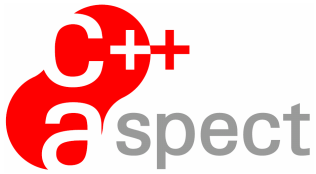
Aspektorientierte Programmierung

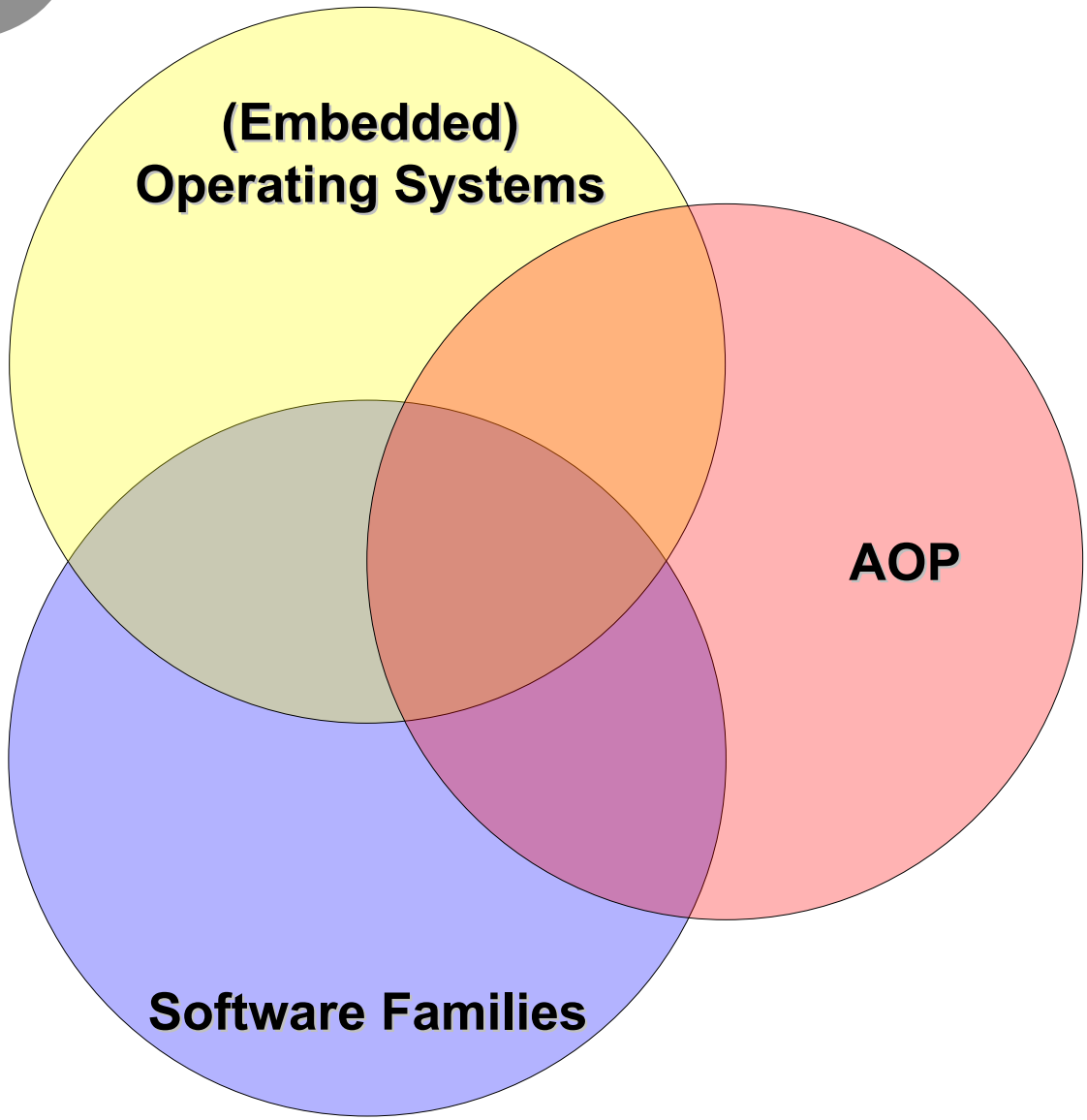
Eine Einführung am Beispiel von AspectC++

Daniel Lohmann
lohmann@informatik.uni-erlangen.de

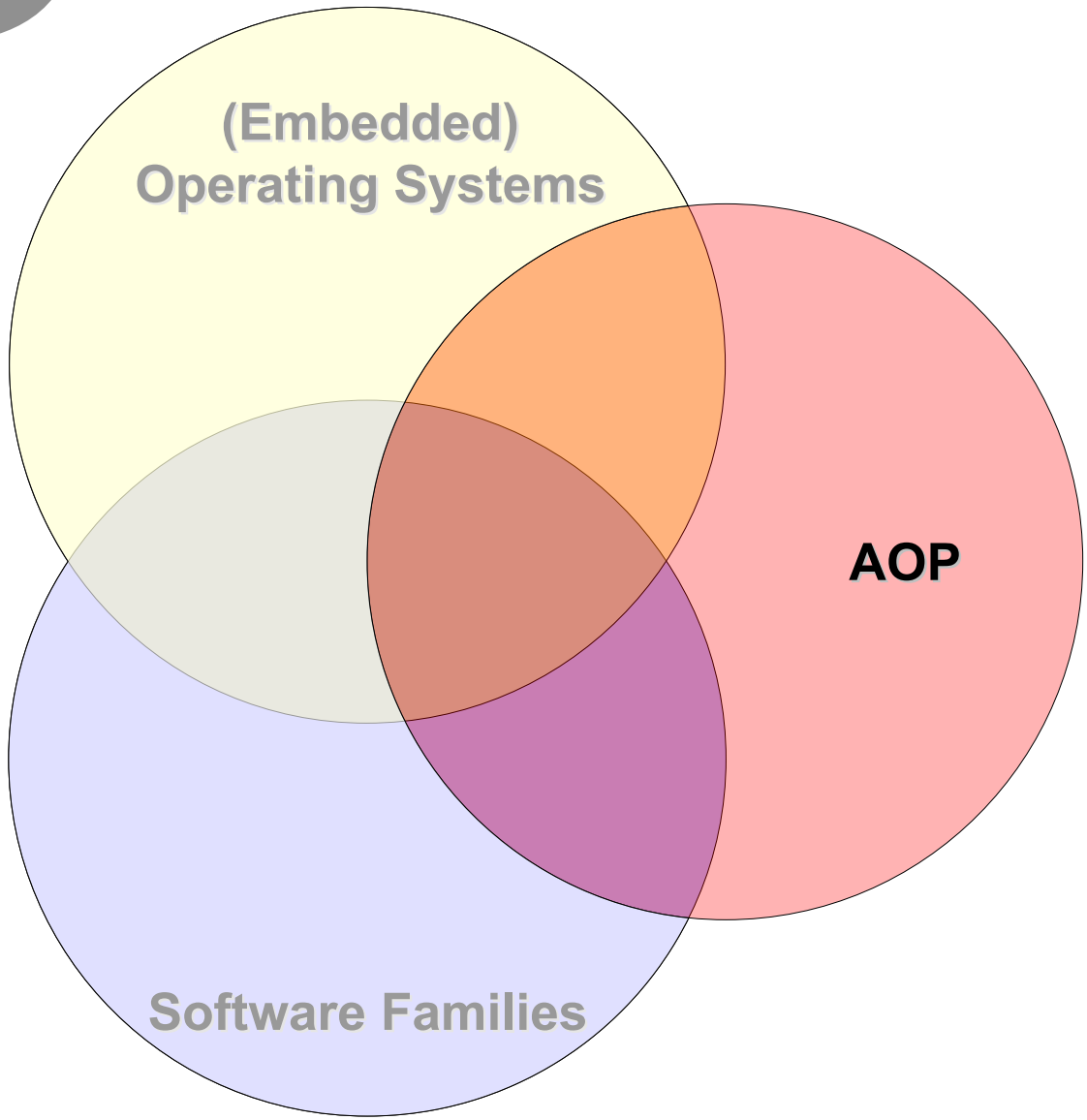
University of Erlangen-Nuremberg
Computer Science 4



