A Quantitative Analysis of Aspects in the eCos Kernel

Daniel Lohmann
Fabian Scheler
Reinhard Tartler
Olaf Spinczyk
Wolfgang Schröder-Preikschat

Department of Computer Science IV
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

http://www4.cs.fau.de/
Motivation: Cross-cutting Concerns

Cross-cutting Concern: A concern, whose implementation is scattered over the implementation of other concerns

- Enforcement of global policies and strategies
  - synchronization
  - instrumentation
  - tracing/logging
  - protection
  - ...

- Enforcement of fine-grained configuration options
  - optional features
  - alternative implementations
  - ...
eCos: An OS Product Line

- Open source OS product line for embedded applications
  - 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: Enforcement of Some Global Policies

- synchronization
- instrumentation
- tracing
eCos: Distribution of a Configuration Option

Variants of the optional mutex priority inversion protocol

- simple
- ceiling
- inheritance
- dynamic
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)
#endif
cygsem_kernel_synch_mutex_priority_inversion_protocol_default_inherit
    protocol = INHERIT;
#endif
#endif // DYNAMIC and DEFAULT defined
#endif // DYNAMIC and DEFAULT defined
#endif // not (DYNAMIC and DEFAULT defined)
#endif
cygsem_kernel_synch_mutex_priority_inversion_protocol_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
#else
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#endif
}
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

```cpp
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

```cpp
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();
```

Well, ...

- comprehensability ?
- extensability ?
- reuseability ?
- maintainability?
Aspect-Oriented Programming

AOP provides language means to encapsulate cross-cutting and scattered concerns

without AOP

with AOP

well modularized concern

badly modularized

aspect
Aspects: The Basic Idea

Separation of *what* from *where*

- **join-points** *(where)*
  - positions in the static structure or dynamic control flow (event)
  - given declaratively by pointcut expressions

- **advice** *(what)*
  - additional elements (members, ...) to introduce at certain join-points of the static structure (classes, structs)
  - additional behaviour (code to execute) to superimpose:
    - before, after
    - around (instead of)
  - certain join-points of the dynamic control flow
Example: Priority Ceiling Protocol

```cpp
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);
        }
    }
    ...
};
```

*what*  *where*
Example: 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);
      }
    }
    ...
};

Introduce a data member ceiling into all classes named Cyg_Mutex

what   where
Example: Priority Ceiling Protocol

```c++
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);
            }
        }
    ...
};
```

Execute *initialization code after* the *construction* of a *Cyg_Mutex* instance.
Example: Priority Ceiling Protocol

```cpp
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);
        }
    }
    ...

};
```

After a call to any overload of `Cyg.Mutex::lock_inner` that occurs...

what       where
Example: Priority Ceiling Protocol

```cpp
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);
                    }
                }
        )
    }

    ...

};
```

**After** a **call** to any overload of *Cyg_Mutex::lock_inner* that occurs...

...while being **within** *Cyg_Mutex::lock* and takes...
Example: 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);
            }
        }
    ...
};
Example: Priority Ceiling Protocol

