Konfigurierbare Systemsoftware (KSS)

VL 4 – Aspect-Aware Development: The CiAO Approach

Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 15 – 2015-05-04

http://www4.informatik.uni-erlangen.de/Lehre/SS15/V_KSS
About this Lecture

Problem Space

Domain Expert

Features and Dependencies

Solution Space

Architect / Developer

Features and Dependencies

Configuration

Variant

System User

Actual implementation

intended properties

intentional side

extensional side
Implementation Techniques: Classification

Decompositional Approaches

- Text-based filtering (untyped)
- Preprocessors

Compositional Approaches

- Language-based composition mechanisms (typed)
- OOP, AOP, Templates

Generative Approaches

- Metamodel-based generation of components (typed)
- MDD, C++ TMP, generators
Implementation Techniques: Goals

General
1. Separation of concerns (SoC)
2. Resource thriftiness

Operational
3. Granularity
Components should be fine-grained. Each artifact should either be mandatory or dedicated to a single feature only.
4. Economy
The use of memory/run-time expensive language features should be avoided as far as possible. Decide and bind as much as possible at generation time.
5. Pluggability
Changing the set of optional features should not require modifications in any other part of the implementation. Feature implements should be able to “integrate themselves”.
6. Extensibility
The same should hold for new optional features, which may be available in a future version of the product line.

How to achieve these with AOP?
Agenda

4.1 AOP Mechanisms Under the Hood
4.2 Study: i4Weathermon AOP
4.3 CiAO
4.4 CiAO Results
4.5 Summary
4.6 References
4.1 AOP Mechanisms Under the Hood
   Diagram Notation
   Obliviousness & Quantification
   AOP Mechanisms: Summary
4.2 Study: i4Weathermon AOP
4.3 CiAO
4.4 CiAO Results
4.5 Summary
4.6 References
Notation

BaseClass

method()

explicitJP()

SomeClass

#protected()
static()
unadvisable()

AnAspect

exec("explicitJP")
intro("SomeClass")

ASlice

state_variable_
anotherMethod()

"explicitJP"

method is explicit join point

introduction [of Slice]

elements belong to same concern

method is unadvisable

method is unadvisable
Szenario:
Optional feature component $F_1$ shall be integrated into SPL component $PL$.

With OOP:
- $PL$ has to call $F_1 \leadsto PL$ has to know $F_1$
- control flows can only be established in the direction of knowledge.

With AOP:
- $F_1$ can give advice to $PL \leadsto F_1$ has to may know $PL$
- control flow is established opposite to the direction of knowledge
- binding is inherently loose $\leadsto$ silently missed, if $PL$ does not exist
**Szenario:**
(Nonfunctional) feature component $F_1$ shall be integrated into (optional) SPL components $PLC_1$...$n$

With AOP:
- binding is inherently loose $\leadsto$ may quantify over $n$ join points
- possible by declarative pointcut concept (here: wildcard in match expression)
Advice inverses the direction in which control-flow relationships are established: \( C \text{ calls } A \implies A \text{ advises } C \)

- Aspects integrate themselves into the surrounding program
  \( \leadsto \) “I make you call me”
- Surrounding program can be kept oblivious of the aspects
  \( \leadsto \) advice-based binding as a means to integrate (optional) features

Pointcuts provide for an implicit quantification of this integration

- Applies to \( 0 \ldots M \ldots n \) join points, depending on the pointcut expression
  \( \leadsto \) Aspects can be kept oblivious of the surrounding program
- Thereby, advice-based binding is inherently loose
  \( \leadsto \) advice-based binding as a means to integrate interacting features
Agenda

4.1 AOP Mechanisms Under the Hood
4.2 Study: i4Weathermon AOP
  Flashback: i4Weathermon
  i4WeatherMon with AOP
  i4WeatherMon with AOP: Results
4.3 CiAO
4.4 CiAO Results
4.5 Summary
4.6 References
How to achieve *Granularity, Economy, Pluggability, Extensibility*?

