

Was bringt die aspektorientierte Programmierung?

Eine Einführung am Beispiel
von AspectC++

Daniel Lohmann, Olaf Spinczyk
{lohmann, spinczyk}@informatik.uni-erlangen.de



Aspektorientierte Programmierung

in den Nachrichten (<http://www.aspectmentor.com/training/reasons.html>)

„AOP is vital for our survival“

Daniel Sabbah



„Microsoft will adopt AOP“

Bill Gates

„Aspects will become mainstream“

Grady Booch



In zehn Jahren in jedem Automobil?

Aspektorientierte Programmierung

in den Nachrichten (<http://www.aspectmentor.com/training/reasons.html>)

„AOP is vital for our survival“



**Was macht AOP so interessant?
Was bringt es dem SW-Entwickler konkret?**

„Aspects will become mainstream“

Grady Booch



In zehn Jahren in jedem Automobil?

Wer wir sind

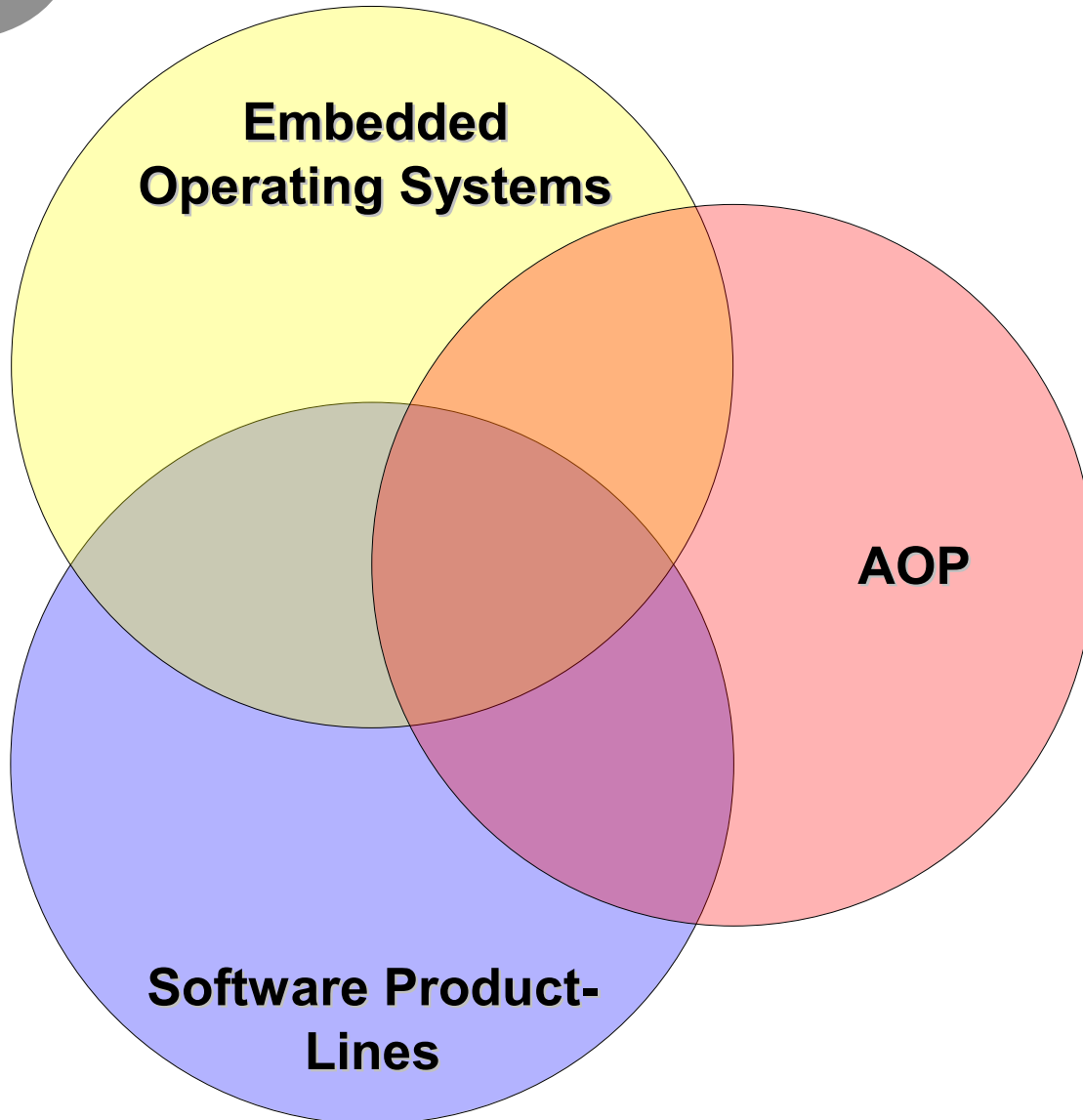
Daniel Lohmann
Dr. Olaf Spinczyk

Lehrstuhl für Informatik IV
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

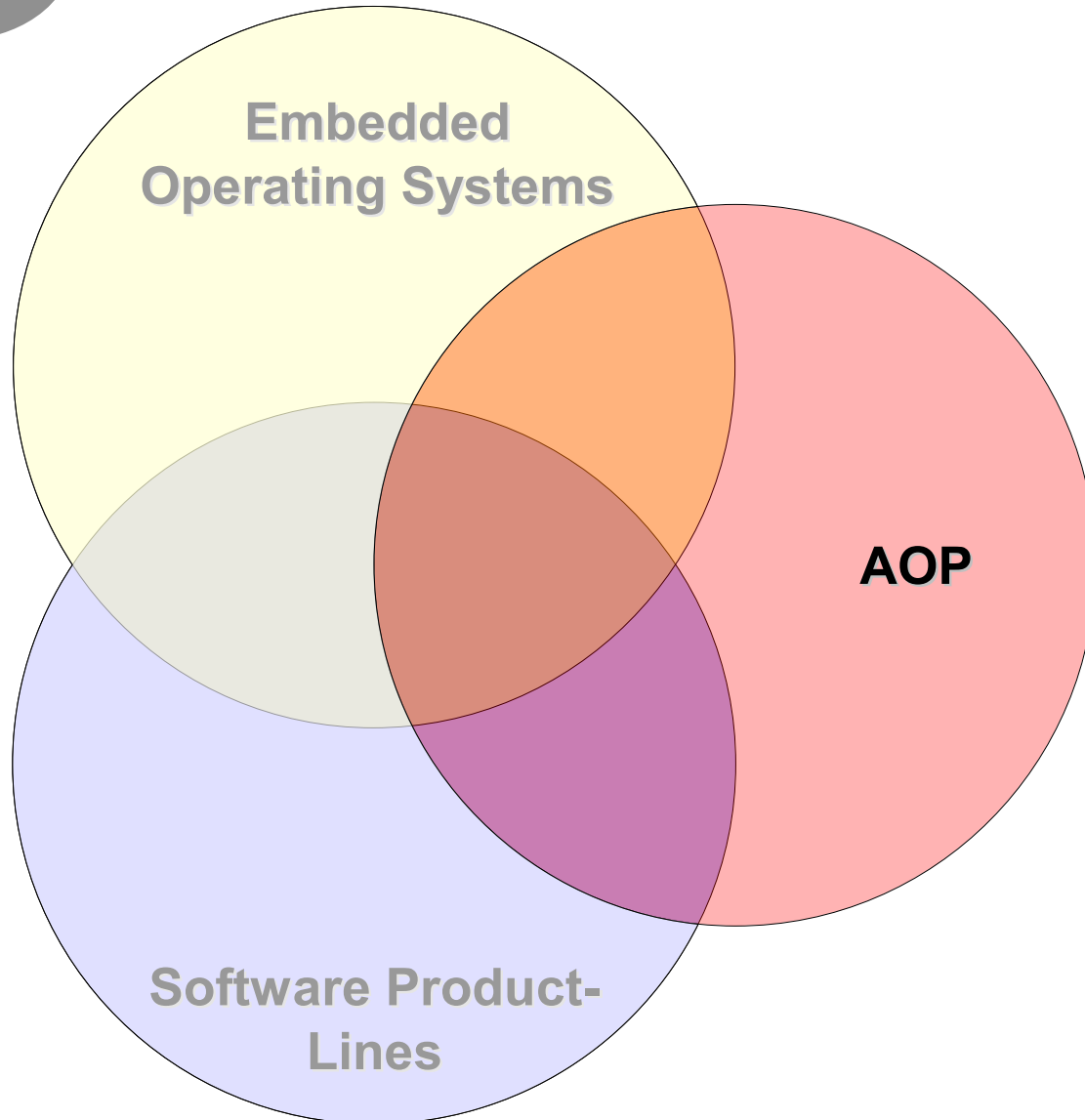
{lohmann, spinczyk}@informatik.uni-erlangen.de



