Aspect-Oriented Programming with C++

Olaf Spinczyk

University of Erlangen-Nuremberg
Computer Science 4

The AspectC++ Project

This talk is about ...

- the importance of C++ for the success of AOP
- different AOP approaches for C++
  - language independent vs. pure C++ vs. extension
- the AspectC++ project
  - history, language, implementation
- the future

Observations: AOP Products

- IBM: AspectJ and AJDT
- BEA: AspectWerkz
- JBoss: JBoss AOP

Observations: AOP Products

- IBM: AspectJ and AJDT
- BEA: AspectWerkz
- JBoss: JBoss AOP

Observations: AOP Products

... for C++ developers:

- Semantic Designs: DMS
- P&P Software: XWeaver
- pure-systems: AspectC++ Add-In

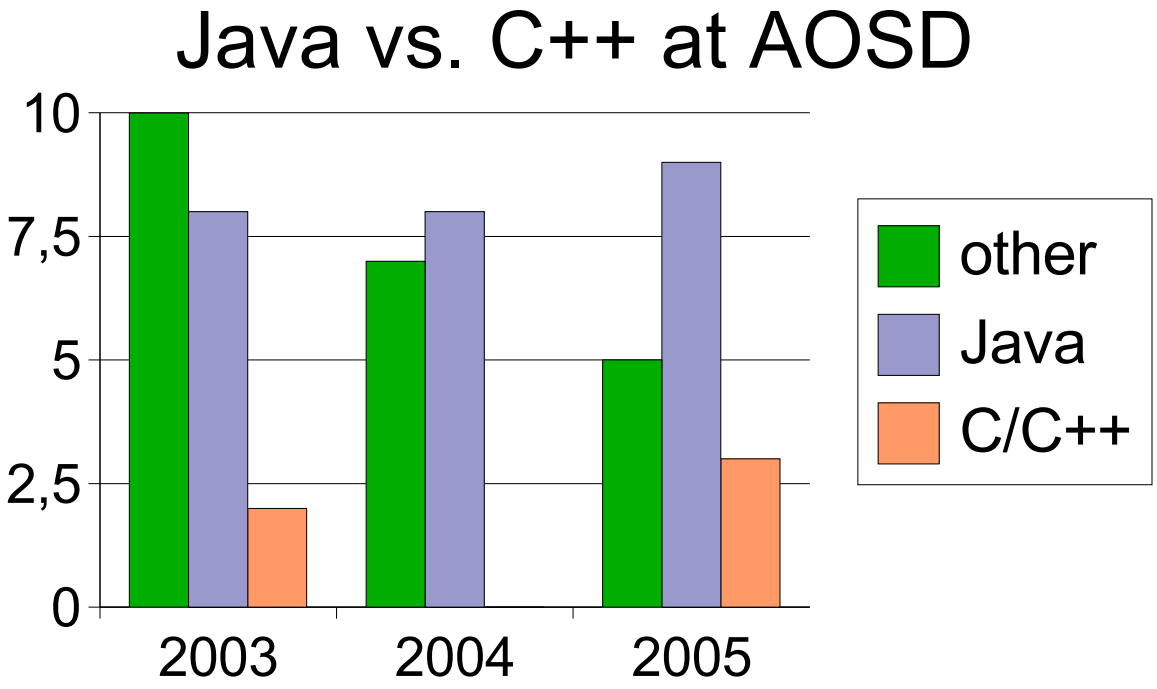
Observations: AOP Products

... for C++ developers:

- Semantic Designs: DMS
- P&P Software: XWeaver
- pure-systems: AspectC++ Add-In

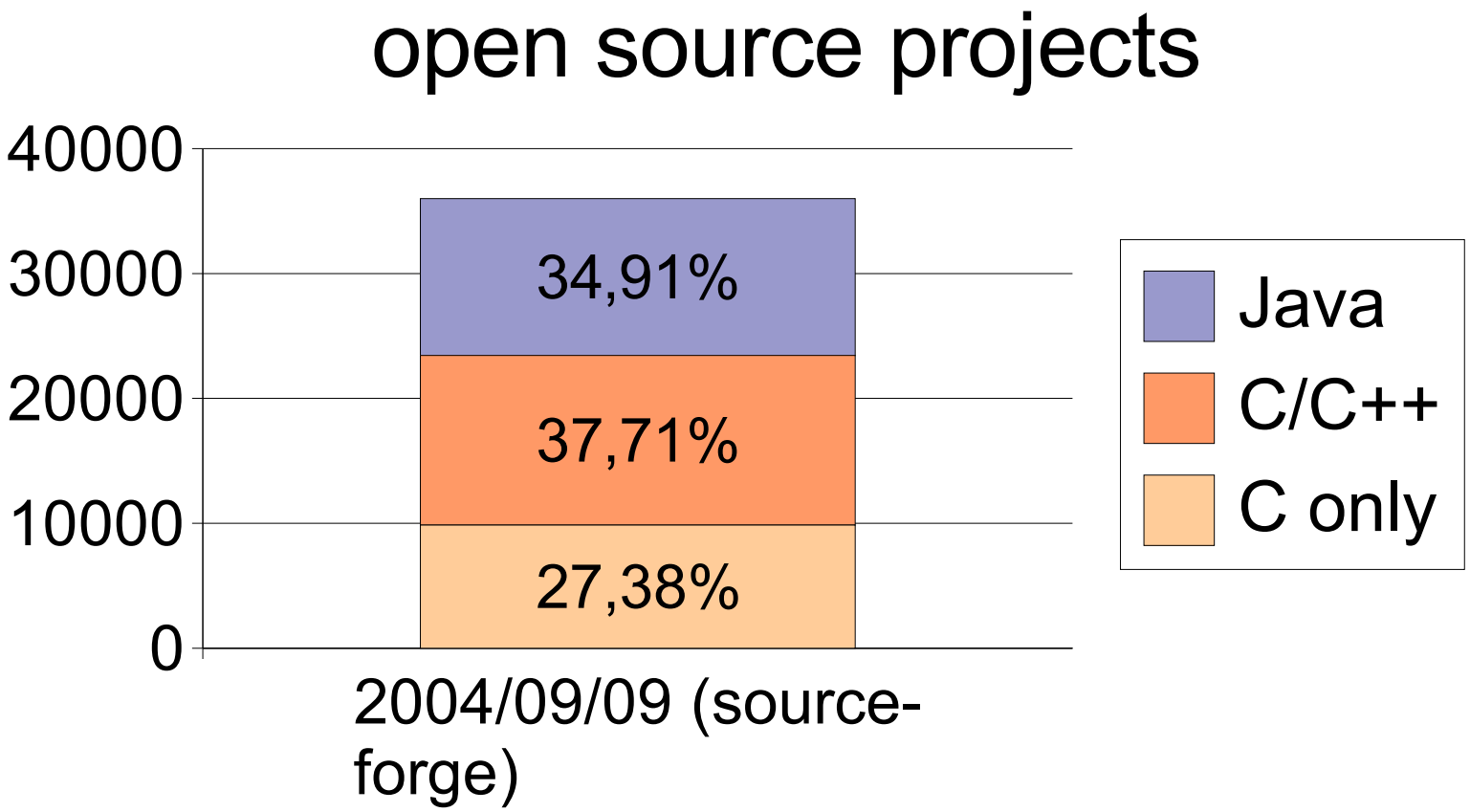
these teams play in a different league

Observations: AOP Research

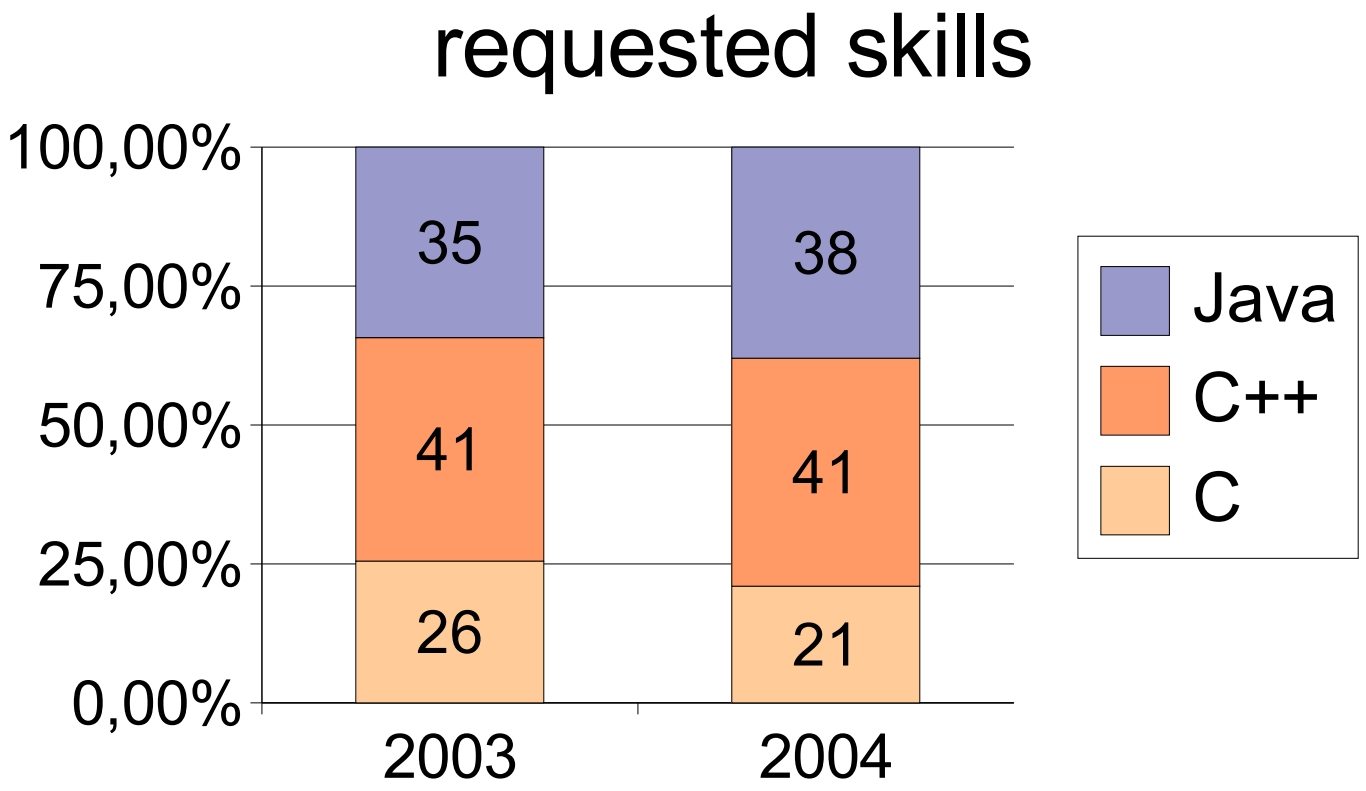


C++ is almost invisible!

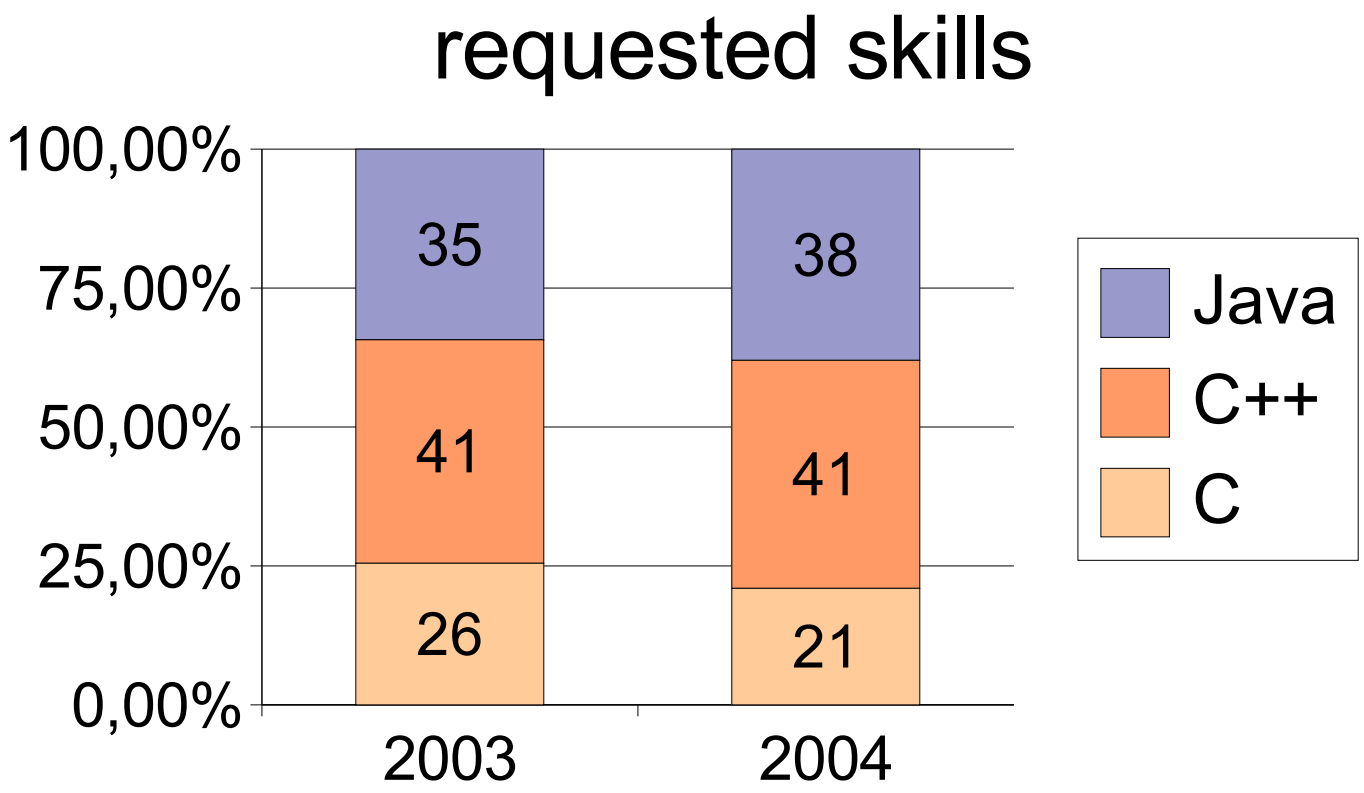
Java vs. C++ in the Real World



Java vs. C++ in the Real World



Java vs. C++ in the Real World



C and C++ are still dominating!

Java vs. C++ in the Real World

AOP research & products don't reflect the real world.

Why?

Reasons

C++ is one of the most complex languages today

- developing tools and extensions is painfully hard
- C++ is not common in academic research
- no transfer from academia to industry

The Problem

AOP research & products don't reflect the real world.

