CiAO: An Aspect-Oriented OS Family for Resource-Constrained Embedded Systems

Daniel Lohmann
Wanja Hofer
Wolfgang Schröder-Preikschat

Friedrich-Alexander University Erlangen-Nuremberg

Jochen Streicher
Olaf Spinczyk

Technical University Dortmund

Motivation: Embedded Systems

Motivation: Embedded Systems

Motivation: Configurability of eCos

```
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_PRIORITY;
#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
  // Otherwise set it to zero.
  ceiling = 0;
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
  CYG_REPORT_RETURN();
}
```

Motivation: Configurability of eCos

```
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_PRIORITY;
#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
  // Otherwise set it to zero.
  ceiling = 0;
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
  CYG_REPORT_RETURN();
}
```

Mutex options:

- PROTOCOL
- CEILING
- INHERIT
- DYNAMIC

Kernel policies: Tracing Instrumentation Synchronization

Motivation: Configurability of eCos

```
void Cyg_Mutex::unlock(void) {
    CYG_REPORT_FUNCTION();
    Cyg_Scheduler::lock();
    CYG_INSTRUMENT_MUTEX(UNLOCK, this, 0);
    if( !queue.empty() ) {
        Cyg_Thread *thread = queue.dequeue();
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_INHERIT
        IF_PROTOCOL_INHERIT
            thread->relay_priority(owner, &queue);
#endif
        thread->set_wake_reason( Cyg_Thread::DONE );
        thread->wake();
        CYG_INSTRUMENT_MUTEX(WAKE, this, thread);
    }
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL
    IF_PROTOCOL_ACTIVE
        owner->uncount_mutex();
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_INHERIT
    IF_PROTOCOL_INHERIT
        owner->disinherit_priority();
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
    IF_PROTOCOL_CEILING
        owner->clear_priority_ceiling();
#endif
    locked      = false;
    owner       = NULL;
    Cyg_Scheduler::unlock();
    CYG_REPORT_RETURN();
}
```

utex
tions:

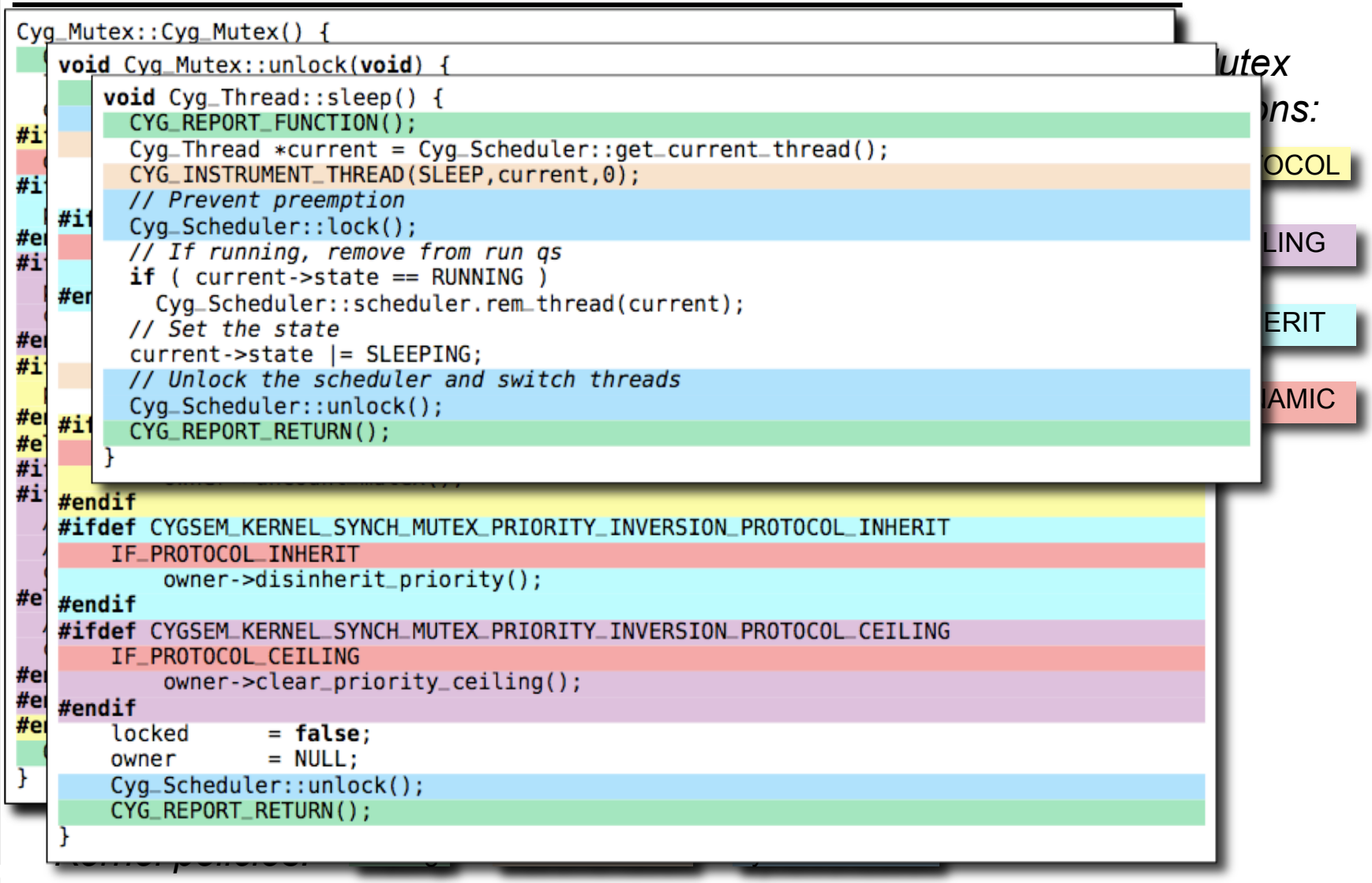
ROTOCOL

CEILING

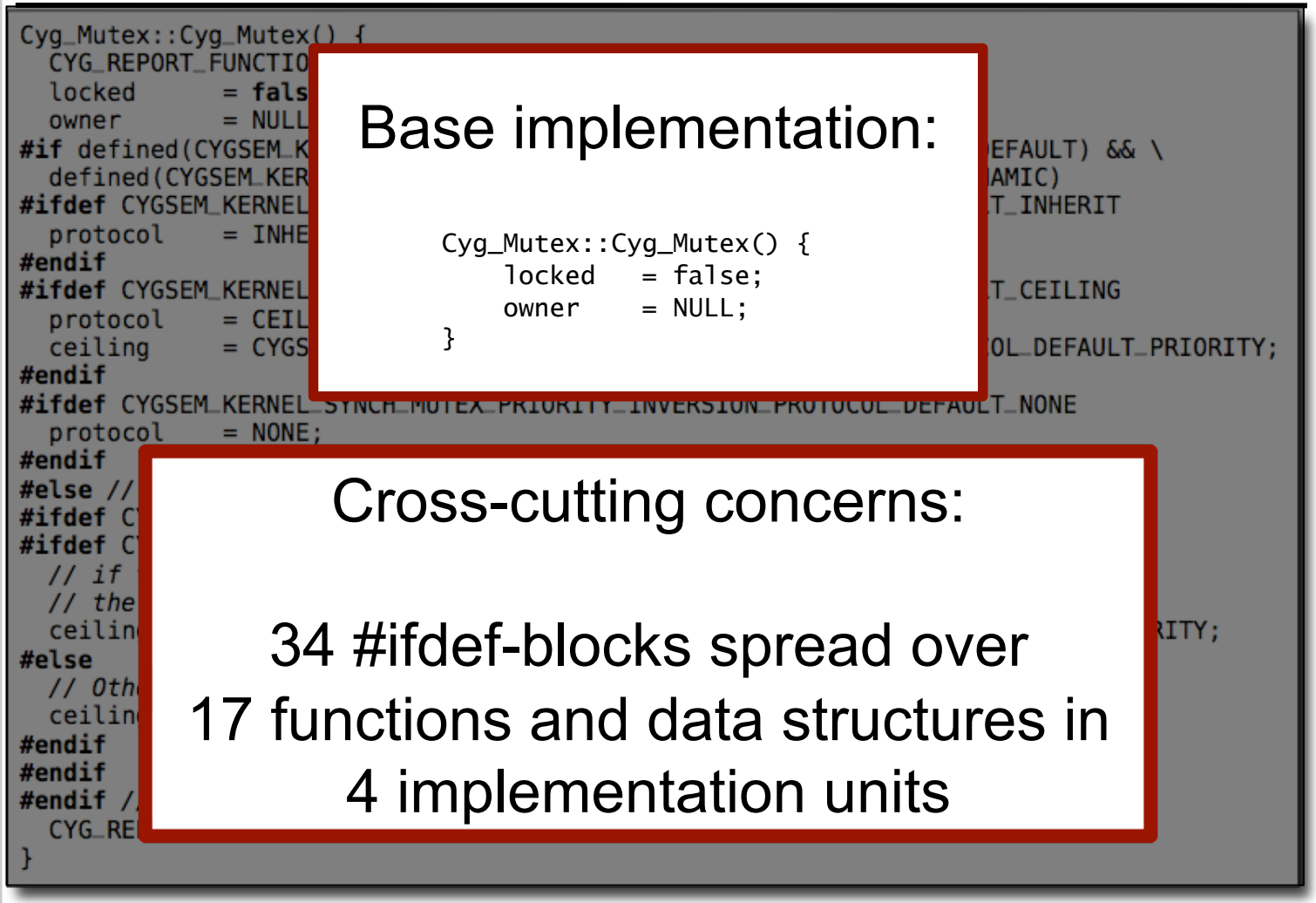
INHERIT

YNAMIC

Motivation: Configurability of eCos



Motivation: Configurability of eCos



Motivation: Configurability of OS

```
#ifdef hell
```

- Difficult to understand, maintain, evolve
- Lack of encapsulation in the implementation