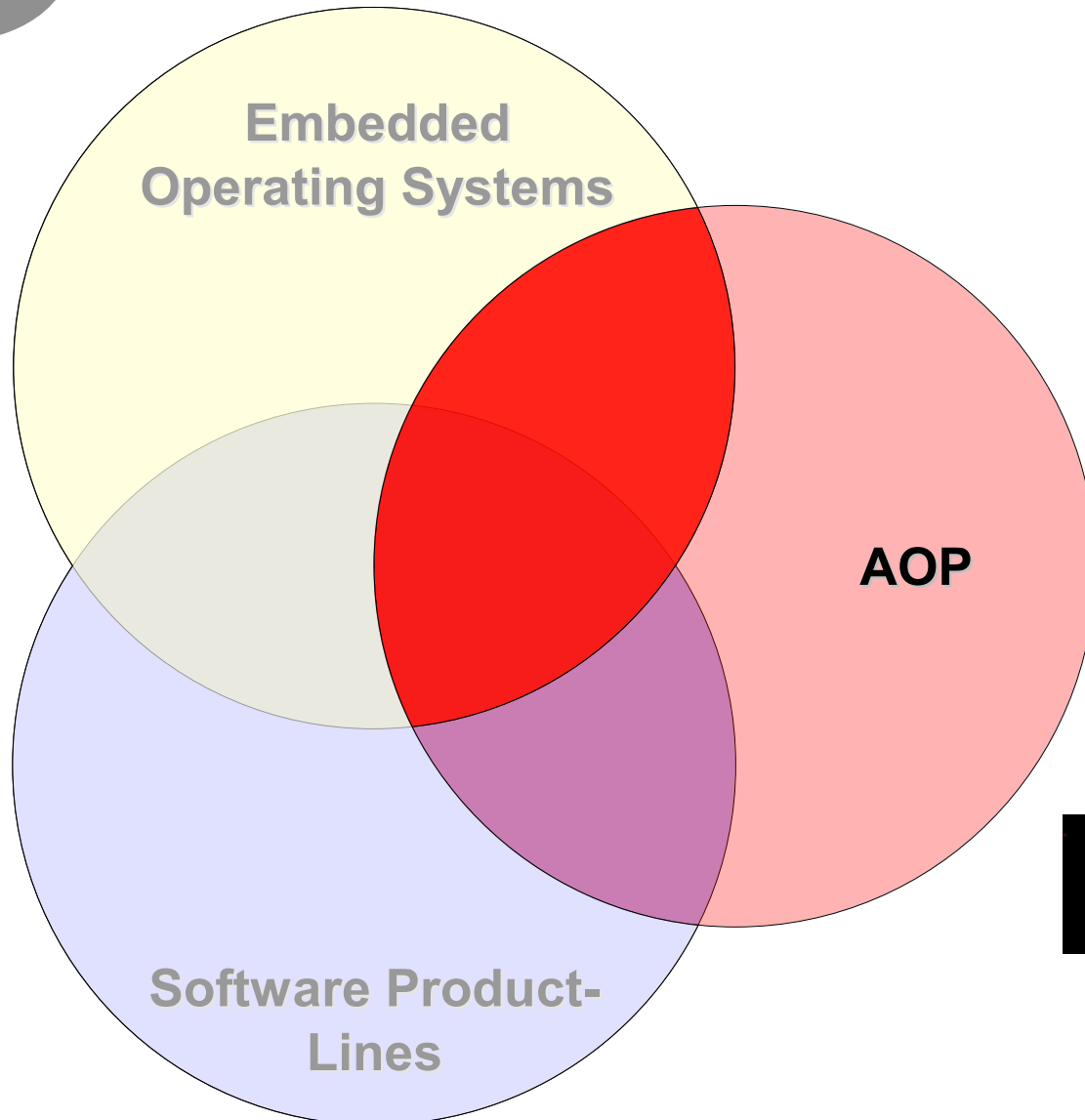
Was wir forschen



Inhalt dieses Vortrags



Inhalt dieses Vortrags

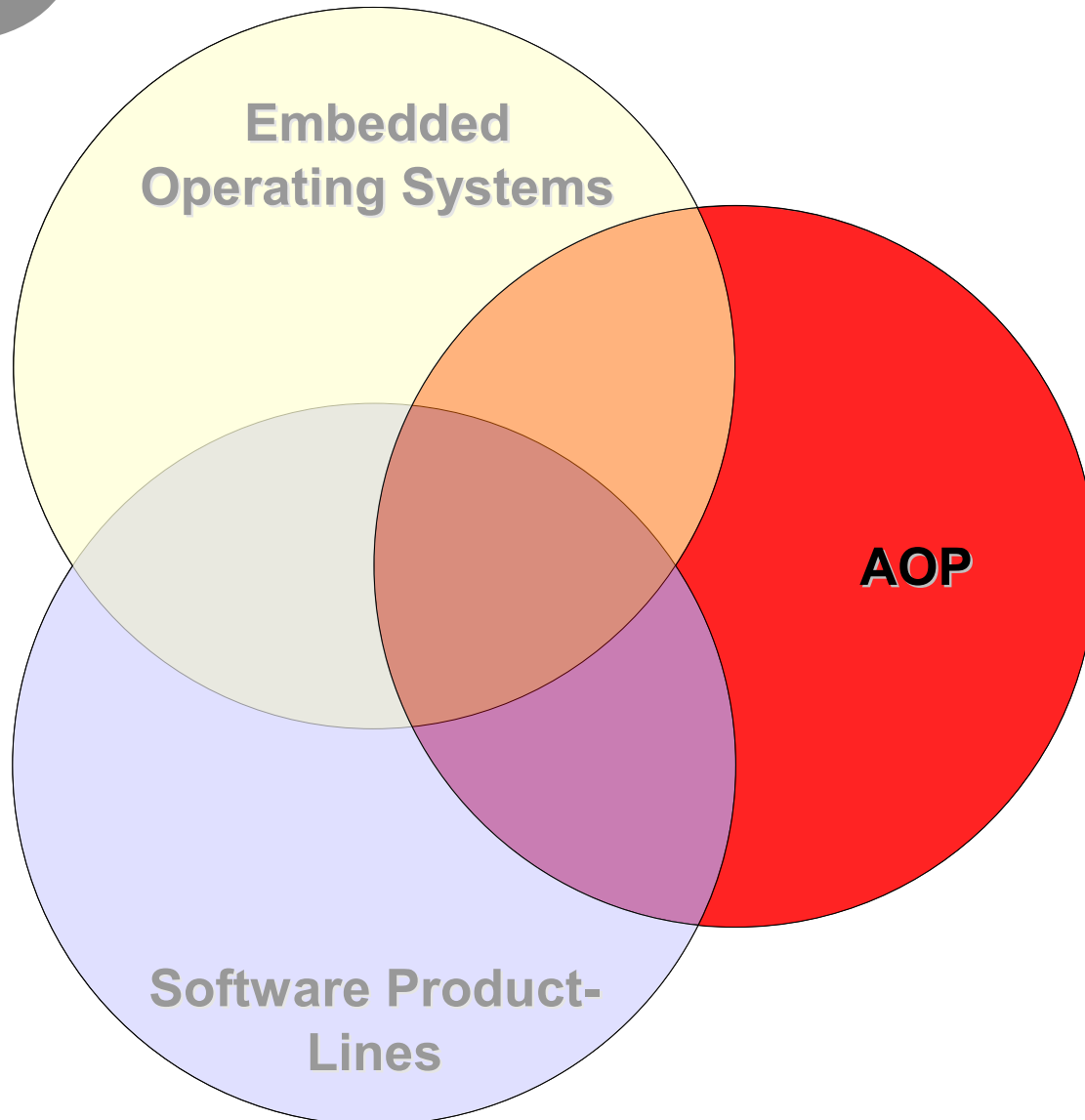


Motivation: Fallstudie eCos

- Das Problem
 - tangled code
 - scattered code
- AOP als Lösungsansatz

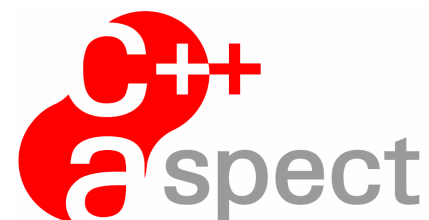


Inhalt dieses Vortrags



Hauptteil: AspectC++

- Die Sprache
 - Grundbegriffe
 - Besonderheiten
- Beispiele
 - Win32 Fehlerbehandlung
 - Observer Pattern



Überblick

➔ Querschneidende Belange in eCos

- Das Problem
- Aspektorientierte Programmierung
 - Der Lösungsansatz
- AspectC++
 - Grundlagen
 - Ergebnisse der eCos-Lösung
 - Erweiterte Sprachmerkmale am Beispiel
 - Vererbung und Wiederverwendung: *Observer Pattern*
 - Generische Programmierung: *Win32 Fehlerbehandlung*
- Zusammenfassung