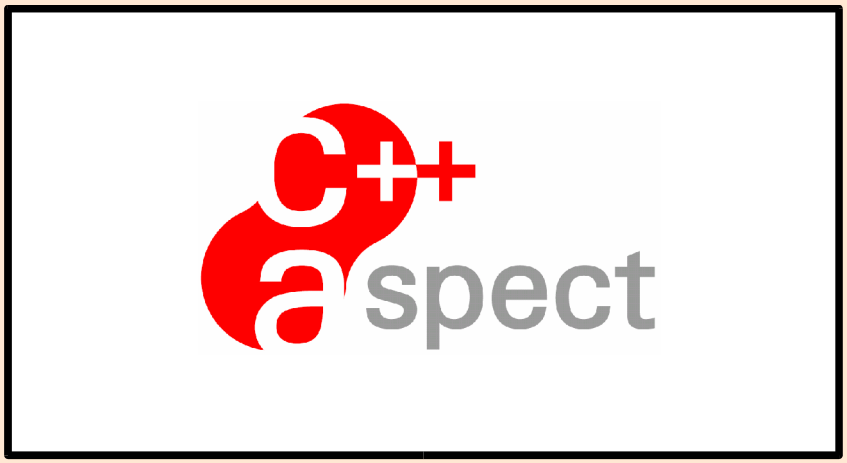
- the **unwanted** message is:

  AOP is Java.

- the **unwanted** consequences are:
  - no large scale adoption by the IT industry
  - billions lines of C/C++ code don't benefit from AOP

* WANTED *

~~DEAD OR~~ ALIVE



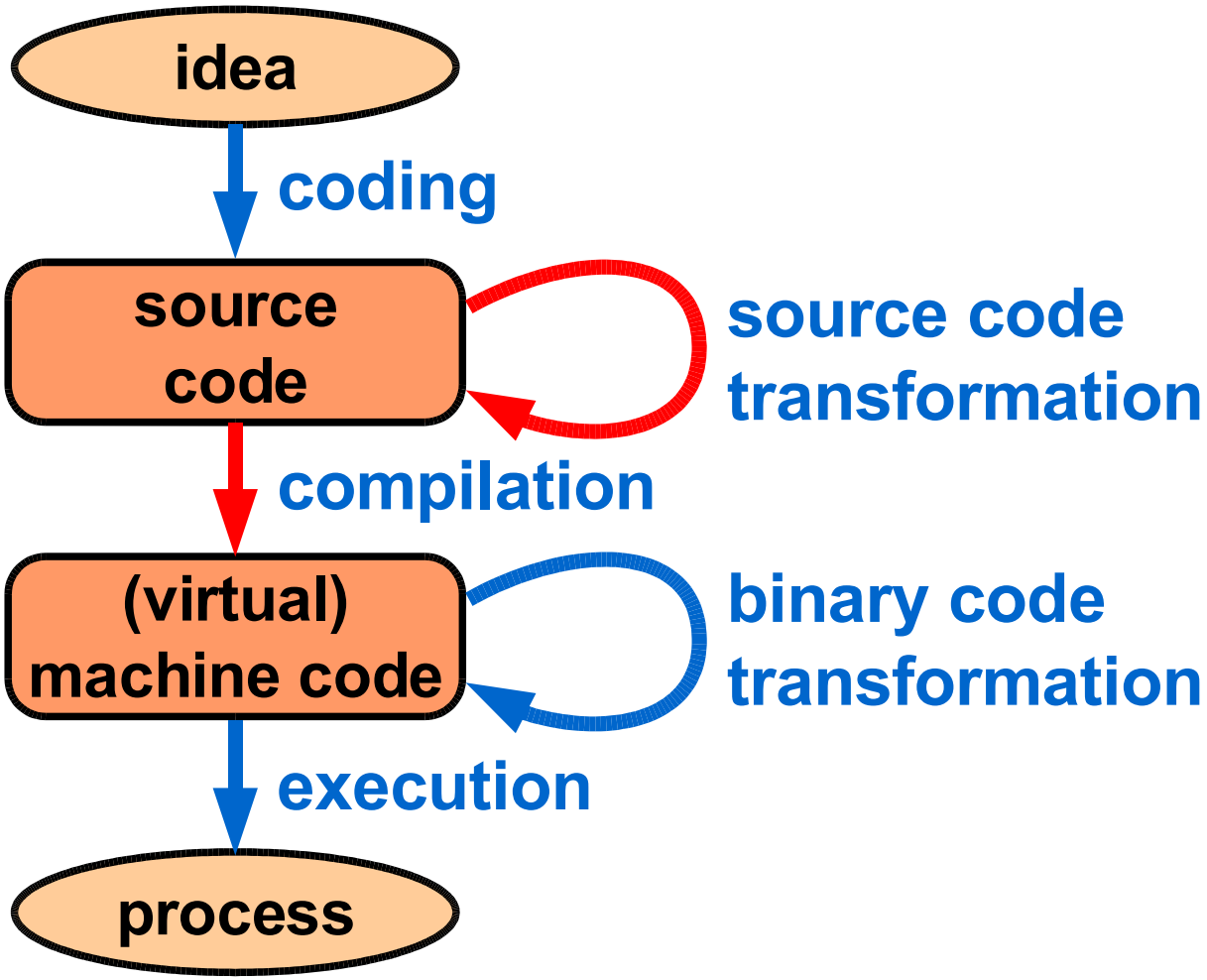
AspectC++

Requirements

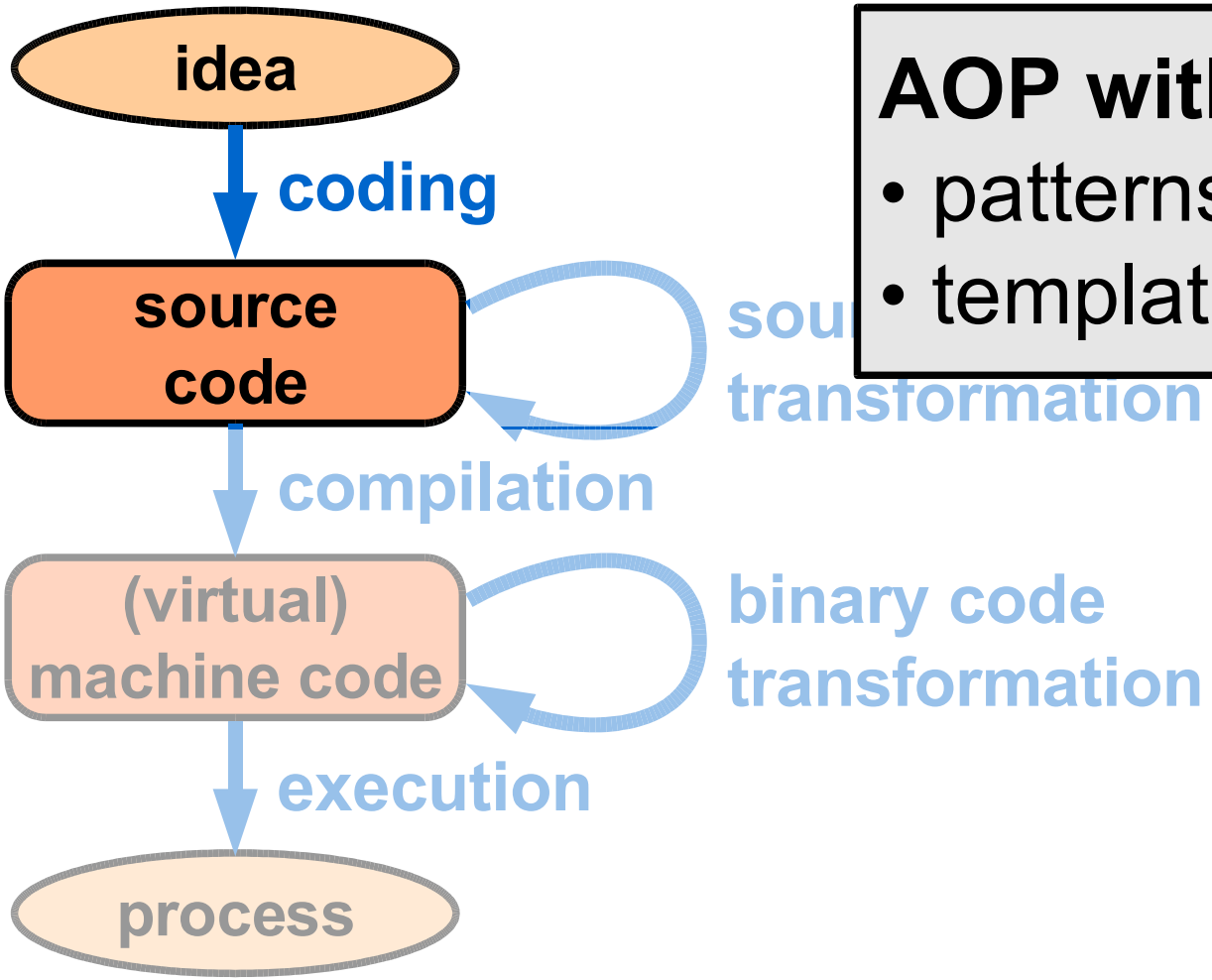
an AOP solution for C++ has to ...

