

# 61 Artefakt- und Modellmanagement in Technikräumen

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
<http://st.inf.tu-dresden.de>  
Version 12-1.0, 23.01.13

- 1) Modellmanagement
  - 1) Einsortige Algebren über Artefakten
  - 2) Zweisortige Algebren
- 2) Technikräume mit Modellmanagement

- ▶ Obligatorisch:
- ▶ Zusätzlich:
  - Siehe CBSE im Sommer
  - Jakob Henriksson, Florian Heidenreich, Steffen Zschaler, Jendrik Johannes, and Uwe Assmann. Extending grammars and metamodels for reuse - the reuseware approach. IET Software Journal Special Issue: Language Engineering, 2008.
  - <http://www.reuseware.org>
  - Model Management 2.0: Manipulating Richer Mappings. Philip A. Bernstein, Sergey Melnik. SIGMOD 07, ACM.

# Problem

- ▶ Wir haben viele Werkzeuge gesehen....
  - die Files, Modelle, Codedateien, Dokumente, etc. bearbeiten

Wie kann man das Management solcher Artefakte vereinheitlichen?

# 61.1 Model Management

- Model management is:
  - model composition with model algebrae
  - model slicing



# ***61.1.1 Einsortige Algebren über Modellen und anderen Artefakten***

Text-Algebren, Modell-Algebren

**Component Model**

**Composition Technique**

**Composition Language**

# Composition with Algebrae

**Component Model:  
Set as Carrier**

**Composition Technique:  
Algebra Operators**

**Composition Language:  
Functional Language,  
Lambda-Calculus**

# Einsortige Algebra über Texten

- ▶ Eine **einsortige Algebra** ist eine Menge von Operatoren über einer Trägermenge (Carrier) eines Typs (einer Sorte)
- ▶ Beispiel: Texte sind Folgen von Zeichen, in Zeilen aufgeteilt
- ▶ Die UNIX Programmers Workbench enthält eine Algebra über Texte, bestehend aus Zeilen:
  - $\text{diff} : \text{Text} \times \text{Text} \rightarrow \text{Transformation (Editiersequenz)}$
  - $\text{cmp} : \text{Text} \times \text{Text} \rightarrow \text{Boolean}$
  - $\text{patch} : \text{Text} \times \text{Editiersequenz} \rightarrow \text{Text}$
  - $\text{diff3} : \text{mine:Text} \times \text{older:Text} \times \text{yours:Text} \rightarrow \text{Editiersequenz}$
  - $\text{split} : \text{Text} \times \text{Splitzeichen} \rightarrow \text{Text}^*$
  - $\text{match} : \text{Text} \times \text{Muster} \rightarrow \text{Text}^*$
  - $\text{check-property} : \text{Text} \times \text{Muster} \rightarrow \text{Boolean}$
  - $\text{is-consistent} : \text{Text} \times \text{Text} \rightarrow \text{Boolean}$
  - $\text{format} : \text{Text} \rightarrow \text{Text}$
  - $\text{expand} : \text{Text-template} \times \text{Text}^* \rightarrow \text{Text}$



# Einsortige Algebra über Ascii-Tabellen

- ▶ Tabellen sind Folgen von Zeilen, in Spalten aufgeteilt, die durch einen Spaltentrenner (TAB , | ) getrennt werden
  - .csv-Dateien (comma separated values)
  - html-Tabellen, tex-Tabellen
- ▶ rdb enthält eine Algebra über Tabellen:
  - diff : Tabelle x Tabelle → Transformation (Editiersequenz)
  - cmp: File x File → Boolean
  - patch: Tabelle x Editiersequenz → Tabelle
  - diff3: mine:Tabelle x older:Tabelle x yours:Tabelle → Editiersequenz
  - split: Tabelle x Splitzeichen → Tabelle\*
  - match: Tabelle x Muster → Tabelle\*
  - check-property: Tabelle x Muster → Boolean
  - is-consistent: Tabelle x Tabelle → Boolean
  - join, sort, group-by...
  - format: Tabelle → Tabelle
  - expand: Tabelle-template x Tabelle\* → Tabelle

## ***61.1.2 Zweisortige Algebren über Artefakten***

Invasive Software Composition with Graybox  
Components  
... preview onto the summer  
(CBSE course)