- Configuration-dependent sensor and actuator sets
  - initialization, integration, interaction of optional feature code
- Generic and nongeneric actuators
  - interacting optional feature code
I4WeatherMon: AOP Solution Space

Pressure
- measure()
- name()
- unit()
- str_val()

PressureHandling
- exec("Weather::measure")
- exec("Sink::process")
- intro("Weather")
- intro("Sink")

Display
- exec("Sink::process")
- exec("Sink::process_data(%)")

Sink
- process()
- process_data(Pressure)
- process_data(Wind)

Wind
- measure()
- name()
- unit()
- str_val()

WindHandling
- exec("Weather::measure")
- exec("Sink::process")
- intro("Weather")
- intro("Sink")

PCConnection
- #send()

SNGConnection
- exec("Sink::process")
- exec("Sink::process_data(Wind")
- exec("Sink::process_data(Pressure")
- exec("Sink::process_data(Temperature")

© dl  KSS (VL 4 | SS 15)  4 The CiAO Approach | 4.2 Study: i4Weathermon AOP
Basic structure

**Weather** and **Sink** are (almost) empty classes.

- Provide a **lexical scope** for sensor / actuator introductions
- Provide **explicit join points** (empty methods `measure() / process()`) that are invoked by the main loop, when measuring / processing should take place
- All further functionality is provided by the aspects!
Sensor integration

A Sensor is implemented as a class with an accompanying Handling aspect

- Slices the sensor singleton instance into Weather
- Gives advice to Weather::measure() to invoke Sensor::measure()
- Slices an explicit join point process_data(Sensor) into Sink
- Gives advice to Sink::process() to invoke process_data(Sensor)
```cpp
class Weather {
public:
    void measure() {} // empty implementation
};

class Sink {
public:
    void process() {} // empty implementation
};

aspect PressureHandling {
   // Weather integration
   advice "Weather" : slice struct{
       Pressure _pressure; // introduce sensor instance (singleton)
   };
   advice execution( "void Weather::measure()" ) : before() {
       theWeather._pressure.measure(); // invoke sensor’s measure()
   }
   // Sink integration
   advice "Sink" : slice struct {
       // introduce sensor-specific explicit join point for actuator aspects
       void process_data( const Pressure & ) {}
   };
   advice execution( "void Sink::process()" ) : after() {
       theSink.process_data( theWeather._pressure ); // trigger it
   }
};
```
**Generic actuator integration**

A generic actuator (processes all sensors) is implemented by an aspect:

- Gives advice to `Sink::process()` to execute processing pre-/post actions.
- Gives **generic advice** to all overloads of `Sink::process_data()` to invoke each sensor (typed) in order to process its data via the generic `str_val()`.

~ Generic actuator does not know the available / possible sensor types.
aspect Display {
    ...
    // display each element of the weather data
    advice execution("void Sink::process_data(%)") : before () {
        typedef JoinPoint::template Arg<0>::ReferredType Data;
        char val[5];
        tjp->arg<0>()->str_val( val );
        print( Data::name(), val, Data::unit() );
    }
};
Nongeneric actuator integration

A nongeneric actuator (processes some sensors) is implemented by an aspect

- Gives advice to Sink::process() to execute processing pre-/post actions
- Gives advice to selected overloads of Sink::process_data() to invoke them in order to process each sensors data via a sensor-specific interface

Nongeneric actuator has to know specific sensor types
aspect SNGConnection : protected PCConnection {
    UInt8 _p, _w, _t1, _t2; // weather record
...
    // let this aspect take a higher precedence than <Sensor>Handling
    advice process () : order ("SNGConnection", "%Handling");
    advice execution("void Sink::process(const Weather&)"
        : before () { ... /* init record */ }
    advice execution("void Sink::process(const Weather&)"
        : after () { ... /* transmit record */ }
}

// collect wind, pressure, temperature data by giving specific advice
advice execution("void Sink::process_data(...)"
    : before (const Wind &wind) {
        _w = wind._w;
    }
    advice execution("void Sink::process_data(...)"
        : before (const Pressure &pressure) {
        _p = pressure._p - 850;
    }
    advice execution("void Sink::process_data(...)"
        : before (const Temperature &temp) {
        _t1 = (UInt8)temp._t1;
        _t2 = temp._t2;
    }
};
AOP is as efficient as CPP-based configurability!
I4WeatherMon (AOP): Evaluation

