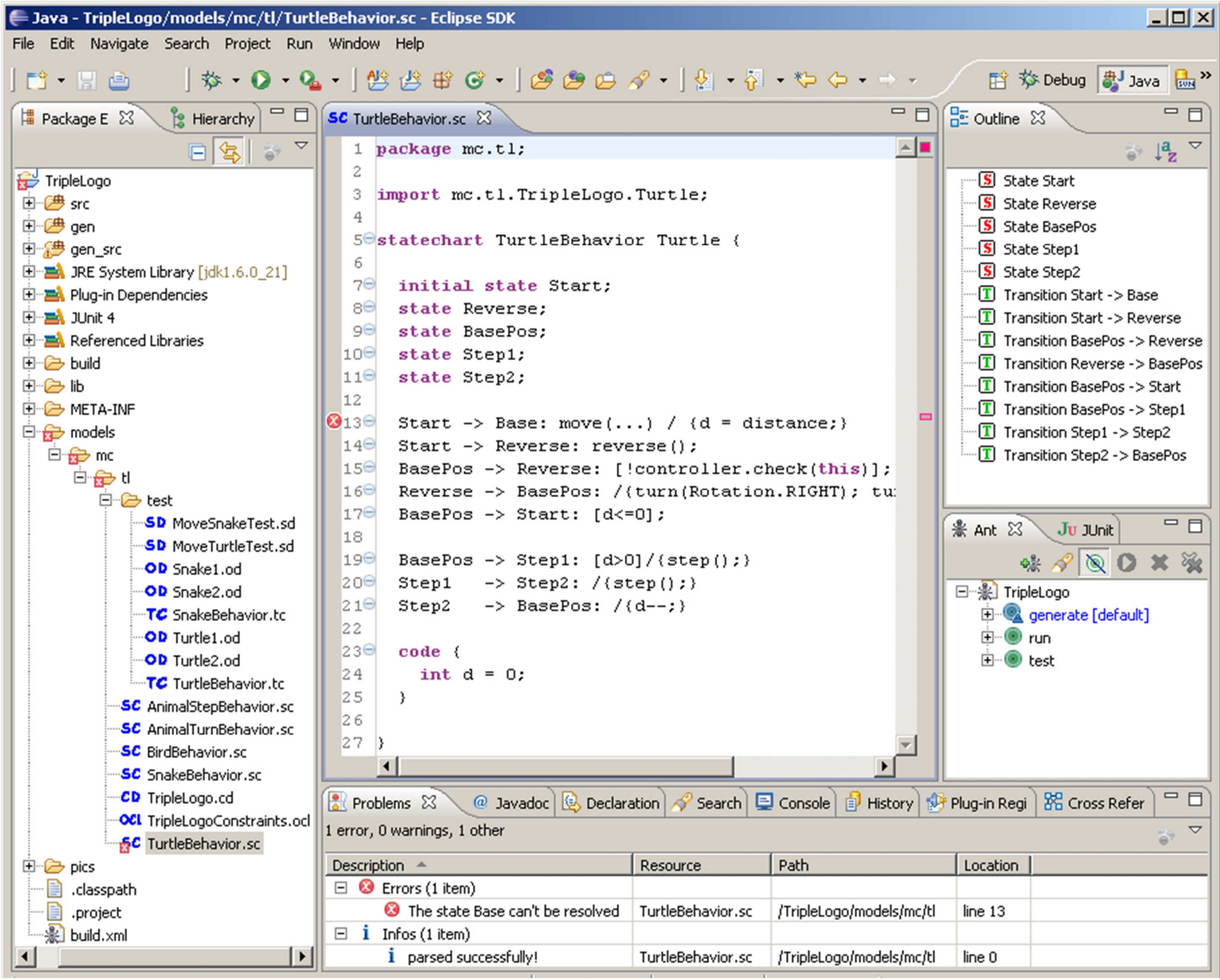
```
// transformation rule:
// redirect transitions to initial substates

state $outer {
  state $inner [[ << initial >> :- ]];
}
// transition
A -> [[ $outer :- $inner ]];
```