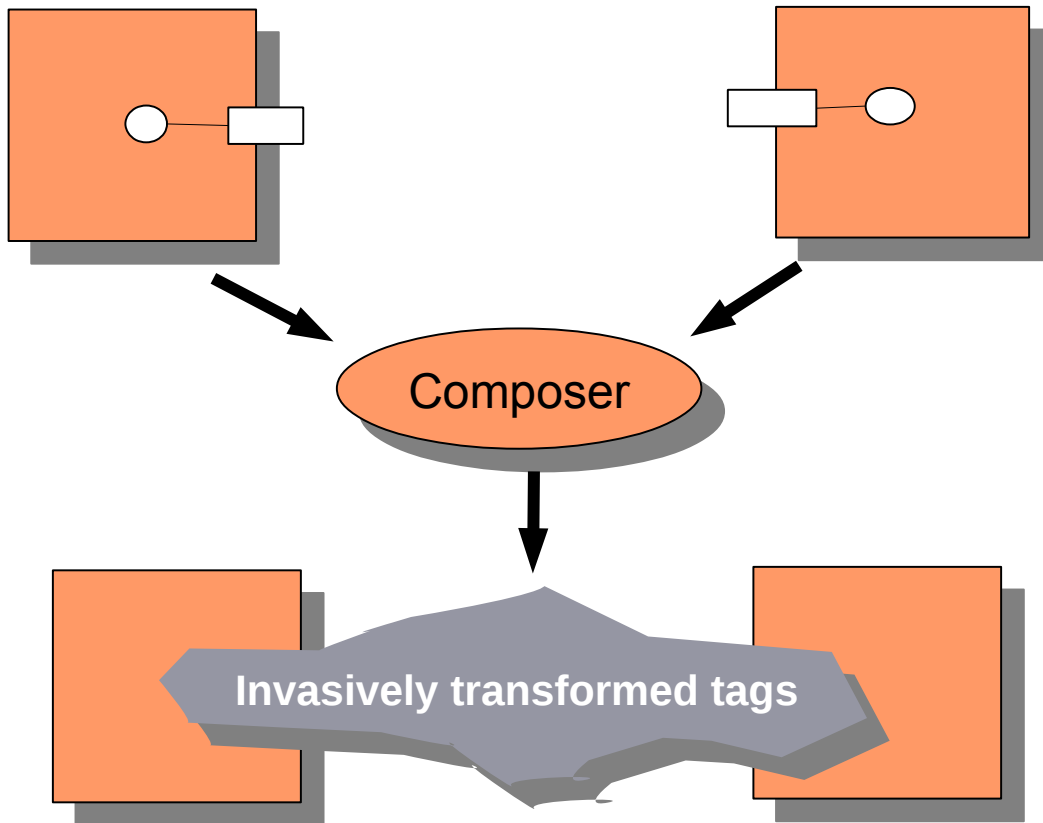
# *“Invasive” Composition with 2-Sorted Algebrae*

**Component Model:  
Fragments of a Language  
Their Hooks**

**Composition Technique:  
Hook Transformation**

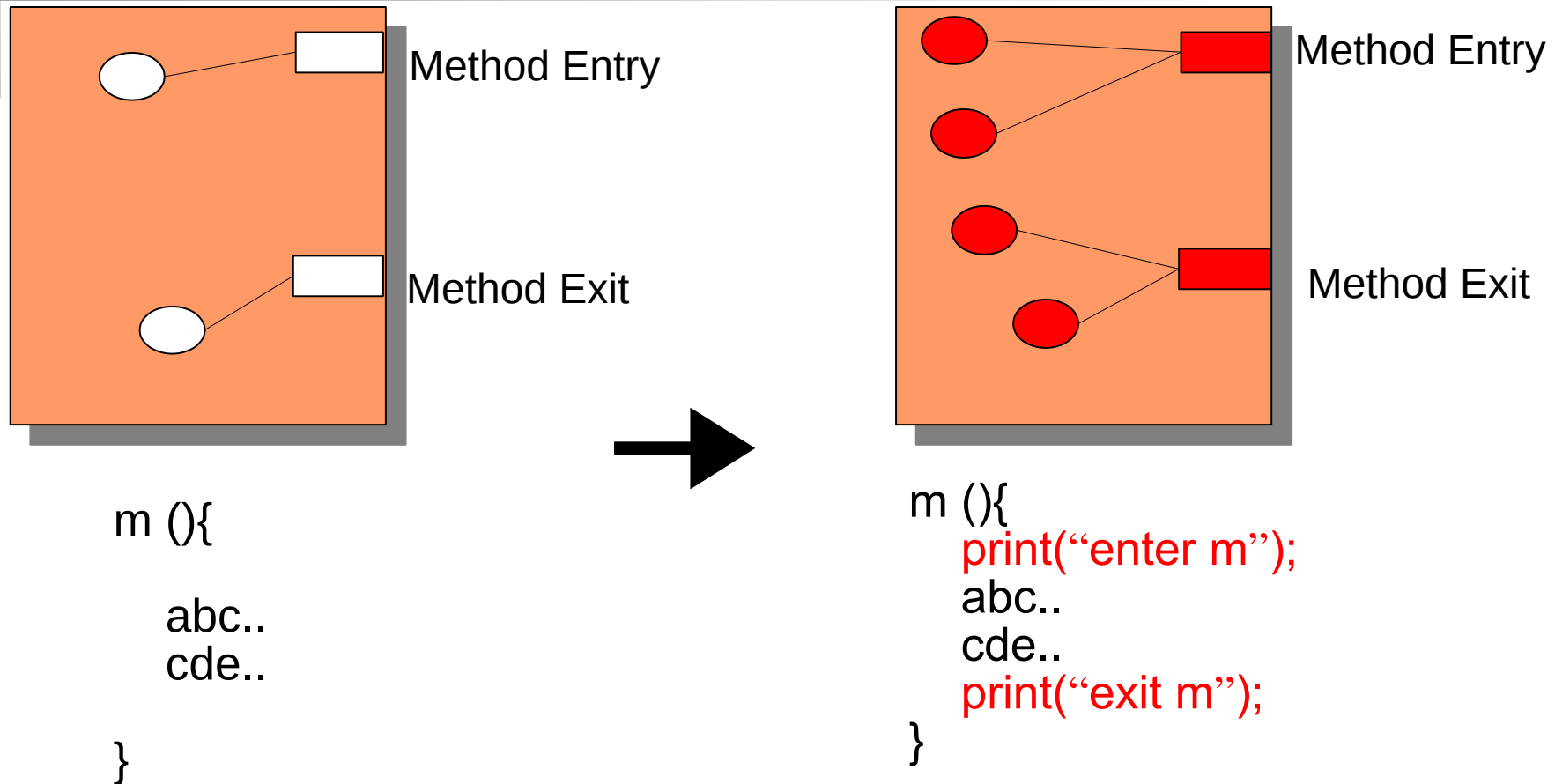
**Composition Language:  
Standard Languages**