General
1. Separation of concerns (SoC) ✔
2. Resource thriftiness ✔

Operational
3. Granularity ✔
   - Every component implements functionality of a single feature only.
4. Economy ✔
   - All control-flow bindings are established at compile time.
5. Pluggability ✔
   - Sensors and actuators integrate themselves by aspects.
6. Extensibility ✔
   - “Plug & Play” of sensor and actuator implementations.
4.1 AOP Mechanisms Under the Hood
4.2 Study: i4Weathermon AOP
4.3 CiAO
   Motivation and Goals
   Design Approach
   Examples: Aspects in Action
   Explicit Join Points
   Further Examples
4.4 CiAO Results
4.5 Summary
4.6 References
CiAO: Motivation and Goals

"Throughout the entire operating-system design cycle, we must be careful to separate policy decisions from implementation details (mechanisms). This separation allows maximum flexibility if policy decisions are to be changed later."

Silberschatz, Gagne, and Galvin 2005: Operating System Concepts [8, p. 72]

- Primary goal: architectural configurability
- Configurability of even fundamental policies
  → synchronization, protection, interaction
- Secondary goal: < the standards >
  → efficiency, configurability in general, portability
- Approach: aspect-aware operating system design
  → strict decoupling of policies and mechanisms in the implementation
  → using aspects as the primary composition technique
CiAO is Aspect-Oriented

A product line of aspect-oriented operating systems
- Implements OSEK VDX / AUTOSAR OS [1, 7]
- Fine-grained configurability of all system policies and abstractions
- Developed from scratch with AOP
CiAO: General Structure

Layered Architecture
- Interface layer (as/ciao)
- System layer (os)
- Hardware layer (hw)

Layers $\mapsto$ C++ namespaces
- Potential join points for cross-layer transitions
- Further refined by sublayers (os::krn, hw::irq)
- Layers as a means of aspect-aware development
  $\leadsto$ “visible transitions”

// yields all hardware invocations from the system layer
pointcut OStoHW() = call("% hw::......::%(...)")
  && within("% os::......::%(...)");
Methodology: Principles of Aspect-Aware Development

<table>
<thead>
<tr>
<th>Design Principles</th>
<th>Development Idioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>loose coupling</strong></td>
<td><em>by</em> advice-based binding</td>
</tr>
<tr>
<td>2. <strong>visible transitions</strong></td>
<td><em>by</em> explicit join points</td>
</tr>
<tr>
<td>3. <strong>minimal extensions</strong></td>
<td><em>by</em> extension slices</td>
</tr>
</tbody>
</table>
The principle of loose coupling. Make sure that aspects can hook into all facets of the static and dynamic integration of system components. The *binding* of components, but also their *instantiation* (e.g., placement in a certain memory region) and the time and order of their *initialization* should all be established (or at least be influenceable) by aspects.

The principle of visible transitions. Make sure that aspects can hook into all control flows that run through the system. All control-flow transitions into, out of, and within the system should be influenceable by aspects. For this they have to be represented on the join-point level as statically evaluable, unambiguous join points.

The principle of minimal extensions. Make sure that aspects can extend all features provided by the system on a fine granularity. System components and system abstractions should be fine-grained, sparse, and extensible by aspects.
Methodology: Roles of Aspects and Classes

What to model as a *class* and what as an *aspect*?

- **< Thing >** is a class if – and only if – it is a distinguishable, instantiable concept of CiAO:
  - A **system component**, instantiated internally on behalf of CiAO
    - The Scheduler, the Alarm Manager, the OS control facility, ...
    - Hold and manage kernel state, singletons by definition
  - A **system abstraction**, instantiated as objects on behalf of the user
    - Task, Event, Resource, ...
    - In AUTOSAR OS: instantiated at compile time
  - Both are **sparse** → provide a minimal implementation only

- Otherwise **< thing >** is an aspect!
Three idiomatic aspect roles

- **Extension aspects**: extend some system component or system abstraction by additional functionality.
- **Policy aspects**: “glue” otherwise unrelated system abstractions or components together to implement some CiAO kernel policy.
- **Upcall aspects**: bind behavior defined by higher layers to events produced in lower layers of the system.
Example: Mechanism Integration