- support full **obliviousness** and **quantification**
  - no preparation of the component code
  - rich pointcut language
- be **strong** were C++ is strong
  - no runtime system
  - support for procedural, object-oriented, and generic code
  - exploit and support the powerful static type system
  - efficient code
- be **usable**
  - simple
  - easy integration

Technical Approaches



Technical Approaches



AOP with pure C++

- patterns
- templates, macros

AOP with pure C++ (1)

C. Czarnecki, U. Eisenecker, L. Dominick:

```
// generic wrapper (aspect) that adds counting to
// any queue class Q, as long as it implements the
// proper interface

template <class Q>
class Counting_Aspect : public Q {
  int counter; // introduction
public:
  void enqueue(Item* item) { // after advice
    Q::enqueue(item); counter++;
  }
};
```

AOP with pure C++ (2)

aspect weaving by template instantiation

```
// component code
class Queue { ... }

// wrappers (aspects)
template <class Q>
class Counting_Aspect : public Q { ... }
template <class Q>
class Tracing_Aspect : public Q { ... }

// template instantiation (weaving)
typedef Counting_Aspect<Queue> CountingQueue;
typedef Trace_Aspect<Counting_Queue> TraceCountingQueue;
```

AOP with pure C++ (3)

obliviousness for the client code

```
namespace components {
   class Queue { ... };
}
namespace aspects {
  template <class Q> class Counting_Aspect : public Q { ... };
}
namespace configuration { // select counting queue
   typedef aspects::Counting_Aspect<components::Queue> Queue;
}

using namespace configuration;
void client_code () {
  Queue queue; // Queue with all configured aspects
  queue.enqueue (new MyItem);
}
```

AOP with pure C++ (4)

C. Diggins: macros hide the template “magic”

```
// the CountingAspect as before
struct CountingAspect {
  // Inc and Dec is advice
  struct Inc { template<...> virtual void OnAfter (...) { ... } };
};

// the DEF_POINTCUT macro describes sets of member functions
DEF_POINTCUT(EnqueuePointcut)
  SET_PROCJOINPOINT1(enqueue, Item*, item)
END_POINTCUT

// the CROSSCUT macro combines a class, a pointcut, and an aspect
typedef CROSSCUT(Queue,EnqueuePointcut,
                 CountingAspect::Inc) CountingQueue;
```

AOP with pure C++ - Review

obliviousness: not given

- component code has to be “prepared”

quantification: not given

- aspects have to be applied manually

strong: not really

- basically supports weaving in
  public (virtual) class member functions

usable: not often

+ no special tool support required

- code is hard to develop, understand, and maintain

AOP with pure C++ - Review

obliviousness: not given

- component code has to be “prepared”

quantification: not given

- aspects have to be applied manually

no tool needed, but too many restrictions

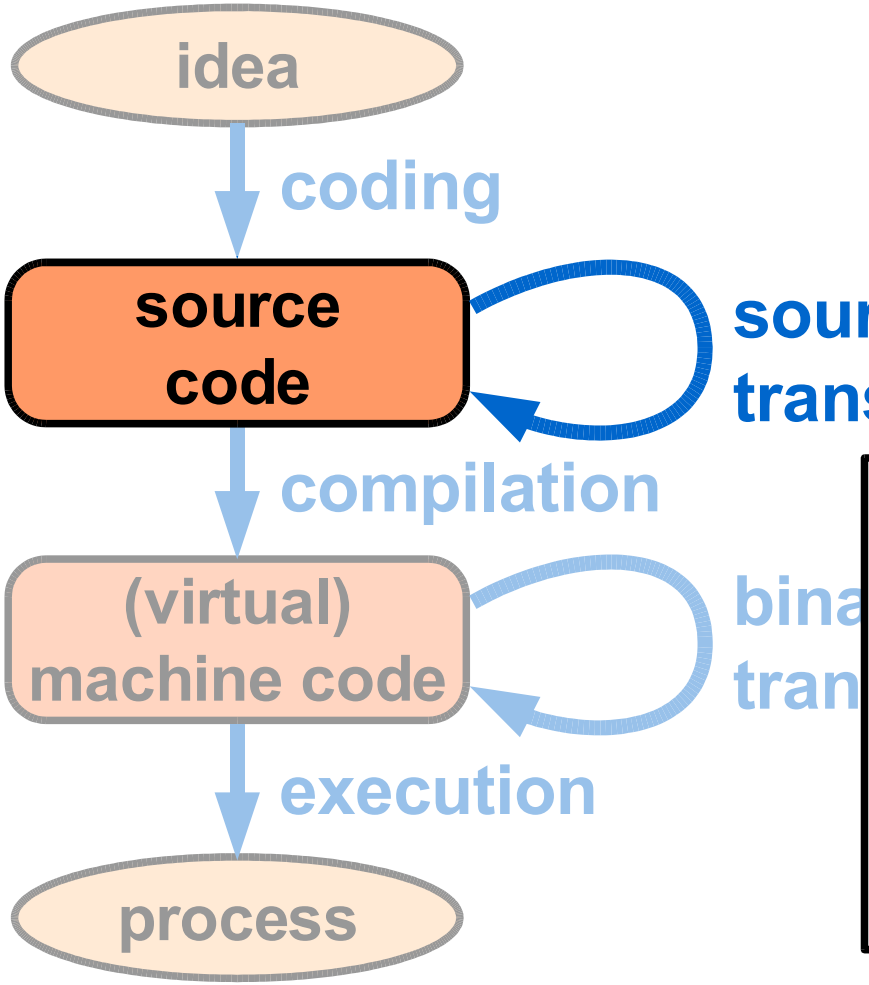
- basically supports weaving in public (virtual) class member functions

