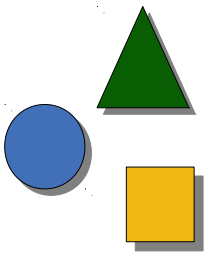
52. Staged Software Architectures with Staged Composition

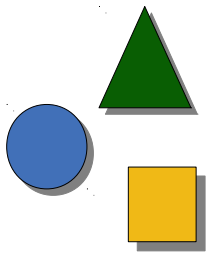
Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnologie
Version 13-1.0, 13.07.13

1) Web programming considered harmful
   1) Problem 1: Untyped template expansion
   2) Problem 2: Staging
   3) Problem 3: Spaghetti Code
2) Staged Architectures

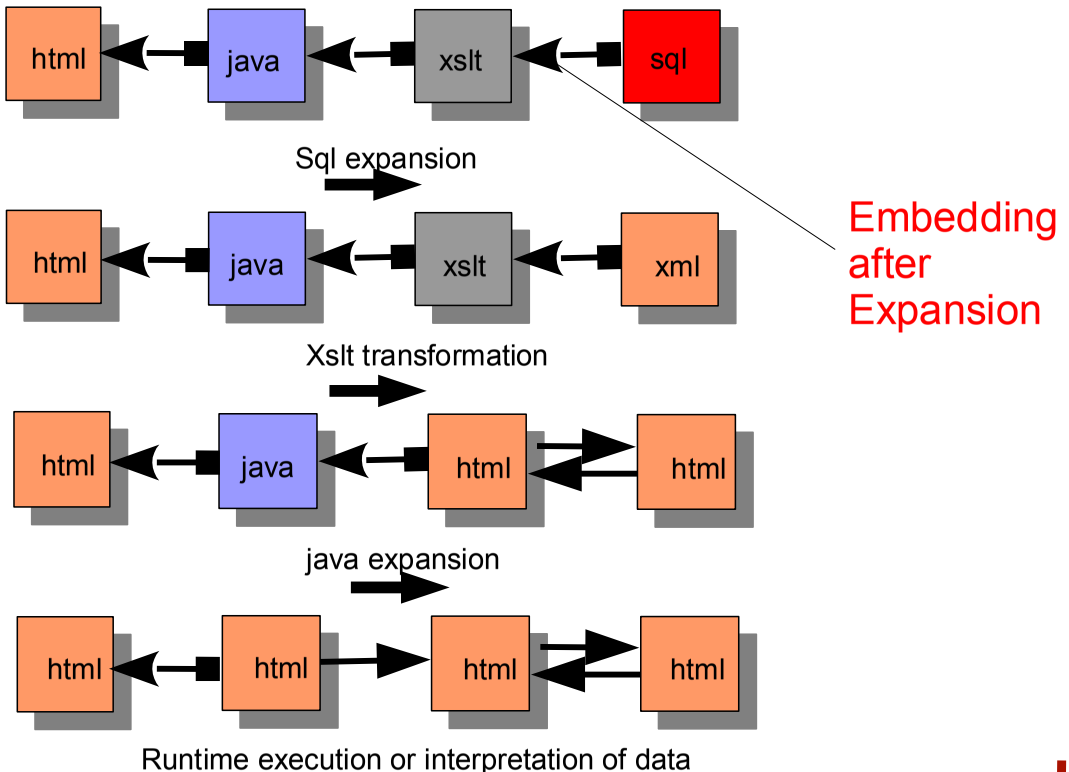


A Staged Architecture from Nature

52.1 Web Programming Considered Harmful



Web Programming: Staged, Untyped Template Expansion



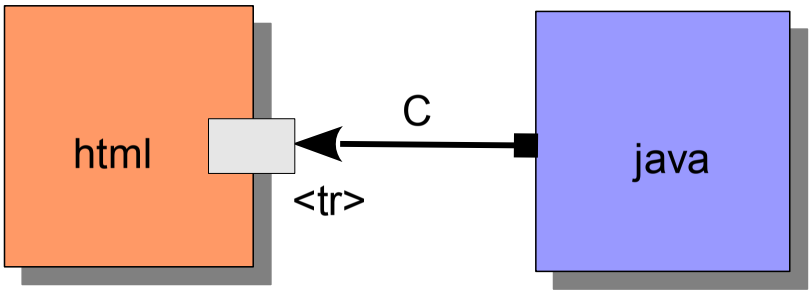
Problems of Web Programming

- Untyped extensions of templates
  - Error-prone
- Comprehension very difficult, due to the different stages
- Spaghetti-code-like programs
  - Scripts mixed with templates
  - Only valuable for programming-in-the-small

52.1.1 Problem 1: Untyped Template Expansion

Type-Safe Template Expansion

- How can you be sure that table rows are filled in?

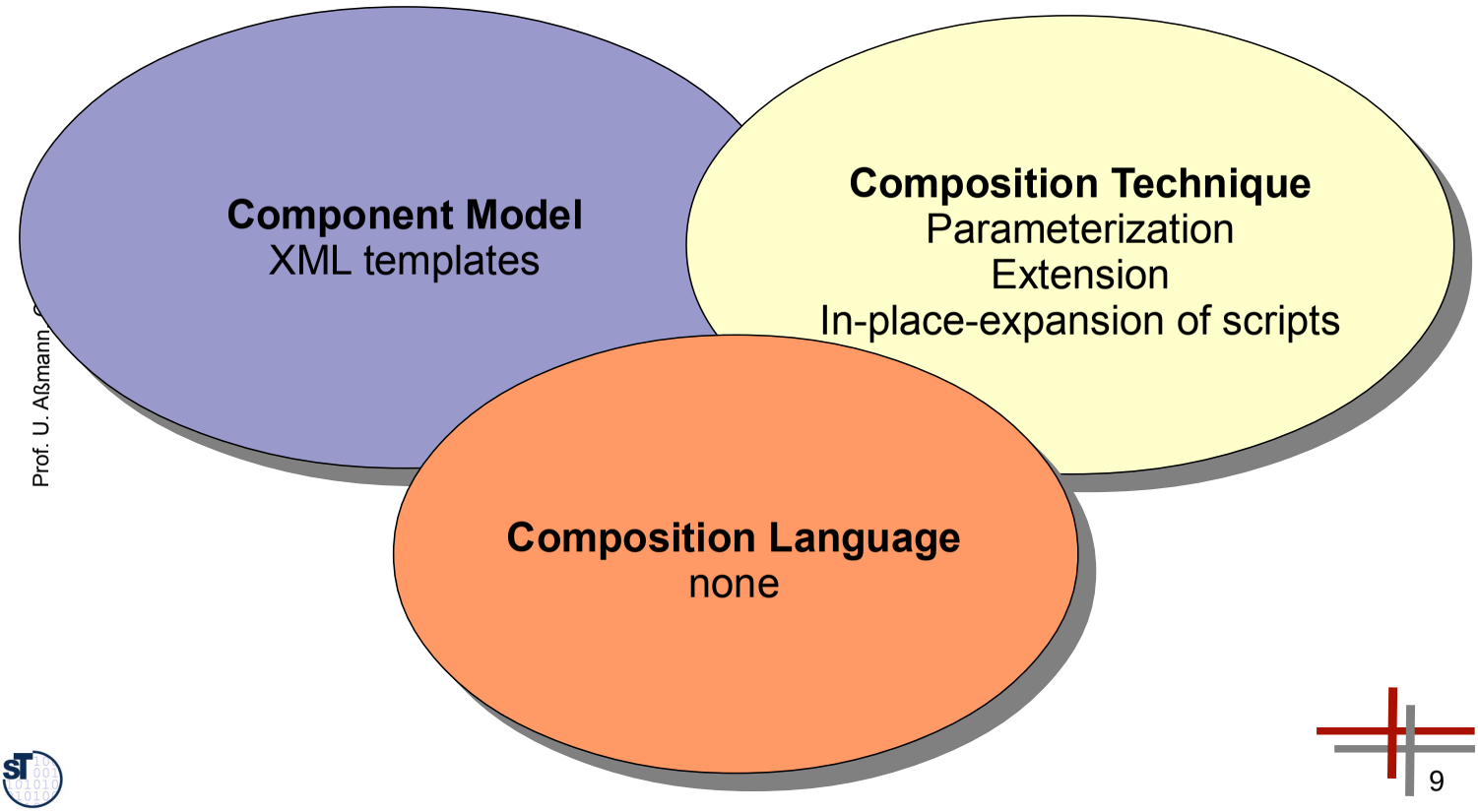


- Answer: in an invasive document composition system, the type checker of the invasive composition program will tell you, when checking the composition step C

Universality of Invasive Composition

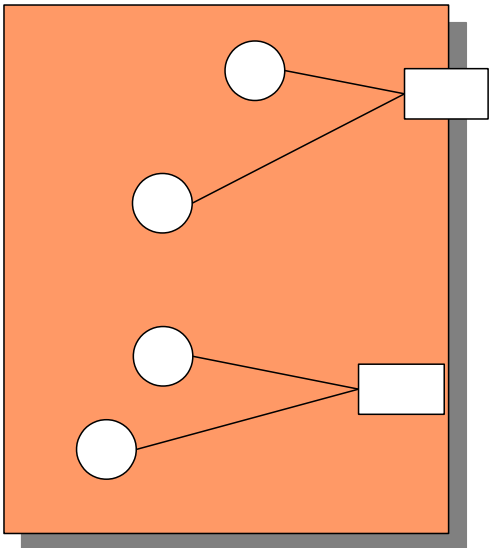
- Invasive composition only depends on a metamodel of the language
  - New hook and slot models can be derived from any language
  - Typing controls the composition of artifacts
- Hence, the method is *universal*
- and can be applied for typed document composition
- See www.reuseware.org, the universal invasive composition environment,
  - Can be tailored for text-based and diagrammatic languages
  - OpenOffice
  - XML dialects
  - EMF-based

Elements of Web Composition Systems



The Component Model of Invasive XML Composition

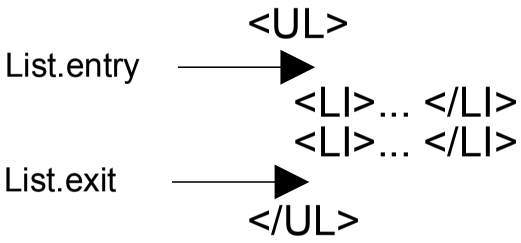
- The component is a fragment component (template)
  - A subword of the language, with *holes*
- Slots are variation points of a component
  - Parameters
  - Positions, which are subject to change
- Hooks are extension points
- Example:
  - A generic XML tree
  - A XML list with extension points

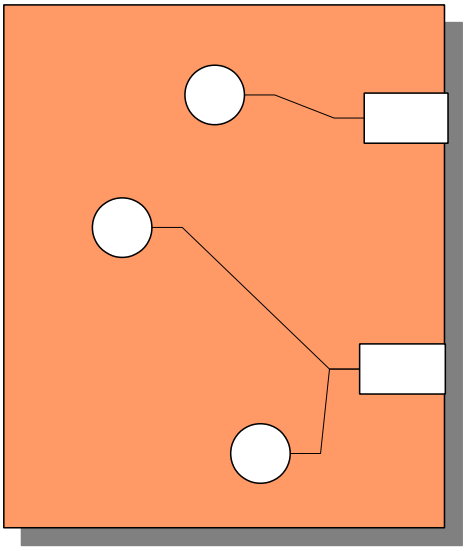


Extension of XML Fragment Components Should can be Typed

- What can be placed into an XML list entry/exit?

Prof. U. Aßmann, CBSE

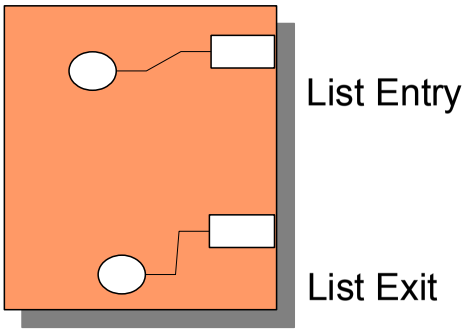


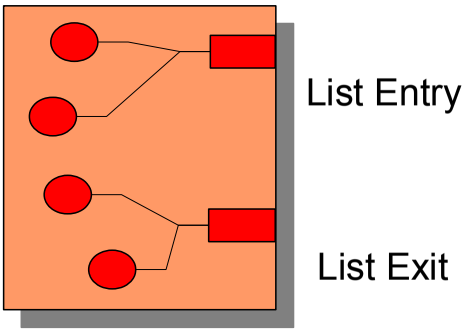


Slot and hook types are given by an XSchema, a metamodel of the XML document

Typed Hook Expansion for XML Components

Prof. U. Aßmann, CBSE






```
<UL>

   <LI>... </LI>
   <LI>... </LI>

</UL>
```

```
<UL>
   <LI>... </LI>
   <LI>... </LI>
   <LI>... </LI>
   <LI>... </LI>
</UL>
```

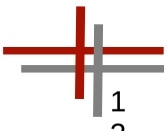
```
XMLcomponent.findHook(„ListEntry“).extend(„<LI>... </LI>”);

XMLcomponent.findHook(„ListExit“).extend(“<LI>... </LI>”);
```