Visible transitions by explicit join points

```c
os::krn
hw::hal
```

```c
Serial0
init()
sendBlock()
```

```c
Sched
init()
ready()
```

```c
Serial0_Init
exec("init")
```

```c
Sched_Init
exec("init")
```

```c
Serial0Ext
exec("init")
```

```c
Serial0Ext_Task
init
```

```c
Task_task
init
```

```
visible transitions by explicit join points
```

advice-based bindings
extension aspects add to mechanisms by extension slices
This might lead to additional functional dependencies

```c
os::krn
hw::hal
```

```c
4–30
```
Example: Mechanism Integration

Binding aspects implement upcalls by advice-based bindings.

binding aspects
implement upcalls by advice-based bindings
Example: Mechanism Integration

- **Serial0**:
  - `init()`
  - `sendBlock()`

- **Sched**:
  - `init()`
  - `ready()`

- **Serial0Ext**:
  - `exec("init")`

- **Serial0Ext_Task**:
  - `Task_task`

- **Serial0_Init**:
  - `exec("init")`

- **Sched_Init**:
  - `exec("init")`

- Uses:
  - `ready`
  - `uses`

- Extension aspects:
  - Add to mechanisms by extension slices

- Binding aspects:
  - Implement upcalls by advice-based bindings

- This might lead to additional functional dependencies
Example: Mechanism Integration

```
aspect Serial0Ext {
    ...
    advice execution( "void hw::hal::init()" )
        : order( "Sched_Init", "Serial0_Init" );
    ...
}
```

This might lead to additional functional dependencies.
Example: Policy Integration

Cooperative Scheduling:

- `enterKernel()`
- `leaveKernel()`

Preemptive Scheduling:

- `reschedule()`
- `setNeedReschedule()`

Visible transitions by explicit join points

- `reschedule()`
- `setNeedReschedule()`

- `exec("leaveKernel")`
- `exec("enterKernel")`
- `exec("setNeedReschedule")`

Cooperative Scheduling Flow:

1. `enterKernel()`
2. `leaveKernel()`
3. `reschedule()`
4. `setNeedReschedule()`

Preemptive Scheduling Flow:

1. `reschedule()`
2. `setNeedReschedule()`
3. `exec("leaveKernel")`
4. `exec("enterKernel")`
5. `exec("setNeedReschedule")`
Example: Policy Integration

Cooperative Scheduling:

- enterKernel()
- leaveKernel()

«policy aspect»
Sched_LeaveBinding
exec("leaveKernel")

reschedule()
setNeedReschedule()

Sched

Preemptive Scheduling:

- enterKernel()
- leaveKernel()

«policy aspect»
Kernel_ASTSync
exec("enterKernel()")
exec("leaveKernel()")

reschedule
"setNeedReschedule"

«policy aspect»
Sched_ASTBinding
exec("ast")
exec("setNeedReschedule")

os::krn
hw::hal

Policy aspects
connect mechanisms
to implement some
CiAO kernel policy
Example: Policy Integration

Cooperative Scheduling:

- `enterKernel()`
- `leaveKernel()`

Preemptive Scheduling:

- `enterKernel()`
- `leaveKernel()`

```
Sched
reschedule()
setNeedReschedule()
```

```
Kernel_ASTSync

Sched_ASTBinding
reschedule()
setNeedReschedule()
```

```
AST0

trigger()
```

```
c`
```
Methodology: Explicit Join Points

- Advice-based binding \(\mapsto\) availability of the “right” join points
  - for all semantically important transitions in the system
  - statically evaluable

- Fine-grained component structure \(\leadsto\) many implicit join points, but
  - amount and precise semantics often implementation dependent
  - aspects have to “know” \(\leadsto\) no obliviousness

- Important transitions not available for technical reasons as JPs
  - target code may be fragile (e.g., context switch) \(\leadsto\) must not be advised
  - target code may be written in assembly \(\leadsto\) transitions not visible as JPs
Methodology: Explicit Join Points (Cont’d)

- Solution: explicit join points
  - empty inline methods for the sole purpose that aspects can bind to them
  - explicitly triggered by components or other aspects
  - well defined semantics

- Upcall join points (U) represent system-internal events that are to be processed on a higher layer
  - exceptions, such as signals or interrupts
  - internal events, such as system initialization or entering of the idle state

- Transition join points (T) represent semantically important control-flow transitions inside the kernel
  - level transitions: \textit{user} $\rightarrow$ \textit{kernel}, \textit{user} $\rightarrow$ \textit{interrupt}
  - context transitions: \textit{threadA} $\rightarrow$ \textit{threadB}
Explicit join points for the support and binding of OSEK OS and AUTOSAR OS user-level hook functions, as specified in [36, p. 39] and [4, p. 46].