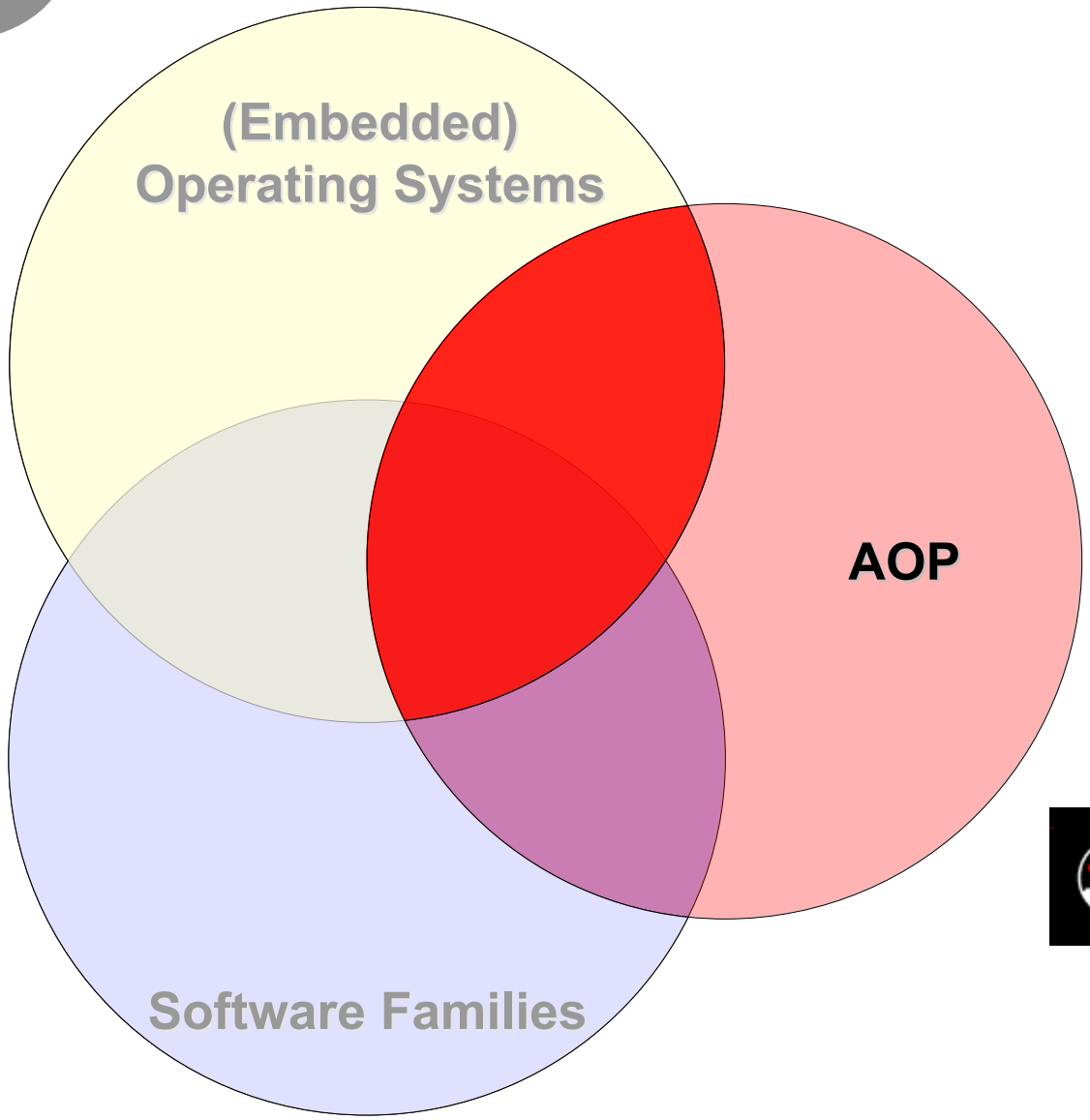
Softwaretechnik in BST



Inhalt dieses Vortrags



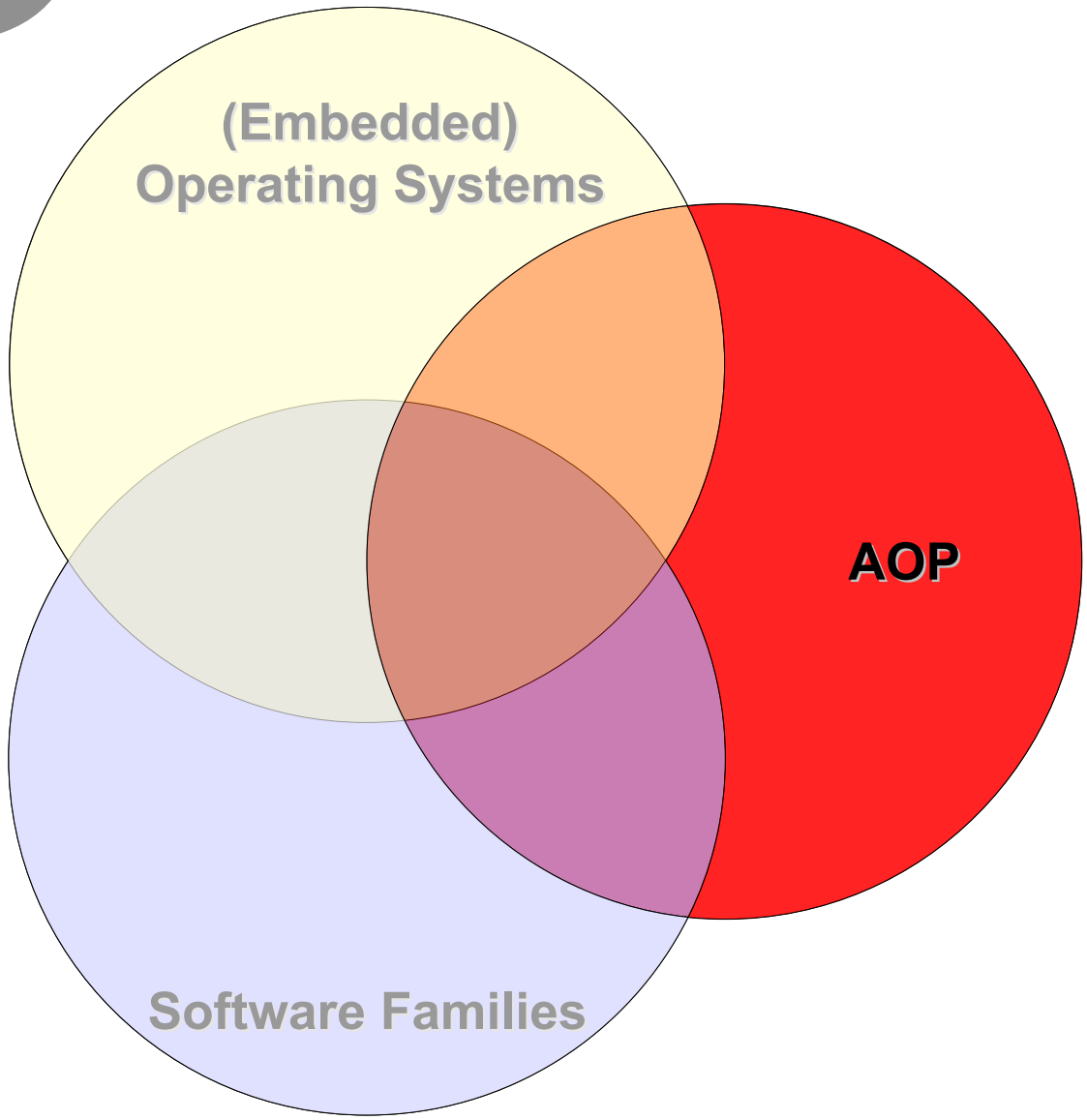
Inhalt dieses Vortrags



Motivation:
Fallstudie eCos

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

Inhalt dieses Vortrags



Hauptteil: AspectC++

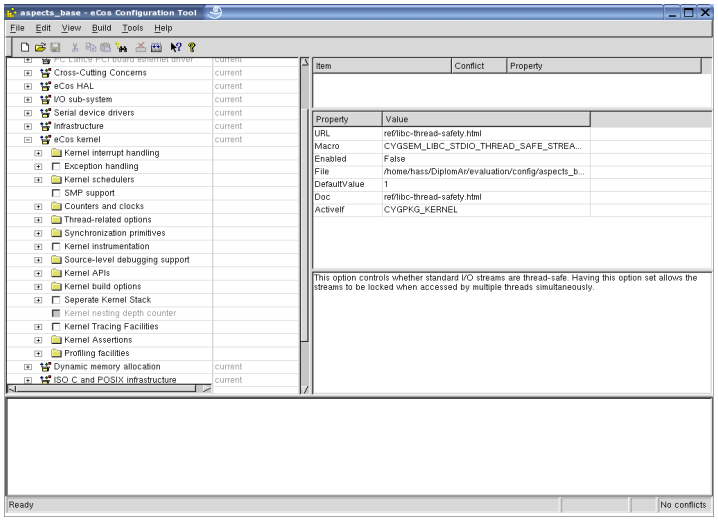
- Die Sprache
  - Grundbegriffe
  - Besonderheiten
- Tutorial
  - "Queue" Familie

Überblick

- **Querschneidende Belange in eCos**
  - Das Problem
- Aspektorientierte Programmierung
  - Der Lösungsansatz
- AspectC++
  - Grundlagen
  - Ergebnisse der eCos-Lösung
  - Tutorial
- 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

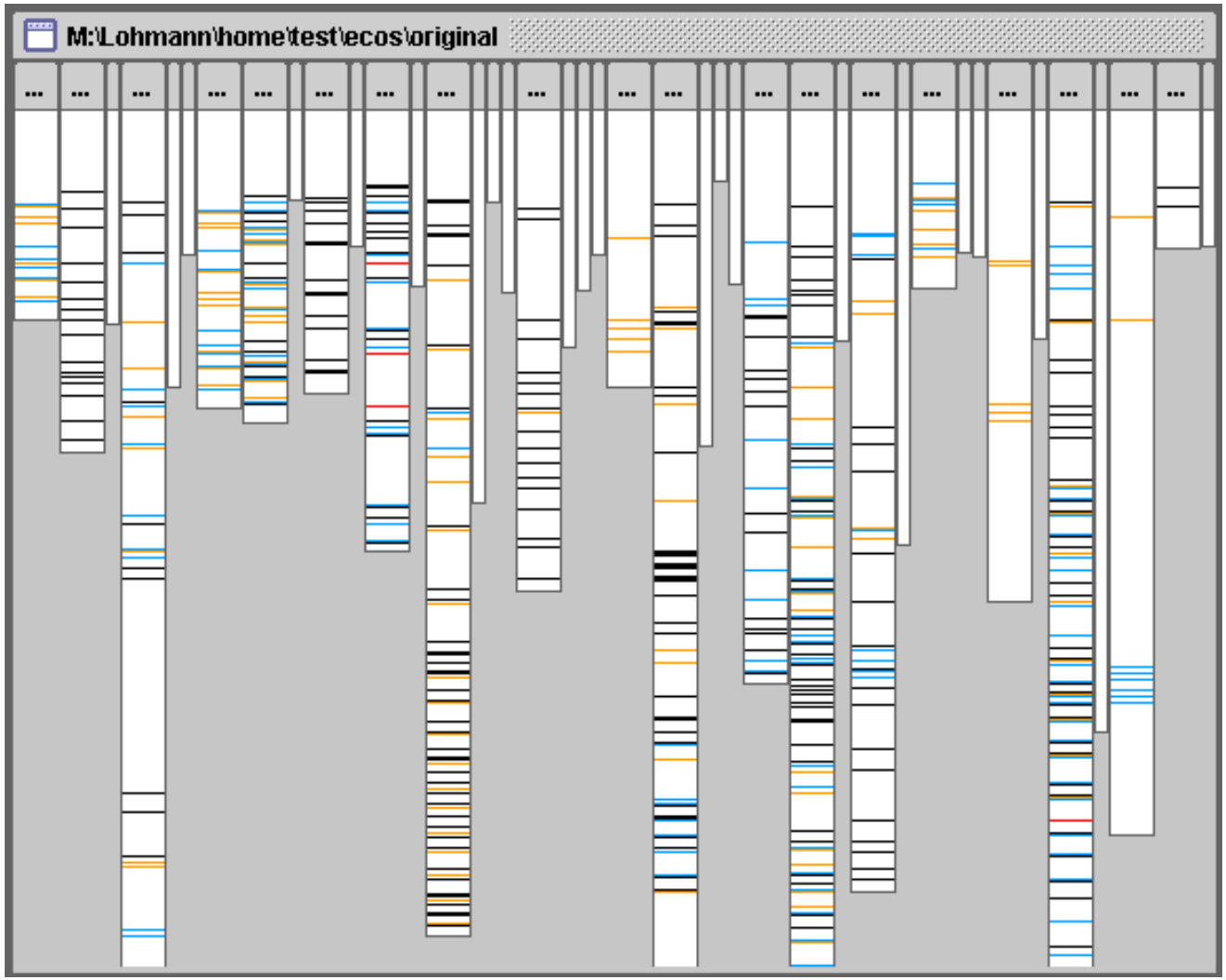
- Configuration approach
  - package selection (coarse-grained configuration)
  - **conditional compilation** (fine-grained configuration)