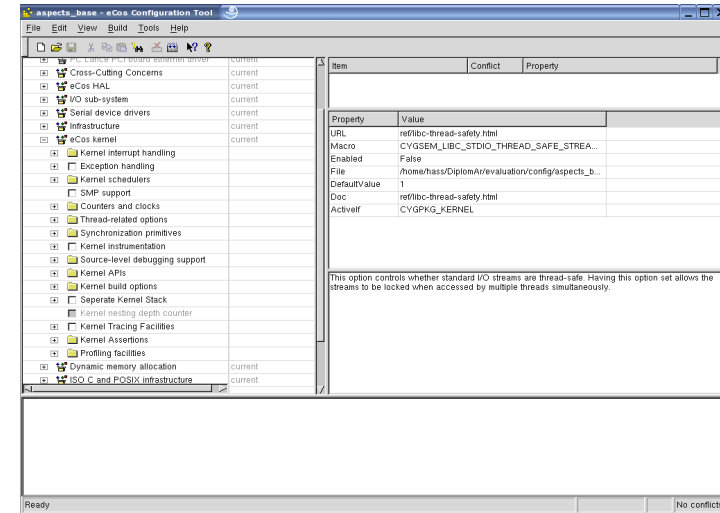
eCos: An OS Product Line



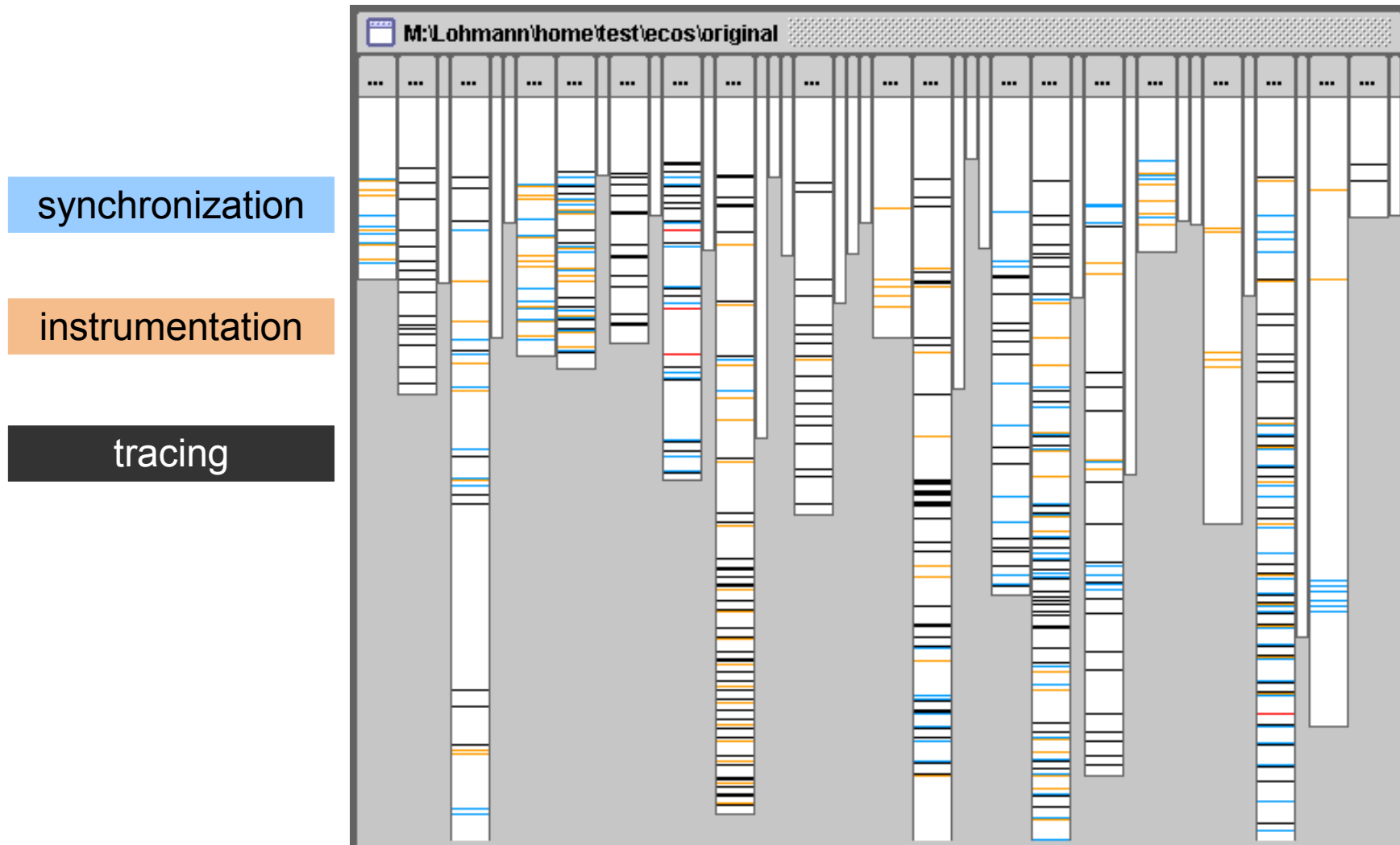
- Open source OS product line for embedded systems
 - developed and maintained by RedHat
 - supports a high number of 16/32 bit architectures
 - kernel written in C++

- Goal: static configurability
 - 63 selectable packages
 - 761 selectable configuration options

- Configuration approach
 - package selection (coarse-grained configuration)
 - **conditional compilation** (fine-grained configuration)

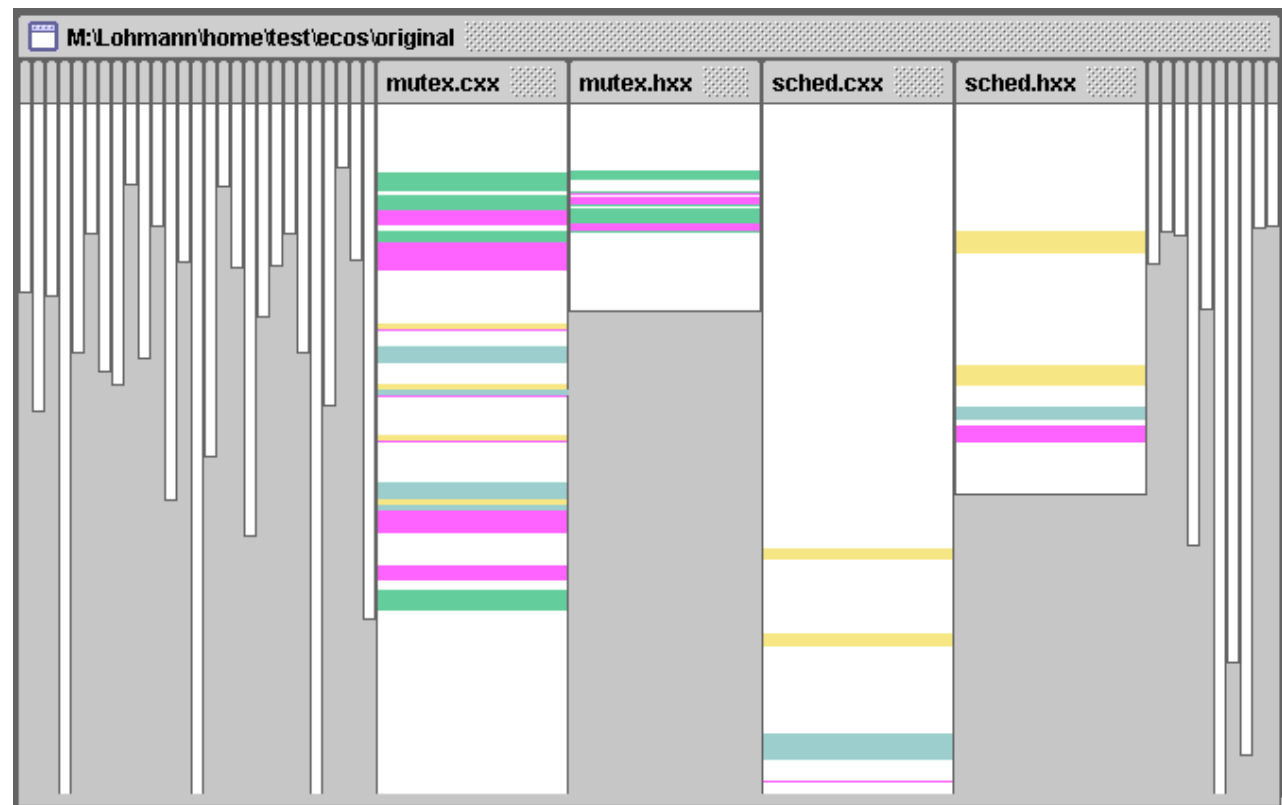


eCos: Global Policies in the Code



eCos: Configuration Options

Variants of the optional mutex priority inversion protocol



eCos: Implementation Example

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner     = NULL;
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```

27 lines of code

eCos: Implementation Example

```

Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner     = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif
CYG_REPORT_RETURN();
}

```

2 lines for the tracing policy

eCos: Implementation Example

```

Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner     = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}

```

21 (almost unreadable) lines for
optional features

eCos: Implementation Example

```

Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner    = NULL;
    #if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
        defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
        protocol = INHERIT;
    #endif
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
        protocol = CEILING;
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
    #endif
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
        protocol = NONE;
    #endif
    #else // not (DYNAMIC and DEFAULT defined)
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
        // if there is a default priority ceiling defined, use that to initialize
        // the ceiling.
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
    #else
        ceiling = 0; // Otherwise set it to zero.
    #endif
    #endif
    #endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}

```

4 lines for the **basic implementation**

eCos: Implementation Example

```

Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner    = NULL;
    #if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
        defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
        protocol = INHERIT;
    #endif
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
        protocol = CEILING;
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
    #endif
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_NONE
        protocol = NONE;
    #endif
    #else // not (DYNAMIC and CEILING)
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_NONE
        protocol = NONE;
    #endif
    #ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
        // if there is a ceiling,
        // the ceiling.
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
    #endif
    #else
        ceiling = 0; // no ceiling
    #endif
    #endif
    #endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}

```

Well, ...

- comprehensability ?
- extensability ?
- reuseability ?
- maintainability ?