# Invasive Composition as Hook Transformations



Invasive Composition  
**adapts** and **extends**  
**components**  
at **hooks**  
by a composition  
operator

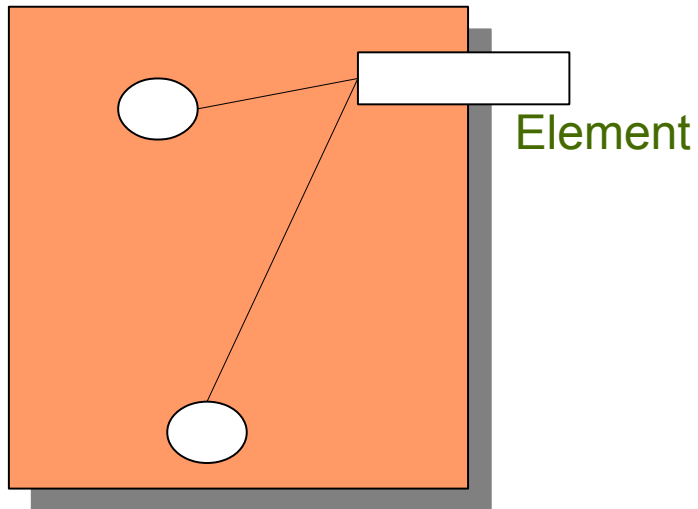
# Binding Implicit Hooks with Fragments



```
box.findHook(„MethodEntry“).extend(“print(\\”enter m\\”);”);
```

```
box.findHook(„MethodExit“).extend(“print(\\”exit m\\”);”);
```

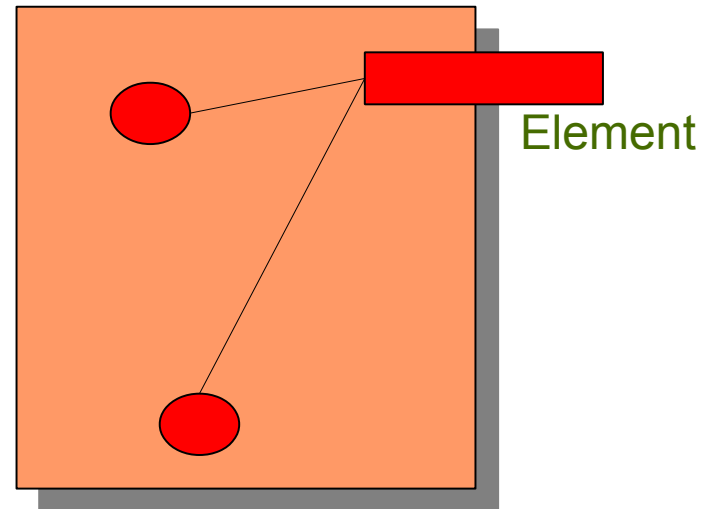
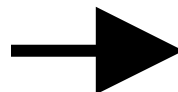
# Binding Declared Hooks with Fragments



```
List(Element) le;
```

```
....  
le.add(new Element());
```

```
...
```



```
List(Apple) le;
```

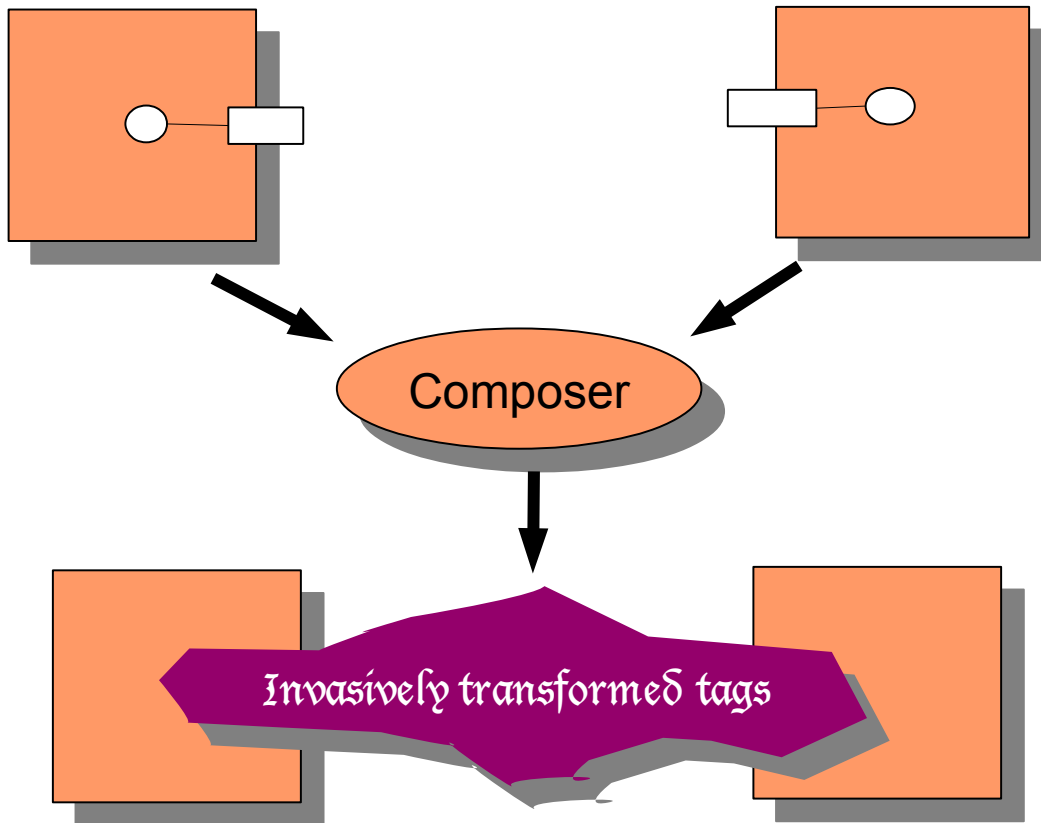
```
....  
le.add(new Apple());
```

```
...
```

```
box.findHook(„Element“).bind(“Apple”);
```