eCos: Global Policies in the Code

synchronization

instrumentation

tracing



eCos: Configuration Options

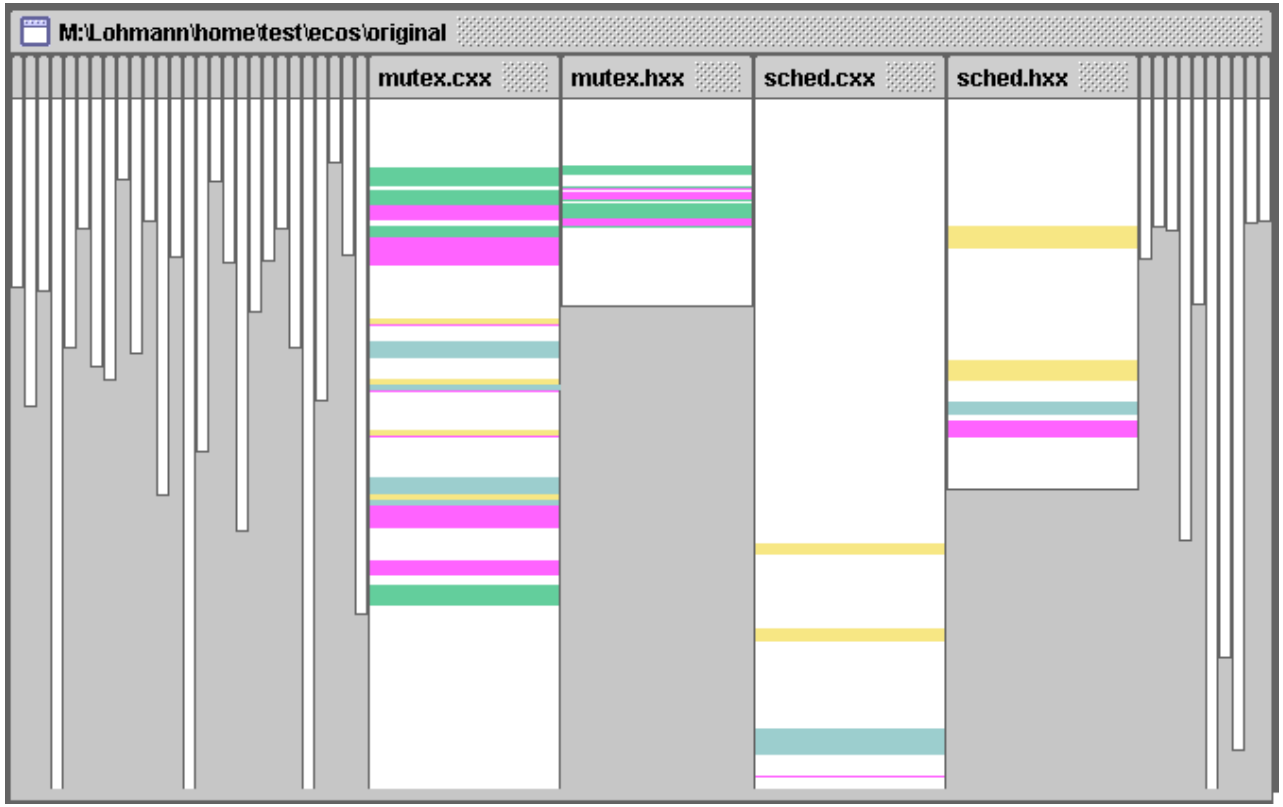
Variants of the optional mutex priority inversion protocol

simple

ceiling

inheritance

dynamic



eCos: Implementation Example

27 lines of code

```
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();
}
```

eCos: Implementation Example

2 lines for the **tracing policy**

```
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();
}
```

eCos: Implementation Example

21 (almost unreadable) lines for **optional features**

```
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();
}
```

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_KERNE                                                  LING
    protocol = CEIL
    ceiling  = CYGS                                                  AULT_PRI;
#endif
#ifdef CYGSEM_KERNE                                                  E
    protocol = NONE
#endif
#else // not (DYNAM
#ifdef CYGSEM_KERNE
#ifdef CYGSEM_KERNE                                                  ORITY
    // if there is a                                                 alize
    // the ceiling.
    ceiling = CYGSE                                                  ULT_PRIORITY;
#else
    ceiling = 0; //
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
     CYG_REPORT_RETURN();
}
```

Well, ...

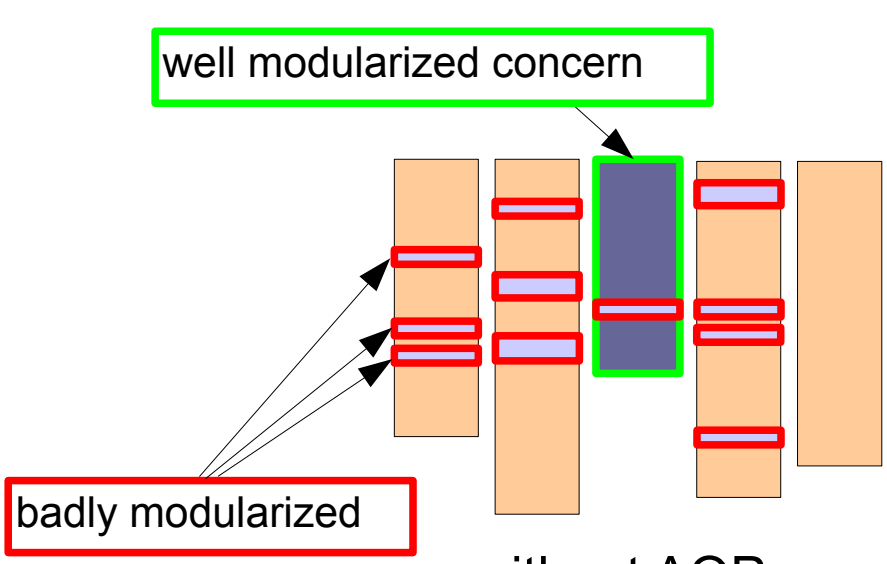
- comprehensability ?
- extensability ?
- reuseability ?
- maintainability ?

Überblick

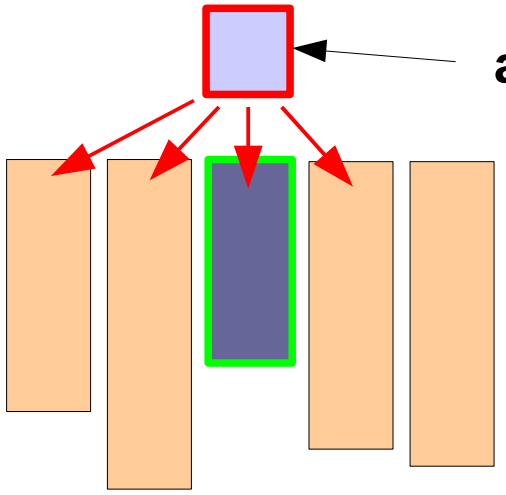
- Querschneidende Belange in eCos
  - Das Problem
- **Aspektorientierte Programmierung**
  - Der Lösungsansatz
- AspectC++
  - Grundlagen
  - Ergebnisse der eCos-Lösung
  - Tutorial
- Zusammenfassung

Aspekt-Orientierte Programmierung

AOP bietet Sprachmittel zur Trennung und Kapselung von querschneidenden Belangen



without AOP



with AOP

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 **deklaritiv** 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

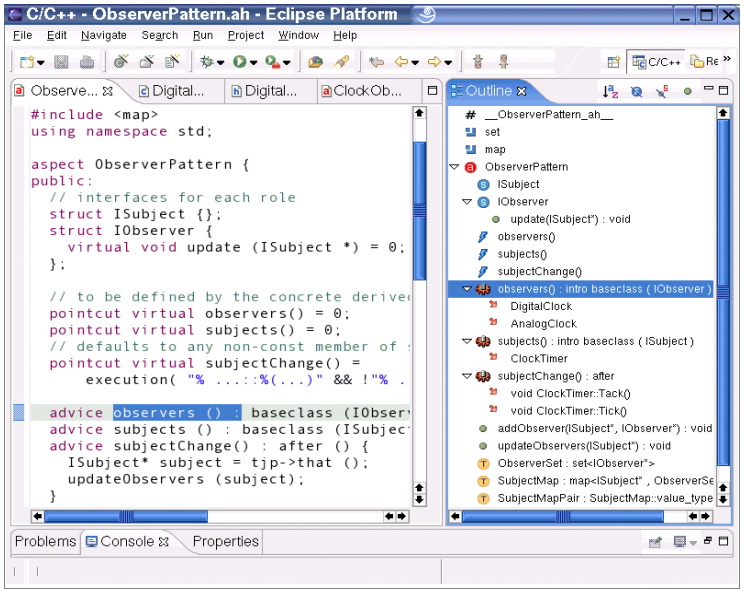
- Querschneidende Belange in eCos
  - Das Problem
- Aspektorientierte Programmierung
  - Der Lösungsansatz
- **AspectC++**
  - Grundlagen
  - Ergebnisse der eCos-Lösung
  - Tutorial
- Zusammenfassung

AspectC++

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

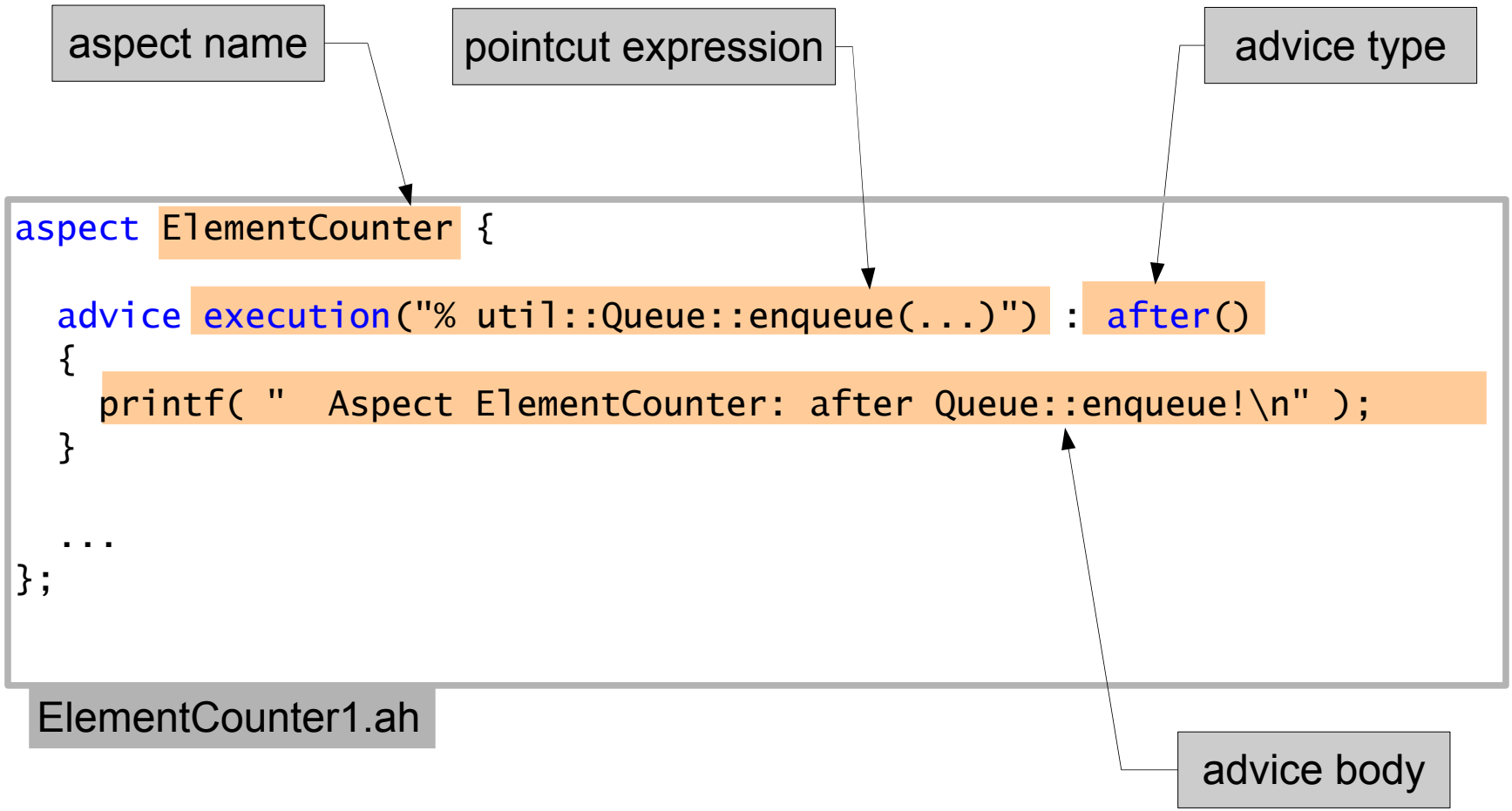
http://www.aspectc.org

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



- Aktive Nutzer-Community

Syntactic Elements



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 paramerter (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 explicitely invoked by `tjp->proceed()`

introductions

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

Before / After Advice

for execution join points:

advice execution(“void ClassA::foo()”) : **before**()

advice execution(“void ClassA::foo()”) : **after**()

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

for call join points:

advice call (“void ClassA::foo()”) : **before**()

advice call (“void ClassA::foo()”) : **after**()

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

Around Advice

for execution join points:

```
advice execution("void ClassA::foo()") :
around()
   before code

   tjp->proceed()

   after code
```

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

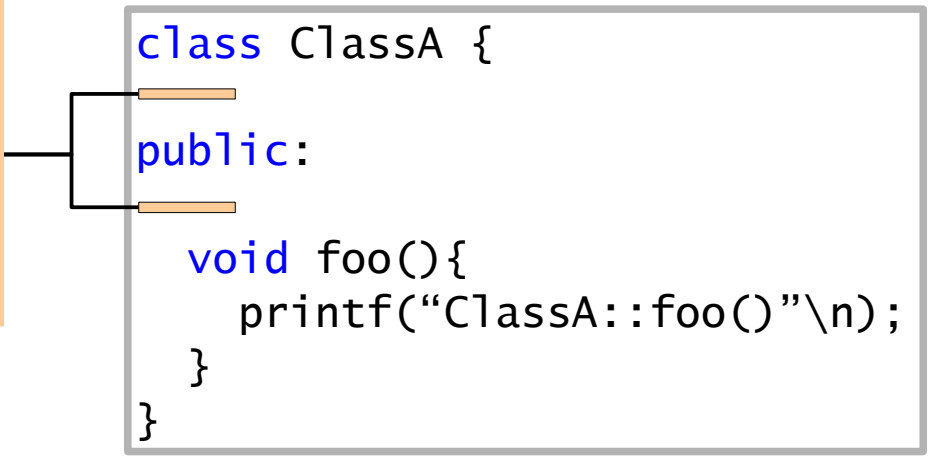
for call join points:

```
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
};
```



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

Example: Synchronization in AspeCos

```
aspect int_sync {

  pointcut sync() = execution(...) // kernel calls to sync
           || construction(...)
           || destruction(...);

  // advise kernel code to invoke lock() and unlock()
  advice sync() : before() {
    Cyg_Scheduler::lock();
  }
  advice sync() : after() {
    Cyg_Scheduler::unlock();
  }

  // In eCos, a new thread always starts with a lock value of 0
  advice execution(
  "%Cyg_HardwareThread::thread_entry(...)") : before() {
    Cyg_Scheduler::zero_sched_lock();
  }
  ...
};
```

Example: Synchronization in AspeCos

```
aspect int_sync {

  pointcut sync() = execution(...) // kernel calls to sync
           || construction(...)
           || destruction(...);

  // advise kernel code to invoke lock() and unlock()
  advice sync() : before() {
    Cyg_Scheduler::lock();
  }
  advice sync() : after() {
    Cyg_Scheduler::unlock();
  }

  // In eCos, a new thread always starts with a lock value of 0
  advice execution(
  "%Cyg_HardwareThread::thread_entry(...)") : before() {
    Cyg_Scheduler::zero_sched_lock();
  }
  ...
};
```

where

Example: Synchronization in AspeCos

```
aspect int_sync {

  pointcut sync() = execution(...) // kernel calls to sync
                  || construction(...)
                  || destruction(...);

  // advise kernel code to invoke lock() and unlock()
  advice sync() : before() {
    Cyg_Scheduler::lock();
  }
  advice sync() : after() {
    Cyg_Scheduler::unlock();
  }