Überblick

- Querschneidende Belange in eCos

- Das Problem



Aspektorientierte Programmierung

- Der Lösungsansatz

- AspectC++

- Grundlagen

- Ergebnisse der eCos-Lösung

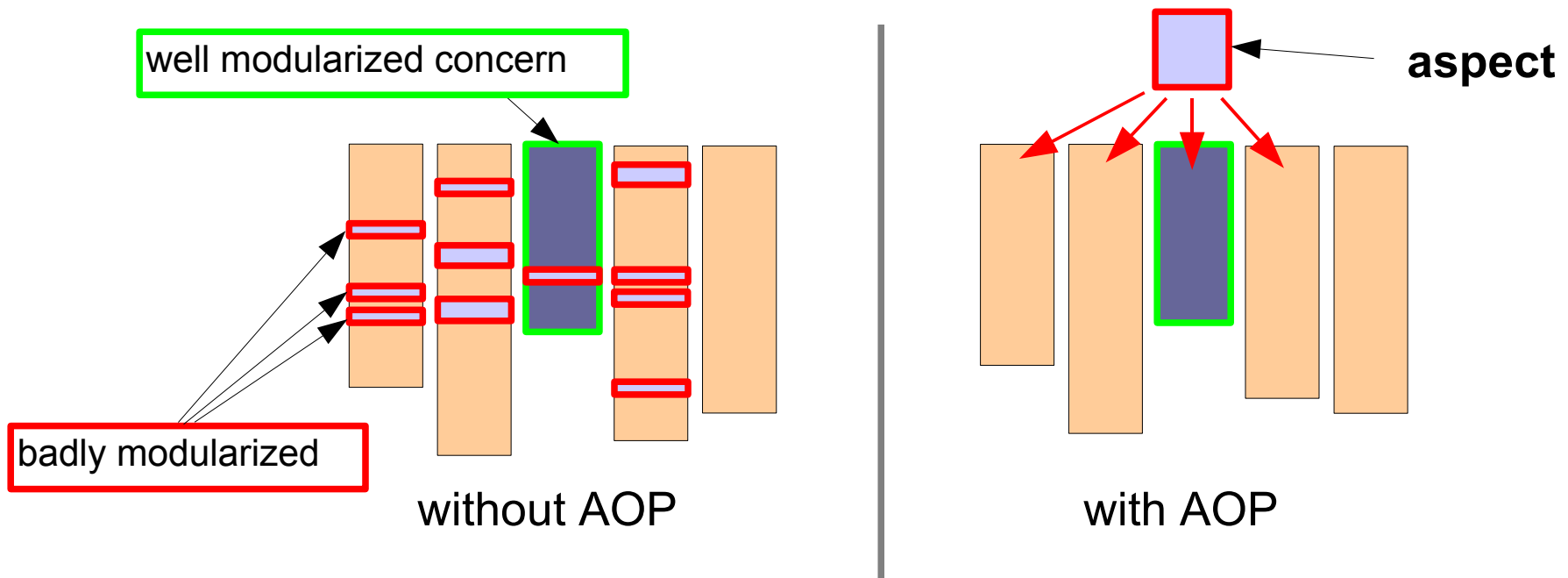
- Erweiterte Sprachmerkmale am Beispiel

- Vererbung und Wiederverwendung: *Observer Pattern*
- Generische Programmierung: *Win32 Fehlerbehandlung*

- Zusammenfassung

Aspekt-Orientierte Programmierung

AOP bietet Sprachmittel zur Trennung und Kapselung von querschnittenden Belangen



AOP: Trennung von **Was** und **Wo**

➤ **Join-Points**

- bezeichnen Stellen in der **Programmstruktur** (name join-points) oder Ereignisse im laufenden **Kontrollfluss** (code join-points)
- werden **deklarativ** spezifiziert durch **pointcut expressions**

➤ **Advice**

- zusätzliche **Elemente** (Methoden, Daten, ...), werden in bestimmte Klassen oder Strukturen eingefügt (statisch)

- zusätzliches **Verhalten** (Code), das

- **before, after**
- **around** (anstelle von)

bestimmten Ereignissen im Kontrollfluss aktiviert wird

Überblick

- Querschneidende Belange in eCos
 - Das Problem
- Aspektorientierte Programmierung
 - Der Lösungsansatz



AspectC++

- Grundlagen
- Ergebnisse der eCos-Lösung
- Erweiterte Sprachmerkmale am Beispiel
 - Vererbung und Wiederverwendung: *Observer Pattern*
 - Generische Programmierung: *Win32 Fehlerbehandlung*
- Zusammenfassung



AspectC++



- AOP Spracherweiterung für C++
 - Syntax und Semantik angelehnt an AspectJ

- Werkzeugunterstützung
 - Source-Level Weaver ac++
 - IDE Integration in Eclipse und (kommerziell) VisualStudio
 - OpenSource Projekt
 - Kommerzieller Support durch pure::systems GmbH

- Aktive Nutzer-Community

```

C/C++ - ObserverPattern.ah - Eclipse Platform
File Edit Navigate Search Run Project Window Help

#include <map>
using namespace std;

aspect ObserverPattern {
public:
  // interfaces for each role
  struct ISubject {};
  struct IObserver {
    virtual void update (ISubject *) = 0;
  };

  // to be defined by the concrete derived
  pointcut virtual observers() = 0;
  pointcut virtual subjects() = 0;
  // defaults to any non-const member of :
  pointcut virtual subjectChange() =
    execution( "% ..::%(...)" && !"% ..::%(...)" );

  advice observers () : baseclass (IObserver) {
  advice subjects () : baseclass (ISubject) {
  advice subjectChange() : after () {
    ISubject* subject = tjp->that ();
    updateObservers (subject);
  }
}
}

```



Syntactic Elements

aspect name

pointcut expression

advice type

```
aspect ElementCounter {  
  advice execution("% util::Queue::enqueue(...)") : after()  
  {  
    printf( " Aspect ElementCounter: after Queue::enqueue!\n" );  
  }  
  ...  
};
```

ElementCounter1.ah

advice body



Pointcut Expressions

- Pointcut expressions are made from ...
 - **match expressions**, e.g. "% util::queue::enqueue(...)"
 - are matched against C++ programm entities → name join-points
 - support wildcards
 - **pointcut functions**, e.g. execution(...), call(...), that(...)
 - **execution**: all points in the control flow, where a function is about to be executed → code join-points
 - **call**: all points in the control flow, where a function is about to be called → code join-points
- Pointcut functions can be combined into expressions
 - using logical connectors: &&, ||, !
 - Example: `call("% util::Queue::enqueue(...)") && within("% main(...)")`



Advice

advice for code join-points (runtime events)

– **before advice**

- advice code runs **before** the original code
- may read and modify function parameter (call + execution)

– **after advice**

- advice runs **after** the original code
- may read and modify function result values (call + execution)

– **around advice**

- advice code run **instead of** the original code
- original code can be explicitly invoked by `tjp->proceed()`

introductions

- further methods, attributes, ... are inserted into classes
- extension of interfaces and class implementations



Before / After Advice

advice execution("void ClassA::foo()") : **before()**

advice execution("void ClassA::foo()") : **after()**

```
class ClassA {  
public:  
    void foo(){  
        printf("ClassA::foo()\n");  
    }  
}
```

advice call ("void ClassA::foo()") : **before()**

advice call ("void ClassA::foo()") : **after()**

```
int main(){  
    printf("main()\n");  
    ClassA a;  
    a.foo();  
}
```



Around Advice

```
advice execution("void ClassA::foo()") :  
around()  
  before code  
  
tjp->proceed()  
  
after code
```

```
class ClassA {  
public:  
  void foo(){  
    printf("ClassA::foo()\n");  
  }  
}
```

```
advice call("void ClassA::foo()") : around()  
  before code  
  
tjp->proceed()  
  
after code
```

```
int main(){  
  printf("main()\n");  
  ClassA a;  
  a.foo();  
}
```



Introductions

```
advice "ClassA" : slice class {  
    private element to introduce  
public:  
    public element to introduce  
};
```

```
class ClassA {  
    public:  
    void foo(){  
        printf("ClassA::foo()\n");  
    }  
}
```

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

    void call_clear_ceiling(Cyg_Thread*);
    ...

    advice "Cyg_Mutex" : cyg_priority ceiling;
    ...

    advice construction("Cyg_Mutex") : after() {
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
    }

    advice call("% Cyg_Mutex::lock_inner(...)")
        && within("% Cyg_Mutex::lock(...)")
        && args(self)
        : after(Cyg_Thread* self)
    {
        if(!(*tjp->result())) {
            call_clear_ceiling(self);
        }
    }
    ...
};

```

Was

Wo

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

    void call_clear_ceiling(Cyg_Thread*);
    ...

    advice "Cyg_Mutex" : cyg_priority ceiling;
    ...

    advice construction("Cyg_Mutex") : after() {
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
    }

    advice call("% Cyg_Mutex::lock_inner(...)")
        && within("% Cyg_Mutex::lock(...)")
        && args(self)
        : after(Cyg_Thread* self)
    {
        if(!(*tjp->result())) {
            call_clear_ceiling(self);
        }
    }
    ...
};

```

Was

Wo

Einfügung einer Variable *ceiling* in alle Klassen mit dem Namen *Cyg_Mutex*

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

    void call_clear_ceiling(Cyg_Thread*);
    ...

    advice "Cyg_Mutex" : cyg_priority ceiling;
    ...

    advice construction("Cyg_Mutex") : after() {
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
    }

    advice call("% Cyg_Mutex::lock_inner(...)")
        && within("% Cyg_Mutex::lock(...)")
        && args(self)
        : after(Cyg_Thread* self)
    {
        if(!(*tjp->result())) {
            call_clear_ceiling(self);
        }
    }
    ...
};

```

Was

Wo

Ausführen der *Initialisierung*
nach dem **Erstellen** einer
Cyg_Mutex Instanz

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

    void call_clear_ceiling(Cyg_Thread*);
    ...

    advice "Cyg_Mutex" : cyg_priority ceiling;
    ...

    advice construction("Cyg_Mutex") : after() {
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
    }

    advice call("% Cyg_Mutex::lock_inner(...)")
        && within("% Cyg_Mutex::lock(...)")
        && args(self)
        : after(Cyg_Thread* self)
    {
        if(!(*tjp->result())) {
            call_clear_ceiling(self);
        }
    }
    ...
};

```

Was

Wo

Nach einem **Aufruf** von
Cyg_Mutex::lock_inner, der
stattfindet...

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

  void call_clear_ceiling(Cyg_Thread*);
  ...

  advice "Cyg_Mutex" : cyg_priority ceiling;
  ...

  advice construction("Cyg_Mutex") : after() {
    tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
  }

  advice call("% Cyg_Mutex::lock_inner(...)")
    && within("% Cyg_Mutex::lock(...)")
    && args(self)
    : after(Cyg_Thread* self)
  {
    if(!(*tjp->result())) {
      call_clear_ceiling(self);
    }
  }
  ...
};

```

Was

Wo

Nach einem **Aufruf** von *Cyg_Mutex::lock_inner*, der stattfindet...

...**innerhalb** der Ausführung von *Cyg_Mutex::lock* und außerdem...

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

    void call_clear_ceiling(Cyg_Thread*);
    ...

    advice "Cyg_Mutex" : cyg_priority ceiling;
    ...

    advice construction("Cyg_Mutex") : after() {
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
    }

    advice call("% Cyg_Mutex::lock_inner(...)")
        && within("% Cyg_Mutex::lock(...)")
        && args(self)
        : after(Cyg_Thread* self)
    {
        if(!(*tjp->result())) {
            call_clear_ceiling(self);
        }
    }
    ...
};

```

Was

Wo

Nach einem **Aufruf** von *Cyg_Mutex::lock_inner*, der stattfindet...