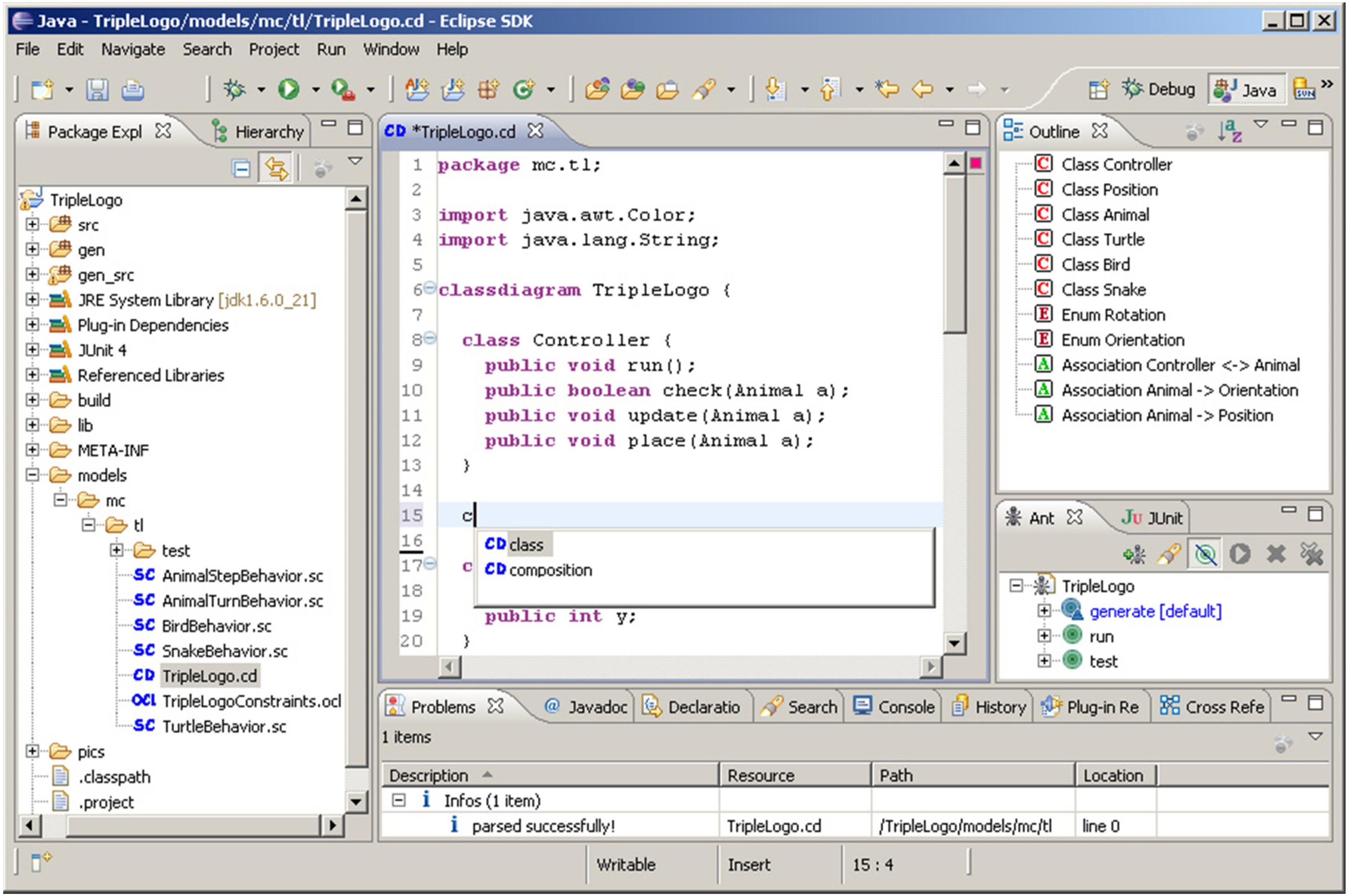
```
statechart S {
  state A;
  state B {  state Sub1, Sub2;
             state Sub3;         }
  A -> Sub1;                // transition
  A -> Sub2;                // transition}
```

Steps of Code Generation



- AST = abstract syntax tree of model


Java - TripleLogo/models/mc/tl/TurtleBehavior.sc - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Debug
Java
Package E
Hierarchy
TripleLogo
src
gen
gen_src
JRE System Library [jdk1.6.0_21]
Plug-in Dependencies
JUnit 4
Referenced Libraries
build
lib
META-INF
models
mc
tl
test
MoveSnakeTest.sd
MoveTurtleTest.sd
Snake1.od
Snake2.od
SnakeBehavior.tc
Turtle1.od
Turtle2.od
TurtleBehavior.tc
AnimalStepBehavior.sc
AnimalTurnBehavior.sc
BirdBehavior.sc
SnakeBehavior.sc
TripleLogo.cd
TripleLogoConstraints.ocl
TurtleBehavior.sc
pics
.classpath
.project
build.xml
TurtleBehavior.sc
package mc.tl;
import mc.tl.TripleLogo.Turtle;
statechart TurtleBehavior Turtle {
initial state Start;
state Reverse;
state BasePos;
state Step1;
state Step2;
Start -> Base: move(...) / {d = distance;}
Start -> Reverse: reverse();
BasePos -> Reverse: [!controller.check(this)];
Reverse -> BasePos: /{turn(Rotation.RIGHT); tu
BasePos -> Start: [d<=0];
BasePos -> Step1: [d>0]/{step();}
Step1 -> Step2: /{step();}
Step2 -> BasePos: /{d--;}
code {
int d = 0;
}
}
Outline
State Start
State Reverse
State BasePos
State Step1
State Step2
Transition Start -> Base
Transition Start -> Reverse
Transition BasePos -> Reverse
Transition Reverse -> BasePos
Transition BasePos -> Start
Transition BasePos -> Step1
Transition Step1 -> Step2
Transition Step2 -> BasePos
Ant
JUnit
TripleLogo
generate [default]
run
test
Problems
Javadoc
Declaration
Search
Console
History
Plug-in Regi
Cross Refer
1 error, 0 warnings, 1 other
Description
Resource
Path
Location
Errors (1 item)
The state Base can't be resolved
TurtleBehavior.sc
/TripleLogo/models/mc/tl
line 13
Infos (1 item)
parsed successfully!
TurtleBehavior.sc
/TripleLogo/models/mc/tl
line 0



screenshot of the editor-plugin for Eclipse with auto-completion