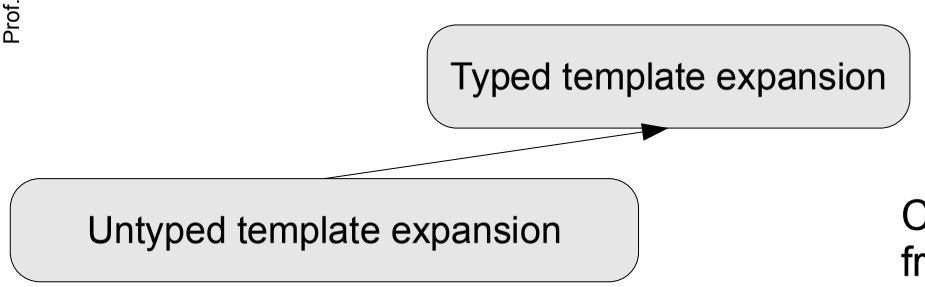
Insight: Web Systems Need Typed Template Processing

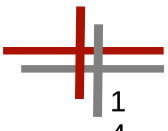
Problem: Web programming is based on *untyped template expansion (frame processing)*

It should be based on typed template expansion (invasive composition)

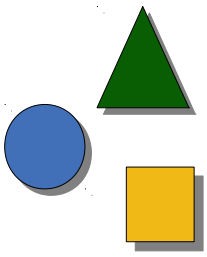


The Hierarchy of Staged Architectures

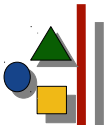


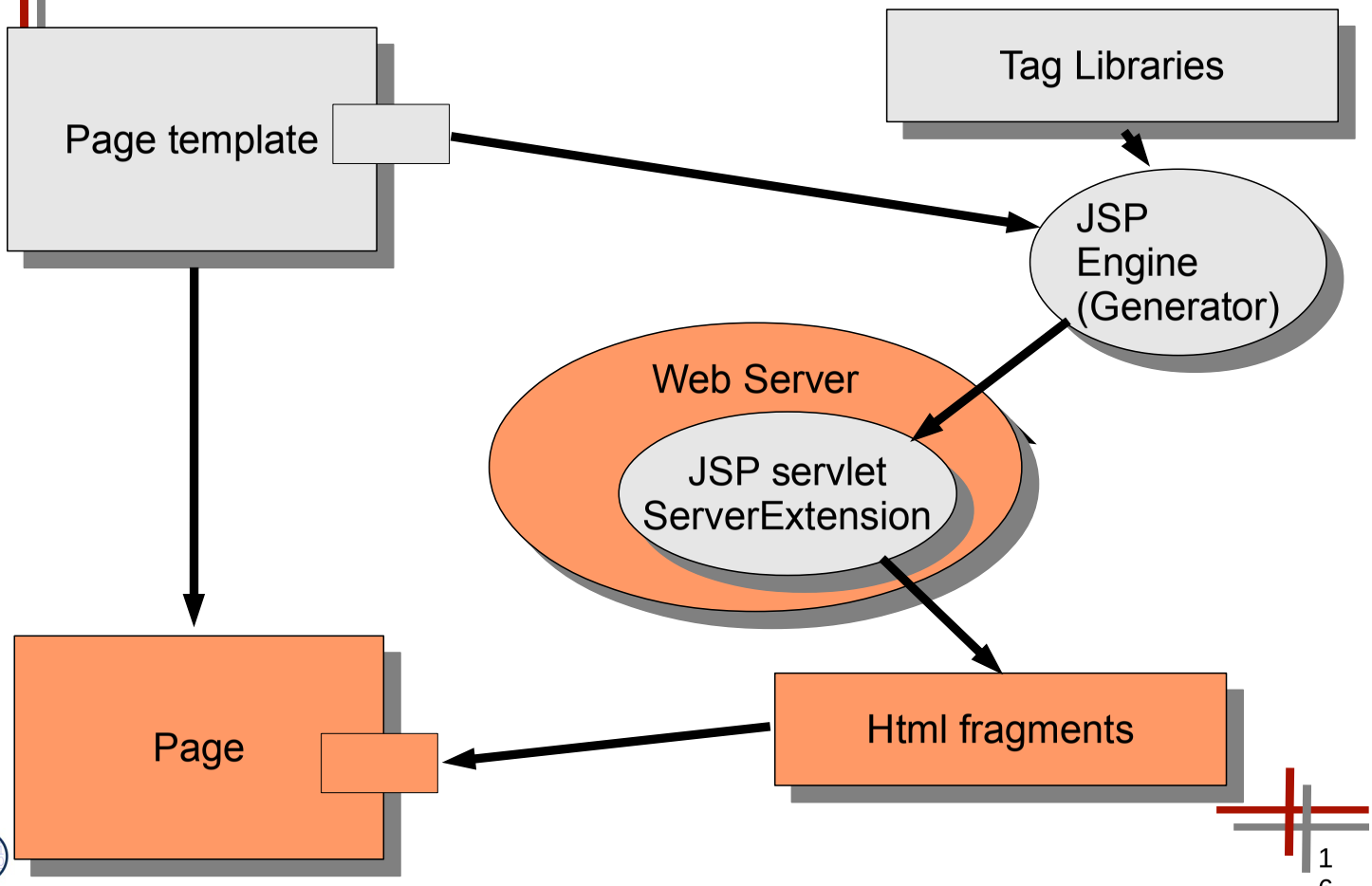


Problem 2: Staging



The JSP Mechanism





Spagetti Code from JSP Tutorial - Belongs to Different Execution Stages

```
<html>
<%@page language=”java” imports=”java.util.*” %>
<h1> Welcome! </h1>
<jsp:useBean id=”clock” class=”jspCalendar” />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
        Good Morning!
<% }else { %>
        Good afternoon...
<% } %>
</p>
<html>
```

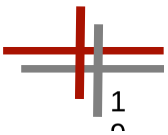

A Web Scripting Language with 5 Stages

```
<xfa1:profession>
  <xfa2:ref pop-up>
    <sql>select arbitrary lastName from bakers</sql> baker
  <xfa2:ref pop-up>
</xfa1:profession>
<xfa:function hello>
  <body>
  <h1>This is My Personal Page with XFA</h1>
  <xfa:if Odd(environment^DATE)>
    <xfa:ref message>
  <xfa:else>
    Even day. No money for <xfa1:profession> :-(
  </xfa:if>
  </body>
</xfa:function>
<xfa:function message>
  Odd day today, dear student. You may visit the <xfa1:profession> shop.
</xfa:function>
```

[until 2003: www.xml4all.com]

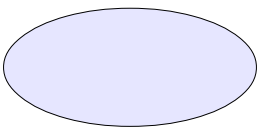
A Possible Solution: Staged Programming

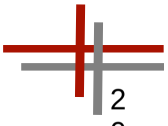
- In the Beginning, there was the Data



Then Came the Programs

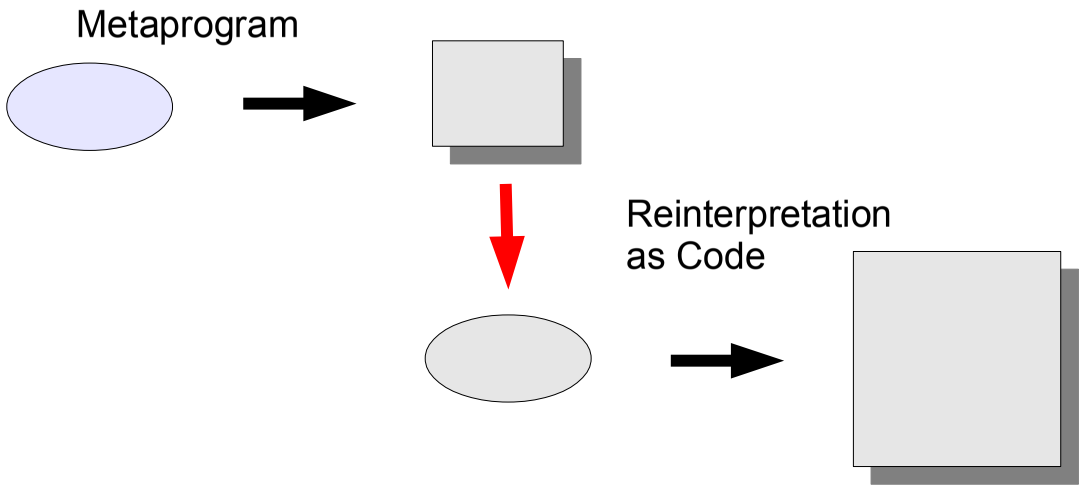
- Producing lots of data out of little code





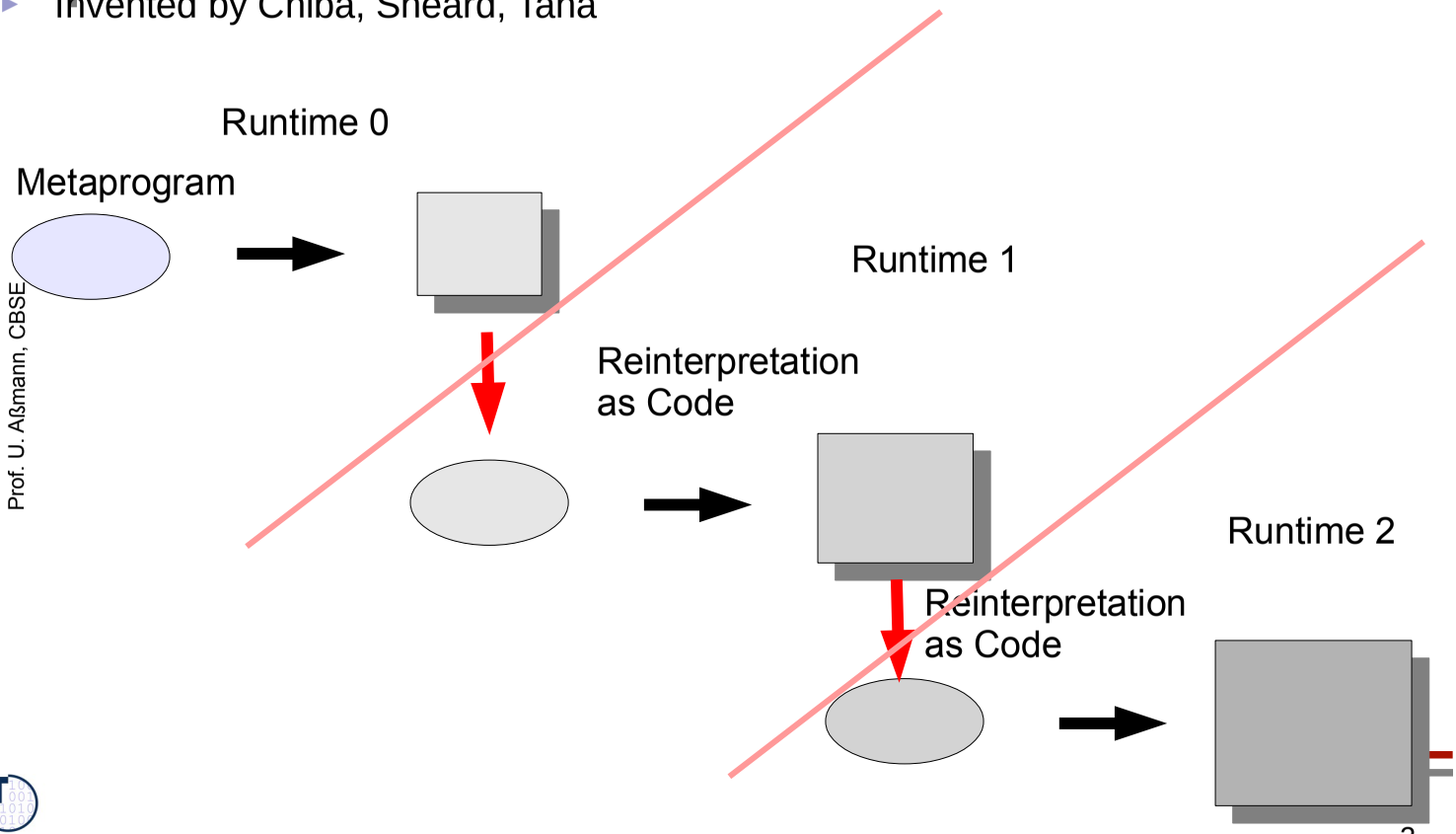
Then Came the Metaprograms

- Producing lots of programs from few metaprograms



Then Came the Staged Metaprograms

- Invented by Chiba, Sheard, Taha

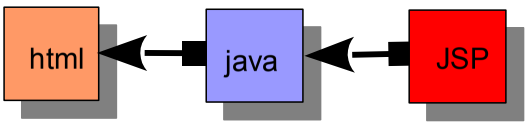


Staged Programming

- Staged programming (e.g., MetaML, MetaOCaML) has pioneered the mix of static metaprograms and programs
  - The metaprograms are expanded statically (stage 1) to produce the final program (stage 2)
  - Metaprograms are typed in the metamodel of the programs (type-safe expansion of metaprograms)
- Example [Taha]:

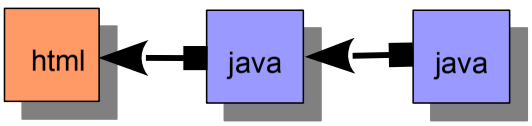
```
# let a = 1+2;;
val a: int = 3
# let a = .<1+2>.;;
val a: int code = .<1+2>.
# let b = .! a;;
val b = 3
```