<table>
<thead>
<tr>
<th>Type representing function or method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>U internalErrorHook()</code></td>
<td>Triggered in case of an error, a protection violation, before (pre) and after (post) at high-level task switch, and at operating-system startup and shutdown time.</td>
</tr>
<tr>
<td><code>U internalProtectionHook(StatusType error)</code></td>
<td>Triggered when a control flow enters (respectively leaves) the kernel domain.</td>
</tr>
<tr>
<td><code>U internalPreTaskHook()</code></td>
<td>Entry point of a new thread (continuation).</td>
</tr>
<tr>
<td><code>U internalPostTaskHook()</code></td>
<td>Triggered immediately before the running continuation is deactivated or terminated; to is going to become the next running continuation.</td>
</tr>
<tr>
<td><code>U internalStartupHook()</code></td>
<td>Triggered immediately after the (new) running continuation got reactivated or started.</td>
</tr>
<tr>
<td><code>U internalShutdownHook()</code></td>
<td>Entry point of the respective AST.</td>
</tr>
<tr>
<td><code>U ThreadFunc()</code></td>
<td>Triggered during system startup after memory busses and stack have been initialized.</td>
</tr>
<tr>
<td><code>T enterKernel()</code></td>
<td>Entry point of the respective interrupt handler. (Interrupts are still disabled.)</td>
</tr>
<tr>
<td><code>T leaveKernel()</code></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

4.1 AOP Mechanisms Under the Hood
4.2 Study: i4Weathermon AOP
4.3 CiAO
4.4 CiAO Results
4.5 Summary
4.6 References
## AUTOSAR-OS Features Implemented in CiAO as Aspects (Excerpt)

<table>
<thead>
<tr>
<th>concern</th>
<th>extension policy</th>
<th>upcall</th>
<th>advice</th>
<th>join points</th>
<th>extension of</th>
<th>advice-based binding to</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISR cat. 1 support</td>
<td>1</td>
<td>2 + m</td>
<td>2 + m</td>
<td>API, OS control</td>
<td>m ISR bindings</td>
<td></td>
</tr>
<tr>
<td>ISR cat. 2 support</td>
<td>1</td>
<td>5 + n</td>
<td>5 + n</td>
<td>API, OS control, scheduler</td>
<td>n ISR bindings</td>
<td></td>
</tr>
<tr>
<td>ISR abortion support</td>
<td>1</td>
<td>2</td>
<td>1 + m + n</td>
<td>scheduler</td>
<td>m + n ISR functions</td>
<td></td>
</tr>
<tr>
<td>Resource support</td>
<td>1 1</td>
<td>3</td>
<td>5</td>
<td>scheduler, API, task</td>
<td>PCP policy implementation</td>
<td></td>
</tr>
<tr>
<td>Resource tracking</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>task, ISR</td>
<td>monitoring of Get/ReleaseResource</td>
<td></td>
</tr>
<tr>
<td>Event support</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>scheduler, API, task, alarm</td>
<td>trigger action JP</td>
<td></td>
</tr>
<tr>
<td>Alarm support</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>API</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS application support</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>scheduler, task, ISR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full preemption</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td></td>
<td>3 points of rescheduling</td>
<td></td>
</tr>
<tr>
<td>Mixed preemption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple activation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack monitoring</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>task</td>
<td>CPU-release JPs</td>
<td></td>
</tr>
<tr>
<td>Context check</td>
<td>1</td>
<td>1</td>
<td>s</td>
<td></td>
<td>s service calls</td>
<td></td>
</tr>
<tr>
<td>Disabled interrupts check</td>
<td>1</td>
<td>1</td>
<td>30</td>
<td></td>
<td>all services except interrupt services</td>
<td></td>
</tr>
<tr>
<td>Enable w/o disable check</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td>enable services</td>
<td></td>
</tr>
<tr>
<td>Missing task end check</td>
<td>1</td>
<td>1</td>
<td>t</td>
<td></td>
<td>t task functions</td>
<td></td>
</tr>
<tr>
<td>Out of range check</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td>alarm set and schedule table start services</td>
<td></td>
</tr>
<tr>
<td>Invalid object check</td>
<td>1</td>
<td>1</td>
<td>25</td>
<td></td>
<td>services with an OS object parameter</td>
<td></td>
</tr>
<tr>
<td>Error hook</td>
<td>1</td>
<td>2</td>
<td>30</td>
<td>scheduler</td>
<td>29 services</td>
<td></td>
</tr>
<tr>
<td>Protection hook</td>
<td>1 1</td>
<td>2</td>
<td>2</td>
<td>API</td>
<td>default policy implementation</td>
<td></td>
</tr>
<tr>
<td>Startup / shutdown hook</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td>explicit hooks</td>
<td></td>
</tr>
<tr>
<td>Pre-task / post-task hook</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td>explicit hooks</td>
<td></td>
</tr>
</tbody>
</table>