- Tangling of many concerns in one implementation unit
- Scattering across several implementation units

Talk Outline

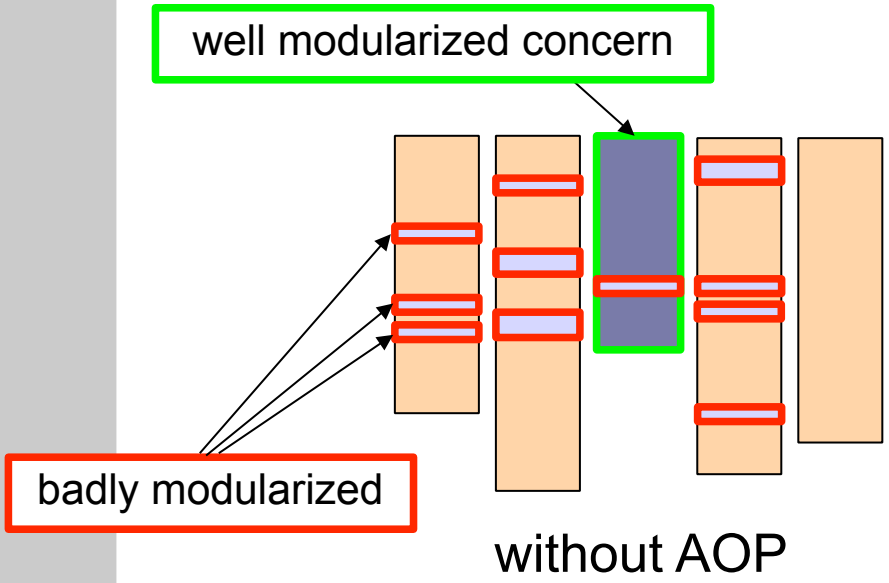
- Cross-cutting in Configurable System Software
- Aspect-Oriented Programming (AOP) to the Rescue?
- CiAO: Aspect-Aware OS Design
- Evaluation: Suitability for Embedded Systems