JSP Uses Staged Programming



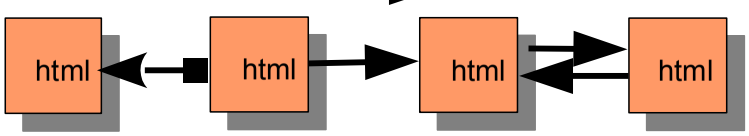
Stage 1

JSP expansion



Stage 2

java expansion



Stage 3

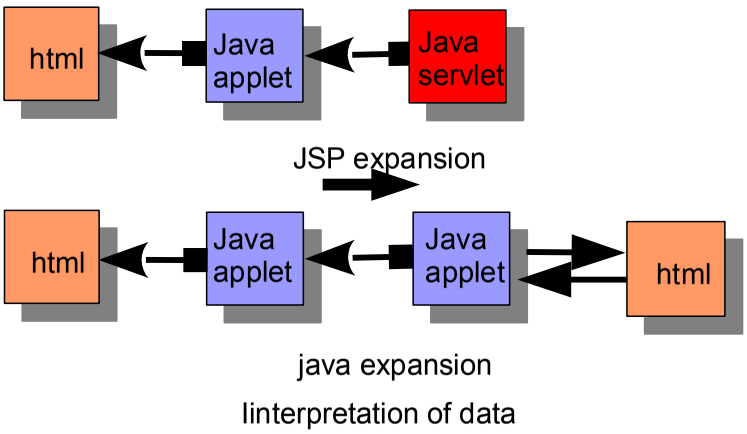
interpretation of data

Spagetti Code Revisited

```
<html>
<%@page language=”java” imports=”java.util.*” %>
<h1> Welcome! </h1>
<jsp:useBean id=”clock” class=”jspCalendar” />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
        Good Morning!
<% }else { %>
        Good afternoon...
<% } %>
</p>
<html>
```

Servlet generator expands blue lines to Java code

Example 2: Staged Servlet/Applet Processing



Insight 2: Web Systems Need Staged Programming

Web programming is often based on
staged programming

- Because for dynamic web pages, code is generated
  - E.g., servlet or applet generation
- Because of the client-server stage separation
- Because legacy tools must be encapsulated into a stage (e.g., databases)

Staged programming should additionally be typed, otherwise it is chaotic

N.B.: Configuration and Variant Selection works with Staged Programming

```
# fun f variant =
  if variant = 1 then .<.fun q x = x*x.>.
                 else .<.fun q x = x/x.>.
;;
```

```
# let variant = 1;;
# fun g = (f variant) 2;;
val g: int code = .<let q x =
  x*x>.
# let res = g 3;;
val res = 9
```

```
# let variant = 2;;
# let g = (f variant) 2;;
val g: int code = .<let q x =
  x/x>.
# let res = g 3;;
val res = 1
```

Different behavior of second stage

Staging Is Used for Variant Management

On stage n-1, control-flow denotes variant selection for stage n

Platforms are often selected by evaluating control-flow in previous stages

Spagetti Code Revisited

```
#ifdef HTML
<html>
#else
<wap>
#endif
<%@page language="java" imports="java.util.*" %>
#ifdef HTML
<h1> Welcome! </h1>
#else
<bold>Welcome!</bold>
#endif
<jsp:useBean id="clock" class="jspCalendar" />
#ifdef HTML
<p>
#endif
.....
```

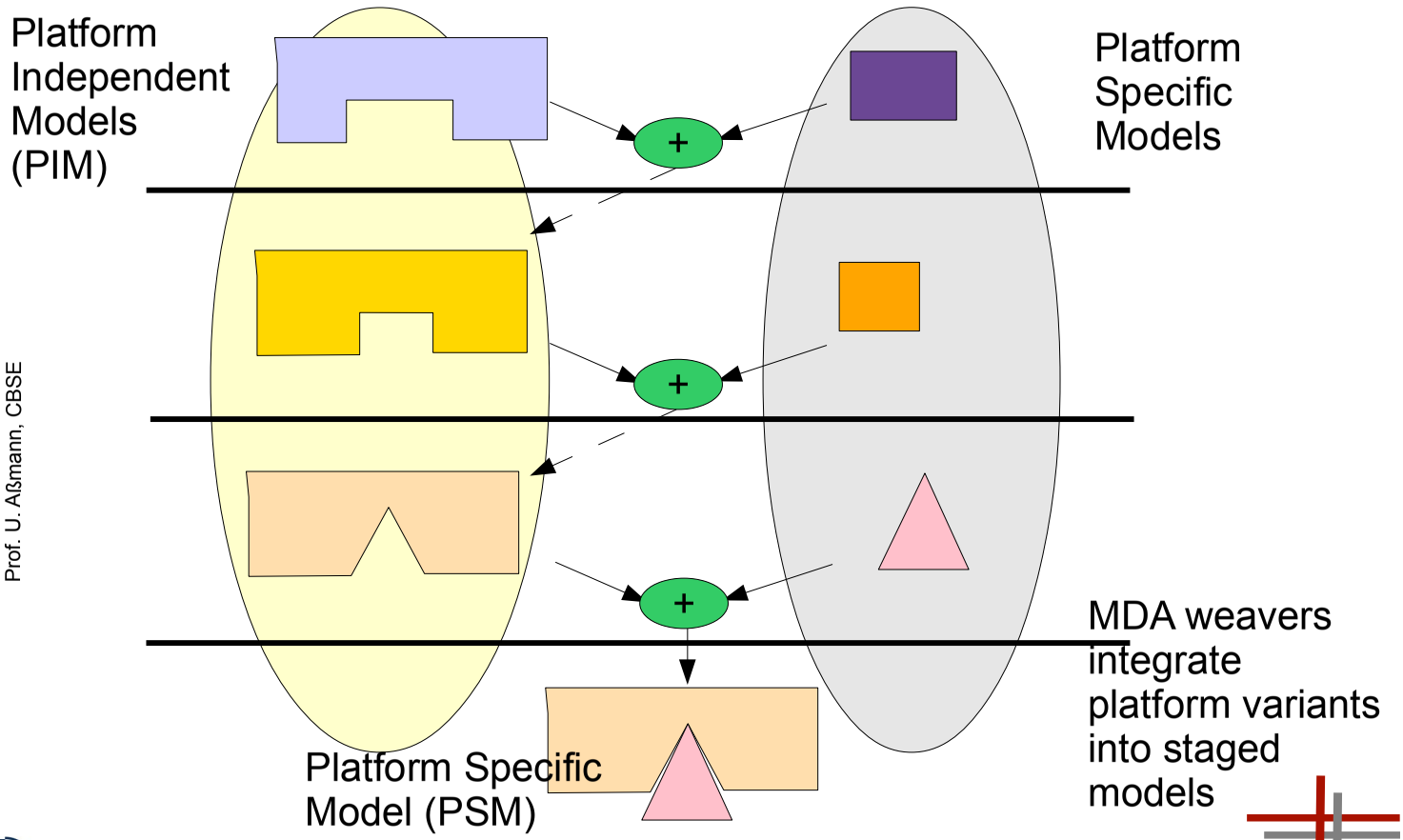
CPP stage selects HTML or WAP

Evaluating the CPP script chooses the platform

The C Preprocessor as Staged Programming System

- Insight: C with #ifdef language is a real staged programming system with CPP-C (State 0) and core-C (Stage 1)
  - That's why it's being used...
  - That's why it's so hard to deal with
- However, there is no component model, not even respect of the syntax of core-C
- The composition language of CPP-C is simple (macros, if-expressions, constant definitions)

A Staged Programming System: MDA

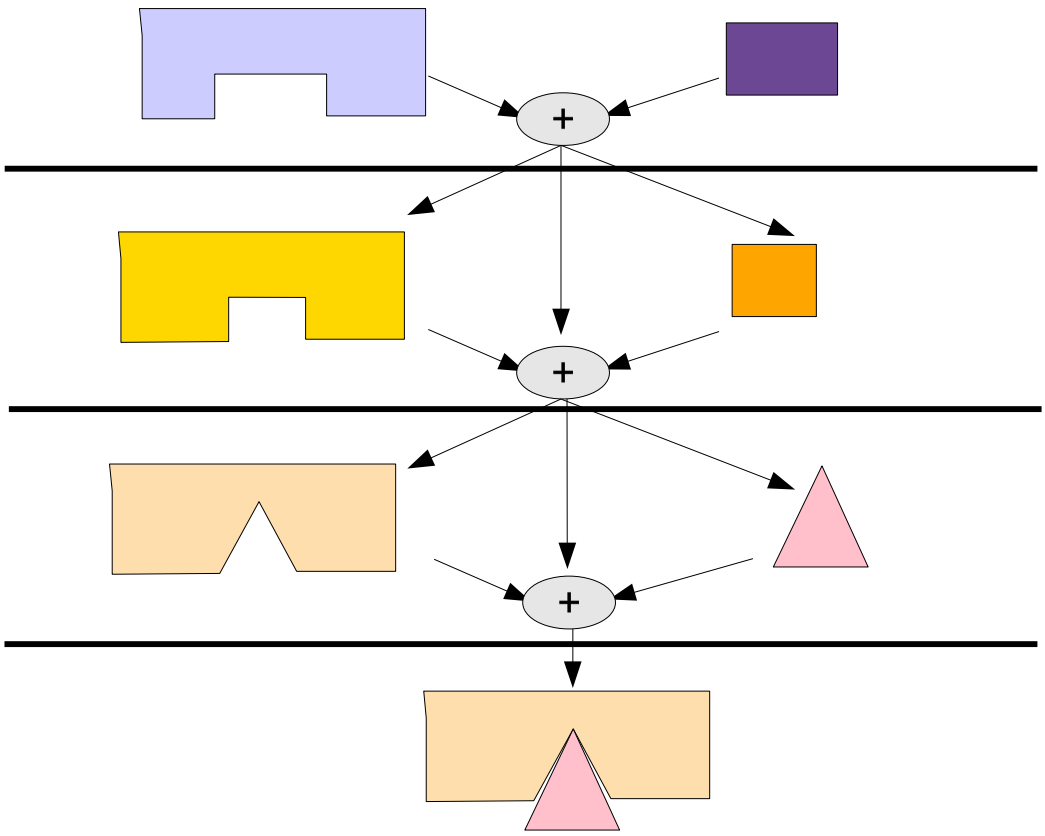


Staged Programming Architectures vs MDA

- MDA is a staged programming approach, but *not* a staged programming architecture, since no architecture, no component models are given
- ... but a staged programming technology for variant selection

... but we can build more powerful forms of MDA by taking in the ideas of staged programming and staged architectures

Staged Architectures Written as Layers



Advantages of Staged Programming

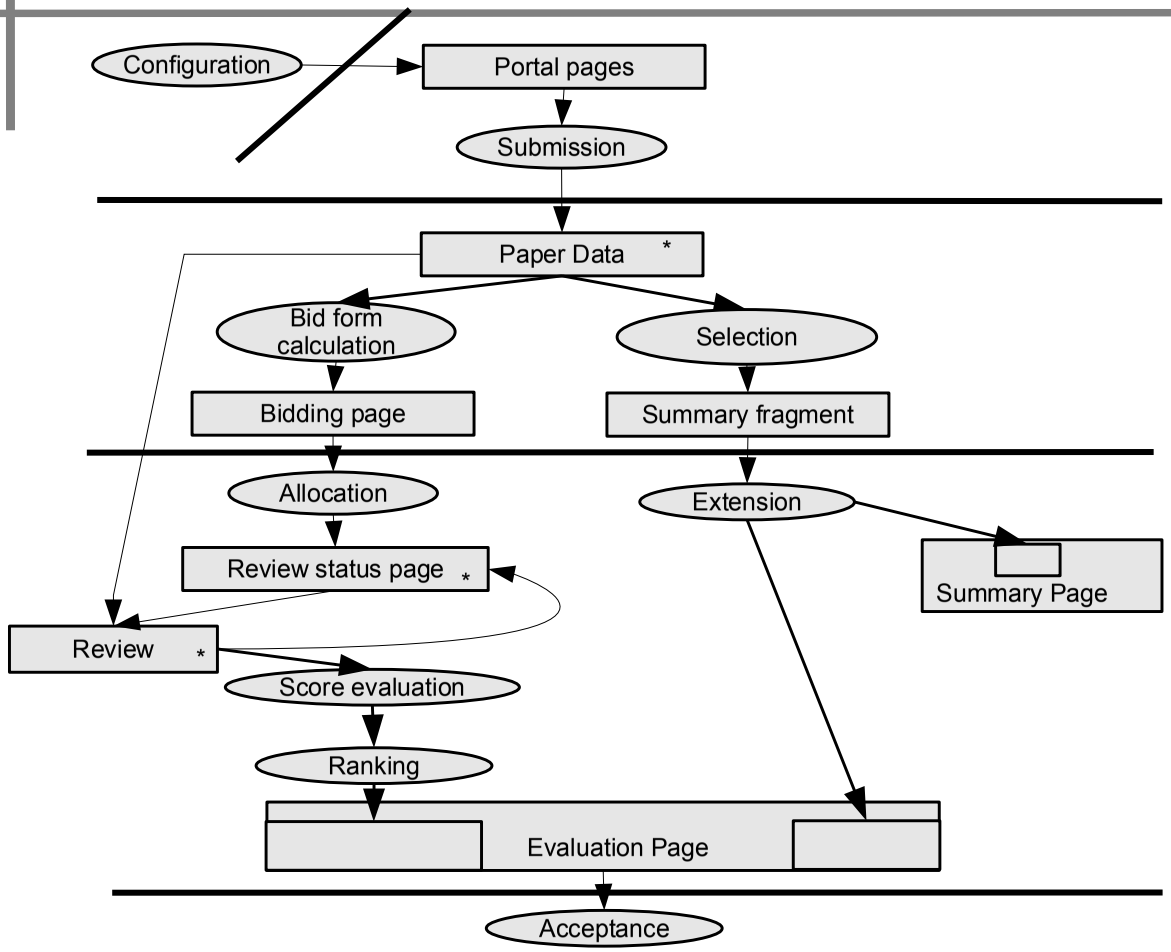
- Typed
  - Type-safe development, less error-prone
- Concise representation of system
  - Representation is expanded through every stage
- Easy to code variants
  - Control flow on a build stage does variant selection

- Problems:
  - Still, lots of spaghetti code.