# Invasive Composition as Hook Transformations

- ▶ Invasive Composition works uniformly on
  - declared hooks
  - implicit hooks
- ▶ Allows for unification of
  - Inheritance
  - Views
  - Aspect weaving
  - Parameterization
  - Role model merging



- ▶ Invasive Softwarekomposition bildet eine zweisortige Algebra
  - Sorten: Fragmentkomponenten mit Haken (hooks)
  - Sowohl Haken als auch Komponenten können komponiert werden

## Simple composition operators

- ▶ **bind** hook (parameterize)
  - generic programming
- ▶ **rename** component, rename hook
- ▶ **remove** value from hook (unbind)
- ▶ **extend** component or hook
  - extensions
- **copy** fragment component

## Compound composition operators

- ▶ **inheritance** from component
  - object-oriented programming
- ▶ **view** of component
  - view-based programming
- ▶ **connect** hook 1 and 2
  - connector-based programming
- ▶ **distribute** component over other component
  - aspect weaving



# *61.2 Technikräume und Algebren über Artefakten*



# Technical Spaces (Technikräume)

	Grammarware (Strings)		Tableware (Tables)		Treeware (Bäume)		Graphware/Modelware				
	Strings	Text	Text-Tabelle	Relationale Algebra	XML	NF2	MOF/OMG	Eclipse	CDIF	MetaEdit +	OWL-Ware
M3	EBNF	EBNF		CWM (common warehouse model)	XSD	NF2-Sprache	MOF	Ecore	ERD	GOPPR	
M2	Grammatik einer Sprache	Grammatik mit Zeilentrennern	csv-header	Relationales Schema	XML Schema-beschreibung, z.B. xhtml	NF2-Schema	UML-CD, -SC, OCL	UML, many others	CDIF-Sprache n	UML, many others	
M1	String, Programm	Text in Zeilen	csv Datei	Relationen	XML-Dokumente	NF2-Baumrelation	Klassen, Programme	Klassen, Programme	CDIF-Modelle	Klassen, Programme	
M0					dynamische Semantik im Browser						