usable: not often

+ no special tool support required

- code is hard to develop, understand, and maintain

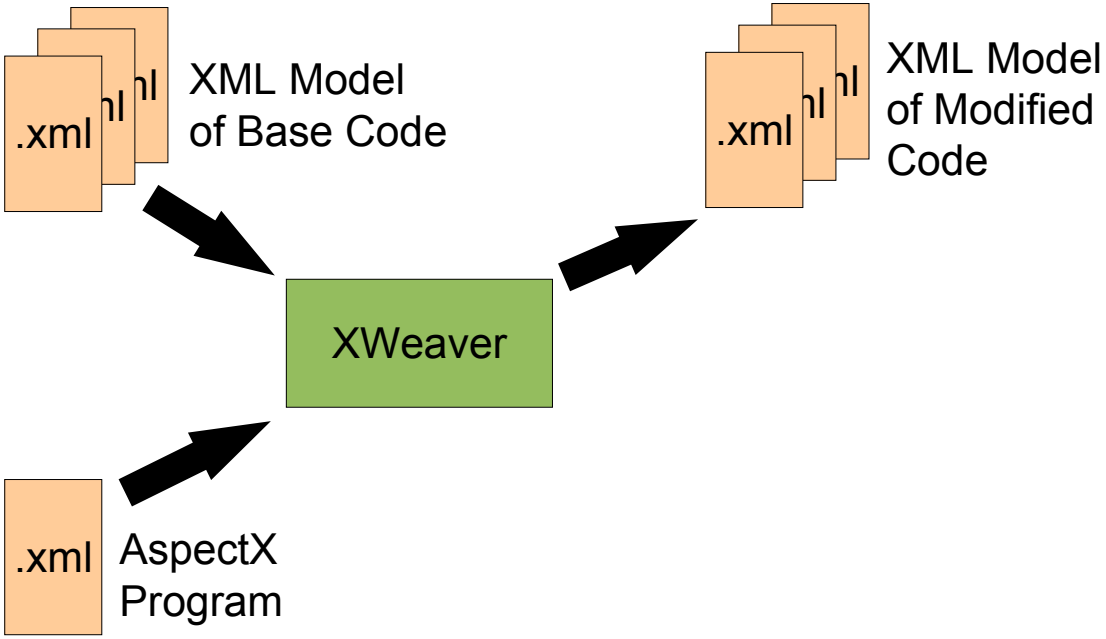
Technical Approaches



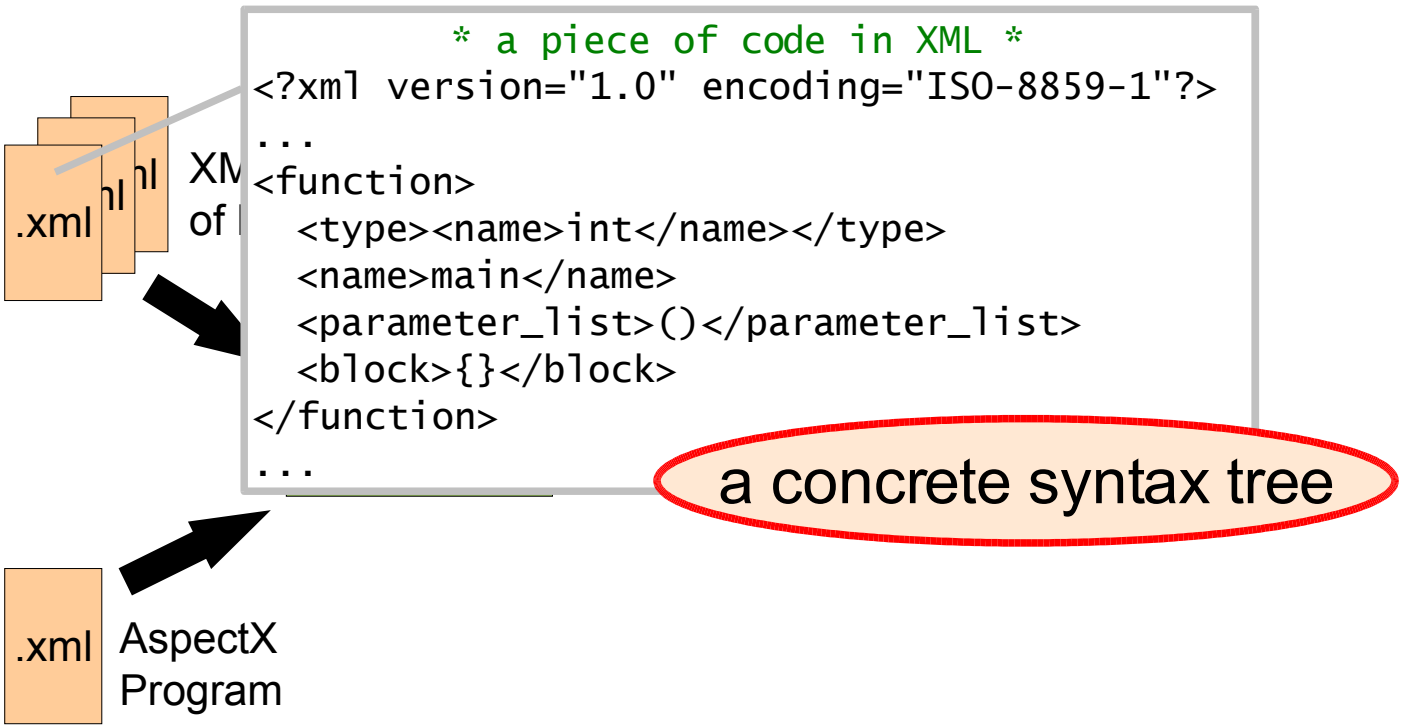
Code Transformers

- AspectC++
- XWeaver
- DMS

XWeaver (1)

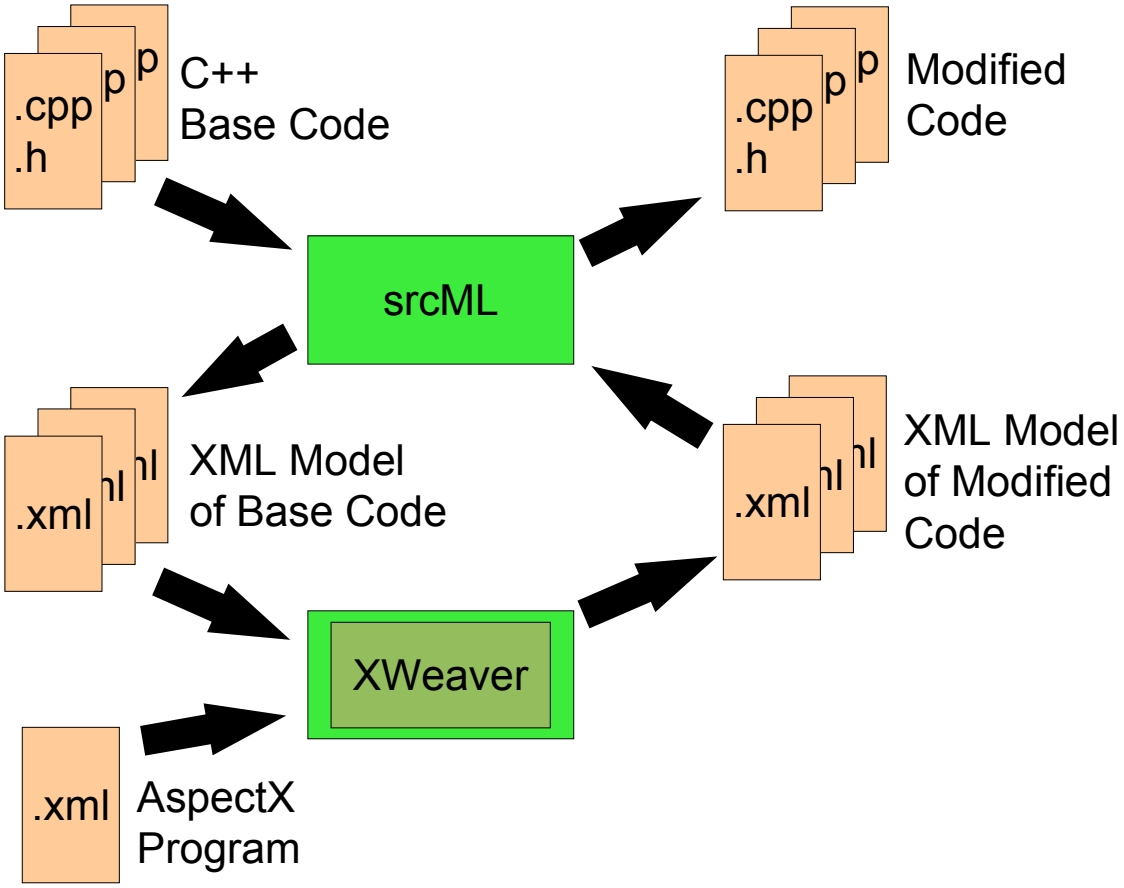


XWeaver (1)



XWeaver (2)

- the language-dependent part



XWeaver (3)

current limitations:

- only supports “embedded C++”
- strongly limited join point model
  - function/constructor/destructor execution only

reason: srcML parser problems

- no semantic analysis, no function call resolution **no call advice**!
- E.g. “`int (*f())(long) {}`” yields **nonsense**

XWeaver (3)

current limitations:

- only supports “embedded C++”
- strongly limited join point model
  - function/constructor/destructor execution only

The transformation approach is only viable with a **fully-fledged C++ parser/analyzer**.