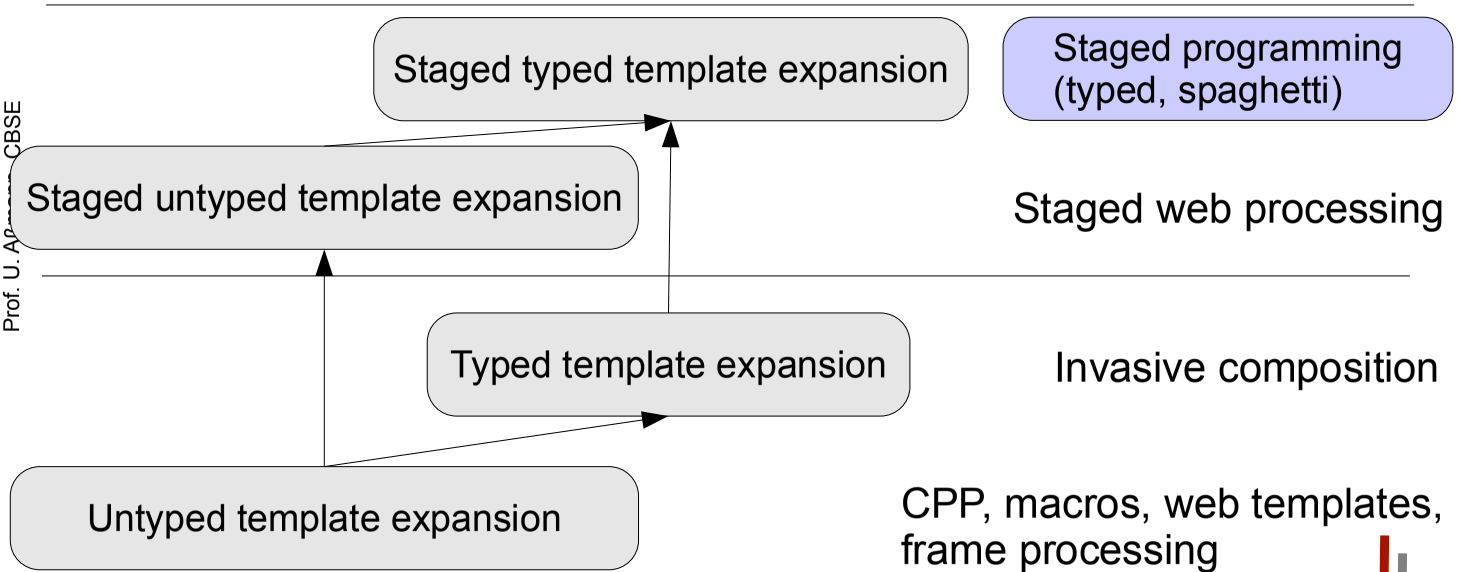
Example: The START Conference Management System

- START is a review management system
  - It has a 5-phase staged template expansion architecture
  - START servlets are composition scriptlets that compose (parameterize, extend) html-templates
- Using invasive composition, we developed a *staged typed template expansion* system
- It is no problem to generate servlets, too. Then we have real staged programming

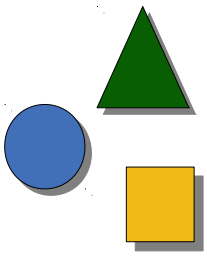
The Staged Template Expansion Architecture of START



The Hierarchy of Staged Architectures



54.1.3 Problem 3: Spaghetti Code



and a possible remedy:
staged architectures

Architecture and Composition

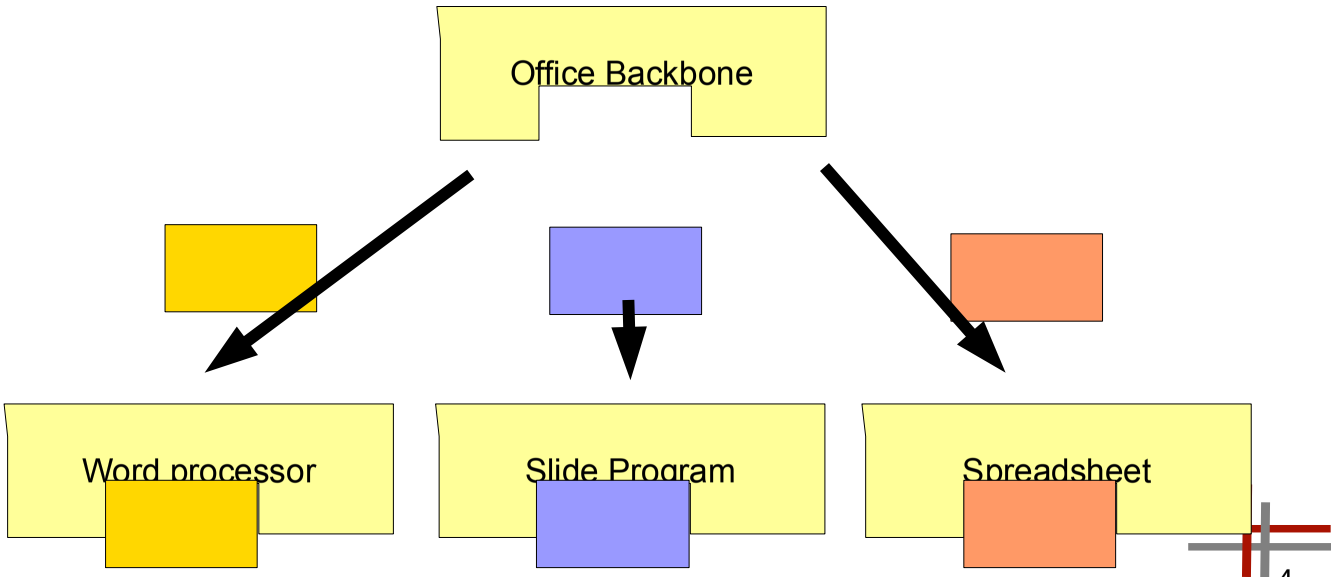
- Two of the central insights of the software engineering in the 1990s are:

Separate architecture from the components

Compose components by a *composition language*

Benefit of Architectures

- Comprehensibility
- Commonalities into the architectural level, variabilities into the application-specific components
- Does this also hold for web programming?



Less Spaghetti Code: A Fragment-Based Template and its Architecture

Component

```
<html>
        <hook id="imports">
<h1> Welcome! </h1>
        <hook id=use">
<p> Today is
        <hook id="year"/>
        -<hook id="day"/>
</p>
<p>
        <hook id="greeting"/>
</p>
<html>
```

Composition Program (Architecture)

```
public class composeTemplate {
  String use = „jspCalendar“
  String imports="java.util.*";

  compose() {
    Template template = read();
    Bean clock = new jspCalendar();
    String year = clock.getYear();
    String day = clock.dayOfTheMonth();
    if (Calender.getInstance().get(Calendar.AM_PM) ==
      Calender.AM)
       greeting = “Good Morning!”;
    else
      greeting = “Good afternoon...”;
    this.merge(template);
  }
}
```

Prof. U. Aßmann, CBSE

Separation of Components and Architecture Allows for Variants

```
public class composeTemplate {
   String use =
   String imports=
   compose() {
     String year =
     String day =
      greeting =
   }
}
```

Composition Program (Architecture)

Component 1

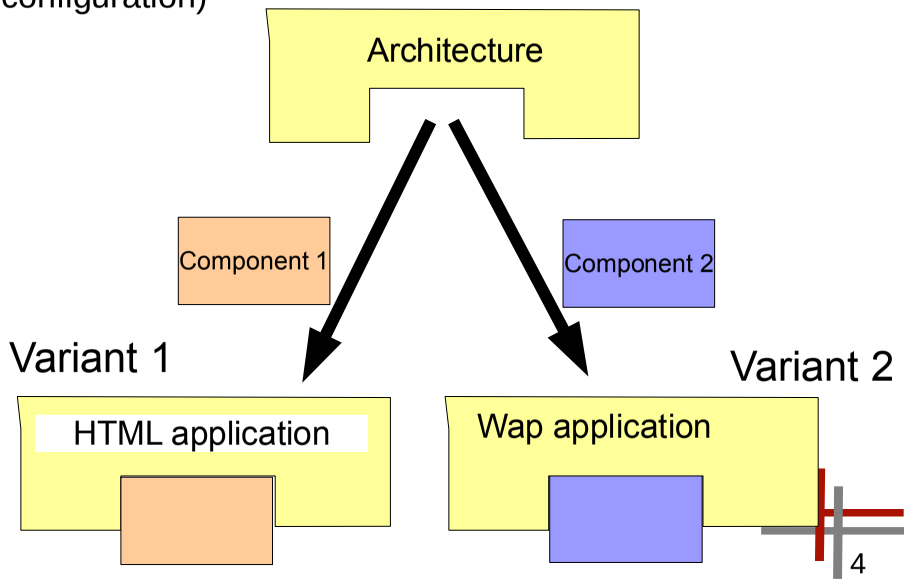
```
<html>
        <hook id="imports">
<h1> Welcome! </h1>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>   <hook id="greeting"/>
</p>
</html>
```

Component 2

```
<wap>
        <hook id="imports">
<bold> Welcome! </bold>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>   <hook id="greeting"/>
</p>
</wap>
```


Architecture and Variants in a Product Line

- Advantages for Separating Architecture From Application Components
  - Isolation of commonalities into frameworks
  - Comprehensibility
    - Programming-in-the-large is separated from programming-in-the-small, components can be abstracted away
    - Less spaghetti
  - Easy variability (variant configuration)



Prof. U. Aßmann, CBSE

Variant Management by Control Flow in Architectural Composition Programs

```
public class composeTemplate {
  if (HTML) then use component 1
            else use component 2
  String use =
  String imports=
  compose() {
    String year =
    String day =
     greeting =
  }
}
```

Variant 1

```
<html>
        <hook id="imports">
<h1> Welcome! </h1>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>   <hook id="greeting"/>
</p>
</html>
```

Variant 2

```
<wap>
        <hook id="imports">
<s1> Welcome! </h1>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>   <hook id="greeting"/>
</p>
</wap>
```


Definition: Staged Data-Flow Architectures

Staged data-flow architectures add an explicit architectural level to staged template processing

- Every stage is executed to produce ***data*** for the next stage (data-flow)
- Every stage is executed at a specific time
- On every stage, there is
  - an architecture,
  - a component model
  - a composition technique,
  - and a composition language
- Every composition language has its own interpreter
  - and is reduced (expanded) at different interpretation times

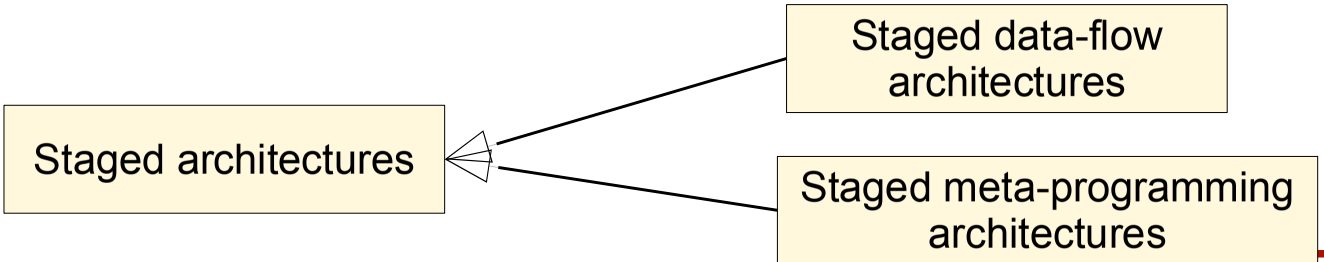
Web Programming needs Staged Data-Flow Architectures

- It would be nice to extend staged typed template expansion in web engineering to
- staged data-flow architectures.