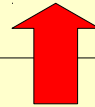
# A Technical Space

- ▶ aka *metapyramid*

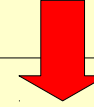
M3 metametamodel level

Modelling concepts

MOF, UML-core, OWL, AG, NS



validInstanceOf

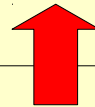


describes

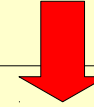
M2 metamodel level

Language descriptions

OWL, UML, CWM, ER



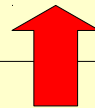
validInstanceOf



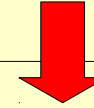
describes

M1 model level

Types, programs, models, domain ontologies



validInstanceOf



describes

M0 Object level

Software objects describing world objects

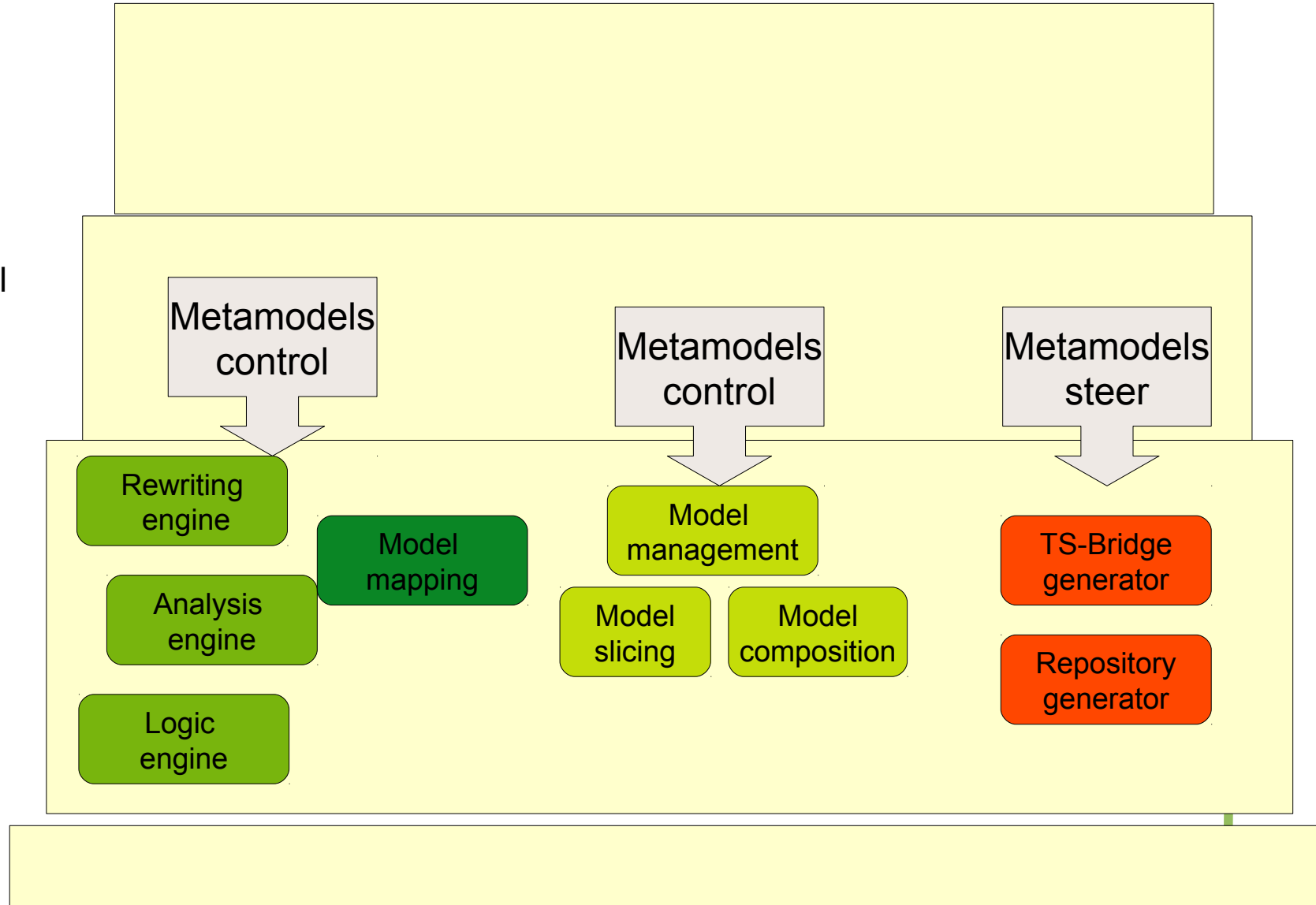
# The Generic Tools of a Technical Space (TS)

M3 metametamodel level  
Metalanguage  
Modelling concepts

M2 metamodel level  
Metamodels  
(languages)

M1 model level  
Models, Programs

M0 object level



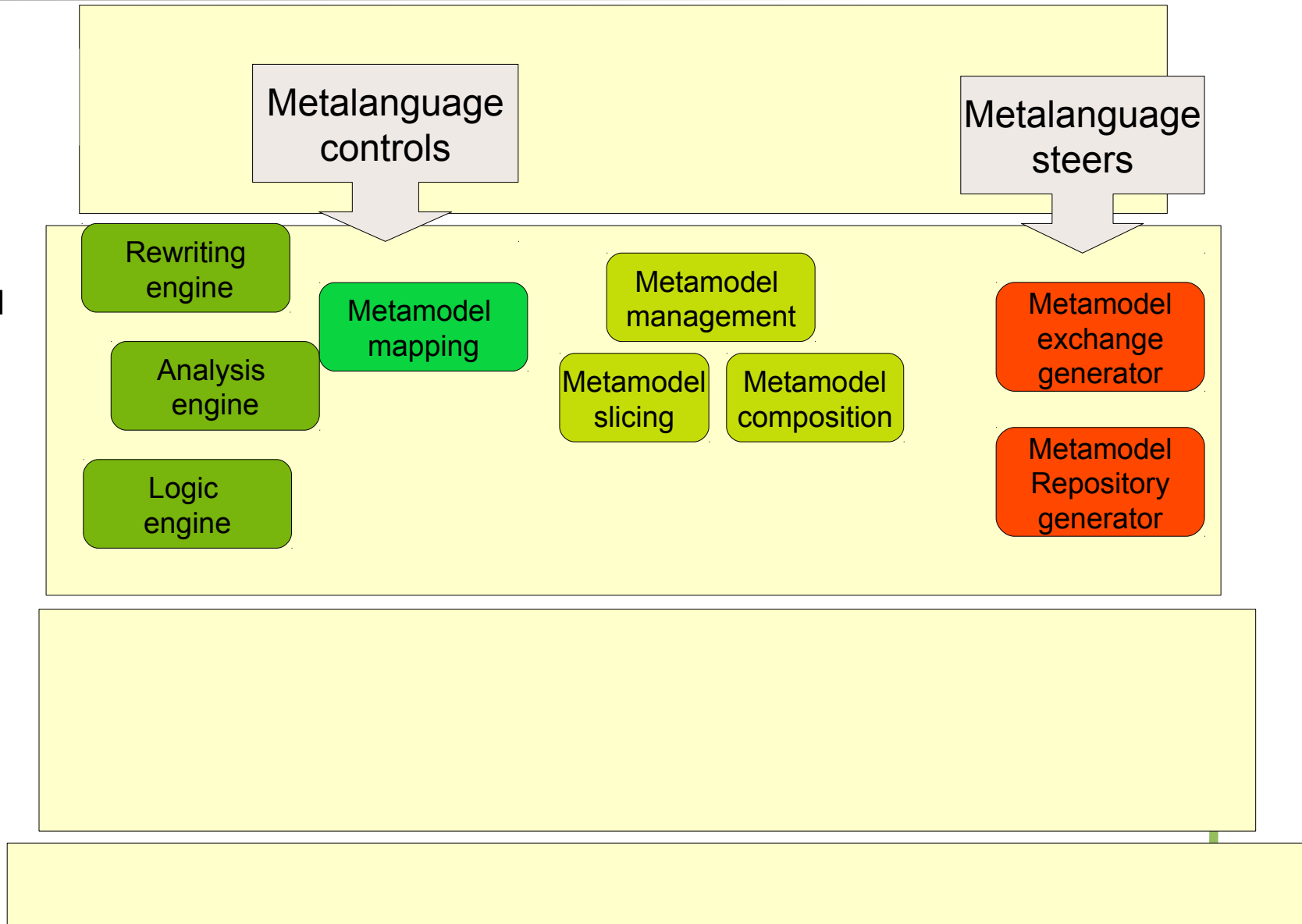
# The Generic Tools of a Technical Space (2)