  - **no call advice**!
- E.g. “`int (*f())(long) {}`” yields **nonsense**

Code Transformation - Review

obliviousness: possible

+ weaver has full control

quantification: possible

+ only a matter of aspect language features

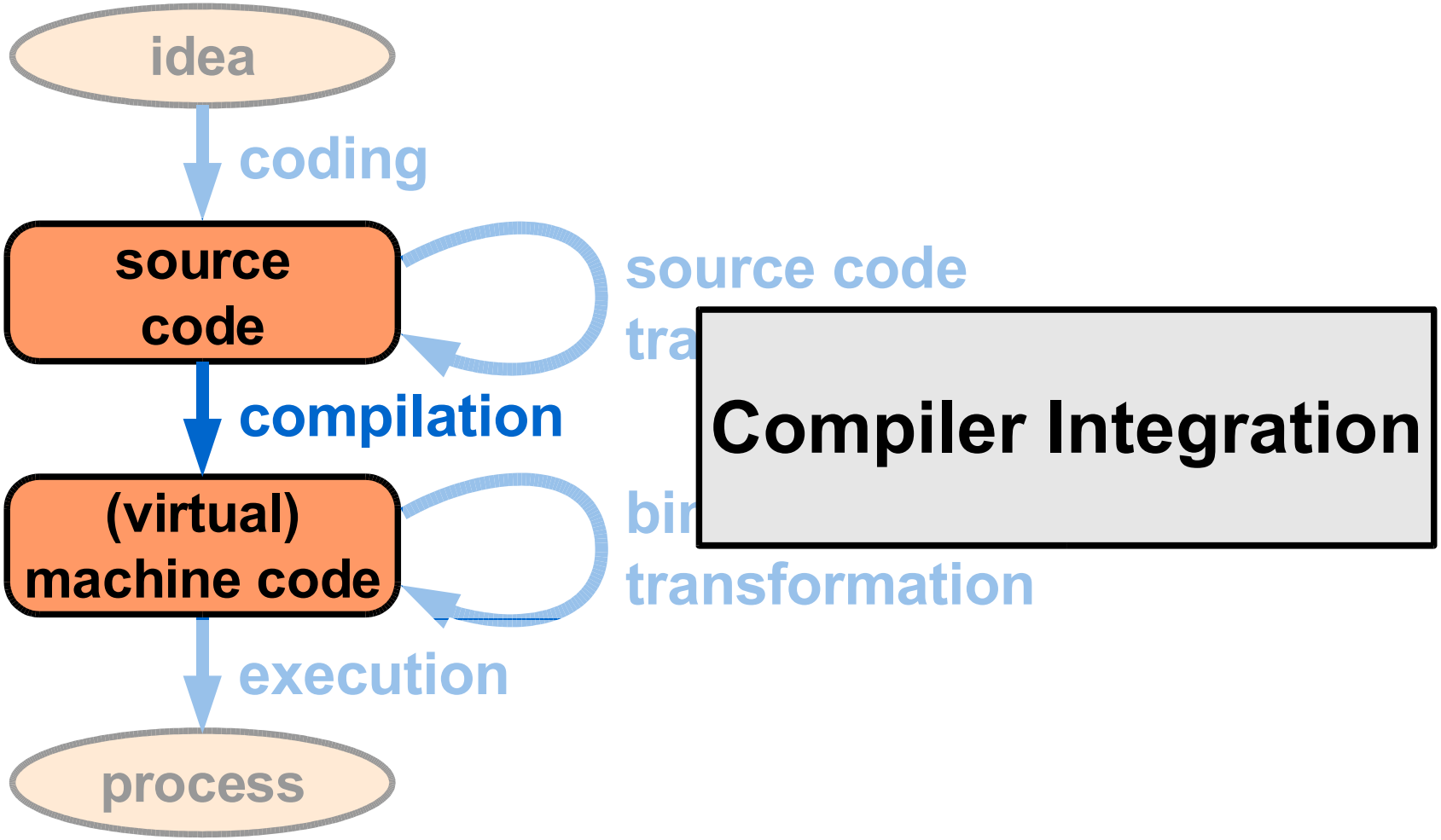
strong: definitely

+ generated code can be as efficient as tangled code

usable: yes

+ AspectJ-like programming model possible
+ easy integration into existing tool chains
+ platform-independent

Technical Approaches



Compiler Integration

Could an AOP extension for C++ make it into...

- commercial compilers?
- the C++ standardization?

ISO/IEC JTC1/SC22/WG21

- very busy with problems like “A<B<C>>”
- Detlef Vollmann summarizes “Aspects of Reflection in C++”
- Daveed Vandevoorde presents a “Metaprogramming Extension” at the ACCU conference
- besides that: **no revolutions**

Compiler Integration

Could an AOP extension for C++ make it into...

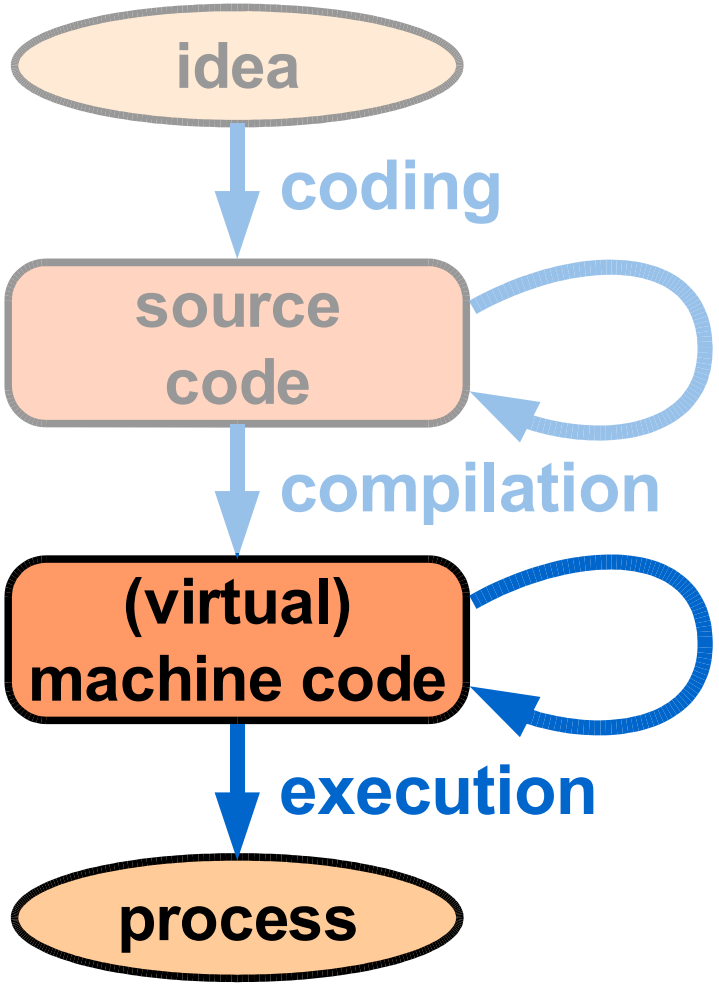
- commercial compilers?