Definition: Staged Architectures

Staged meta-programming architectures combine *staged programming with an explicit architectural level*

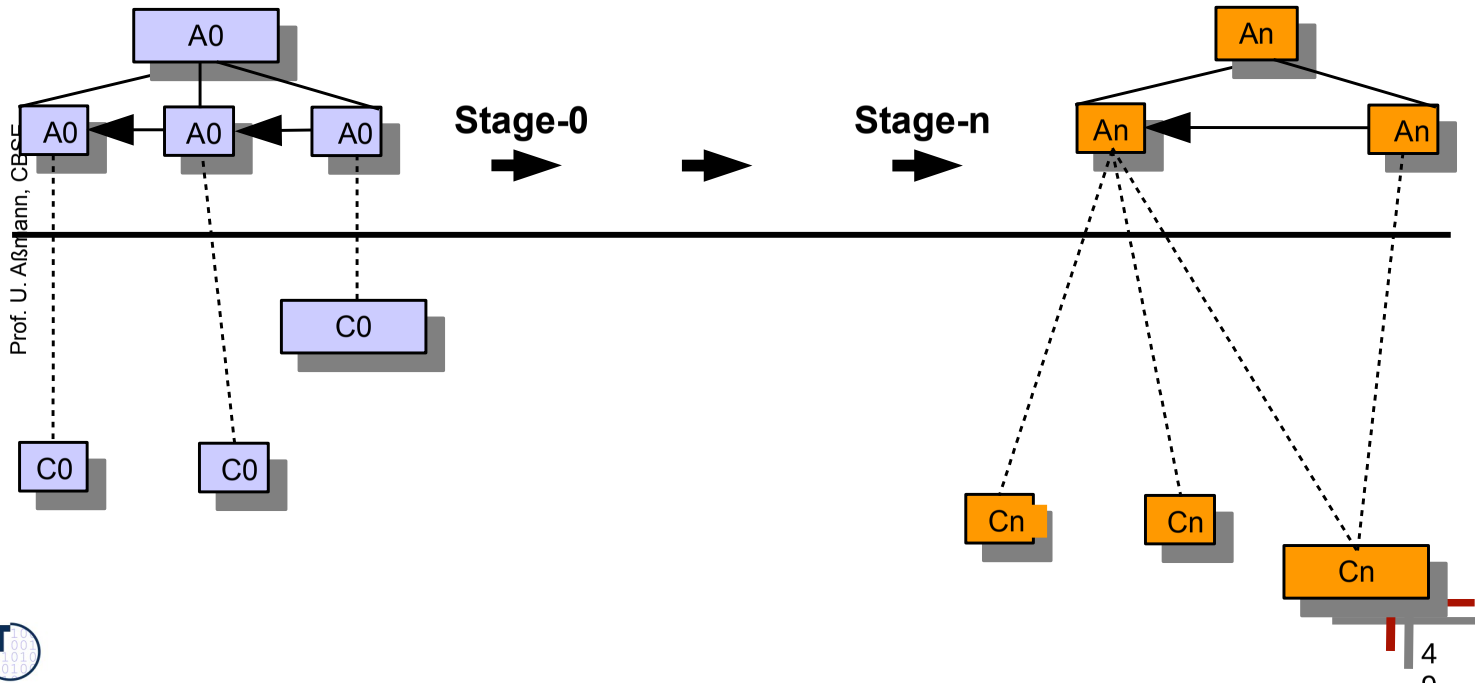
- Every stage is executed to produce ***code*** for the next stage
  - The final runtime code (architecture and components) is computed over several stages
  - The initial architecture is very small, the final architecture can be very large
  - Composition expressions, specifications, or programs may be hidden in components of a previous stage



Staged Metaprogramming Architectures Separate Large from Small

Stage-A0 architecture in composition language A0 Component language C0

Generated **Stage-An architecture in composition language An Component language Cn**

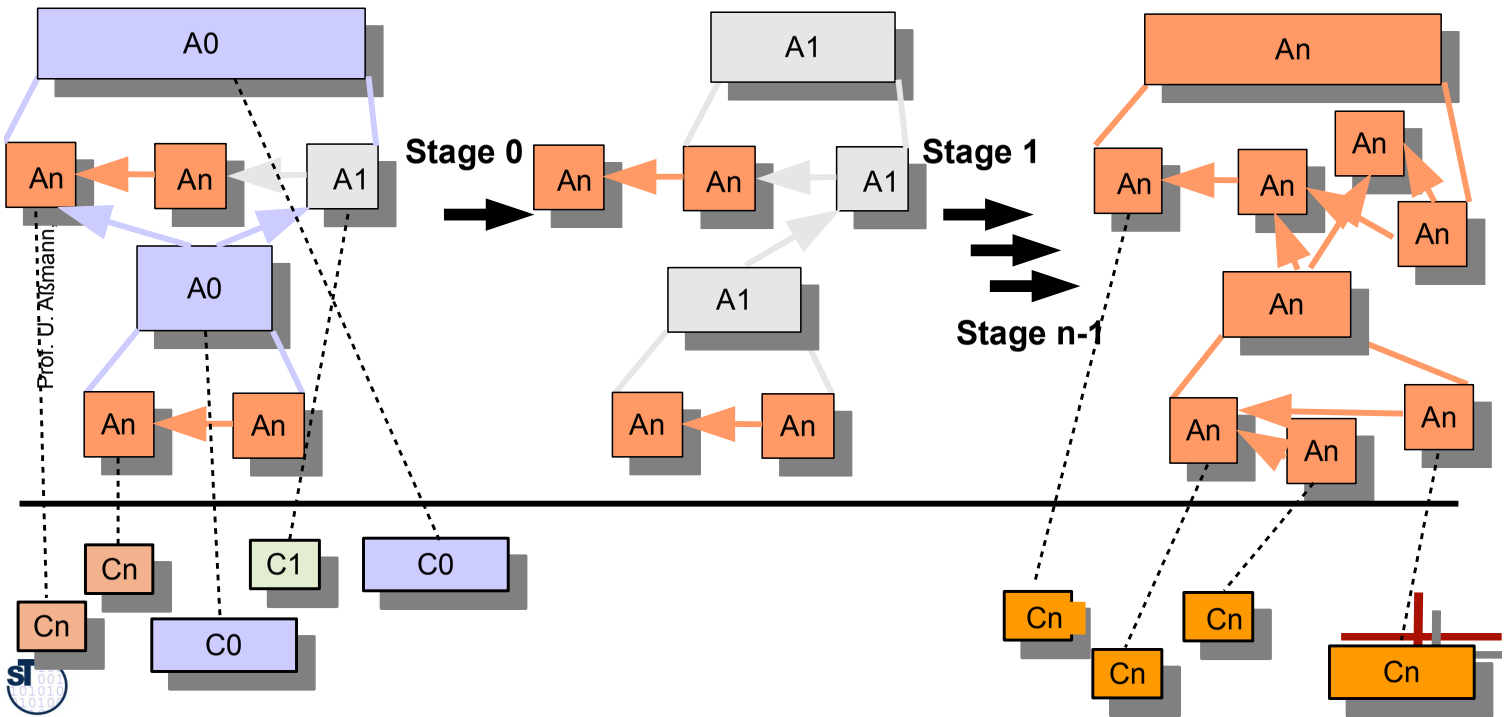


Staged Metaprogramming Architectures may have Different Component Models on Each Stage

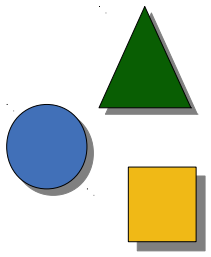
Stage-0 architecture in composition language A0 Component language C0

Stage 0 produces Stage-1 architecture in composition language A1 Component language C1

Stage n-1 produces Stage-n architecture in composition language An Component language Cn

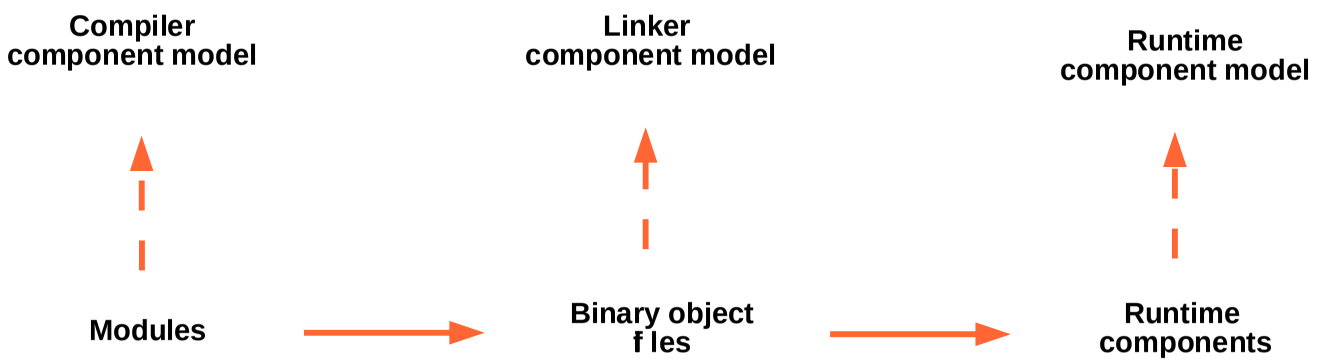


52.4 Staged Metaprogramming Architectures in Software Engineering



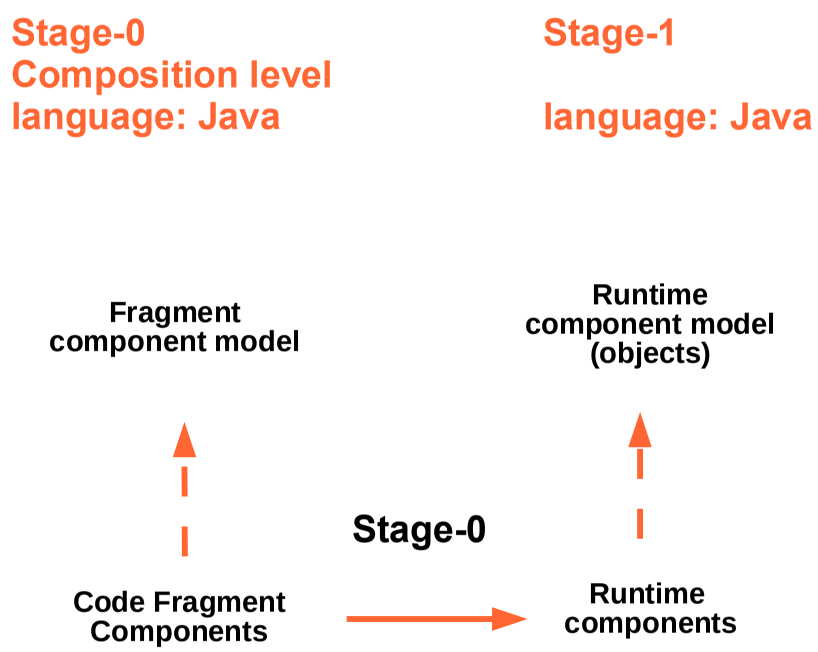
Build Management is Staged Composition

- Software build management is code composition in several stages
- Composition language: Make, ant, maven, etc.
  - Make is a composition tool with a lazy rule-based language
  - Expressions are applications of UNIX tools (compiler, linker, generator, preprocessor)
- Different component models on all stages



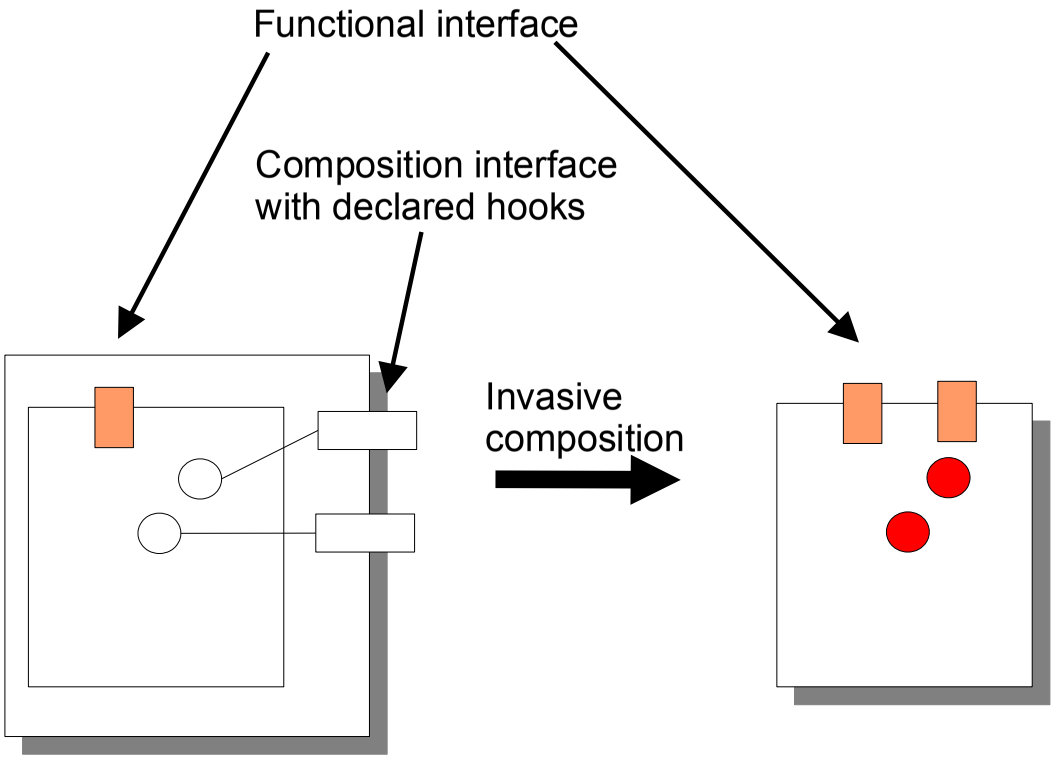
Invasive Software Composition

- Produces code from typed templates by parameterization and expansion



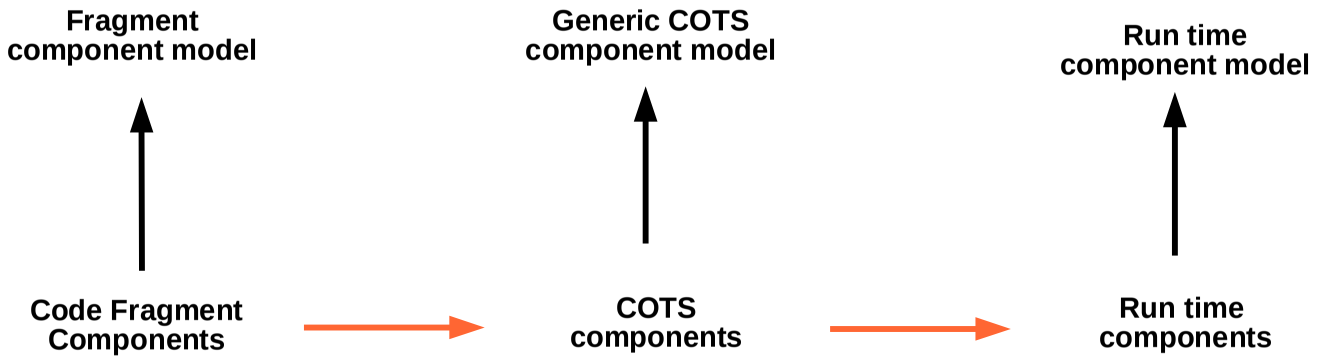
Invasive Composition Produces Functional from Composition Interfaces