```cpp
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);
        }
    }
}
```

After a call to any overload of `Cyg_Mutex::lock_inner` that occurs...

...while being within `Cyg_Mutex::lock` and takes...

...an argument of type `Cyg_Thread*`...

...check the result value and clear ceiling, if required

what where
Summary: AOP

- AOP provides additional means for separation of concerns
  - declarative match mechanism
  - join-points denote events/positions with specific **semantics**
  - it's not just patching!

- Does it really help?
  - several publications say so
    - separating prefetching code in FreeBSD (Coady, 2001)
    - integrating the Bossa scheduler into Linux (Ǻberg, 2003)
    - ... “...improved evolvability / comprehensability / configurability..
  - counter-examples have been published as well
    - separating network code optimizations (Siadat, 2006)
      “...comprehensability was **not improved.**”

- More **experience** and **larger studies** required!
Summary: AOP

- AOP provides additional means for separation of concerns
  - declarative match mechanism
  - join-points denote events/positions with specific semantics...
  - it's not just patching!

Does it really help?

- several publications say so
  - separating prefetching code in FreeBSD (Coady, 2001)
  - integrating the Bossa scheduler into Linux (Ǻberg, 2003)
  - improved evolvability / comprehensability / configurability...
- counter-examples have been published as well
  - separating network code optimizations (Siadat, 2006)
    “...comprehensability was not improved.”

- More experience and larger studies required!

And what does it cost?
Quantifying the Overhead of AOP

**Target:** AspectC++ language and weaver
- open source aspect weaver for C++
- transforms AspectC++ code into C/C++ code
- platform/compiler-independent

**Conducted:**

1. A series of μ-Benchmarks for AspectC++ constructs
2. Larger comparative study by refactoring eCos

http://www.aspectc.org
1. Costs of AspectC++ language features

**a) incrementer**

<table>
<thead>
<tr>
<th>advice</th>
<th>cycles</th>
<th>stack</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>tangled</td>
<td>4 abs</td>
<td>0 abs</td>
<td>4128</td>
</tr>
<tr>
<td>before</td>
<td>6 Δ 6</td>
<td>0 Δ 0</td>
<td>4128 0</td>
</tr>
<tr>
<td>after</td>
<td>6 Δ 6</td>
<td>0 Δ 0</td>
<td>4128 0</td>
</tr>
<tr>
<td>around</td>
<td>6 Δ 6</td>
<td>0 Δ 0</td>
<td>4128 0</td>
</tr>
</tbody>
</table>

**b) multiaspect**

<table>
<thead>
<tr>
<th># aspect</th>
<th>cycles</th>
<th>stack</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6 Δ 6</td>
<td>0 Δ 0</td>
<td>4080</td>
</tr>
<tr>
<td>2</td>
<td>5 Δ -1</td>
<td>0 Δ 0</td>
<td>4080 0</td>
</tr>
<tr>
<td>3</td>
<td>6 Δ 1</td>
<td>0 Δ 0</td>
<td>4096 16</td>
</tr>
</tbody>
</table>

**c) parameters, jp-api**

<table>
<thead>
<tr>
<th>tjp-&gt;</th>
<th>cycles</th>
<th>stack</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain, n=0</td>
<td>5 Δ 5</td>
<td>16 Δ 16</td>
<td>3968</td>
</tr>
<tr>
<td>that()</td>
<td>7 Δ 7</td>
<td>20 Δ 20</td>
<td>3968 0</td>
</tr>
<tr>
<td>target()</td>
<td>8 Δ 8</td>
<td>20 Δ 20</td>
<td>3968 0</td>
</tr>
<tr>
<td>result()</td>
<td>11 Δ 11</td>
<td>16 Δ 16</td>
<td>3968 0</td>
</tr>
<tr>
<td>plain, n=1</td>
<td>13 Δ 13</td>
<td>24 Δ 24</td>
<td>3968</td>
</tr>
<tr>
<td>arg&lt;0&gt;()</td>
<td>13 Δ 13</td>
<td>24 Δ 24</td>
<td>3968 0</td>
</tr>
<tr>
<td>plain, n=2</td>
<td>13 Δ 13</td>
<td>32 Δ 32</td>
<td>3984</td>
</tr>
<tr>
<td>arg&lt;1&gt;()</td>
<td>13 Δ 13</td>
<td>32 Δ 32</td>
<td>3984 0</td>
</tr>
</tbody>
</table>

**d) dynamic pointcuts**

<table>
<thead>
<tr>
<th>pointcut</th>
<th>Δ cycle</th>
<th>Δ stack</th>
<th>Δ code</th>
<th>Δ data</th>
</tr>
</thead>
<tbody>
<tr>
<td>cflow()</td>
<td>6 Δ 6</td>
<td>16 Δ 16</td>
<td>8 Δ 8</td>
<td>4</td>
</tr>
<tr>
<td>enter/leave</td>
<td>12 Δ 12</td>
<td>52 Δ 52</td>
<td>56 Δ 56</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>that()</td>
<td>10 Δ 10</td>
<td>24 Δ 24</td>
<td>128 Δ 128</td>
<td>50</td>
</tr>
<tr>
<td>target()</td>
<td>10 Δ 10</td>
<td>24 Δ 24</td>
<td>144 Δ 144</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Costs of AspectC++ language features

### a) incrementer

<table>
<thead>
<tr>
<th>Advice</th>
<th>Cycles</th>
<th>Stack</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>tangled</td>
<td>4</td>