The transition from C to C++ took many years.

We should not expect this process to be faster in the case of AOP.

- Daveed Vandevoorde presents a “Metaprogramming Extension” at ACCU
- besides that: **no revolutions**

Technical Approaches



Language-independent AOP

binary code transformation

Runtime Weaving

- Arachne

Weaving in Byte/Machine Code

completely decouples the weaver from the parser, but ...

- aspect weaving in **machine code** ...
  - strictly limits available AOP features
  - has to be implemented for numerous platforms
- aspect weaving in **virtual machine code** ...
  - is not feasible in most C++ dominated domains
  - compromizes the strengths of C++

Runtime Aspect Weaving

- same restrictions as static binary code weaving
- advantage:
  - dynamicity – needed in some application scenarios
- disadvantage
  - additional runtime system required

Runtime Aspect Weaving

- same restrictions as static binary code weaving
- advantage:

Many C++ projects have problems
with crosscutting concerns.

Dynamic weaving is nice to have, but we should come up with a viable static AOP solution first.

Binary Code Weaving - Review

obliviousness: given

quantification: limited

- restricted set of join point types and possible transformations
- introductions are a huge problem

strong: not really

- binary code weaving conflicts with code optimization
- potential loss of static type information

usable: specific cases

- highly platform dependent
- aspect program expressivenes depends on machine model

Summary

	pure C++	source level	binary level
obliviousness	-	+	+
quantification	-	+	o
strong	-	+	-
usable	o	+	-

Summary

	pure C++	source level	binary level
obliviousness	-	+	+
quantification	-	+	o
strong	-	+	-
usable	o	+	-

Sustained success of AOP in the C++ world requires:

- a source level weaving approach
- a convincing freely available implementation

The AspectC++ Project

"We are definitely not targeting C++ for our work."

(Gregor Kiczales, July 2001)

- Language Level Goals:
  - AspectJ-like syntax and semantic
  - AspectC++ should fit well into the C++ philosophy
- Implementation Level Goals
  - support for various hardware platforms and C++ dialects
  - IDE integrations

AspectC++ vs. AspectJ

AspectJ

```
aspect SimpleTracing {
    pointcut tracedCall() :
        call(void FigureElement.draw(GraphicsContext));
    before() : tracedCall() {
        System.out.println("Entering: " + thisJoinPoint);
}   }
```

AspectC++

```
aspect SimpleTracing {
    pointcut tracedCall() =
        call("void FigureElement::draw(GraphicsContext&)");
    advice tracedCall() : before () {
        cout << "Entering: " << JoinPoint::signature ());
}   };
```

The C++ Philosophy

- C compatibility
  - re-use of billions lines of code
  - but: untyped pointers, preprocessor, ...
- strong focus on static typing
  - generic programming
  - function and operator overloading
- multi-paradigm development
  - object-oriented, procedural, generic
- generative programming
- efficiency in time and space

Consequences for AspectC++

AspectC++ has to cope with the C++ philosophy

- weaving in C-style code
- statically typed aspect implementations
  - “generic advice”
- multi-paradigm AOP
  - advice for C-style functions
  - advice for classes and objects
  - advice for generic code and template instances
  - advice for operator functions and conversion functions
  - ...
- generation of efficient code

AspectC++ – Joinpoint API

Compile-Time Joinpoint API

JoinPoint::That	Type of affected class (call/execution)
JoinPoint::Target	Type of the target class (call)
JoinPoint::Result	Type of the function result
JoinPoint::Arg< ***i*** >::Type	Type of the i^{th} function argument
JoinPoint::Arg< ***i*** >::ReferredType	(with 0 ≤ i < ARGS)
JoinPoint::ARGS	Number of arguments
JoinPoint::JPID	Unique identifier for this joinpoint
JoinPoint::JPTYPE	Type of the joinpoint (call/execution)

Runtime Joinpoint API

That* that()	current object instance
Target* target()	target object instance (call)
Result* result()	result value
Arg< ***i*** >::ReferredType* arg< ***i*** >()	value of i^{th} argument

...

AspectC++ – Joinpoint API

Compile-Time Joinpoint API

JoinPoint::That	Type of affected class (call/execution)
JoinPoint::Target	Type of the target class (call)
JoinPoint::Result	Type of the function result
JoinPoint::Arg< *i* >::Type	Type of the i^{th} function argument
JoinPoint::Arg< *i* >::ReferredType	(with 0 ≤ i < ARGS)
JoinPoint::ARGS	Number of arguments
JoinPoint::JPID	Unique identifier for this joinpoint
JoinPoint::JPTYPE	Type of the joinpoint (call/execution)

Complete signature of the affected function is available

Runtime Joinpoint API

That* that()	
Target* target()	
Result* result()	
Arg< *i* >::ReferredType* arg< *i* >()	value of i^{th} argument

...

AspectC++ – Joinpoint API

Compile-Time Joinpoint API

JoinPoint::That	Type of affected class (call/execution)
JoinPoint::Target	Type of the target class (call)
JoinPoint::Result	Type of the function result
JoinPoint::Arg< *i* >::Type	Type of the i^{th} function argument
JoinPoint::Arg< *i* >::ReferredType	(with 0 ≤ i < ARGS)
JoinPoint::ARGS	Number of arguments
JoinPoint::JPID	Unique identifier for this joinpoint
JoinPoint::JPTYPE	

Type-safe access to actual values at runtime

Runtime Joinpoint API

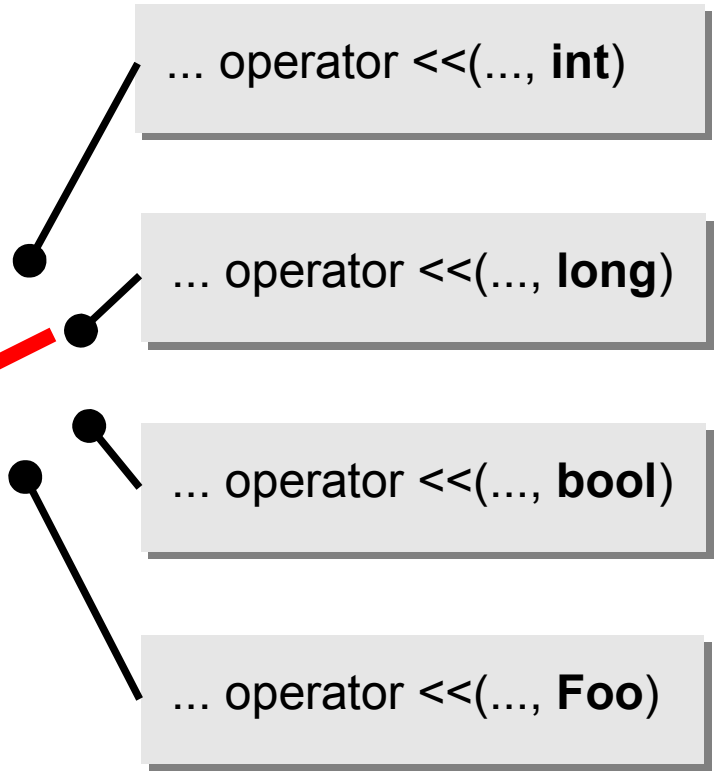
That* that()	
Target* target()	target object instance (call)
Result* result()	result value
Arg< *i* >::ReferredType* arg< *i* >()	value of i^{th} argument

...