- Two different component models



Component Models on Different Levels in the Software Process

- Standard COTS models are just models for binary code



The Dresden Staged Architecture Development Process

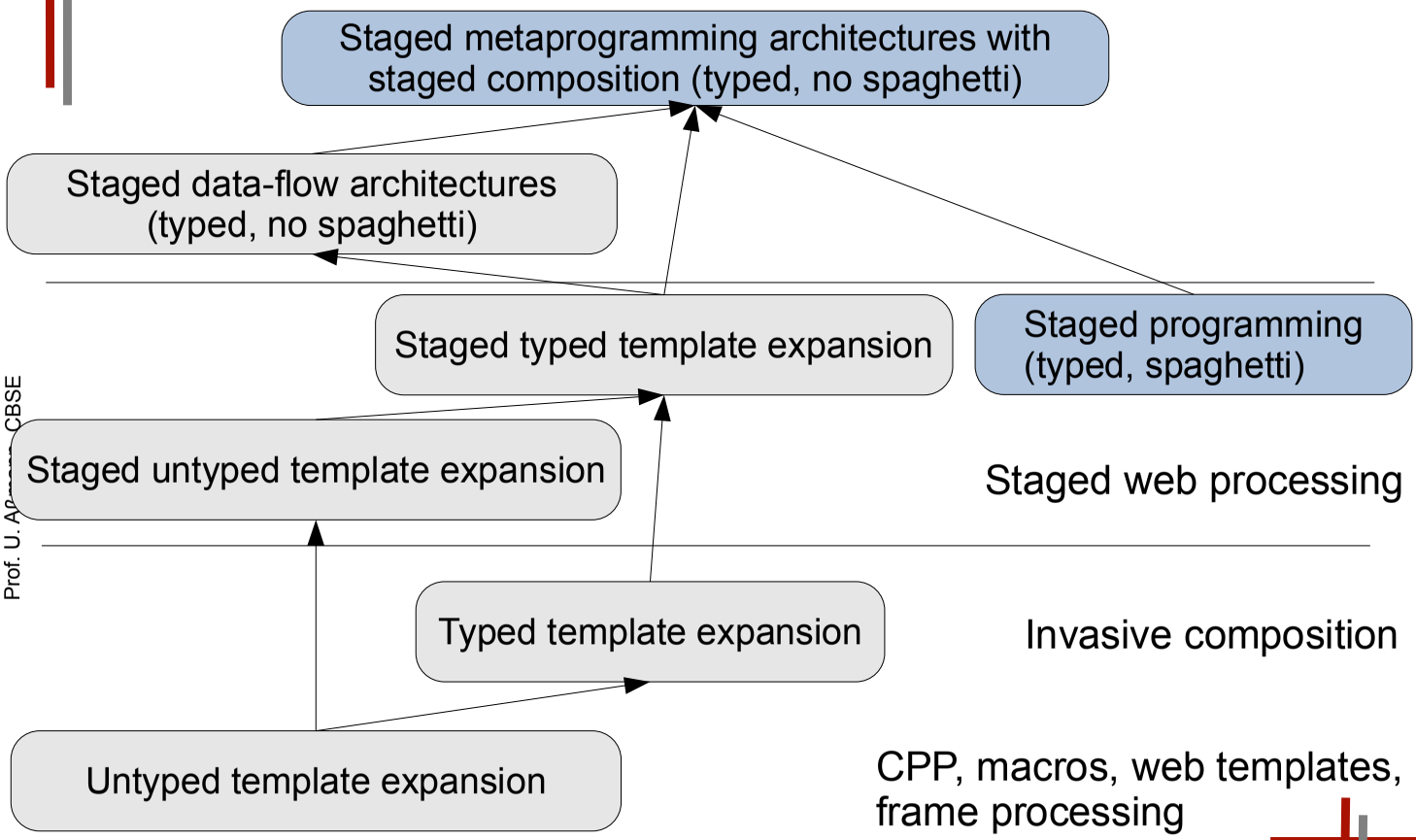
- Fix the stages
  - Decide on a staged processing or programming architecture
- Fix the component models for every stage
  - Interface concepts, composition operations, composition language
  - Design a concrete component model with Reuseware toolkit
- Fix the architectures
  - Decide on a composition language on each level
- Fix the variant management
- Fix the components

- And you'll have a pretty comprehensible product line!

The Vision of Staged Systems

- The staged programming principle is powerful, so future systems will employ it
- We need tools to support staged architectures
  - Visualize them
  - Debug them
  - Support the component models on all stages
  - .... that's a lot of work...

The Hierarchy of Staged Architectures



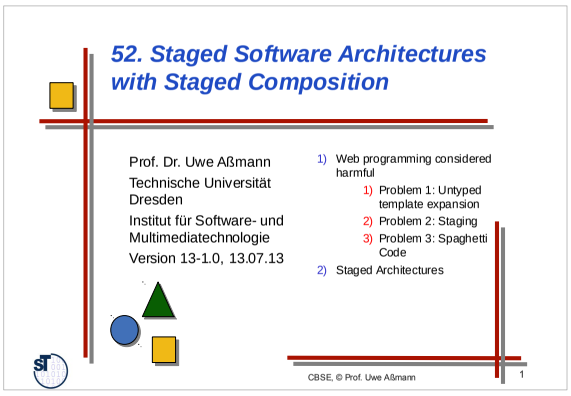
What Have We Learned?

- Large systems have *staged architectures* based on
  - *staged programming,*
  - *architectures,*
  - and *typed composition*
- On every stage, there is a component model and composition system
- All component models, composition systems and architectures have to work in synchronization
- Special cases:
  - The refinement-based software process (e.g., MDA)
  - Web systems, active documents
  - Invasive software composition
  - Standard build management

The Beauty of a Staged Programming Architecture

The End

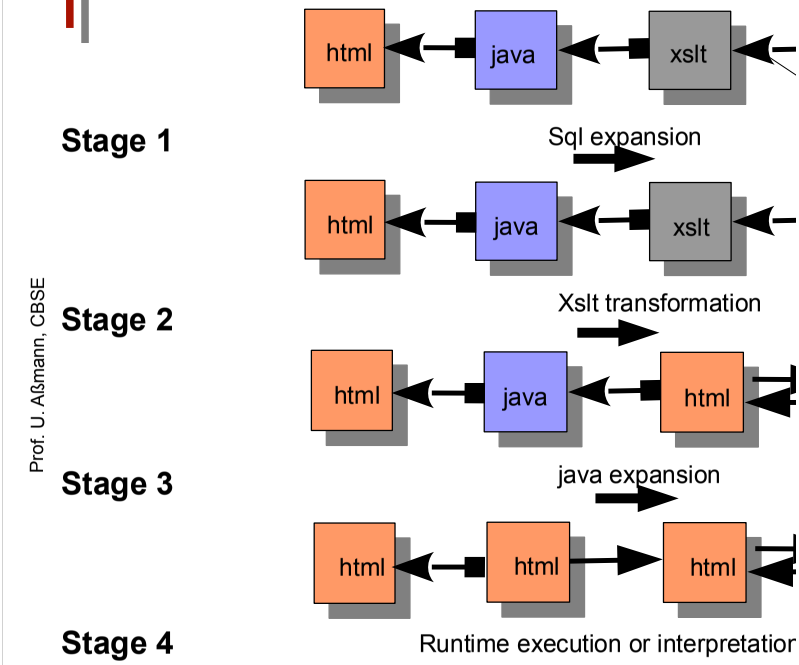
- www.easycomp.org
- http://www.the-compost-system.org
- U. Aßmann. Invasive Software Composition, 2003, Springer.
- U. Aßmann. Architectural Styles for Active Documents. Special Issue “Software Composition” Science of Computer Programming, Elsevier, 2005.
- Walid Taha. A Gentle Introduction to Multi-Stage Programming. Domain-Specific Program Generation, 2003, LNCS, pp. 30-50 http://www.springerlink.com/index/JEMT0D8VYN5JB2L8.pdf
- Tim Sheard: Accomplishments and Research Challenges in Meta-programming. SAIG 2001: Proceedings of the Second International Workshop on Semantics, Applications, and Implementation of Program Generation, pp. 2-44, LNCS 2196, Springer-Verlag, 2001.



A Staged Architecture from Nature

52.1 Web Programming Considei Harmful

Web Programming: Staged, Unty Expansion



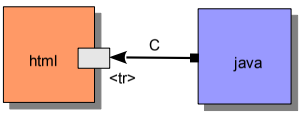
Problems of Web Programming

- Untyped extensions of templates
  - Error-prone
- Comprehension very difficult, due to the different stages
- Spaghetti-code-like programs
  - Scripts mixed with templates
  - Only valuable for programming-in-the-small

52.1.1 Problem 1: Untyped Template Expansion

Type-Safe Template Expansion

- How can you be sure that table rows are filled in?

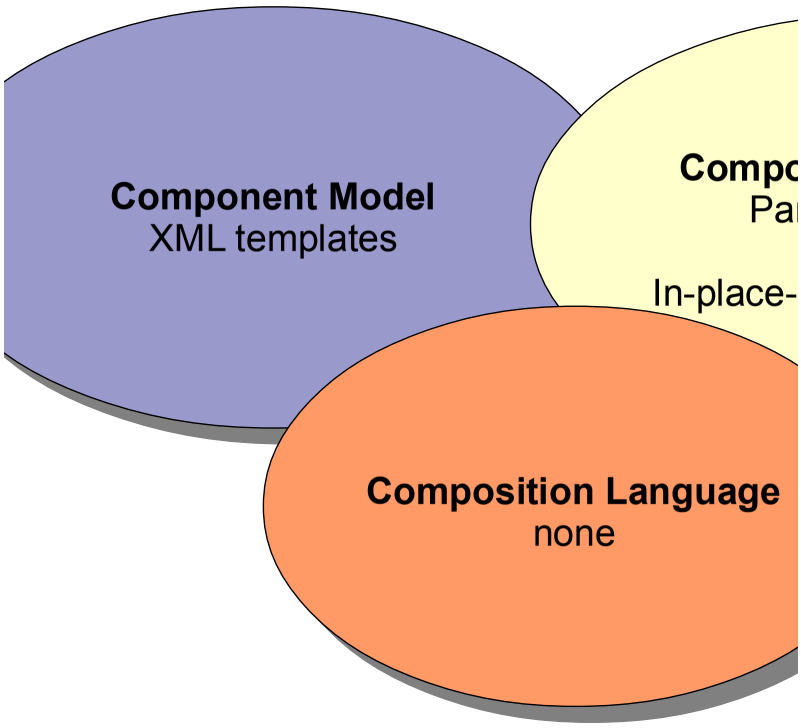


- Answer: in an invasive document composition system, the type checker of the invasive composition program will tell you, when checking the composition step C

Universality of Invasive Composition

- Invasive composition only depends on a metamodel of the language
  - New hook and slot models can be derived from any language
  - Typing controls the composition of artifacts
- Hence, the method is *universal*
- and can be applied for typed document composition
- See www.reuseware.org, the universal invasive composition environment,
  - Can be tailored for text-based and diagrammatic languages
  - OpenOffice
  - XML dialects
  - EMF-based

Elements of Web Composition Sys

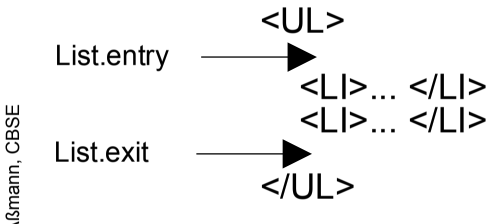


The Component Model of Invasive Composition

- The component is a fragment component (template)
  - A subword of the language, with *holes*
- Slots are variation points of a component
  - Parameters
  - Positions, which are subject to change
- Hooks are extension points
- Example:
  - A generic XML tree
  - A XML list with extension points

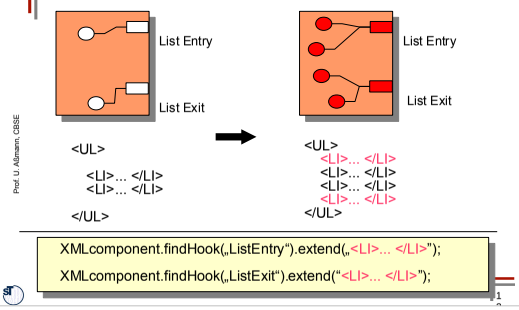
Extension of XML Fragment Com can be Typed

- What can be placed into an XML list entry/exit



Slot and hook types are given by an XSchema, a document

Typed Hook Expansion for XML Components



XMLcomponent.findHook(„ListEntry“).extend(„<LI>... </LI>“);

XMLcomponent.findHook(„ListExit“).extend(“<LI>... </LI>”);