...**innerhalb** der Ausführung von *Cyg_Mutex::lock* und außerdem...

...ein **Argument** vom Typ *Cyg_Thread** übergibt...

Beispiel: eCos Priority Ceiling Protocol

```

aspect priority_ceiling {

    void call_clear_ceiling(Cyg_Thread*);
    ...

    advice "Cyg_Mutex" : cyg_priority ceiling;
    ...

    advice construction("Cyg_Mutex") : after() {
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;
    }

    advice call("% Cyg_Mutex::lock_inner(...)")
        && within("% Cyg_Mutex::lock(...)")
        && args(self)
        : after(Cyg_Thread* self)
    {
        if(!(*tjp->result())) {
            call_clear_ceiling(self);
        }
    }
    ...
};

```

Was

Wo

Nach einem **Aufruf** von *Cyg_Mutex::lock_inner*, der stattfindet...

...**innerhalb** der Ausführung von *Cyg_Mutex::lock* und außerdem...

...ein **Argument** vom Typ *Cyg_Thread** übergibt...

...überprüfe, ob der Mutex verlassen wurde und passe ggfs. die Priorität an

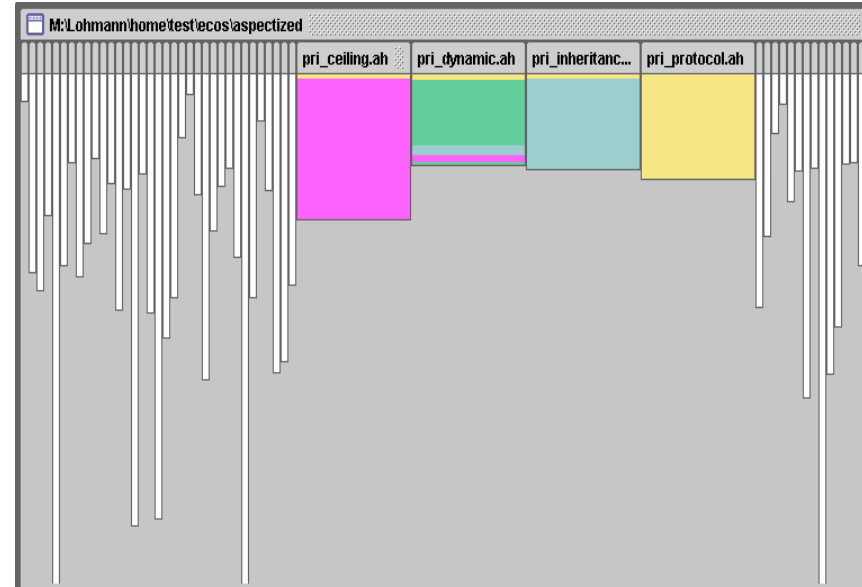
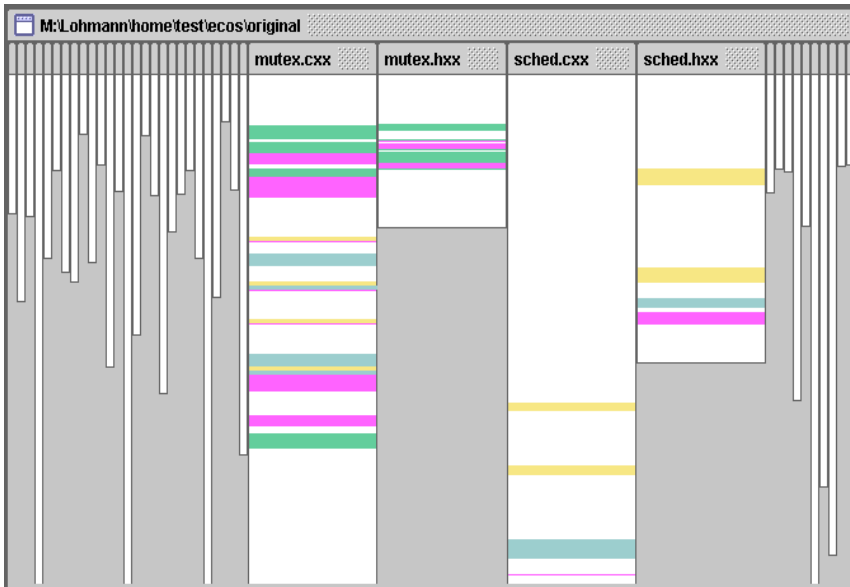
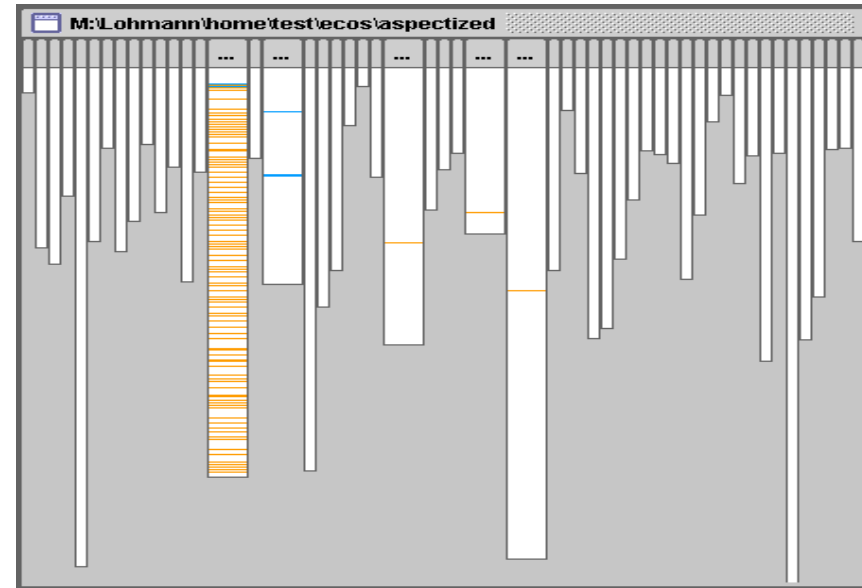
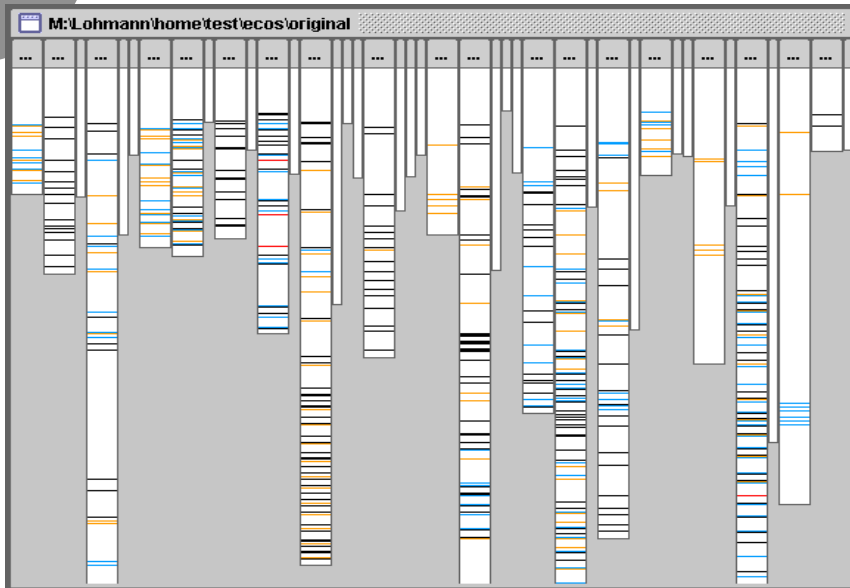
Results with eCos

- **Refactored:** *original kernel* → *aspectized kernel*
 - **3 cross-cutting policies**
 - interrupt synchronization 187 invocations → 160 code join-points
 - kernel instrumentation 162 invocations → 139 code join-points
 - tracing 336 invocations → 632 code join-points
 - **12 configuration options**
 - mutex features
 - thread features

- **Compared:** *original kernel* ↔ *aspectized kernel*
 - scattering, performance, memory footprint