  // In eCos, a new thread always starts with a lock value of 0
  advice execution(
  "%Cyg_HardwareThread::thread_entry(...)") : before() {
    Cyg_Scheduler::zero_sched_lock();
  }
  ...
};
```

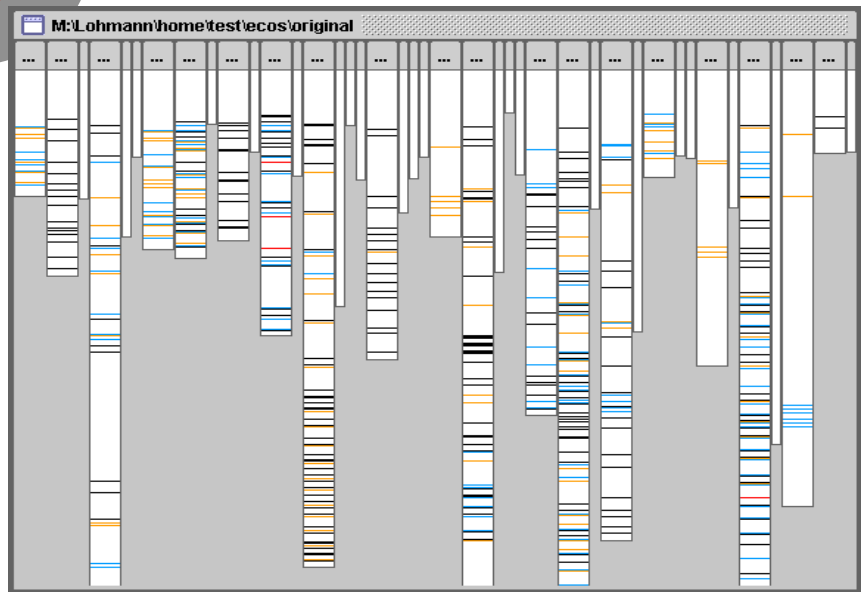
where

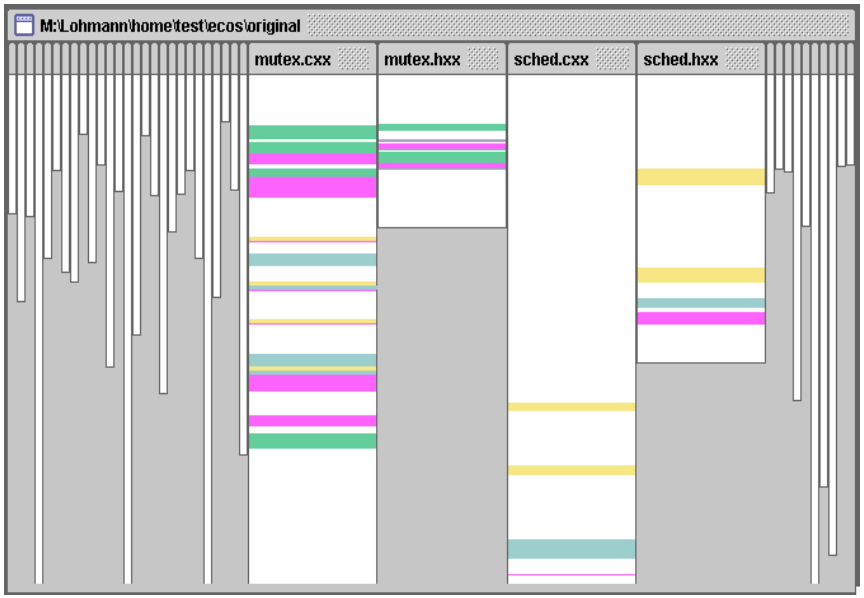
what

Results with eCos

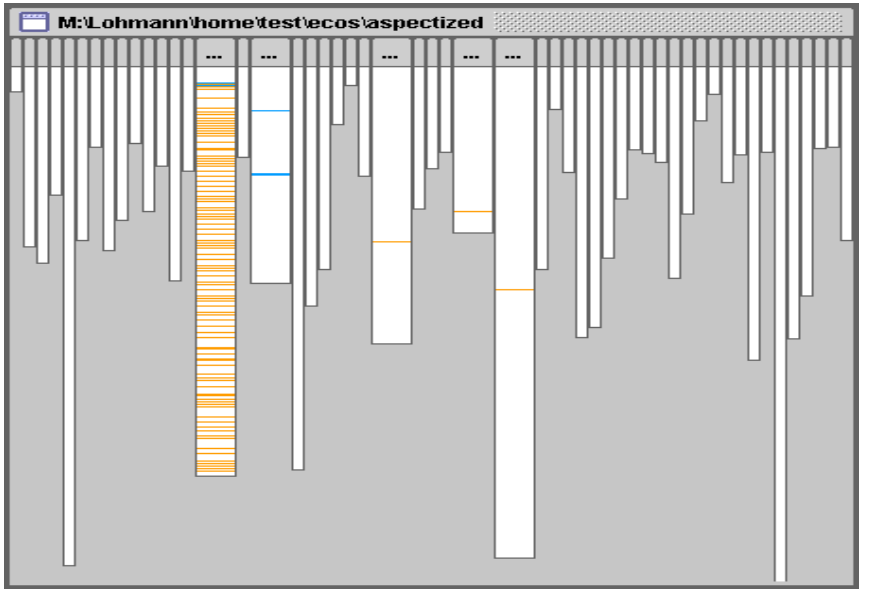
- **Refactored: *original kernel* → *aspectized kernel***
  - **3 cross-cutting policies**
    - interrupt synchronization — 187 invocations → 160 code join-points
    - kernel instrumentation — 162 invocations → 139 code join-points
    - tracing — 336 invocations → 632 code join-points
  - **12 configuration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

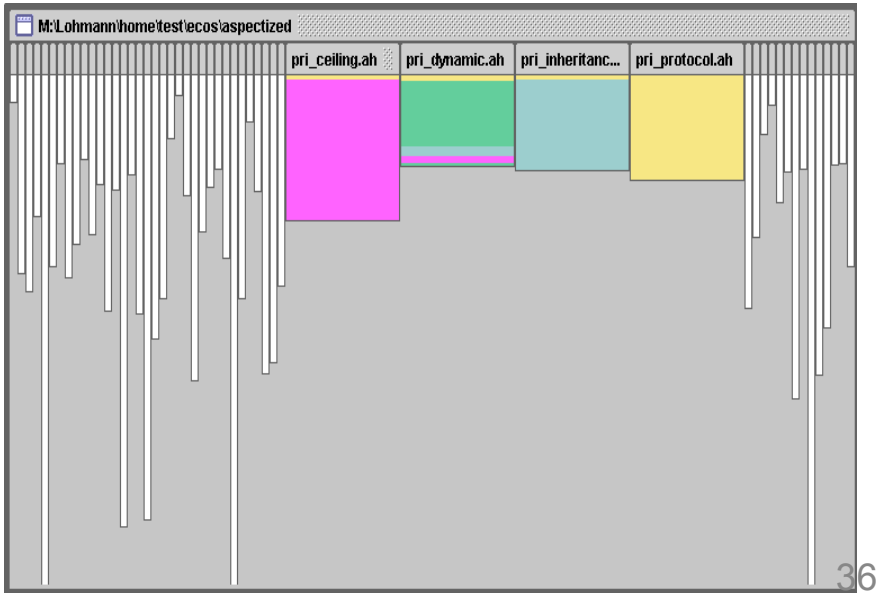
original kernel





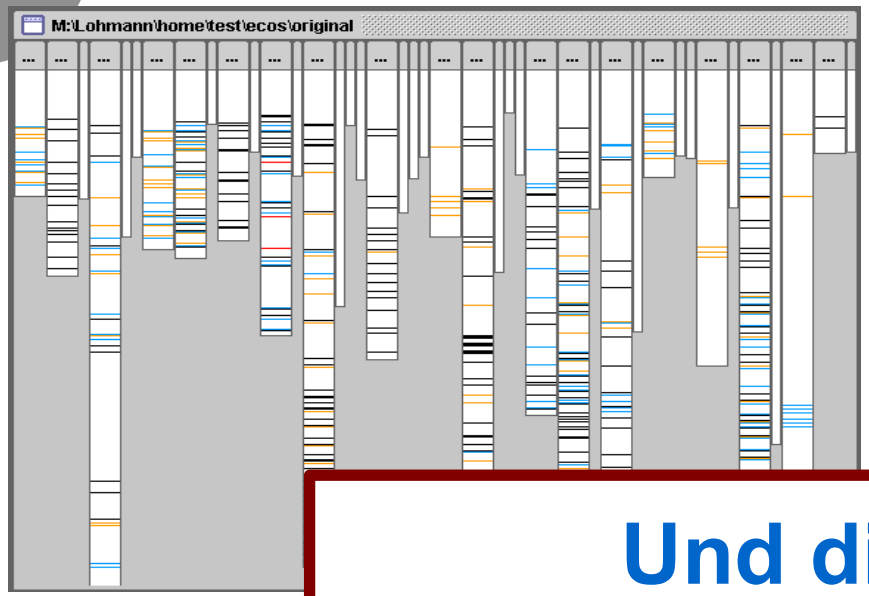
aspectized kernel

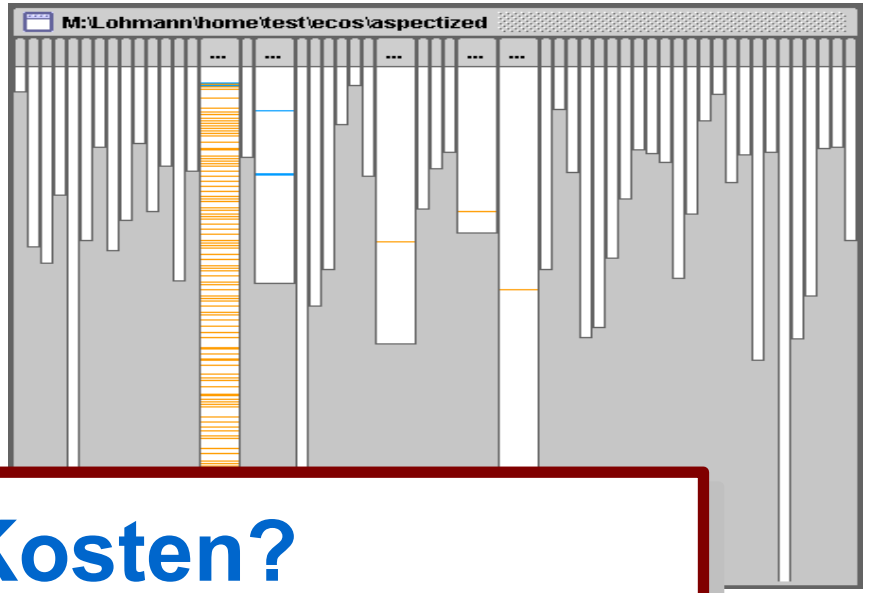




original kernel

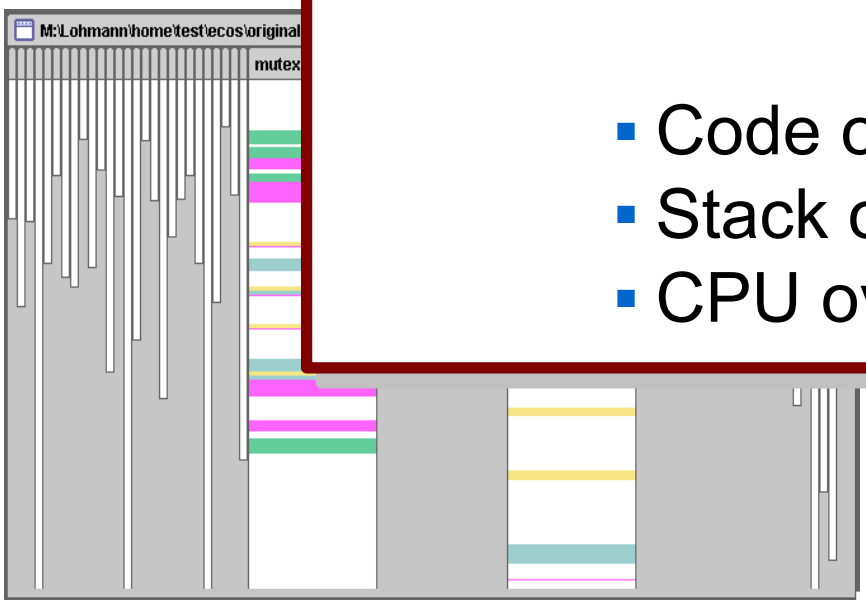
aspectized kernel

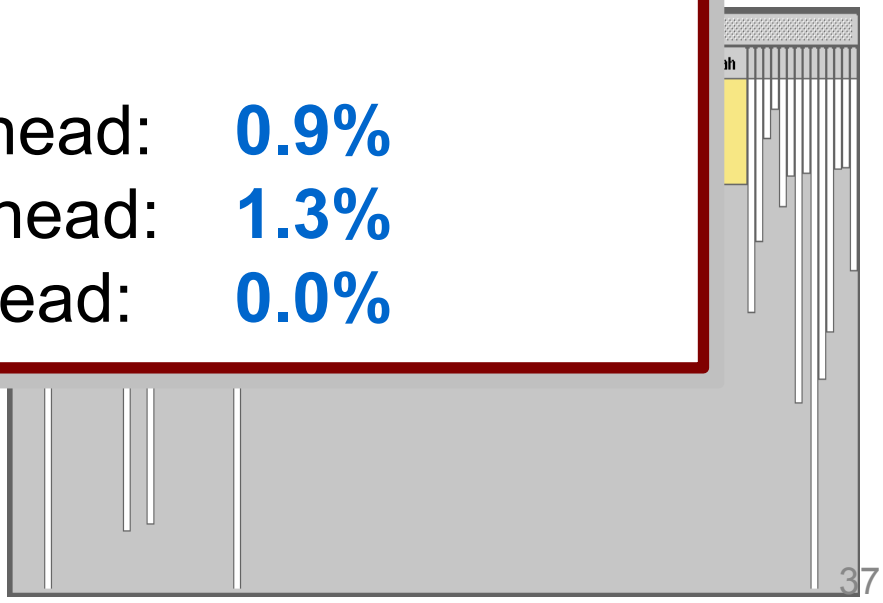




Und die Kosten?

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





Evaluation Case Study: CiAO-AS

Document Title	Specification of

AUTOSAR

Specification of Operating System
V2.0.1

OS093: If interrupts are disabled and any OS services, excluding the interrupt services, are called outside of hook routines, then the Operating System shall return `E_OS_DISABLEDINT`

Evaluation Case Study: CiAO-AS

AUTOSAR

Specification of Operating System
V2.0.1

OS093: If interrupts are disabled and any OS services, excluding the interrupt services, are called outside of hook routines, then the Operating System shall return `E_OS_DISABLEDINT`

```
aspect DisabledIntCheck {
  advice call( pcOSServices() && !pcInterruptServices() )
  && !within( pcHookRoutines() ) : around() {
    if( interruptsDisabled() )
      *tjp->result() = E_OS_DISABLEDINT;
    else
      tjp->proceed();
} };
```

Evaluation Case Study: CiAO-AS

AUTOSAR Specification of Operating System V2.0.1
Document Title Specification of

AUTOSAR

Specification of Operating System
V2.0.1

OS093: If interrupts are disabled and any OS services, excluding the interrupt services, are called outside of hook routines, then the Operating System shall return `E_OS_DISABLEDINT`

```
aspect DisabledIntCheck {
  advice call( pcOSServices() && !pcInterruptServices() )
  && !within( pcHookRoutines() ) : around() {
    if( interruptsDisabled() )
      *tjp->result() = E_OS_DISABLEDINT;
    else
      tjp->proceed();
} };
```

Evaluation Case Study: CiAO-AS

AUTOSAR

Specification of Operating System
V2.0.1

OS093: If interrupts are disabled and any OS services, excluding the interrupt services, are called outside of hook routines, then the Operating System shall return `E_OS_DISABLEDINT`

```
aspect DisabledIntCheck {
  advice call( pcOSServices() && !pcInterruptServices() )
  && !within( pcHookRoutines() ) : around() {
    if( interruptsDisabled() )
      *tjp->result() = E_OS_DISABLEDINT;
    else
      tjp->proceed();
} };
```

Evaluation Case Study: CiAO-AS

AUTOSAR

Specification of Operating System
V2.0.1

OS093: If interrupts are disabled and any OS services, excluding the interrupt services, are called outside of hook routines, then the Operating System shall return `E_OS_DISABLEDINT`