**Plug & Play of optional features and policies!**
<table>
<thead>
<tr>
<th>test scenario</th>
<th>CiAO</th>
<th>OSEK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>full</td>
</tr>
<tr>
<td>(a) voluntary task switch</td>
<td>160</td>
<td>178</td>
</tr>
<tr>
<td>(b) forced task switch</td>
<td>108</td>
<td>127</td>
</tr>
<tr>
<td>(c) preemptive task switch</td>
<td>192</td>
<td>219</td>
</tr>
<tr>
<td>(d) system startup</td>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>(e) resource acquisition</td>
<td>19</td>
<td>56</td>
</tr>
<tr>
<td>(f) resource release</td>
<td>14</td>
<td>52</td>
</tr>
<tr>
<td>(g) resource release with preemption</td>
<td>240</td>
<td>326</td>
</tr>
<tr>
<td>(h) category 2 ISR latency</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>(i) event blocking with task switch</td>
<td>141</td>
<td>172</td>
</tr>
<tr>
<td>(j) event setting with preemption</td>
<td>194</td>
<td>232</td>
</tr>
<tr>
<td>(k) comprehensive application</td>
<td>748</td>
<td>748</td>
</tr>
</tbody>
</table>

Execution time [clock ticks] on TC1796@50 MHz

(ac++1.0pre3 with tricore-g++3.4.3 -O3 -fno-rtti -funit-at-a-time -ffunction-sections -Xlinker --gc-sections)
# Kernel Latency Comparison with “OSEK”

<table>
<thead>
<tr>
<th>Test Scenario</th>
<th>CiAO min</th>
<th>CiAO full</th>
<th>OSEK min</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) voluntary task switch</td>
<td>160</td>
<td>178</td>
<td>218</td>
</tr>
<tr>
<td>(b) forced task switch</td>
<td>108</td>
<td>127</td>
<td>280</td>
</tr>
<tr>
<td>(c) preemptive task switch</td>
<td>192</td>
<td>219</td>
<td>274</td>
</tr>
<tr>
<td>(d) system startup</td>
<td>194</td>
<td>194</td>
<td>399</td>
</tr>
<tr>
<td>(e) resource acquisition</td>
<td>19</td>
<td>56</td>
<td>54</td>
</tr>
<tr>
<td>(f) resource release</td>
<td>14</td>
<td>52</td>
<td>41</td>
</tr>
<tr>
<td>(g) resource release with preemption</td>
<td>240</td>
<td>326</td>
<td>294</td>
</tr>
<tr>
<td>(h) category 2 ISR latency</td>
<td>47</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>(i) event blocking with task switch</td>
<td>141</td>
<td>172</td>
<td>224</td>
</tr>
<tr>
<td>(j) event setting with preemption</td>
<td>194</td>
<td>232</td>
<td>201</td>
</tr>
<tr>
<td>(k) comprehensive application</td>
<td>748</td>
<td>748</td>
<td>1216</td>
</tr>
</tbody>
</table>

CiAO outperforms the marked leader in 11 out of 12 cases by up to **260 percent**.