M3 metametamodel level  
Metalanguage  
Modelling concepts

M2 metamodel level  
Metamodels  
(languages)

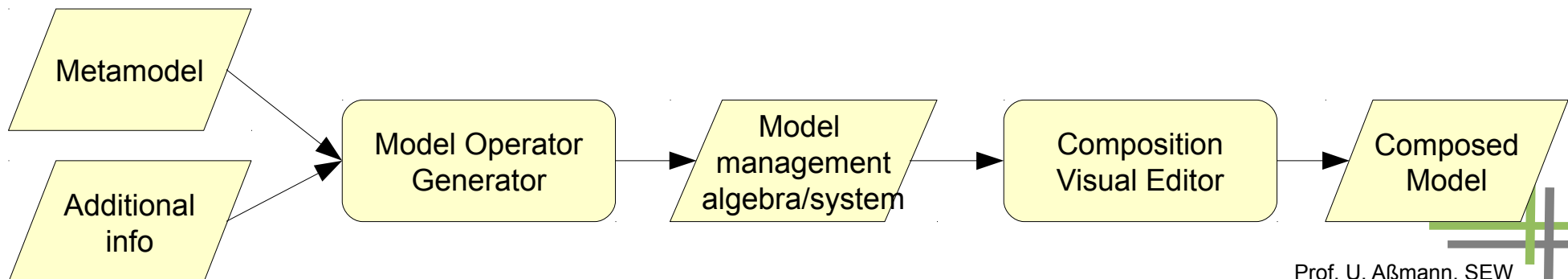
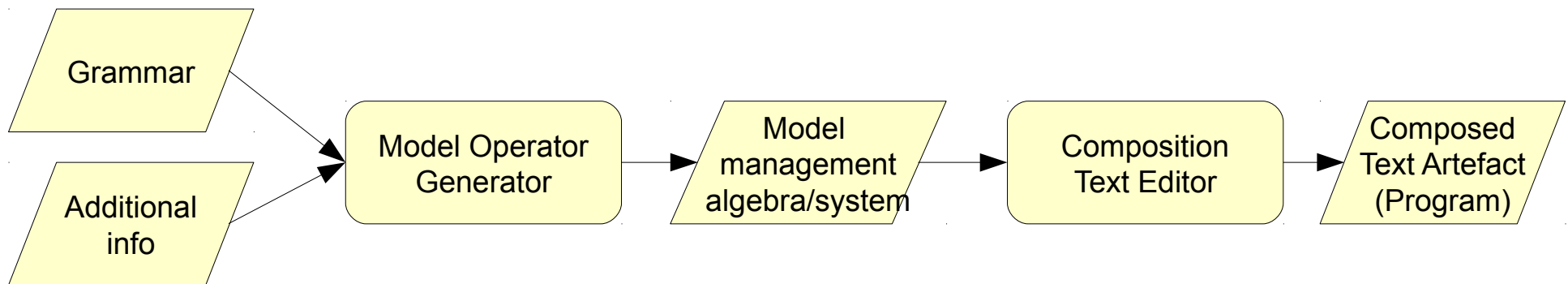
M1 model level  
Models, Programs