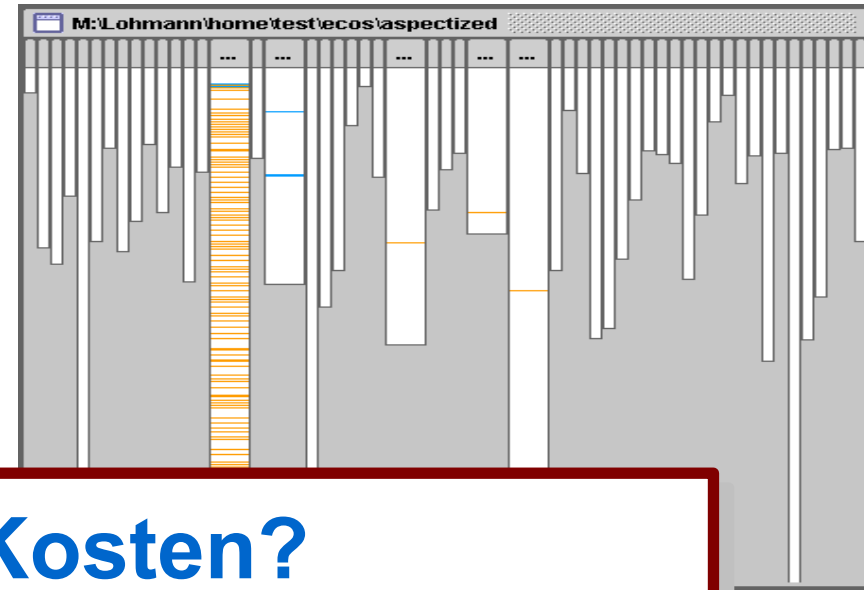
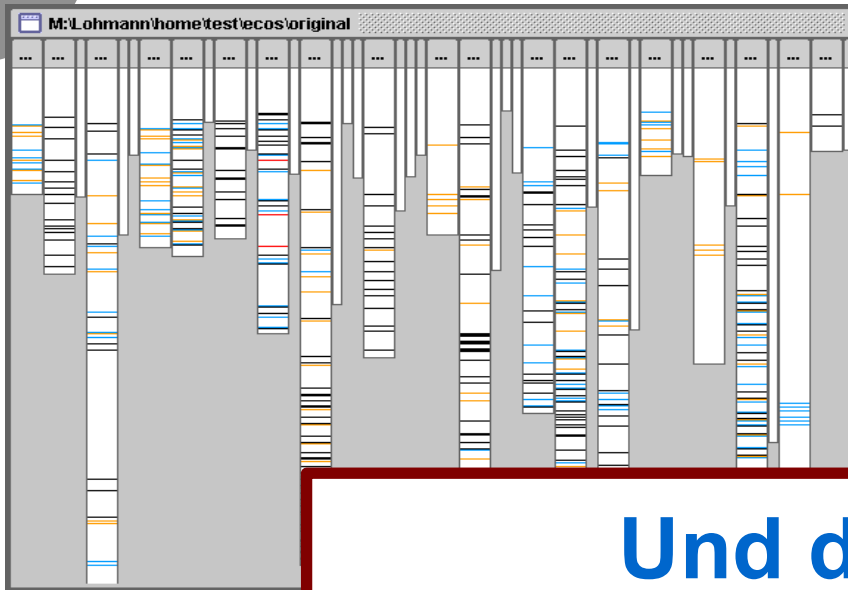
original kernel

aspectized kernel



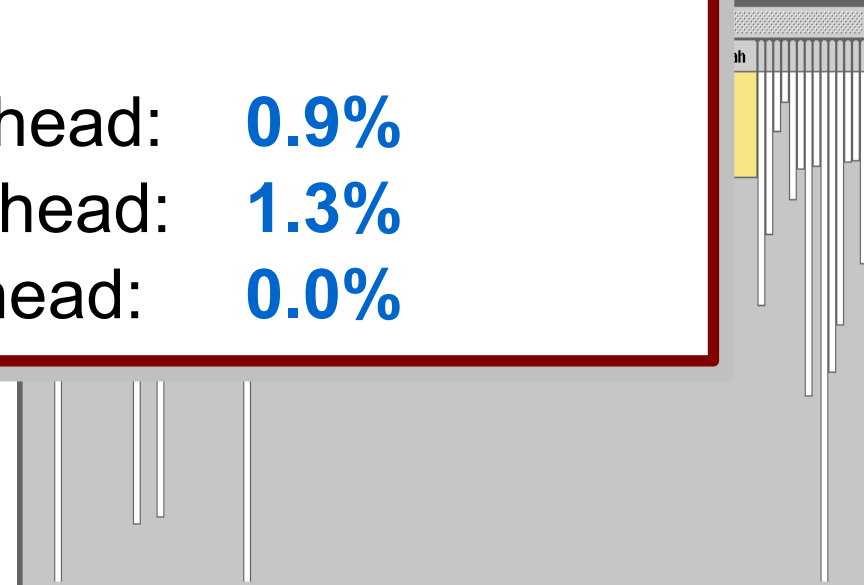
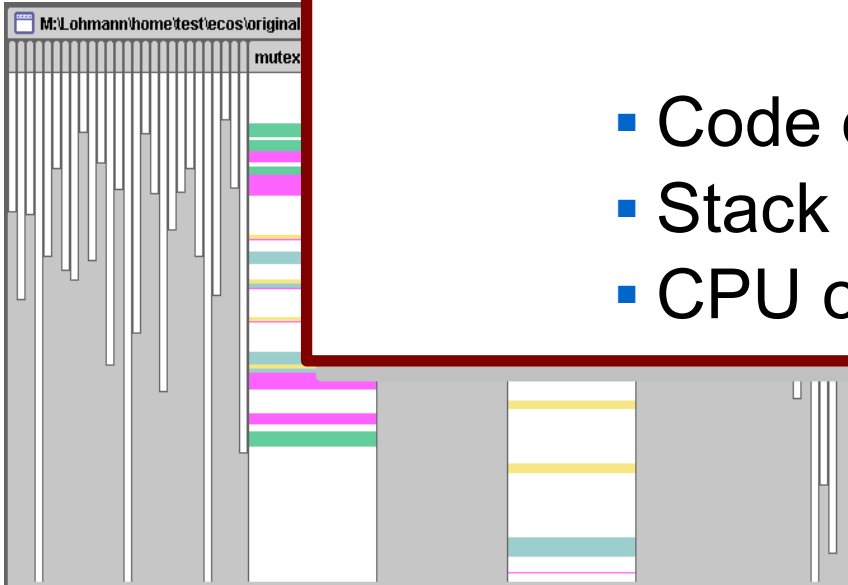
original kernel

aspectized kernel



Und die Kosten?

- Code overhead: **0.9%**
- Stack overhead: **1.3%**
- CPU overhead: **0.0%**

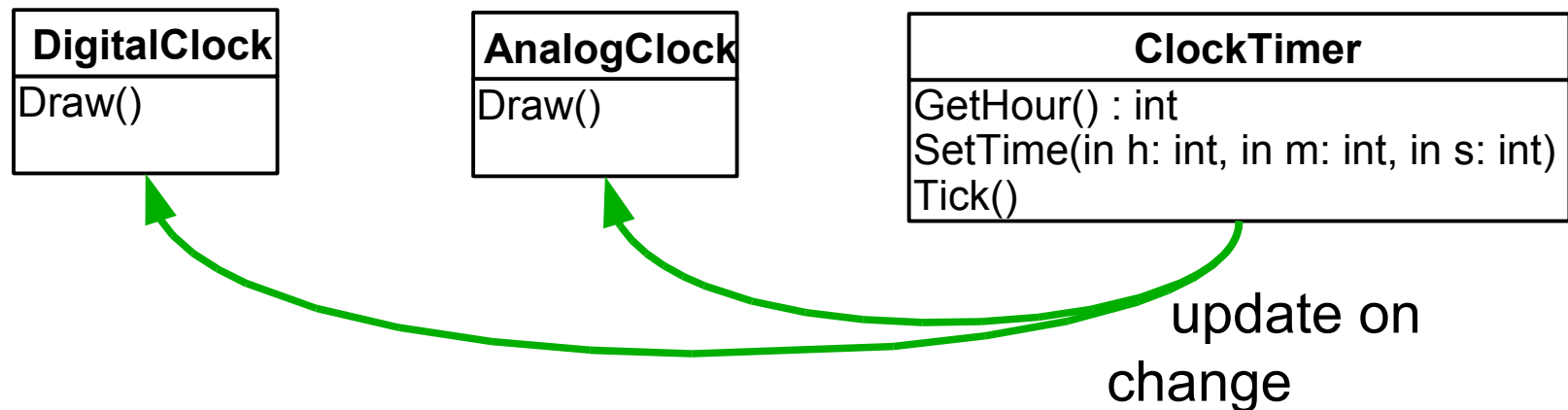


Überblick

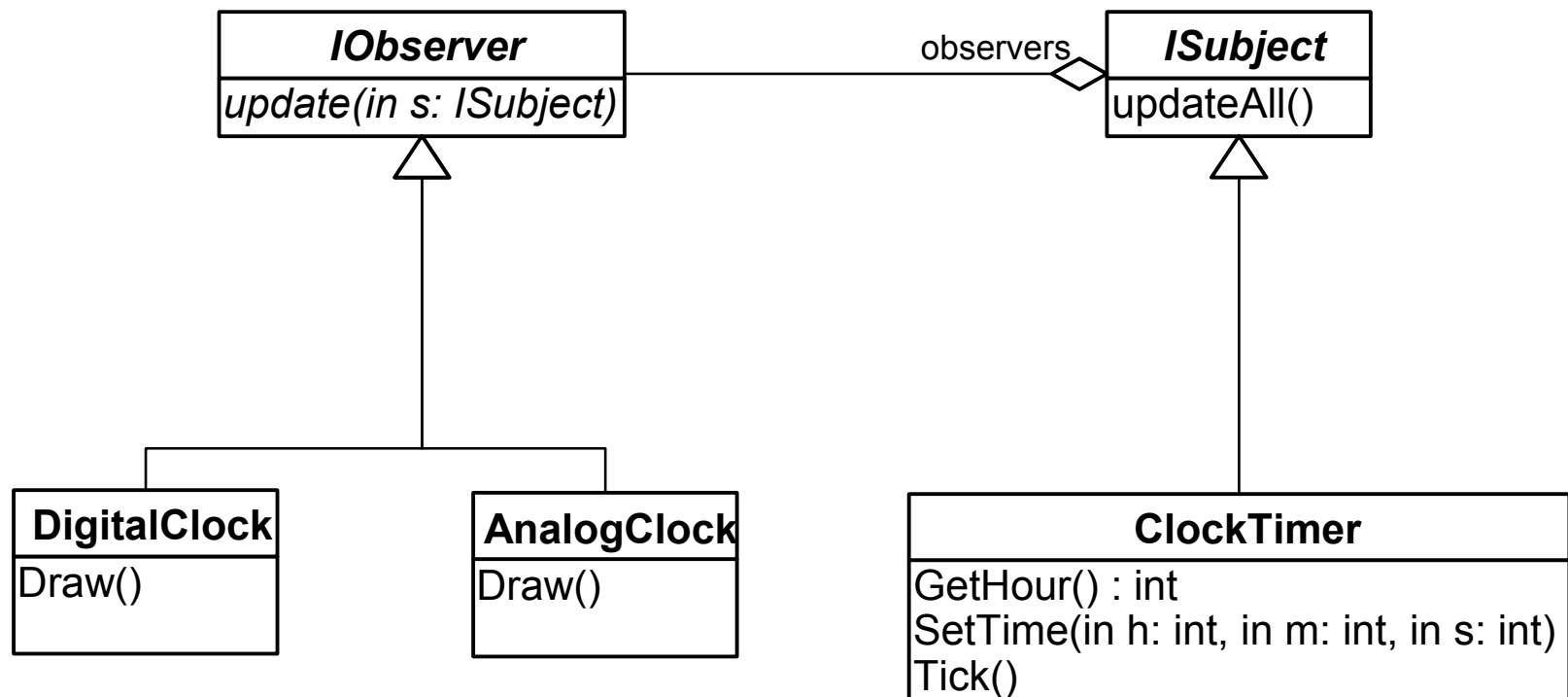
- Querschneidende Belange in eCos
 - Das Problem
- Aspektorientierte Programmierung
 - Der Lösungsansatz
- AspectC++
 - Grundlagen
 - Ergebnisse der eCos-Lösung
 - Erweiterte Sprachmerkmale am Beispiel
 - **Vererbung und Wiederverwendung: *Observer Pattern***
 - Generische Programmierung: *Win32 Fehlerbehandlung*
- Zusammenfassung