```
aspect DisabledIntCheck {
  advice call( pcOSServices() && !pcInterruptServices() )
  && !within( pcHookRoutines() ) : around() {
    if( interruptsDisabled() )
      *tjp->result() = E_OS_DISABLEDINT;
    else
      tjp->proceed();
} };
```

Evaluation Case Study: CiAO-AS

Document Title	Specification of

AUTOSAR

Specification of Operating System
V2.0.1

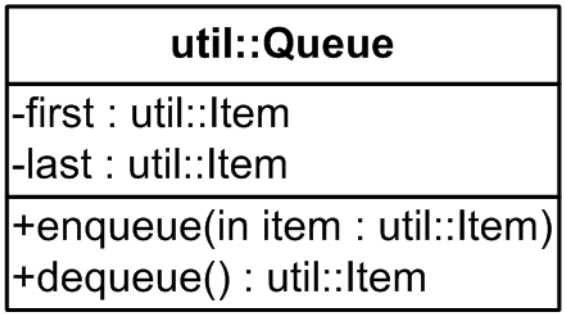
OS093: If interrupts are disabled and any OS services, excluding the interrupt services, are called outside of hook routines, then the Operating System shall return `E_OS_DISABLEDINT`

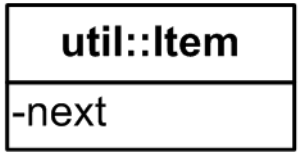
```
aspect DisabledIntCheck {
  advice call( pcOSServices() && !pcInterruptServices() )
  && !within( pcHookRoutines() ) : around() {
    if( interruptsDisabled() )
      *tjp->result() = E_OS_DISABLEDINT;
    else
      tjp->proceed();
} };
```

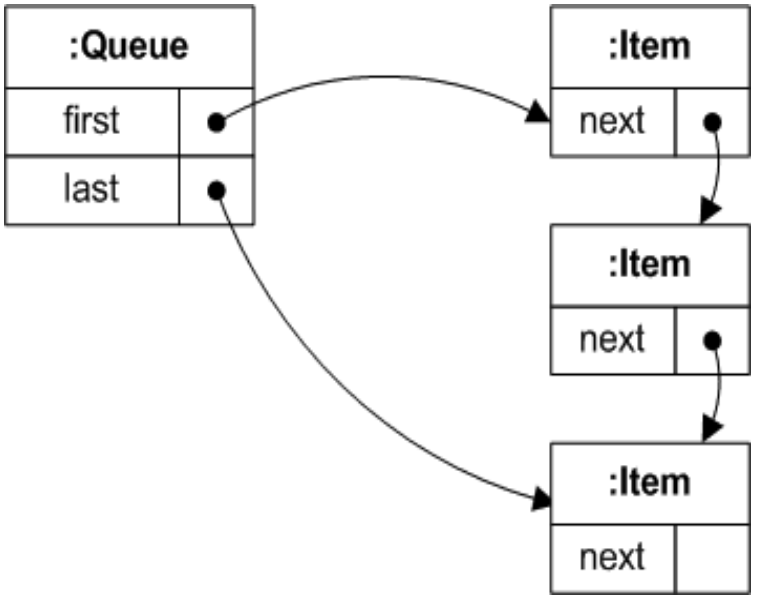
Überblick

- Querschneidende Belange in eCos
  - Das Problem
- Aspektorientierte Programmierung
  - Der Lösungsansatz
- AspectC++
  - Grundlagen
  - Ergebnisse der eCos-Lösung
  - **Tutorial**
- Zusammenfassung

Scenario: A Queue utility class







The Simple Queue Class

```
namespace util {
  class Item {
    friend class Queue;
    Item* next;
  public:
    Item() : next(0){}
  };

  class Queue {
    Item* first;
    Item* last;
  public:
    Queue() : first(0), last(0) {}

    void enqueue( Item* item ) {
      printf( "  > Queue::enqueue()\n" );
      if( last ) {
        last->next = item;
        last = item;
      } else
        last = first = item;
      printf( "  < Queue::enqueue()\n" );
    }

    Item* dequeue() {
      printf("  > Queue::dequeue()\n");
      Item* res = first;
      if( first == last )
        first = last = 0;
      else
        first = first->next;
      printf("  < Queue::dequeue()\n");
      return res;
    }
  }; // class Queue
} // namespace util
```

Scenario: The Problem

Various users of Queue demand extensions:

I want Queue to throw exceptions!

Please extend the Queue class by an element counter!

Queue should be thread-safe!

The Not So Simple Queue Class

```
class Queue {
  Item *first, *last;
  int counter;
  os::Mutex lock;
public:
  Queue () : first(0), last(0) {
    counter = 0;
  }
  void enqueue(Item* item) {
    lock.enter();
    try {
      if (item == 0)
        throw QueueInvalidItemError();
      if (last) {
        last->next = item;
        last = item;
      } else { last = first = item; }
      ++counter;
    } catch (...) {
      lock.leave(); throw;
    }
    lock.leave();
  }
```

```
  Item* dequeue() {
    Item* res;
    lock.enter();
    try {
      res = first;
      if (first == last)
        first = last = 0;
      else first = first->next;
      if (counter > 0) --counter;
      if (res == 0)
        throw QueueEmptyError();
    } catch (...) {
      lock.leave();
      throw;
    }
    lock.leave();
    return res;
  }
  int count() { return counter; }
}; // class Queue
```

What Code Does What?

```
class Queue {
  Item *first, *last;
  int counter;
  os::Mutex lock;
public:
  Queue () : first(0), last(0) {
    counter = 0;
  }
  void enqueue(Item* item) {
    lock.enter();
    try {
      if (item == 0)
        throw QueueInvalidItemError();
      if (last) {
        last->next = item;
        last = item;
      } else { last = first = item; }
      ++counter;
    } catch (...) {
      lock.leave(); throw;
    }
    lock.leave();
  }
```

```
    Item* dequeue() {
      Item* res;
      lock.enter();
      try {
        res = first;
        if (first == last)
          first = last = 0;
        else first = first->next;
        if (counter > 0) --counter;
        if (res == 0)
          throw QueueEmptyError();
      } catch (...) {
        lock.leave();
        throw;
      }
      lock.leave();
      return res;
    }
    int count() { return counter; }
  }; // class Queue
```

The Simple Queue Class Revisited

```
namespace util {
  class Item {
    friend class Queue;
    Item* next;
  public:
    Item() : next(0){}
  };

  class Queue {
    Item* first;
    Item* last;
  public:
    Queue() : first(0), last(0) {}

    void enqueue( Item* item ) {
      printf( "  > Queue::enqueue()\n" );
      if( last ) {
        last->next = item;
        last = item;
      } else
        last = first = item;
      printf( "  < Queue::enqueue()\n" );
    }

    Item* dequeue() {
      printf("  > Queue::dequeue()\n");
      Item* res = first;
      if( first == last )
        first = last = 0;
      else
        first = first->next;
      printf("  < Queue::dequeue()\n");
      return res;
    }
  }; // class Queue

} // namespace util
```

Queue: Demanded Extensions

I. Element counting

Please extend the Queue class by an element counter!

II. Errorhandling
(signaling of errors by exceptions)

III. Thread safety
(synchronization by mutex variables)

Element counting: The Idea

- Increment a counter variable after each execution of `util::Queue::enqueue()`
- Decrement it after each execution of `util::Queue::dequeue()`

ElementCounter1

```
aspect ElementCounter {

  int counter;
  ElementCounter() {
    counter = 0;
  }

  advice execution("% util::Queue::enqueue(...)") : after() {
    ++counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
  advice execution("% util::Queue::dequeue(...)") : after() {
    if( counter > 0 ) --counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
};
```

ElementCounter1.ah

ElementCounter1 - Elements

```
aspect ElementCounter {

  int counter;
  ElementCounter() {
    counter = 0;
  }

  advice execution("% util::Queue::enqueue(...)") : after() {
    ++counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
  advice execution("% util::Queue::dequeue(...)") : after() {
    if( counter > 0 ) --counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
};
```

ElementCounter1.ah

We introduced a new **aspect** named *ElementCounter*.
An aspect starts with the keyword aspect and is syntactically much like a class.

ElementCounter1 - Elements

```
aspect ElementCounter {

  int counter;
  ElementCounter() {
    counter = 0;
  }

  advice execution("% util::Queue::enqueue(...)") : after() {
    ++counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
  advice execution("% util::Queue::dequeue(...)") : after() {
    if( counter > 0 ) --counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
};
```

Like a class, an aspect can define data members, constructors and so on

ElementCounter1.ah

ElementCounter1 - Elements

```
aspect ElementCounter {

  int counter;
  ElementCounter() {
    counter = 0;
  }

  advice execution("% util::Queue::enqueue(...)") : after() {
    ++counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
  advice execution("% util::Queue::dequeue(...)") : after() {
    if( counter > 0 ) --counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
};
```

We give **after advice** (= some crosscuting code to be executed after certain control flow positions)

ElementCounter1.ah

ElementCounter1 - Elements

This **pointcut expression** denotes where the advice should be given. (After **execution** of methods that match the pattern)

```
aspect ElementCounter {

  int counter;
  ElementCounter() {
    counter = 0;
  }

  advice execution("% util::Queue::enqueue(...)") : after() {
    ++counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
  advice execution("% util::Queue::dequeue(...)") : after() {
    if( counter > 0 ) --counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
};
```

ElementCounter1.ah

ElementCounter1 - Elements

```
aspect ElementCounter {

  int counter;
  ElementCounter() {
    counter = 0;
  }

  advice execution("% util::Queue::enqueue(...)") : after() {
    ++counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
  advice execution("% util::Queue::dequeue(...)") : after() {
    if( counter > 0 ) --counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", counter );
  }
};
```

Aspect member elements can be accessed from within the advice body

ElementCounter1.ah

ElementCounter1 - Result

```
int main() {
  util::Queue queue;

  printf("main(): enqueueing an item\n");
  queue.enqueue( new util::Item );

  printf("main(): dequeueing two items\n");
  Util::Item* item;
  item = queue.dequeue();
  item = queue.dequeue();
}
```

main.cc

```
main(): enqueueing am item
  > Queue::enqueue(00320FD0)
  < Queue::enqueue(00320FD0)
  Aspect ElementCounter: # of elements = 1
 main(): dequeueing two items
  > Queue::dequeue()
  < Queue::dequeue() returning 00320FD0
  Aspect ElementCounter: # of elements = 0
  > Queue::dequeue()
  < Queue::dequeue() returning 00000000
  Aspect ElementCounter: # of elements = 0
```

<Output>

ElementCounter1 – What's next?

- The aspect is not the ideal place to store the counter, because it is shared between all Queue instances
- Ideally, counter becomes a member of Queue
- In the next step, we
  - move counter into Queue by **introduction**
  - **expose context** about the aspect invocation to access the current Queue instance

ElementCounter2

```
aspect ElementCounter {

  advice "util::Queue" : slice class {
    int counter;
  public:
    int count() const { return counter; }
  };
  advice execution("% util::Queue::enqueue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    ++queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice execution("% util::Queue::dequeue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    if( queue.count() > 0 ) --queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice construction("util::Queue")
                   && that(queue) : before( util::Queue& queue ) {
    queue.counter = 0;
  }
};
```

ElementCounter2.ah

ElementCounter2 - Elements

```
aspect ElementCounter {

  advice "util::Queue" : slice class {
    int counter;
  public:
    int count() const { return counter; }
  };
  advice execution("% util::Queue::enqueue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    ++queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice execution("% util::Queue::dequeue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    if( queue.count() > 0 ) --queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice construction("util::Queue")
                   && that(queue) : before( util::Queue& queue ) {
    queue.counter = 0;
  }
};
```

Introduces a **slice** of members into all classes denoted by the pointcut "util::Queue"

ElementCounter2.ah

ElementCounter2 - Elements