M0 object level



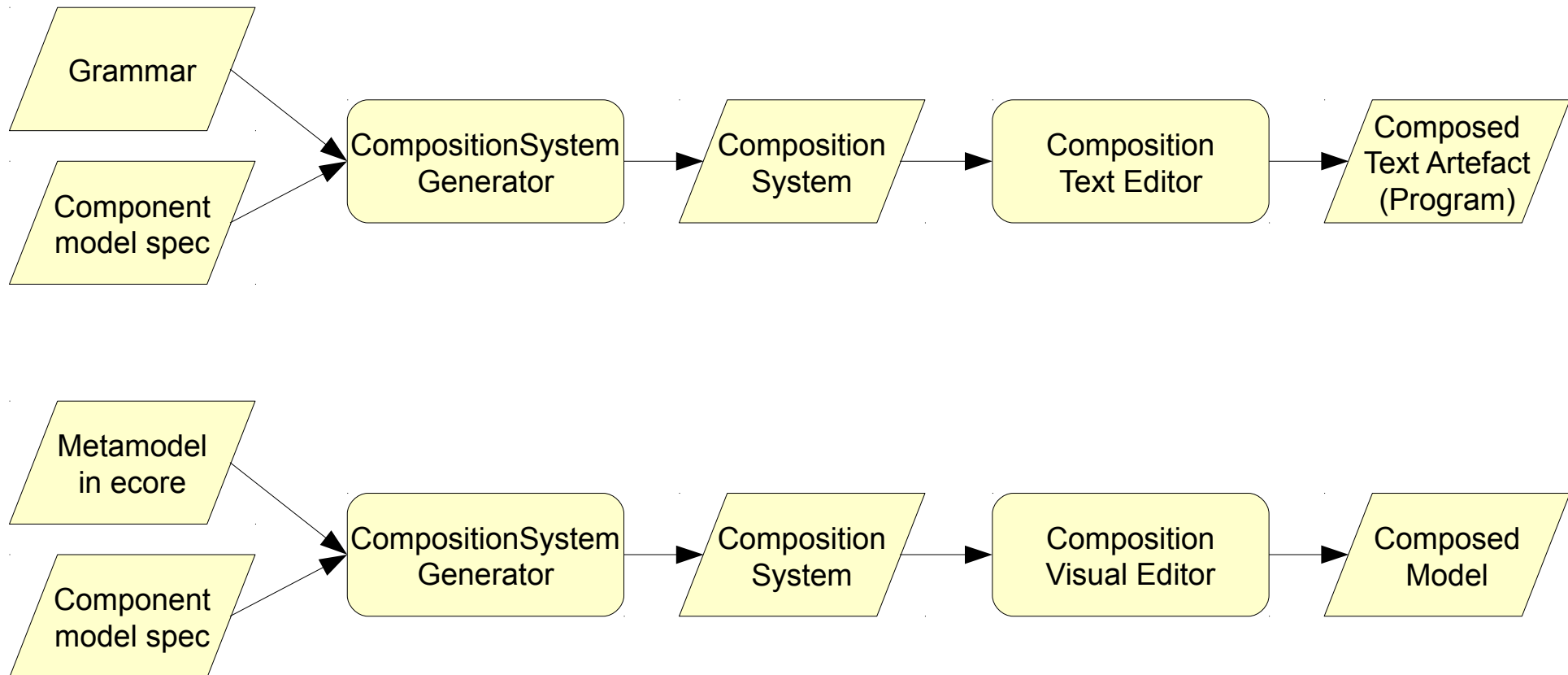
# Modelmanagement

- ▶ Eine **Modelmanagement-Umgebung** verwaltet Modelle eines Technikraumes mit
  - Komposition mit einer einheitlichen einsortigen Algebra, oder auch einer zweisortigen invasiven Algebra (invasives Kompositionssystem)
  - Slicing mit einer Reachability Engine



# Universale Invasive Komposition

- Für Grammarware, Tableware, Treeware und Modelware können invasive Kompositionssysteme generiert werden





## *Was haben wir gelernt?*

---

- ▶ Zukünftige IDE enthalten für jeden Technologieraum ein universelles Modelmanagement und sprach-universelles invasives Kompositionssystem.





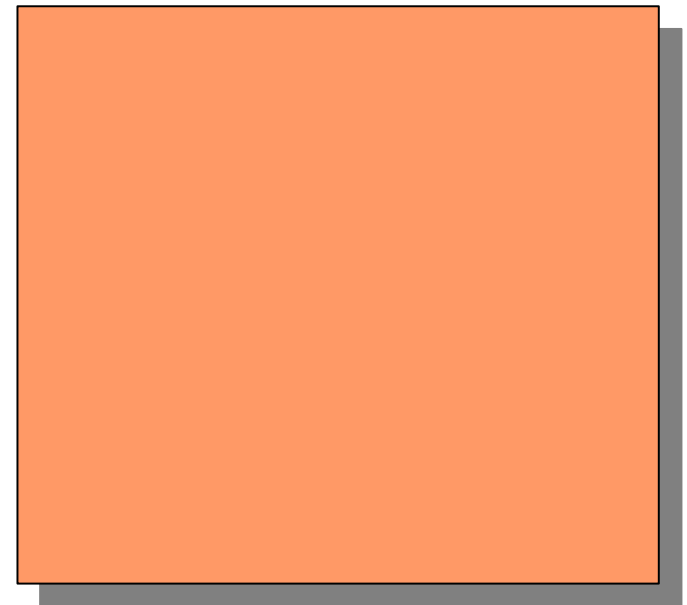
# *The End - Appendix*

---

---

# The Component Model of Invasive Composition

- ▶ A **fragment component** is a set of program fragments (program elements)
- ▶ For instance
  - a class
  - a set of classes
  - a package
  - a set of packages
  - a method
  - an aspect
  - a metadata description



# Boxes have Hooks

- ▶ Examples:
  - beginning/end of lists
  - method entries/exits
  - generic parameters