<td>0</td>
<td>4128</td>
</tr>
<tr>
<td>before</td>
<td>6 2</td>
<td>0 0</td>
<td>4128 0</td>
</tr>
<tr>
<td>after</td>
<td>6 2</td>
<td>0 0</td>
<td>4128 0</td>
</tr>
<tr>
<td>around</td>
<td>6 2</td>
<td>0 0</td>
<td>4128 0</td>
</tr>
</tbody>
</table>

### b) multiaspect

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Cycles</th>
<th>Stack</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4080</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>4080</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>4096</td>
</tr>
</tbody>
</table>

### c) parameters, jp-api

<table>
<thead>
<tr>
<th>Function</th>
<th>Cycles</th>
<th>Stack</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain, n=0</td>
<td>5</td>
<td>16</td>
<td>3968</td>
</tr>
<tr>
<td>that()</td>
<td>7</td>
<td>20</td>
<td>3968</td>
</tr>
<tr>
<td>target()</td>
<td>8</td>
<td>20</td>
<td>3968</td>
</tr>
<tr>
<td>result()</td>
<td>11</td>
<td>16</td>
<td>3968</td>
</tr>
</tbody>
</table>

### d) dynamic pointcuts

<table>
<thead>
<tr>
<th>Pointcut</th>
<th>Δ Cycles</th>
<th>Δ Stack</th>
<th>Δ Code</th>
<th>Δ Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>cflow()</td>
<td>6</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>enter/leave test</td>
<td>12</td>
<td>52</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>that()</td>
<td>10</td>
<td>24</td>
<td>128</td>
<td>50</td>
</tr>
<tr>
<td>target()</td>
<td>10</td>
<td>24</td>
<td>144</td>
<td>50</td>
</tr>
</tbody>
</table>
1. Costs of AspectC++ language features

### Results

- **simple** before/after/around advice for call/execution join-points does **not induce costs**
- accessing **join-point context** (arg, result, ...) may **induce** some stack and CPU **costs** – but very low
- **dynamic pointcut functions** (cflow, that, target) **induce noticeable overhead**
2. Comparative Study with eCos

■ Refactored: original kernel \(\rightarrow\) aspectized kernel
  ▪ 3 cross-cutting policies
    - interrupt synchronization \(187\) invocations \(\rightarrow\) \(160\) code join-points
    - kernel instrumentation \(162\) invocations \(\rightarrow\) \(139\) code join-points
    - tracing \(336\) invocations \(\rightarrow\) \(632\) code join-points
  ▪ 12 configuration options
    - mutex features
    - thread features

■ Compared: original kernel \(\leftrightarrow\) aspectized kernel
  ▪ scattering
  ▪ performance
  ▪ memory footprint
Evaluation: Performance and Memory Costs

Evaluation based on 3 small multi-threaded test programs
- specifically use affected kernel functions
- overhead, if any, should become evident with this setup
- each measured with 13 different eCos configurations

<table>
<thead>
<tr>
<th>a) thread cyg_thread…</th>
<th>b) mutex cyg_mutex…</th>
<th>c) semaphore cyg_semaphore…</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 …_create()</td>
<td>215</td>
<td>…_init()</td>
</tr>
<tr>
<td>2 …_resume()</td>
<td>327</td>
<td>…_lock()</td>
</tr>
<tr>
<td>3 …_resume()</td>
<td>127</td>
<td>…_unlock()</td>
</tr>
<tr>
<td>4 …_yield()</td>
<td>274</td>
<td>…_try_lock()</td>
</tr>
<tr>
<td>5 …_exit()</td>
<td>354</td>
<td>…_try_lock()</td>
</tr>
<tr>
<td>6 …_yield()</td>
<td>77</td>
<td>…_lock()</td>
</tr>
<tr>
<td>7 …_resume()</td>
<td>91</td>
<td>…_unlock()</td>
</tr>
<tr>
<td>8 …_kill()</td>
<td>102</td>
<td>…_destroy()</td>
</tr>
<tr>
<td>9 …_suspend()</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>10 …_suspend()</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>11 …_suspend()</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>12 …_resume()</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>13 …_resume()</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>14 …_delete()</td>
<td>38</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation: Performance Costs

3 Testcases
a 13 configurations
Average AOP runtime cost factor: 0.99
(with caching disabled: 1.00)

Evaluation: Performance Costs
### Evaluation: Memory Costs

<table>
<thead>
<tr>
<th>[bytes]</th>
<th>ROM</th>
<th></th>
<th>RAM</th>
<th></th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>△</td>
<td>%</td>
<td>△</td>
<td>%</td>
<td>△</td>
</tr>
<tr>
<td>01 mutex_base</td>
<td>101</td>
<td>0,6</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>02 mutex_except_hand</td>
<td>55</td>
<td>0,3</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>03 mutex_except_hand_dec</td>