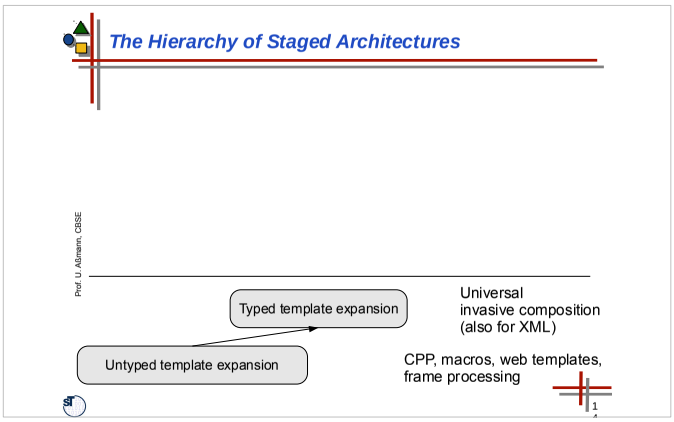
Prof. U. Aßmann, CBSE

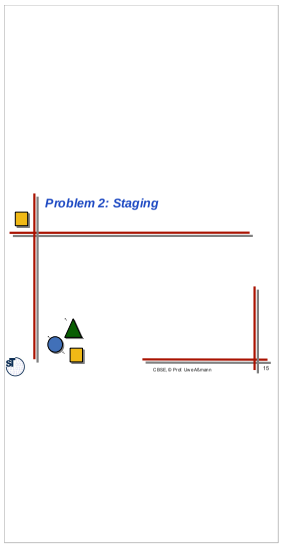
Insight: Web Systems Need Typed Template Processing

Problem: Web programming is based on *untyped template expansion (frame processing)*

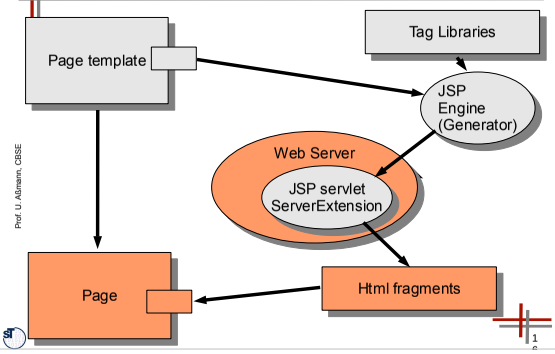
It should be based on typed template expansion (invasive composition)

Prof. U. Aßmann, CBSE


The Hierarchy of Staged Architectures
Prof. U. Aßmann, CBSE
Typed template expansion
Universal
invasive composition
(also for XML)
Untyped template expansion
CPP, macros, web templates,
frame processing


Problem 2: Staging
CBSE, © Prof. Uwe Aßmann
15

The JSP Mechanism



Spagetti Code from JSP Tutorial - Belongs to Different Execution Stages

```
<html>
<%@page language="java" imports="java.util.*" %>
<h1> Welcome! </h1>
<jsp:useBean id="clock" class="jspCalendar" />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
        Good Morning!
<% }else { %>
        Good afternoon...
<% } %>
</p>
<html>
```

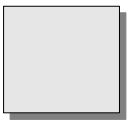
A Web Scripting Language with 5 Stages

```
<xfa1:profession>
  <xfa2:ref pop-up>
    <sql>select arbitrary lastName from bakers</sql> baker
  <xfa2:ref pop-up>
</xfa1:profession>
<xfa:function hello>
  <body>
  <h1>This is My Personal Page with XFA</h1>
  <xfa:if Odd(environment^DATE)>
    <xfa:ref message>
  <xfa:else>
    Even day. No money for <xfa1:profession> :-(
  </xfa:if>
  </body>
</xfa:function>
<xfa:function message>
  Odd day today, dear student. You may visit the <xfa1:profession> shop.
</xfa:function>
```

[until 2003: www.xml4all.com]

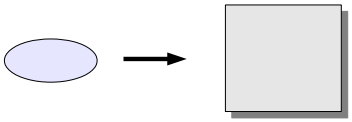
A Possible Solution: Staged Programming

- In the Beginning, there was the Data



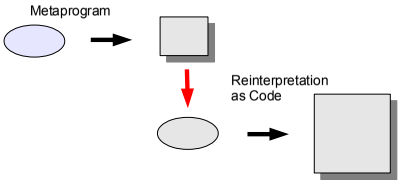
Then Came the Programs

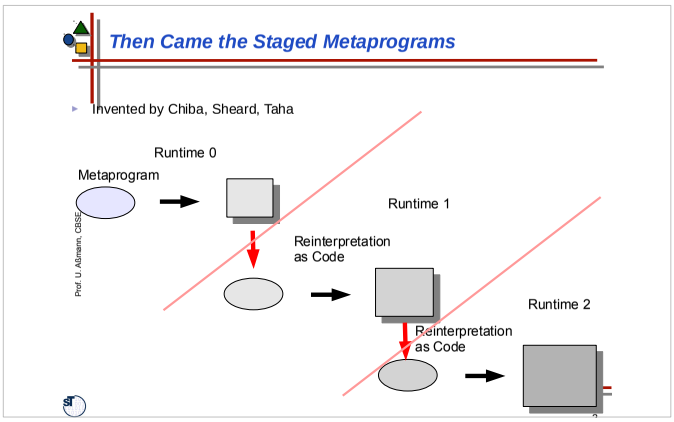
- Producing lots of data out of little code



Then Came the Metaprograms

- Producing lots of programs from few metaprograms



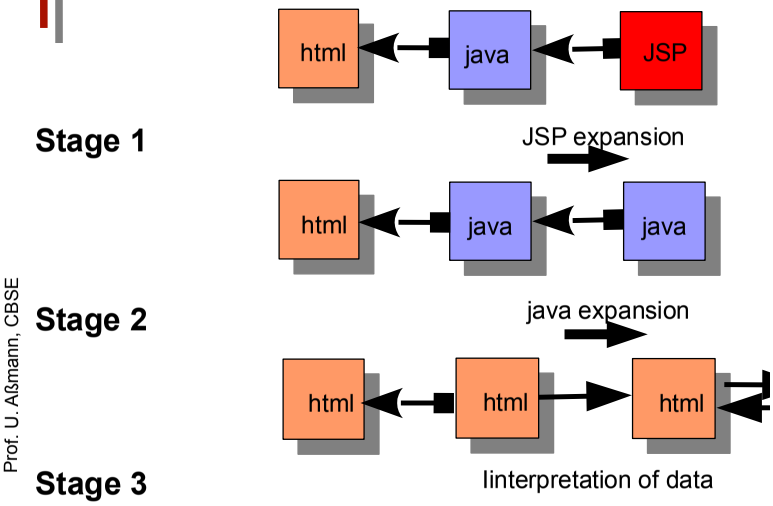


Staged Programming

- Staged programming (e.g., MetaML, MetaOCa
  static metaprograms and programs
  - The metaprograms are expanded statically (sta
    program (stage 2)
  - Metaprograms are typed in the metamodel of tl
    of metaprograms)
- Example [Taha]:

```
# let a = 1+2;;
val a: int = 3
# let a = .<1+2>.;;
val a: int code = .<1+2>.
# let b = .! a;;
val b = 3
```

JSP Uses Staged Programming

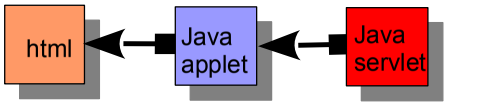


Spagetti Code Revisited

```
<html>
<%@page language="java" imports="java.util.*" %>
<h1> Welcome! </h1>
<jsp:useBean id="clock" class="jspCalendar" />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
        Good Morning!
<% }else { %>
        Good afternoon...
<% } %>
</p>
<html>
```

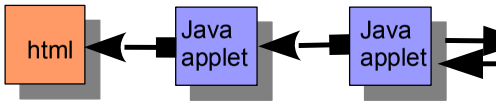
Servlet generator expands blue lines to Java code

Example 2: Staged Servlet/Applet Processin



Stage 1

JSP expansion



Stage 2

java expansion

interpretation of data

Insight 2: Web Systems Need Staged Programming

Web programming is often based on *staged programming*

- Because for dynamic web pages, code is generated
  - E.g., servlet or applet generation
- Because of the client-server stage separation
- Because legacy tools must be encapsulated into a stage (e.g., databases)

Staged programming should additionally be typed, otherwise it is chaotic

N.B.: Configuration and Variant Selection works with Staged Programming

```
# fun f variant =
  if variant = 1 then .<.fun q x = x*x.>.
                 else .<.fun q x = x/x.>.
;;
```

```
# let variant = 1;;
# fun g = (f variant) 2;;
val g: int code = .<let q x =
  x*x>.
# let res = g 3;;
val res = 9
```

Different behavior of second stage

```
# let variant = 2;;
# let g = (f variant) 2;;
val g: int code = .<let q x =
  x/x>.
# let res = g 3;;
val res = 1
```


Staging Is Used for Variant Management

On stage n-1, control-flow denotes variant selection for stage n

Platforms are often selected by evaluating control-flow in previous stages

Prof. U. Aßmann, CBSE

Spagetti Code Revisited

```
#ifdef HTML
<html>
#else
<wap>
#endif
<%@page language="java" imports="java.util.*" %>
#ifdef HTML
<h1> Welcome! </h1>
#else
<bold>Welcome!</bold>
#endif
<jsp:useBean id="clock" class="jspCalendar" />
#ifdef HTML
<p>
#endif
.....
```

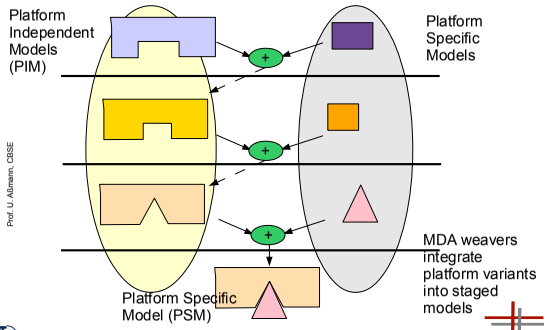
CPP stage selects HTML or WAP

Evaluating the CPP script chooses the platform

The C Preprocessor as Staged Programming System

- Insight: C with #ifdef language is a real staged programming system with CPP-C (State 0) and core-C (Stage 1)
  - That's why it's being used...
  - That's why it's so hard to deal with
- However, there is no component model, not even respect of the syntax of core-C
- The composition language of CPP-C is simple (macros, if-expressions, constant definitions)

A Staged Programming System: MDA

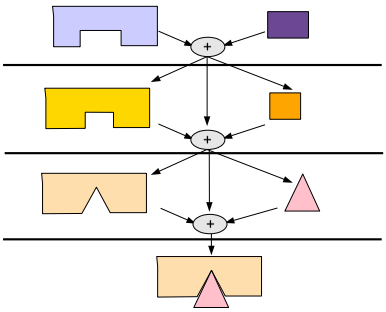


Staged Programming Architectures vs MDA

- MDA is a staged programming approach, but *not* a staged programming architecture, since no architecture, no component models are given
- ... but a staged programming technology for variant selection

... but we can build more powerful forms of MDA by taking in the ideas of staged programming and staged architectures

Staged Architectures Written as Layers



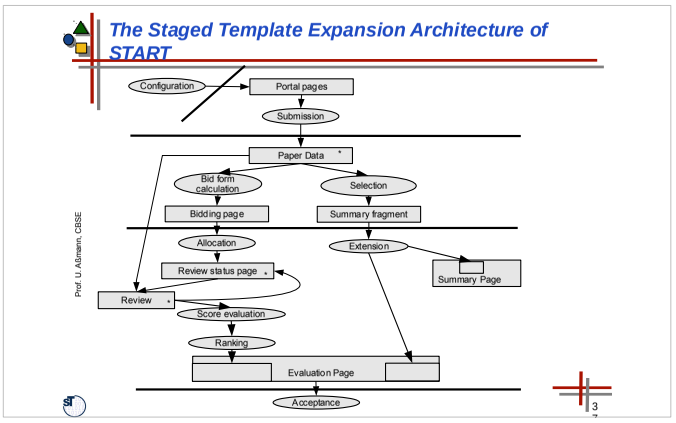
Advantages of Staged Programming

- Typed
  - Type-safe development, less error-prone
- Concise representation of system
  - Representation is expanded through every stage
- Easy to code variants
  - Control flow on a build stage does variant selection

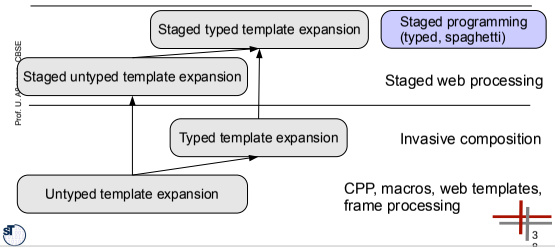
- Problems:
  - Still, lots of spaghetti code.