Observer Pattern: Scenario

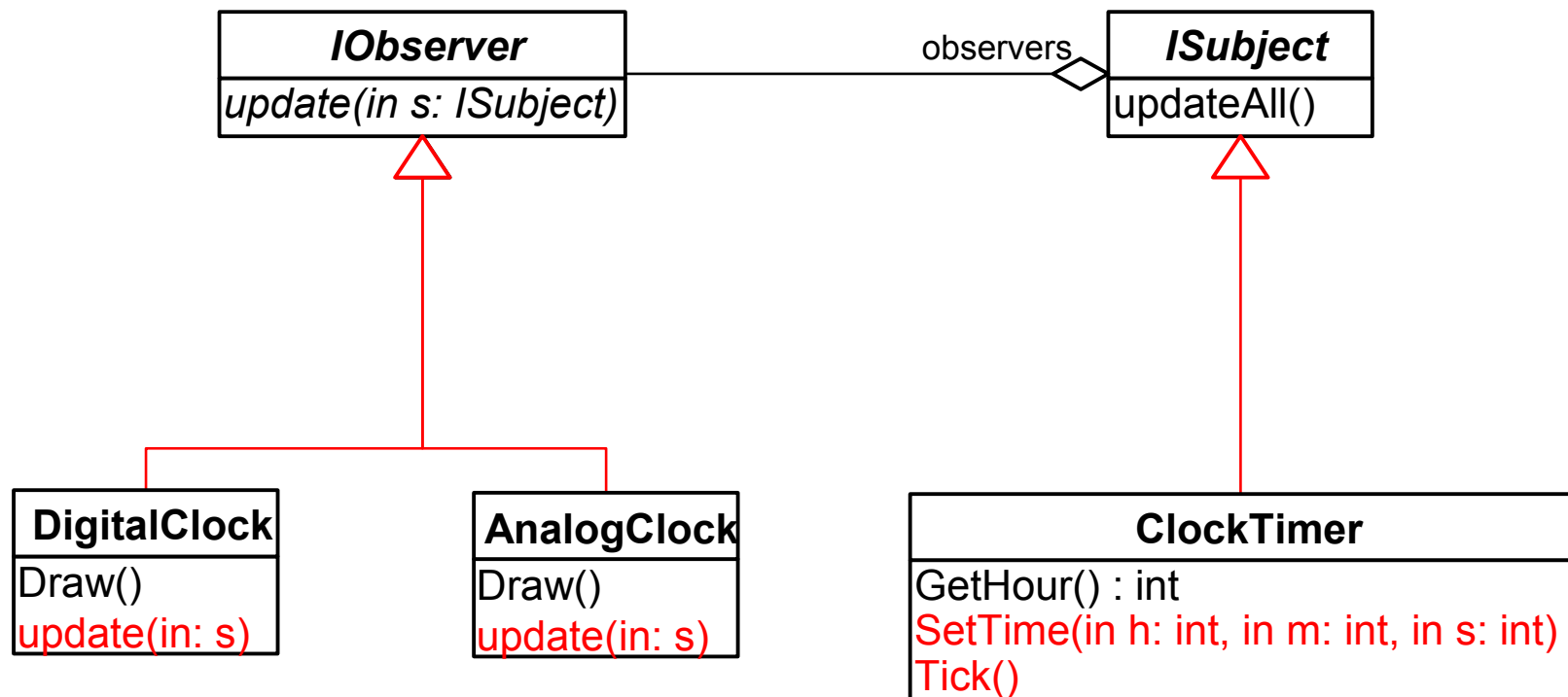


Observer Pattern: Implementation



Observer Pattern: Implementation

The 'Observer Protocol' concern...



...**crosscuts** the module structure



Aspects and Inheritance

- Aspects can inherit from other aspects...
 - Reuse aspect definitions
 - Override methods and pointcuts
- Pointcuts can be pure virtual
 - Postpone the concrete definition to derived aspects
 - An aspect with a pure virtual pointcut is called **abstract aspect**
- Common usage: Reusable aspect implementations
 - Abstract aspect defines advice code, but pure virtual pointcuts
 - Aspect code uses the join-point API to expose context
 - Concrete aspect inherits the advice code and overrides pointcuts

Solution: Generic Observer Aspect

```

aspect ObserverPattern {
    ...
public:
    struct ISubject {};
    struct IObserver {
        virtual void update (ISubject *) = 0;
    };

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;

    pointcut virtual subjectChange() = execution( "% ..::%(...)"
        && !"%. ..::%(...) const" ) && within( subjects() );

    advice observers () : slice class : public ObserverPattern::IObserver;
    advice subjects()   : slice class : public ObserverPattern::ISubject;

    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }

    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver* observer ) { ... }
    void remObserver( ISubject* subject, IObserver* observer ) { ... }
};

```

Solution: Generic Observer Aspect

```

aspect ObserverPattern {
    ...
public:
    struct ISubject {};
    struct IObserver {
        virtual void update (ISubject *) = 0;
    };

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;

    pointcut virtual subjectChange() = execution( "% ..::%(...)"
        && !"%. ..::%(...) const" ) && within( subjects() );

    advice observers () : slice class : public ObserverPattern::IObserver;
    advice subjects()   : slice class : public ObserverPattern::ISubject;

    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }

    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver* observer ) { ... }
    void remObserver( ISubject* subject, IObserver* observer ) { ... }
};

```

interfaces for the subject/observer roles

Solution: Generic Observer Aspect

```

aspect ObserverPattern {
    ...
public:
    struct ISubject {};
    struct IObserver {
        virtual void update (ISubject *) = 0;
    };
    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;

    pointcut virtual subjectChange() = execution( "% ..::%(...)"
        && !"%. ..::%(...) const" ) && within( subjects() );

    advice observers () : slice class : public ObserverPattern::IObserver;
    advice subjects()   : slice class : public ObserverPattern::ISubject;

    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }

    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver* observer ) { ... }
    void remObserver( ISubject* subject, IObserver* observer ) { ... }
};

```

abstract pointcuts that
define subjects/observers

(need to be overridden by
a derived aspect)

Solution: Generic Observer Aspect

```

aspect ObserverPattern {
    ...
public:
    struct ISubject {};
    struct IObserver {
        virtual void update (ISubject *) = 0;
    };

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;

    pointcut virtual subjectChange() = execution( "% ..::%(...)"
        && !"%.::%(...) const" ) && within( subjects() );

    advice observers () : slice class : public ObserverPattern::IObserver;
    advice subjects()   : slice class : public ObserverPattern::ISubject;

    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }

    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver* observer ) { ... }
    void remObserver( ISubject* subject, IObserver* observer ) { ... }
};

```

virtual pointcut defining all state-changing methods.

(defaults to the execution of any non-const methods in subjects)

Solution: Generic Observer Aspect

```

aspect ObserverPattern {
    ...
public:
    struct ISubject {};
    struct IObserver {
        virtual void update (ISubject *) = 0;
    };

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;

    pointcut virtual subjectChange() = execution( "% ..::%(...)"
        && !"%. ..::%(...)" const ) && within( subjects() );

advice observers () : slice class : public ObserverPattern::IObserver;
advice subjects() : slice class : public ObserverPattern::ISubject;

advice subjectChange() : after () {
    ISubject* subject = tjp->that();
    updateObservers( subject );
}

void updateObservers( ISubject* subject ) { ... }
void addObserver( ISubject* subject, IObserver* observer ) { ... }
void remObserver( ISubject* subject, IObserver* observer ) { ... }
};

```

introduction of the role interface as additional **baseclass** into subjects and observers

Solution: Generic Observer Aspect

```

aspect ObserverPattern {
    ...
public:
    struct ISubject {};
    struct IObserver {
        virtual void update (ISubject *) = 0;
    };

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;

    pointcut virtual subjectChange() = execution( "% ..::%(...)"
        && !"%. ..::%(...) const" ) && within( subjects() );

    advice observers () : slice class : public ObserverPattern::IObserver;
    advice subjects()   : slice class : public ObserverPattern::ISubject;

    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }

    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver* observer ) { ... }
    void remObserver( ISubject* subject, IObserver* observer ) { ... }
};

```

after advice to update observers after the execution of state-changing methods

Putting Everything Together

Applying the Generic Observer Aspect to the clock example

```
aspect ClockObserver : public ObserverPattern {
    // define the participants
    pointcut subjects() = "ClockTimer";
    pointcut observers() = "DigitalClock" || "AnalogClock";

public:
    // define what to do in case of a notification
    advice observers() : slice class {
public:
        void update( ObserverPattern::ISubject* s ) {
            Draw();
        }
    };
};
```