```
aspect ElementCounter {

  advice "util::Queue" : slice class {
    int counter;
  public:
    int count() const { return counter; }
  };
  advice execution("% util::Queue::enqueue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    ++queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice execution("% util::Queue::dequeue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    if( queue.count() > 0 ) --queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice construction("util::Queue")
                   && that(queue) : before( util::Queue& queue ) {
    queue.counter = 0;
  }
};
```

We introduce a private *counter* element and a public method to read it

ElementCounter2.ah

ElementCounter2 - Elements

A **context variable** *queue* is bound to *that* (the calling instance). The calling instance has to be an util::Queue

```
aspect ElementCounter {

  advice "util::Queue" : slice class {
    int counter;
  public:
    int count() const { return counter; }
  };
  advice execution("% util::Queue::enqueue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    ++queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice execution("% util::Queue::dequeue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    if( queue.count() > 0 ) --queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice construction("util::Queue")
                   && that(queue) : before( util::Queue& queue ) {
    queue.counter = 0;
  }
};
```

ElementCounter2.ah

ElementCounter2 - Elements

```
aspect ElementCounter {

  advice "util::Queue" : slice class {
    int counter;
  public:
    int count() const { return counter; }
  };
  advice execution("% util::Queue::enqueue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    ++queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice execution("% util::Queue::dequeue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    if( queue.count() > 0 ) --queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice construction("util::Queue")
                   && that(queue) : before( util::Queue& queue ) {
    queue.counter = 0;
  }
};
```

The context variable *queue* is used to access the calling instance.

ElementCounter2.ah

ElementCounter2 - Elements

```
aspect ElementCounter {

  advice "util::Queue" : slice class {
    int counter;
  public:
    int count() const { return counter; }
  };
  advice execution("% util::Queue::enqueue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    ++queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice execution("% util::Queue::dequeue(...)")
                   && that(queue) : after( util::Queue& queue )  {
    if( queue.count() > 0 ) --queue.counter;
    printf( "  Aspect ElementCounter: # of elements = %d\n", queue.count() );
  }
  advice construction("util::Queue")
                   && that(queue) : before( util::Queue& queue ) {
    queue.counter = 0;
  }
};
```

By giving **construction advice** we ensure that counter gets initialized

ElementCounter2.ah

ElementCounter2 - Result

```
int main() {
  util::Queue queue;
  printf("main(): Queue contains %d items\n", queue.count());
  printf("main(): enqueueing some items\n");
  queue.enqueue(new util::Item);
  queue.enqueue(new util::Item);
  printf("main(): Queue contains %d items\n", queue.count());
  printf("main(): dequeueing one items\n");
  util::Item* item;
  item = queue.dequeue();
  printf("main(): Queue contains %d items\n", queue.count());
}
```

main.cc

ElementCounter2 - Result

```
int main() {
  util::Queue queue;
  printf("main(): Queue contains %d items\n", queue.count());
  printf("main(): enqueueing some items\n");
  queue.enqueue(new util::Item);
  queue.enqueue(new util::Item);
  printf("main(): Queue contains
  printf("main(): dequeueing one
  util::Item* item;
  item = queue.dequeue();
  printf("main(): Queue contains
}
```

main.cc

```
main(): Queue contains 0 items
main(): enqueueing some items
  > Queue::enqueue(00320FD0)
  < Queue::enqueue(00320FD0)
  Aspect ElementCounter: # of elements = 1
  > Queue::enqueue(00321000)
  < Queue]::enqueue(00321000)
  Aspect ElementCounter: # of elements = 2
main(): Queue contains 2 items
main(): dequeueing one items
  > Queue::dequeue()
  < Queue::dequeue() returning 00320FD0
  Aspect ElementCounter: # of elements = 1
main(): Queue contains 1 items
```

<Output>

Queue: Demanded Extensions

I. Element counting

II. Errorhandling
(signaling of errors by exceptions)

III. Thread safety
(synchronization by mutex variables)

Errorhandling: The Idea

- We want to check the following constraints:
  - enqueue() is never called with a NULL item
  - dequeue() is never called on an empty queue
- In case of an error an exception should be thrown

- To implement this, we need access to ...
  - the parameter passed to enqueue()
  - the return value returned by dequeue()

  ... from within the advice

ErrorException

```
namespace util {
  struct QueueInvalidItemError {};
  struct QueueEmptyError {};
}

aspect ErrorException {

  advice execution("% util::Queue::enqueue(...)") && args(item)
      : before(util::Item* item) {
    if( item == 0 )
      throw util::QueueInvalidItemError();
  }
  advice execution("% util::Queue::dequeue(...)") && result(item)
      : after(util::Item* item) {
    if( item == 0 )
      throw util::QueueEmptyError();
  }
};
```

ErrorException.ah

ErrorException - Elements

```
namespace util {
  struct QueueInvalidItemError {};
  struct QueueEmptyError {};
}

aspect ErrorException {

  advice execution("% util::Queue::enqueue(...)") && args(item)
      : before(util::Item* item) {
    if( item == 0 )
      throw util::QueueInvalidItemError();
  }
  advice execution("% util::Queue::dequeue(...)") && result(item)
      : after(util::Item* item) {
    if( item == 0 )
      throw util::QueueEmptyError();
  }
};
```

We give advice to be executed *before* enqueue() and *after* dequeue()

ErrorException.ah

ErrorException - Elements

```
namespace util {
  struct QueueInvalidItemEr
  struct QueueEmptyError {}
}

aspect ErrorException {

  advice execution("% util::Queue::enqueue(...)") && args(item)
      : before(util::Item* item) {
    if( item == 0 )
      throw util::QueueInvalidItemError();
  }
  advice execution("% util::Queue::dequeue(...)") && result(item)
      : after(util::Item* item) {
    if( item == 0 )
      throw util::QueueEmptyError();
  }
};
```

ErrorException.ah

A **context variable** *item* is bound to the first **argument** of type *util::Item** passed to the matching methods

ErrorException - Elements

```
namespace util {
  struct QueueInvalidItemEr
  struct QueueEmptyError {}
}

aspect ErrorException {

  advice execution("% util::Queue::enqueue(...)") && args(item)
      : before(util::Item* item) {
    if( item == 0 )
      throw util::QueueInvalidItemError();
  }
  advice execution("% util::Queue::dequeue(...)") && result(item)
      : after(util::Item* item) {
    if( item == 0 )
      throw util::QueueEmptyError();
  }
};
```

ErrorException.ah

Here the **context variable** *item* is bound to the **result** of type *util::Item** returned by the matching methods

Queue: Demanded Extensions

I. Element counting

II. Errorhandling
(signaling of errors by exceptions)

III. Thread safety
(synchronization by mutex variables)

Thread Safety: The Idea

- Protect enqueue() and dequeue() by a mutex object

- To implement this, we need to
  - introduce a mutex variable into class Queue
  - lock the mutex before the execution of enqueue() / dequeue()
  - unlock the mutex after execution of enqueue() / dequeue()

- The aspect implementation should be exception safe!
  - in case of an exception, pending after advice is not called
  - solution: use around advice

LockingMutex

```
aspect LockingMutex {
  advice "util::Queue" : slice class { os::Mutex lock; };

  pointcut sync_methods() = "% util::Queue::%queue(...)";

  advice execution(sync_methods()) && that(queue)
  : around( util::Queue& queue ) {
    queue.lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      queue.lock.leave();
      throw;
    }
    queue.lock.leave();
  }
};
```

LockingMutex.ah

LockingMutex - Elements

```
aspect LockingMutex {
  advice "util::Queue" : slice class { os::Mutex lock; };

  pointcut sync_methods() = "% util::Queue::%queue(...)";

  advice execution(sync_methods()) && that(queue)
  : around( util::Queue& queue ) {
    queue.lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      queue.lock.leave();
      throw;
    }
    queue.lock.leave();
  }
};
```

We introduce a mutex member into class Queue

LockingMutex.ah

LockingMutex - Elements

```
aspect LockingMutex {
  advice "util::Queue" : slice class { os::Mutex lock; };

  pointcut sync_methods() = "% util::Queue::%queue(...)";

  advice execution(sync_methods()) && that(queue)
  : around( util::Queue& queue ) {
    queue.lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      queue.lock.leave();
      throw;
    }
    queue.lock.leave();
  }
};
```

LockingMutex.ah

Pointcuts can be named. *sync_methods* describes all methods that have to be synchronized by the mutex

LockingMutex - Elements

```
aspect LockingMutex {
  advice "util::Queue" : slice class { os::Mutex lock; };

  pointcut sync_methods() = "% util::Queue::%queue(...)";

  advice execution(sync_methods()) && that(queue)
  : around( util::Queue& queue ) {
    queue.lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      queue.lock.leave();
      throw;
    }
    queue.lock.leave();
  }
};
```

LockingMutex.ah

sync_methods is used to give around advice to the execution of the methods

LockingMutex - Elements

```
aspect LockingMutex {
  advice "util::Queue" : slice class { os::Mutex lock; };

  pointcut sync_methods() = "% util::Queue::%queue(...)";

  advice execution(sync_methods()) && that(queue)
  : around( util::Queue& queue ) {
    queue.lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      queue.lock.leave();
      throw;
    }
    queue.lock.leave();
  }
};
```

LockingMutex.ah

By calling tjp->proceed() the original method is executed

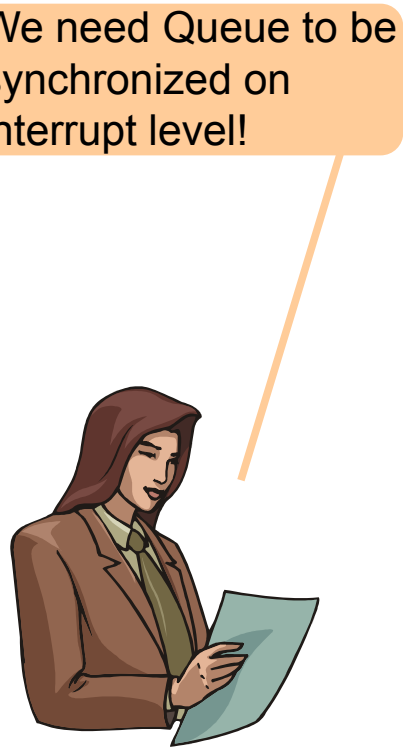
Queue: A new Requirement

I. Element counting

II. Errorhandling
(signaling of errors by exceptions)

III. Thread safety
(synchronization by mutex variables)

IV. Interrupt safety
(synchronization on interrupt level)



Interrupt Safety: The Idea

- Scenario
  - Queue is used to transport objects between kernel code (interrupt handlers) and application code
  - If application code accesses the queue, interrupts must be disabled first
  - If kernel code accesses the queue, interrupts must not be disabled
- To implement this, we need to distinguish
  - if the call is made from kernel code, or
  - if the call is made from application code

LockingIRQ1

```
aspect LockingIRQ {

  pointcut sync_methods() = "% util::Queue::%queue(...)";
  pointcut kernel_code()  = "% kernel::%(...)";

  advice call(sync_methods()) && !within(kernel_code()) : around() {
    os::disable_int();
    try {
      tjp->proceed();
    }
    catch(...) {
      os::enable_int();
      throw;
    }
    os::enable_int();
  }
};
```

LockingIRQ1.ah

LockingIRQ1 – Elements