Generic Advice

A compile-time switch with overloaded functions

```
aspect TraceService {
  advice call(...) : after() {
    ...
    cout << *tjp->result();
  }
};
```



Generic Advice

A compile-time switch with overloaded functions

```
aspect TraceService {
  advice call(...) : after() {
    ...
    cout << *tjp->result();
  }
};
```

... operator <<(..., **int**)

... operator <<(..., **long**)

... operator <<(..., **bool**)

... operator <<(..., **Foo**)

- **no runtime type checks are needed**
- **unhandled types are detected at compile-time**
- **functions can be inlined**

Generic Advice

Instantiation of template metaprograms

```
template<class TJP, int i> struct ArgPrinter {
  static void work( TJP* tjp ) {
    ArgPrinter<TJP, i-1>::work(tjp);
    cout << ", " << *tjp->arg<i>();
  }
};
template<class TJP, 0> struct ArgPrinter {...};

aspect TraceService {
  advice call(...) : after() {
    ...
    ArgPrinter<JoinPoint, JoinPoint::ARGS>::work(tjp);
  }
};
```

Generic Advice

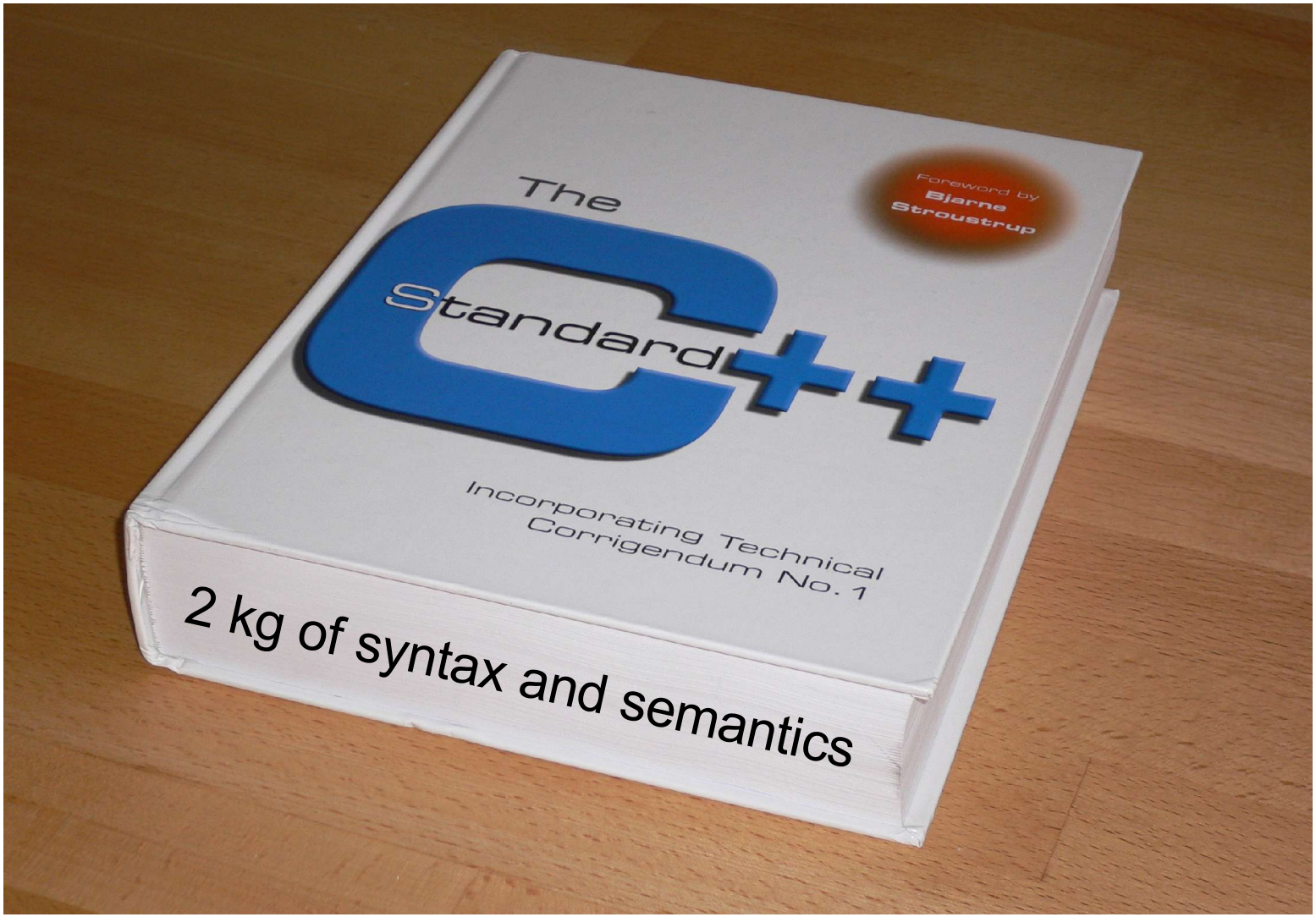
Instantiation of template metaprograms

```
template<class TJP, int i> struct ArgPrinter {
  static void work( TJP* tjp ) {
    ArgPrinter<TJP, i-1>::work(tjp);
    cout << ", " << *tjp->arg<i>();
  }
};
ter

asp
  a
    ...
    ArgPrinter<JoinPoint, JoinPoint::ARGS>::work(tjp);
  }
};
```

full power of template metaprogramming is available for aspects

AspectC++ Implementation



Dealing with Real-World C++ Code

- a standard compliant C++ parser is HUGE
  - currently 70.000 lines of AspecC++ code
- even commercial compilers are not fully compliant
  - EDG announced to have the first fully standard compliant parser a few years ago!
- compiler-specific language extensions
- the standard is interpreted differently

State of the Implementation

- works
  - parser handles real-world code
  - rich aspect language
- does not work yet
  - weaving in template instances
- should be improved
  - performance
  - dependency management
  - weaving in C code

AspectC++ IDEs

- AspectC++ Add-In for Visual Studio .NET
  - commercial Visual Studio extension by pure-systems GmbH
- AspectC++ Development Tools for Eclipse (ACDT)
  - open source Eclipse plugin (demo!)

User Community

- 150 subscribed AspectC++ users
  - most of them from ...com
- 500 downloads of AspectC++ 0.9.1 (binary version) since published at february, 10th
- application areas
  - mobile phones and PDAs: Nokia, Siemens
  - telecommunications: Samsung
  - real-time databases: Linkøping University
  - operating systems: Unversity of Erlangen (CiAO, ECOS, L4)

AspectC++ in the Future

- documentation
  - language
  - resource consumption
  - application class coverage
- integration into Linux distributions
  - Debian is on the way
- C support
  - extension of the join point model
- more partners
  - who wants to support the AspectC++ project?

References

K. Czarnecki, U.W. Eisenecker et. al.: *"Aspektorientierte Programmierung in C++"*, iX – Magazin für professionelle Informationstechnik, 08/09/10, 2001

- A comprehensive analysis of doing AOP with pure C++: what's possible and what not
- http://www.heise.de/ix/artikel/2001/08/143

C. Diggins: *"Aspect-oriented Programming in C++"*, Dr. Dobb's Journal, August, 2004.

Semantic Designs, Inc.: *"Aspect-oriented Programming with DMS"*

- http://www.semdesigns.com/Products/DMS/AspectOrientedProgramming.html

pure-systems GmbH: *AspectC++ Add-In for Visual Studio .NET*

- http://www.pure-systems.com

P&P Software GmbH: *"The XWeaver Project"*

- http:://www.pnp-software.com/XWeaver

Thank you for your attention!