Observer Pattern: Conclusions

- Applying the observer protocol is now very easy!
 - all necessary transformations are performed by the generic aspect
 - programmer just needs to define participants and behaviour
 - multiple subject/observer relationships can be defined
- More reusable and less error-prone component code
 - observer no longer “hard coded” into the design and code
 - no more forgotten calls to update() in subject classes

Überblick

- Querschneidende Belange in eCos
 - Das Problem
- Aspektorientierte Programmierung
 - Der Lösungsansatz
- AspectC++
 - Grundlagen
 - Ergebnisse der eCos-Lösung
 - Erweiterte Sprachmerkmale am Beispiel
 - Vererbung und Wiederverwendung: *Observer Pattern*
 - **Generische Programmierung: *Win32 Fehlerbehandlung***
- Zusammenfassung



Error Handling in Win32: Scenario

```
LRESULT WINAPI WndProc( HWND hWnd, UINT nMsg, WPARAM wParam, LPARAM lParam ) {
    HDC dc = NULL; PAINTSTRUCT ps = {0};

    switch( nMsg ) {
        case WM_PAINT:
            dc = BeginPaint( hWnd, &ps );
            ...
            EndPaint(hWnd, &ps);
            break;
        ...
    }
}

int WINAPI WinMain( ... ) {
    HANDLE hConfigFile = CreateFile( "app.config", GENERIC_READ, ... );

    WNDCLASS wc = {0, WndProc, 0, 0, ... , "Example_Class"};
    RegisterClass( &wc );
    HWND hWndMain = CreateWindowEx( 0, "Example_Class", "Example", ... );
    UpdateWindow( hWndMain );

    MSG msg;
    while( GetMessage( &msg, NULL, 0, 0 ) ) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
    return 0;
}
```

A typical Win32
application

Error Handling in Win32: Scenario

```
LRESULT WINAPI WndProc( HWND hWnd, UINT nMsg, WPARAM wParam, LPARAM lParam ) {
    HDC dc = NULL; PAINTSTRUCT ps = {0};

    switch( nMsg ) {
        case WM_PAINT:
            dc = BeginPaint( hWnd, &ps );
            ...
            EndPaint(hWnd, &ps);
            break;
        ...
    }
}

int WINAPI WinMain( ... ) {
    HANDLE hConfigFile = CreateFile( "app.config", GENERIC_READ, ... );

    WNDCLASS wc = {0, WndProc, 0, 0, ... , "Example_Class"};
    RegisterClass( &wc );
    HWND hWndMain = CreateWindowEx( 0, "Example_Class", "Example", ... );
    UpdateWindow( hWndMain );

    MSG msg;
    while( GetMessage( &msg, NULL, 0, 0 ) ) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
    return 0;
}
```

**these Win32 API
functions may fail**

Win32 Error Handling: Goals

- Detect failed calls of Win32 API functions
 - by giving after advice for any call to a Win32 function
- Throw a *helpful* exception in case of a failure
 - describing the exact circumstances and reason of the failure
- Problem: Win32 failures are indicated by a “magic” return value
 - magic value to compare against depends on the **return type** of the function
 - error reason (GetLastError()) only valid in case of a failure

return type	magic value
BOOL	FALSE
ATOM	(ATOM) 0
HANDLE	INVALID_HANDLE_VALUE or NULL
HWND	NULL



The Join-Point API

- Inside an advice body, the current join-point context is available via the **implicitly passed tjp** variable:

```
advice ... {  
    struct JoinPoint {  
        ...  
    } *tjp;    // implicitly available in advice code  
    ...  
}
```

- You have already seen how to use **tjp**, to ...
 - execute the original code in around advice with **tjp->proceed()**
- The join-point API provides a rich interface
 - to expose context **independently** of the aspect target
 - this is especially useful in writing **reusable aspect code**



The Join-Point API (Excerpt)

Types (compile-time)

```
// object type (initiator)
That

// object type (receiver)
Target

// result type of the affected function
Result

// type of the i'th argument of the affected
// function (with 0 <= i < ARGS)
Arg<i>::Type
Arg<i>::ReferredType
```

Consts (compile-time)

```
// number of arguments
ARGS

// unique numeric identifier for this join point
JPID

// numeric identifier for the type of this join
// point (AC::CALL, AC::EXECUTION, ...)
JPTYPE
```

Values (runtime)

```
// pointer to the object initiating a call
That* that()

// pointer to the object that is target of a call
Target* target()

// pointer to the result value
Result* result()

// typed pointer the i'th argument value of a
// function call (compile-time index)
Arg<i>::ReferredType* arg()

// pointer the i'th argument value of a
// function call (runtime index)
void* arg( int i )

// textual representation of the join-point
// (function/class name, parameter types...)
static const char* signature()

// executes the original join-point code
// in an around advice
void proceed()

// returns the runtime action object
AC::Action& action()
```

Failure Detection: Generic Advice

```
advice call(win32API ()) :  
after () {  
  if (isError (*tjp->result()))  
    // throw an exception  
}
```

bool isError(ATOM);

bool isError(BOOL);

bool isError(HANDLE);

bool isError(HWND);

...

Error Report: Generative Advice

```

template <int I> struct ArgPrinter {
  template <class JP> static void work (JP &tjp, ostream &s) {
    ArgPrinter<I-1>::work (tjp, s);
    s << ", " << *tjp. template arg<I-1>();
  }
};

```

```

advice call(win32API ()) : after () {
  // throw an exception
  ostringstream s;
  DWORD code = GetLastError();
  s << "WIN32 ERROR " << code << ...
    << win32::GetErrorText( code ) << ... <<
    << tjp->signature() << "WITH: " << ...;
  ArgPrinter<JoinPoint::ARGS>::work (*tjp, s);

  throw win32::Exception( s.str() );
}

```

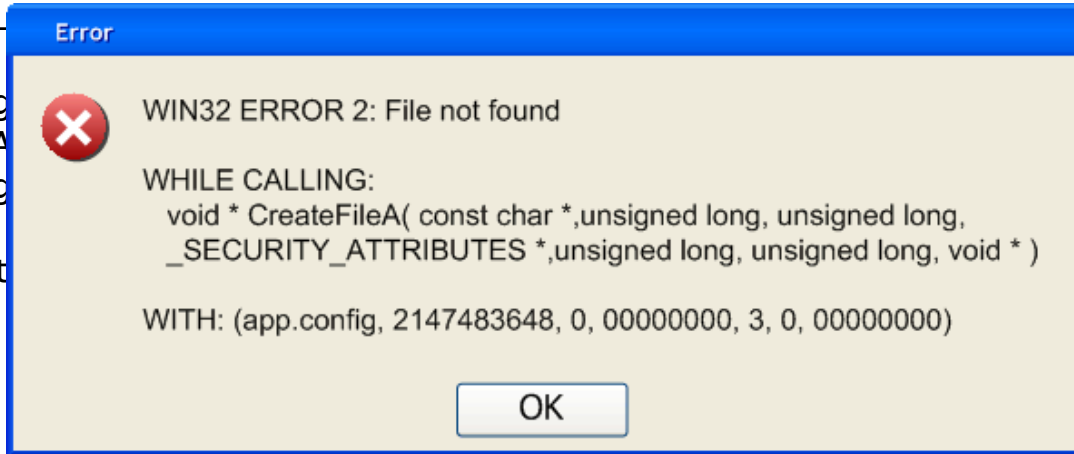
Reporting the Error

```

LRESULT WINAPI WndProc( HWND hWnd, UINT nMsg, WPARAM wParam, LPARAM lParam ) {
    HDC dc = NULL;

    switch( nMsg )
    case WM_PAINT:
        dc = BeginPaint( hWnd, &ps );
        ...
        EndPaint( hWnd, ps );
        break;
    ...
}
}

```



```

int WINAPI WinMain( ... ) {
    HANDLE hConfigFile = CreateFile( "app.config", GENERIC_READ, ... );

    WNDCLASS wc = {0, WndProc, 0, 0, ... , "Example_Class"};
    RegisterClass( &wc );
    HWND hwndMain = CreateWindowEx( 0, "Example_Class", "Example", ... );
    UpdateWindow( hwndMain );

    MSG msg;
    while( GetMessage( &msg, NULL, 0, 0 ) ) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
    return 0;
}
}

```

Error Handling in Win32: Conclusions

- Easy to apply error handling for Win32 applications
 - previously undetected failures are reported by exceptions
 - rich context information is provided
- Uses advanced AspectC++ techniques
 - error detection by generic advice
 - context propagation by generative advice

Überblick

- Querschneidende Belange in eCos
 - Das Problem
- Aspektorientierte Programmierung
 - Der Lösungsansatz
- AspectC++
 - Grundlagen
 - Ergebnisse der eCos-Lösung
 - Erweiterte Sprachmerkmale am Beispiel
 - Vererbung und Wiederverwendung: *Observer Pattern*
 - Generische Programmierung: *Win32 Fehlerbehandlung*

Zusammenfassung

Zusammenfassung

- AOP ...
 - versucht das Problem der querschneidenden Belange zu lösen: Vermeidung von *tangling* und *scattering*
 - modulare Implementierung durch Aspekte
 - Trennung von WO und WAS
- AspectC++ ...
 - erlaubt AOP mit C++
 - ähnelt AspectJ
 - wird durch IDEs unterstützt
- Beispiele zeigen beispielsweise, dass ...
 - viele Entwurfsmuster von Aspekten profitieren
 - redundanter Code vermieden werden kann

Weitere Informationen

- **das** Web Portal der Community: www.aosd.net
 - weitere AOP Sprachen/Werkzeuge
 - Konferenzen/Workshops
 - mailing lists
- AspectC++: www.aspectc.org
 - alle Infos zum AspectC++ Projekt
 - mailing list
- Literatur: "Aspect-Oriented Software Development"
 - von R. Filman, T. Elrad, S. Clarke, M. Aksit



Vielen Dank für's zuhören!