```
aspect LockingIRQ {

  pointcut sync_methods() = "% util::Queue::%queue(...)";
  pointcut kernel_code()  = "% kernel::%(...)";

  advice call(sync_methods()) && !within(kernel_code()) : around() {
    os::disable_int();
    try {
      tjp->proceed();
    }
    catch(...) {
      os::enable_int();
      throw;
    }
    os::enable_int();
  }
};
```

We define two pointcuts. One for the methods to be synchronized and one for all kernel functions

LockingIRQ1.ah

LockingIRQ1 – Elements

```
aspect LockingIRQ {

  pointcut sync_methods() = "% util::Queue::%queue(...)";
  pointcut kernel_code()  = "% kernel::%(...)";

  advice call(sync_methods()) && !within(kernel_code()) : around() {
    os::disable_int();
    try {
      tjp->proceed();
    }
    catch(...) {
      os::enable_int();
      throw;
    }
    os::enable_int();
  }
};
```

This pointcut expression matches any call to a *sync_method* that is **not** done from *kernel_code*

LockingIRQ1.ah

LockingIRQ1 – Result

```
util::Queue queue;
void do_something() {
  printf("do_something()\n");
  queue.enqueue( new util::Item );
}
namespace kernel {
  void irq_handler() {
    printf("kernel::irq_handler()\n");
    queue.enqueue(new util::Item);
    do_something();
  }
}
int main() {
  printf("main()\n");
  queue.enqueue(new util::Item);
  kernel::irq_handler();  // irq
  printf("back in main()\n");
  queue.dequeue();
}
```

main.cc

```
main()
os::disable_int()
  > Queue::enqueue(00320FD0)
  < Queue::enqueue()
os::enable_int()
kernel::irq_handler()
  > Queue::enqueue(00321030)
  < Queue::enqueue()
do_something()
os::disable_int()
  > Queue::enqueue(00321060)
  < Queue::enqueue()
os::enable_int()
back in main()
os::disable_int()
  > Queue::dequeue()
  < Queue::dequeue() returning 00320FD0
os::enable_int()
```

<Output>

LockingIRQ1 – Problem

```
util::Queue queue;
void do_something() {
  printf("do_something()\n");
  queue.enqueue( new util::Item );
}
namespace kernel {
  void irq_handler() {
    printf("kernel::irq_handler()\n");
    queue.enqueue(new util::Item);
    do_something();
  }
}
int main() {
  printf("main()\n");
  queue.enqueue(new util::Item);
  kernel::irq_handler();  // irq
  printf("back in main()\n");
  queue.dequeue();
}
```

main.cc

The pointcut within(*kernel_code*) does not match any **indirect** calls to *sync_methods*

```
ma
os
  > Queue::enqueue(00320FD0)
  < Queue::enqueue()
os::enable_int()
kernel::irq_handler()
  > Queue::enqueue(00321030)
  < Queue::enqueue()
do_something()
os::disable_int()
  > Queue::enqueue(00321060)
  < Queue::enqueue()
os::enable_int()
back in main()
os::disable_int()
  > Queue::dequeue()
  < Queue::dequeue() returning 00320FD0
os::enable_int()
```

<Output>

LockingIRQ2

```
aspect LockingIRQ {

  pointcut sync_methods() = "% util::Queue::%queue(...)";
  pointcut kernel_code()  = "% kernel::%(...)";

  advice execution(sync_methods())
  && !cflow(execution(kernel_code())) : around() {
    os::disable_int();
    try {
      tjp->proceed();
    }
    catch(...) {
      os::enable_int();
      throw;
    }
    os::enable_int();
  }
};
```

Solution
Using the **cflow** pointcut function

LockingIRQ2.ah

LockingIRQ2 – Elements

```
aspect LockingIRQ {

  pointcut sync_methods() = "% util::Queue::%queue(...)";
  pointcut kernel_code()  = "% kernel::%(...)";

  advice execution(sync_methods())
  && !cflow(execution(kernel_code())) : around() {
    os::disable_int();
    try {
      tjp->proceed();
    }
    catch(...) {
      os::enable_int();
      throw;
    }
    os::enable_int();
  }
};
```

LockingIRQ2.ah

This pointcut expression matches the execution of *sync_methods* if no *kernel_code* is on the call stack.
cflow checks the call stack (control flow) at runtime.

LockingIRQ2 – Result

```
util::Queue queue;
void do_something() {
  printf("do_something()\n");
  queue.enqueue( new util::Item );
}
namespace kernel {
  void irq_handler() {
    printf("kernel::irq_handler()\n");
    queue.enqueue(new util::Item);
    do_something();
  }
}
int main() {
  printf("main()\n");
  queue.enqueue(new util::Item);
  kernel::irq_handler();  // irq
  printf("back in main()\n");
  queue.dequeue();
}
```

main.cc

```
main()
os::disable_int()
  > Queue::enqueue(00320FD0)
  < Queue::enqueue()
os::enable_int()
kernel::irq_handler()
  > Queue::enqueue(00321030)
  < Queue::enqueue()
do_something()
  > Queue::enqueue(00321060)
  < Queue::enqueue()
back in main()
os::disable_int()
  > Queue::dequeue()
  < Queue::dequeue() returning 00320FD0
os::enable_int()
```

<Output>

Überblick

- **Querschneidende Belange in eCos**
  - Das Problem
- Aspektorientierte Programmierung
  - Der Lösungsansatz
- AspectC++
  - Grundlagen
  - Ergebnisse der eCos-Lösung
  - Tutorial
- Zusammenfassung

AspectC++: Advanced Concepts

- Join Point API
  - provides a uniform interface to the aspect invocation context, both at runtime and compile-time
- Abstract Aspects and Aspect Inheritance
  - comparable to class inheritance, aspect inheritance allows to reuse parts of an aspect and overwrite other parts
- Generic Advice
  - exploits static type information in advice code
- Aspect Ordering
  - allows to specify the invocation order of multiple aspects
- Aspect Instantiation
  - allows to implement user-defined aspect instantiation models

The Joinpoint API

- Inside an advice body, the current joinpoint 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 joinpoint 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 joinpoint
// (function/class name, parameter types...)
static const char* signature()

// executes the original joinpoint code
// in an around advice
void proceed()

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

Abstract 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 joinpoint API to expose context
  - Concrete aspect inherits the advice code and overrides pointcuts

Abstract Aspects and Inheritance

The abstract locking aspect declares two **pure virtual pointcuts** and uses the **joinpoint API** for an context-independent advice implementation.

```
#include "mutex.h"
aspect LockingA {
  pointcut virtual sync_classes() = 0;
  pointcut virtual sync_methods() = 0;

  advice sync_classes() : slice class {
    os::Mutex lock;
  };
  advice execution(sync_methods()) : around() {
    tjp->that()->lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      tjp->that()->lock.leave();
      throw;
    }
    tjp->that()->lock.leave();
  }
};
```

LockingA.ah

```
#include "LockingA.ah"

aspect LockingQueue : public LockingA {
  pointcut sync_classes() =
    "util::Queue";
  pointcut sync_methods() =
    "% util::Queue::%queue(...)";
};
```

LockingQueue.ah

Abstract Aspects and Inheritance

```
#include "mutex.h"
aspect LockingA {
  pointcut virtual sync_classes() = 0;
  pointcut virtual sync_methods() = 0;

  advice sync_classes() : slice class {
    os::Mutex lock;
  };
  advice execution(sync_methods()) : around() {
    tjp->that()->lock.enter();
    try {
      tjp->proceed();
    }
    catch(...) {
      tjp->that()->lock.leave();
      throw;
    }
    tjp->that()->lock.leave();
  }
};
```

LockingA.ah

The concrete locking aspect **derives** from the abstract aspect and **overrides** the pointcuts.

```
#include "LockingA.ah"

aspect LockingQueue : public LockingA {
  pointcut sync_classes() =
    "util::Queue";
  pointcut sync_methods() =
    "% util::Queue::%queue(...)";
};
```

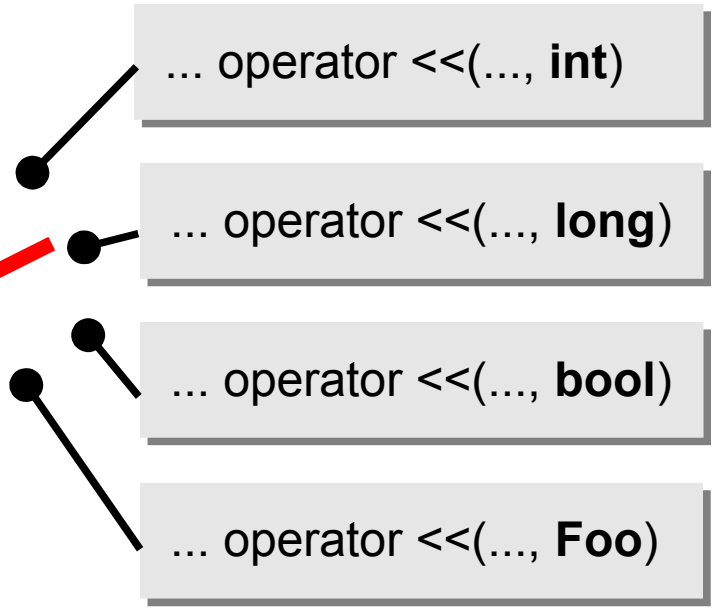
LockingQueue.ah

Generic Advice

Uses static JP-specific type information in advice code

- in combination with C++ overloading
- to instantiate C++ templates and template meta-programs

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



Generic Advice

Uses static JP-specific type information in advice code

- in combination with C++ overloading

Resolves to the **statically typed** return value

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

nplate meta-programs

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

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

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

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

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

Aspect Ordering

- Aspects should be independent of other aspects
  - However, sometimes inter-aspect dependencies are unavoidable
  - Example: Locking should be activated before any other aspects
- Order advice
  - The aspect order can be defined by ***order advice***
    advice *pointcut-expr* : order(*high, ..., low*)
  - Different aspect orders can be defined for different pointcuts
- Example

```
advice "% util::Queue::%queue(...)"
     : order( "LockingIRQ", "%" && !"LockingIRQ" );
```

Aspect Instantiation

- Aspects are singletons by default
  - **aspectof()** returns pointer to the one-and-only aspect instance
- By overriding aspectof() this can be changed
  - e.g. one instance per client or one instance per thread

```
aspect MyAspect {
  // ....
  static MyAspect* aspectof() {
    static __declspec(thread) MyAspect* theAspect;
        if( theAspect == 0 )
          theAspect = new MyAspect;
    return theAspect;
  }
};
```

MyAspect.ah

Example of an user-defined aspectof() implementation for per-thread aspect instantiation by using thread-local storage.

(Visual C++)

Überblick

- **Querschneidende Belange in eCos**
  - Das Problem
- Aspektorientierte Programmierung
  - Der Lösungsansatz
- AspectC++
  - Grundlagen
  - Ergebnisse der eCos-Lösung
  - Tutorial

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 ...
  - konfigurierbare Software von Aspekten profitiert
  - 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!