**Hooks** are arbitrary fragments or spots  
in a box

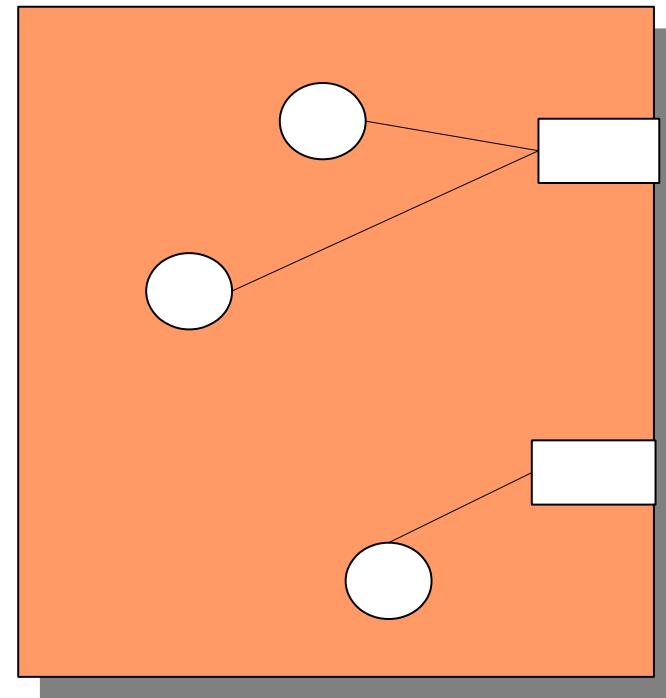
which are subject to change

# Implicit Hooks (aka Static Join Points)

- ▶ An implicit hook is a program point, given by the programming language, the DTD or Xschema
  - Example method entry/exit

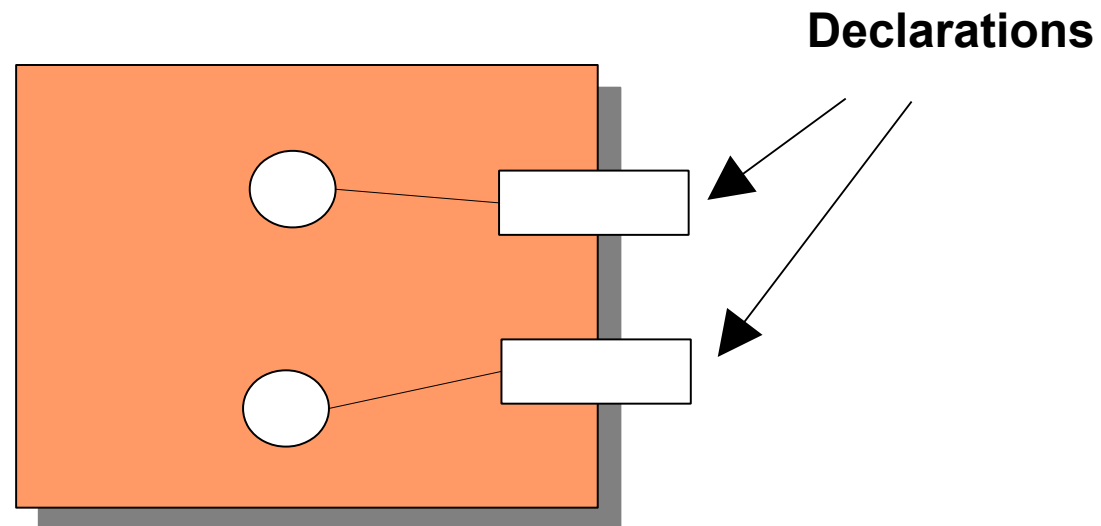
Method.entry → `m () {`  
`abc..`  
`cde..`

Method.exit → `}`



## Declared Hooks (Generic Parameters)

**Declared Hooks** are declared by the box writer as variables in the hook's tags.



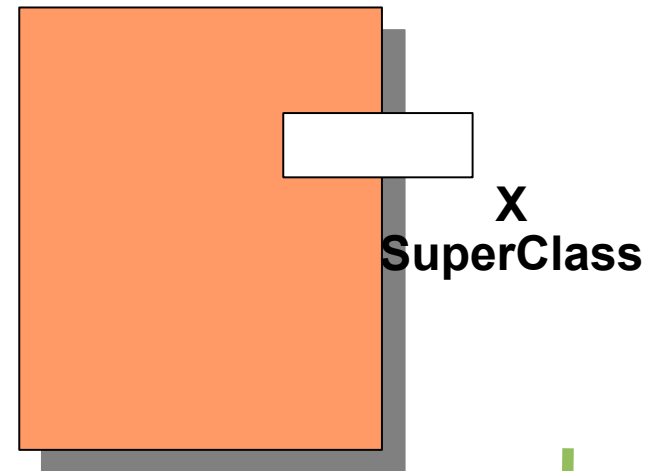
# Declaration of Hooks

- ▶ Markup Tags
- ▶ Language Extensions (keywords..)
- ▶ Standardized Names
- ▶ Comment Tags

```
<superclasshook> X </superclasshook>
```

```
class Set extends genericXSuperClass { }
```

```
class Set /* @superClass */
```



# *The Composition Technique of Invasive Composition*

**Invasive Composition  
adapts and extends  
components  
at hooks  
by transformation**