Execution time [clock ticks] on TC1796@50 MHz

(ac++1.0pre3 with tricore-g++3.4.3 -O3 -fno-rtti -funit-at-a-time -ffunction-sections -Xlinker --gc-sections)
<table>
<thead>
<tr>
<th>feature</th>
<th>with feature or instance</th>
<th>text</th>
<th>data</th>
<th>bss</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base system (OS control and tasks)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>per task</td>
<td>+ func</td>
<td>+ 20</td>
<td>+ 16</td>
<td></td>
</tr>
<tr>
<td>per application mode</td>
<td></td>
<td>0</td>
<td>+ 4</td>
<td>0</td>
</tr>
<tr>
<td>ISR cat. 1 support</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per ISR</td>
<td>+ func</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per disable–enable</td>
<td></td>
<td>+ 4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resource support</td>
<td></td>
<td>+ 128</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per resource</td>
<td></td>
<td>0</td>
<td>+ 4</td>
<td>0</td>
</tr>
<tr>
<td><strong>CiAO achieves excellent granularity!</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>per task</td>
<td></td>
<td>0</td>
<td>+ 8</td>
<td>0</td>
</tr>
<tr>
<td>per alarm</td>
<td></td>
<td>0</td>
<td>+ 12</td>
<td>0</td>
</tr>
<tr>
<td>Full preemption</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per join point</td>
<td>+ 12</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mixed preemption</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per join point</td>
<td>+ 44</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per task</td>
<td>0</td>
<td>+ 4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wrong context check</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per void join point</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per StatusType join point</td>
<td>+ 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Interrupts disabled check</td>
<td></td>
<td>0</td>
<td>+ 64</td>
<td>0</td>
</tr>
<tr>
<td>per join point</td>
<td>+ 64</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Invalid parameters check</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>per join point</td>
<td>+ 36</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Error hook</td>
<td></td>
<td>0</td>
<td>0</td>
<td>+ 4</td>
</tr>
<tr>
<td>per join point</td>
<td>+ 54</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Startup hook or shutdown hook</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pre-task hook or post-task hook</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Discussion: Aspect-Aware Development

By AAD CiAO achieves excellent properties [3–5]
- configurability and granularity even for fundamental kernel policies
- complete separation of concerns in the implementation

The approach has also been applied to other system software
- PUMA, the C/C++ transformation framework behind ac++ [9]
- CiAOIP, an aspect-oriented IP stack for embedded systems [2]

Issues: comprehensibility & tool support
- CiAO: aspect code/base code = 1/2.4
  \[\sim \text{where the heck xyz is implemented?}\]
- calls for additional tool support
- ac++ weaver implementation is stable, but not as mature as gcc
  - missing or confusing error messages
  - no support for weaving in template code
  - no C++0x support

<table>
<thead>
<tr>
<th></th>
<th>Base code</th>
<th>Aspect code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Files LOC</td>
<td>Files LOC</td>
</tr>
<tr>
<td>CiAO kernel only</td>
<td>423 21,086</td>
<td>333 5,923</td>
</tr>
<tr>
<td>CiAO COM</td>
<td>112 8,689</td>
<td>297 5,552</td>
</tr>
<tr>
<td>CiAO IP stack</td>
<td>45 5,038</td>
<td>96 3,230</td>
</tr>
<tr>
<td>CiAO overall</td>
<td>580 34,813</td>
<td>726 14,705</td>
</tr>
</tbody>
</table>

This is research, after all :-)}
4.1 AOP Mechanisms Under the Hood
4.2 Study: i4Weathermon AOP
4.3 CiAO
4.4 CiAO Results
4.5 Summary
4.6 References
Aspect-Aware Development exploits AOP mechanisms to achieve separation of concerns in configurable system software

- Advice inverses the direction in which control-flow relationships are established: $C \text{ calls } A \implies A \text{ advises } C$
- Advice-based binding as a means to integrate (optional) features
- Pointcuts provide for an implicit quantification of this integration
  - Advice-based binding as a means to integrate interacting features

CiAO applies these concepts from the very beginning

- Loose coupling by advice-based binding
- Visible transitions by explicit join points
- Minimal extensions by extension slices

The results are compelling

- Configurability of even all fundamental system policies
- Excellent granularity and footprint