Example: The START Conferenc

- START is a review management system
  - It has a 5-phase staged template expansion ar
  - START servlets are composition scriptlets that html-templates
- Using invasive composition, we developed a *s expansion* system
- It is no problem to generate servlets, too. Ther programming

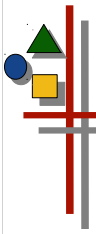


The Hierarchy of Staged Architectures



54.1.3 Problem 3: Spaghetti Code

and a possible remedy: staged architectures

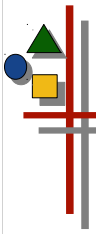


Architecture and Composition

- Two of the central insights of the software engi

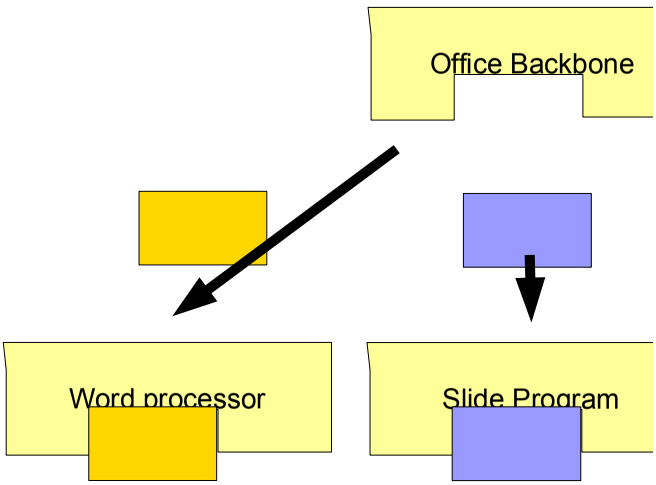
Separate architecture from the

Compose components by a *compo*



Benefit of Architectures

- Comprehensibility
- Commonalities into the architectural level, vari specific components
- Does this also hold for web programming?



Less Spaghetti Code: A Fragment-Based Template and its Architecture

Component

```
<html>
        <hook id="imports">
<h1> Welcome! </h1>
        <hook id=use">
<p> Today is
        <hook id="year"/>
        -<hook id="day"/>
</p>
<p>
        <hook id="greeting"/>
</p>
<html>
```

Composition Program (Architecture)

```
public class composeTemplate {
  String use = „jspCalendar"
  String imports="java.util.*";

  compose() {
    Template template = read();
    Bean clock = new jspCalendar();
    String year = clock.getYear();
    String day = clock.dayOfTheMonth();
    if (Calender.getInstance().get(Calendar.AM_PM) ==
      Calender.AM)
      greeting = "Good Morning!";
    else
      greeting = "Good afternoon...";
    this.merge(template);
  }
}
```

Prof. U. Aßmann, CBSE

Separation of Components and Architecture Allows for Variants

Composition Program (Architecture)

```
public class composeTemplate {
  String use =
  String imports=
  compose() {
    String year =
    String day =
      greeting =
  }
}
```

Component 1

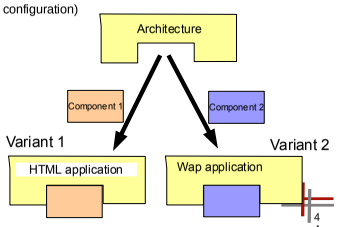
```
<html>
        <hook id="imports">
<h1> Welcome! </h1>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>    <hook id="greeting"/>
</p>
</html>
```

Component 2

```
<wap>
        <hook id="imports">
<bold> Welcome! </bold>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>    <hook id="greeting"/>
</p>
</wap>
```

Architecture and Variants in a Product Line

- Advantages for Separating Architecture From Application Components
  - Isolation of commonalities into frameworks
  - Comprehensibility
    - Programming-in-the-large is separated from programming-in-the-small, components can be abstracted away
    - Less spaghetti
  - Easy variability (variant configuration)



Variant Management by Control Flow in Architectural Composition Programs

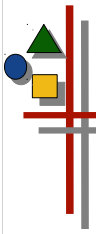
```
public class composeTemplate {
  if (HTML) then use component 1
            else use component 2
  String use =
  String imports=
  compose() {
    String year =
    String day =
      greeting =
  }
}
```

Variant 1

```
<html>
        <hook id="imports">
<h1> Welcome! </h1>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>   <hook id="greeting"/>
</p>
</html>
```

Variant 2

```
<wap>
        <hook id="imports">
<s1> Welcome! </h1>
        <hook id=use">
<p> Today is <hook id="year"/>
        -<hook id="day"/>
</p><p>   <hook id="greeting"/>
</p>
</wap>
```

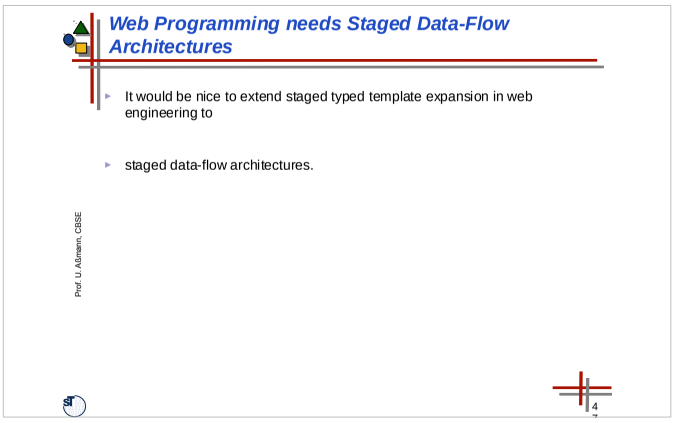


Definition: Staged Data-Flow Arc

Staged data-flow architectures add an exp
to staged template processing

- Every stage is executed to produce ***data*** for th
- Every stage is executed at a specific time
- On every stage, there is
  - an architecture,
  - a component model
  - a composition technique,
  - and a composition language
- Every composition language has its own interp
  - and is reduced (expanded) at different interpret

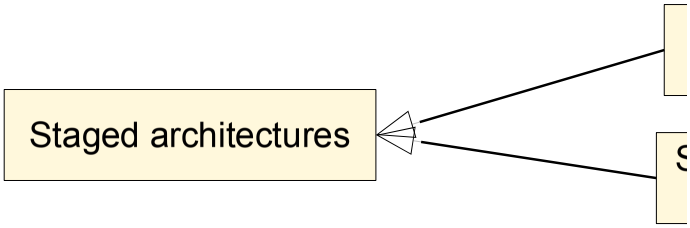
Prof. U. Aßmann, CBSE



Definition: Staged Architectures

Staged meta-programming architectures
programming with an explicit architectura

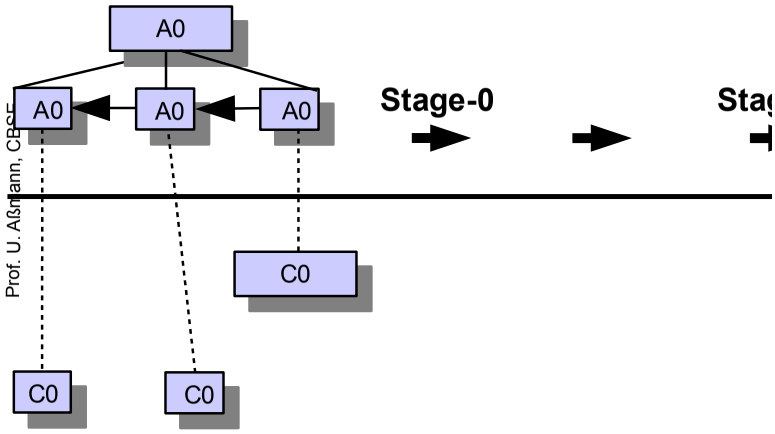
- Every stage is executed to produce ***code*** for tl
  - The final runtime code (architecture and compc stages
  - The initial architecture is very small, the final ar
  - Composition expressions, specifications, or prc components of a previous stage



Staged Metaprogramming Archit Large from Small

Stage-A0 architecture in composition language A0 Component language C0

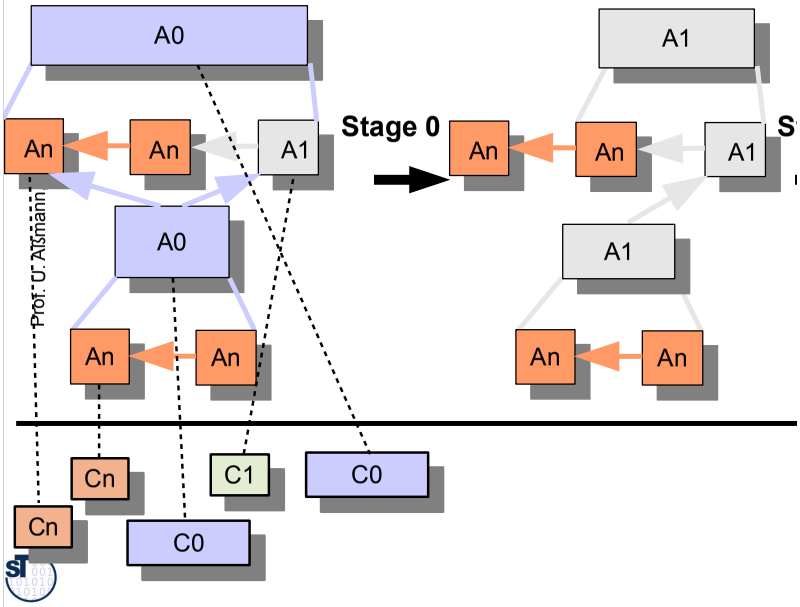
Gen Stag com Com

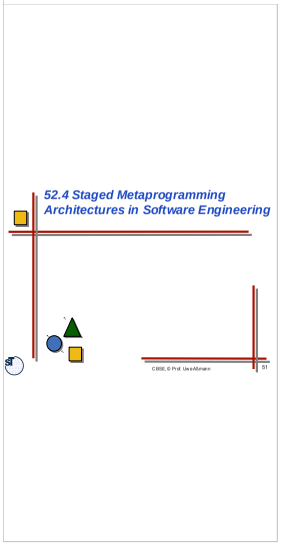


Staged Metaprogramming Archi
Different Component Models on

Stage-0 architecture in
composition language A0
Component language C0

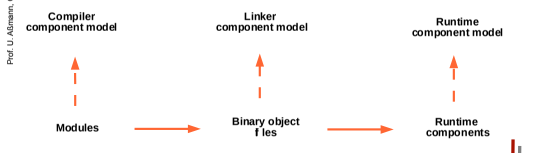
Stage 0 produces
Stage-1 architecture i
composition languag
Component language





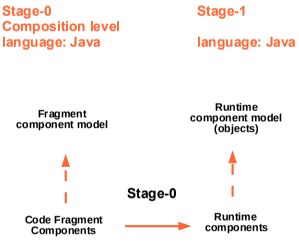
Build Management is Staged Composition

- Software build management is code composition in several stages
- Composition language: Make, ant, maven, etc.
  - Make is a composition tool with a lazy rule-based language
  - Expressions are applications of UNIX tools (compiler, linker, generator, preprocessor)
- Different component models on all stages



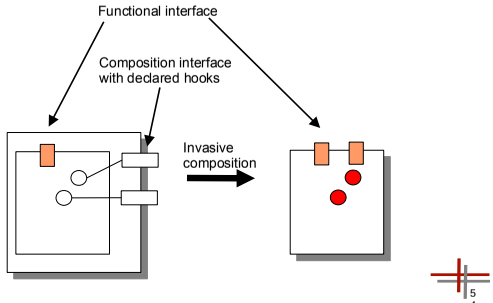
Invasive Software Composition

- Produces code from typed templates by parameterization and expansion



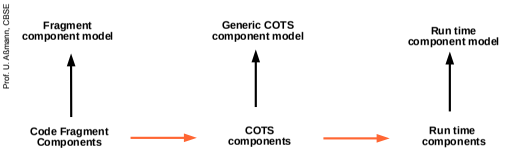
Invasive Composition Produces Functional from Composition Interfaces

- Two different component models



Component Models on Different Levels in the Software Process

- Standard COTS models are just models for binary code



The Dresden Staged Architecture Development Process

- Fix the stages
  - Decide on a staged processing or programming architecture
- Fix the component models for every stage
  - Interface concepts, composition operations, composition language
  - Design a concrete component model with Reuseware toolkit
- Fix the architectures
  - Decide on a composition language on each level
- Fix the variant management
- Fix the components

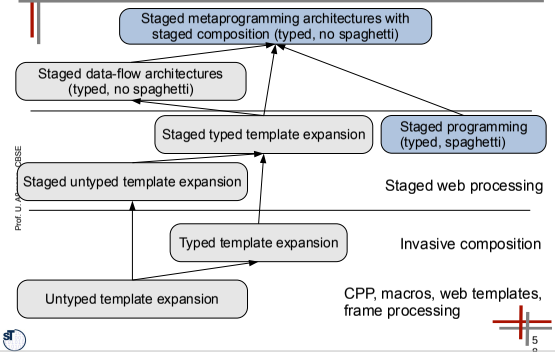
- And you'll have a pretty comprehensible product line!

The Vision of Staged Systems

- The staged programming principle is powerful, so future systems will employ it
- We need tools to support staged architectures
  - Visualize them
  - Debug them
  - Support the component models on all stages

  - .... that's a lot of work...

The Hierarchy of Staged Architectures



What Have We Learned?

- Large systems have *staged architectures* based on
  - *staged programming,*
  - *architectures,*
  - and *typed composition*
- On every stage, there is a component model and composition system
- All component models, composition systems and architectures have to work in synchronization
- Special cases:
  - The refinement-based software process (e.g., MDA)
  - Web systems, active documents
  - Invasive software composition
  - Standard build management

The End

- www.easycomp.org
- http://www.the-compost-system.org
- U. Aßmann. Invasive Software Composition, 2
- U. Aßmann. Architectural Styles for Active Do
  “Software Composition” Science of Computer
  2005.
- Walid Taha. A Gentle Introduction to Multi-Sta
  Specific Program Generation, 2003, LNCS, p
  http://www.springerlink.com/index/JEMT0D8V
- Tim Sheard: Accomplishments and Research
  programming. SAIG 2001: Proceedings of the
  Workshop on Semantics, Applications, and Im
  Generation, pp. 2-44, LNCS 2196, Springer-V