<td>-35</td>
<td>-0,2</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>04 mutex_except_hand_global</td>
<td>92</td>
<td>0,5</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>05 mutex_except_hand_global_dec</td>
<td>85</td>
<td>0,5</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>06 mutex_instrumentation</td>
<td>543</td>
<td>3,0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>07 mutex_thread_data</td>
<td>74</td>
<td>0,4</td>
<td>0</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>08 mutex_thread_destruct</td>
<td>74</td>
<td>0,4</td>
<td>0</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>09 mutex_thread_destruct_perthread</td>
<td>224</td>
<td>1,3</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>10 mutex_thread_linked_list</td>
<td>88</td>
<td>0,5</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>11 mutex_thread_name</td>
<td>146</td>
<td>0,9</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>12 mutex_thread_stack_limit</td>
<td>311</td>
<td>1,8</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>13 mutex_thread_stackusage</td>
<td>341</td>
<td>2,0</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>14 sem_base</td>
<td>91</td>
<td>0,5</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>15 ...</td>
<td>51</td>
<td>0,3</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>40 Average [%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0,9</td>
</tr>
</tbody>
</table>
## Evaluation: Memory Costs

<table>
<thead>
<tr>
<th>cost factor aop/plain [bytes]</th>
<th>ROM</th>
<th>RAM</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Δ</td>
<td>%</td>
<td>Δ</td>
</tr>
<tr>
<td>mutex_base</td>
<td>101</td>
<td>0,6</td>
<td>0</td>
</tr>
<tr>
<td>mutex_except_hand</td>
<td>55</td>
<td>0,3</td>
<td>0</td>
</tr>
<tr>
<td>mutex_except_hand_dec</td>
<td>-35</td>
<td>-0,2</td>
<td>0</td>
</tr>
<tr>
<td>mutex_except_hand_global</td>
<td>92</td>
<td>0,5</td>
<td>0</td>
</tr>
<tr>
<td>mutex_except_hand_global_dec</td>
<td>85</td>
<td>0,5</td>
<td>0</td>
</tr>
<tr>
<td>mutex_instrumentation</td>
<td>543</td>
<td>3,0</td>
<td>0</td>
</tr>
<tr>
<td>mutex_thread_data</td>
<td>74</td>
<td>0,4</td>
<td>0</td>
</tr>
<tr>
<td>Average [AOP ROM cost factor]</td>
<td>1.009</td>
<td>1.013</td>
<td></td>
</tr>
<tr>
<td>Average [AOP stack cost factor]</td>
<td></td>
<td></td>
<td>1.3</td>
</tr>
</tbody>
</table>
Lessons Learned

- “Aspectizing” the eCos kernel was relatively easy
  - concerns **conceptually** already separated
  - fine-grained implementation structure offered enough join-points

- Once aspectized, policy-changes could be applied easily
  - scheduler activation points
  - synchronization points
  - design becomes more explicit in the code

- Compile-time resolvable join-points were success factor
  - overhead goes up, if dynamic pointcut functions have to be used
  - see *kernel-stack* extension example in the paper
Conclusions

- Separation of concerns is an issue in system software
  - enforcement of cross-cutting policies
  - fine-grained optional features in product lines

- Aspect-oriented programming is promising
  - provides additional means for separation of concerns
  - declarative concepts to separate *what* from *where*

- It can be applied *cost-neutral*
  - developers can get better separation of concerns for free!
  - well suited for system software
Future Work: The CiAO Project

- **Approach:** Use AOP concepts in OS development from the very beginning
  - if we use aspects from scratch, how far can we get?
  - what are the fundamental design patterns?

- **Goal:** Configurability of **architectural** properties
  - protection (e.g. single address space vs. memory protection)
  - interaction (e.g. procedural vs. message-based)
  - synchronization (e.g. fine-grained vs. coarse-grained)

http://www.aspectc.org
Future Work: The CiAO Project

- **Approach:** Use AOP concepts in OS development from the very beginning
  - if we use aspects from scratch, how far can we get?
  - what are the fundamental design patterns?

- **Goal:** Configurability of **architectural** properties
  - protection (e.g. single address space vs. memory protection)
  - interaction (e.g. procedural vs. message-based)
  - synchronization (e.g. fine-grained vs. coarse-grained)

Thank you very much for your attention

http://www.aspectc.org