Talk Outline

- Cross-cutting in Configurable System Software
- Aspect-Oriented Programming (AOP) to the Rescue?
- CiAO: Aspect-Aware OS Design
- Evaluation: Suitability for Embedded Systems

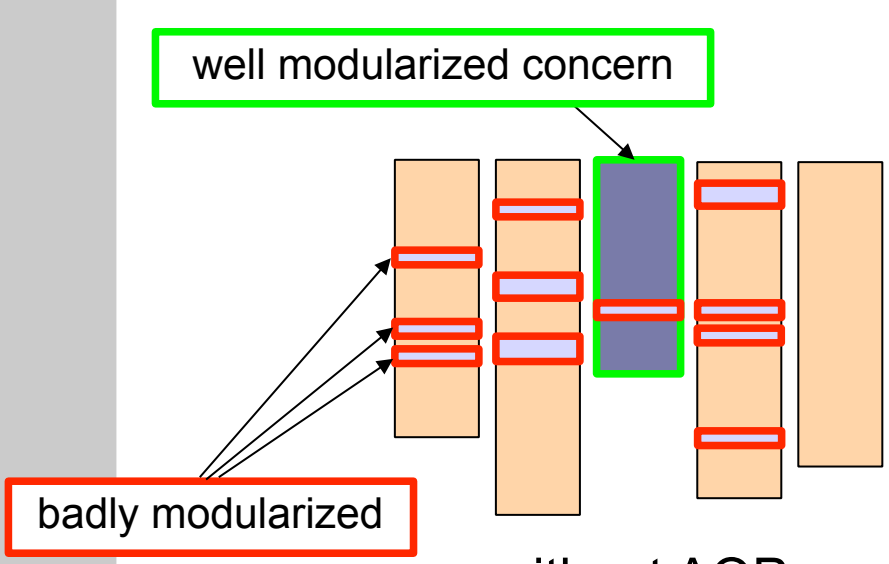
AOP to the Rescue?

- AOP aids modularization of cross-cutting concerns

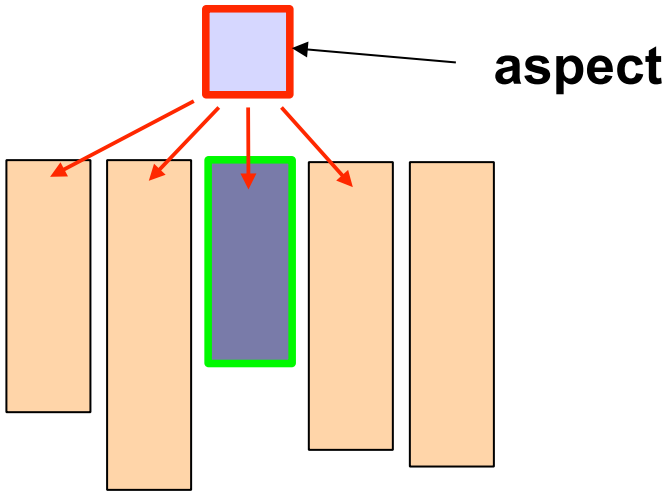


AOP to the Rescue?

- AOP aids modularization of cross-cutting concerns
- It does so by means of **aspects**



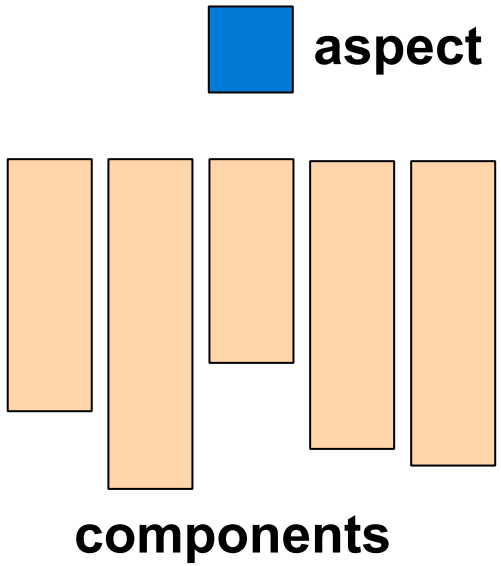
without AOP



with AOP

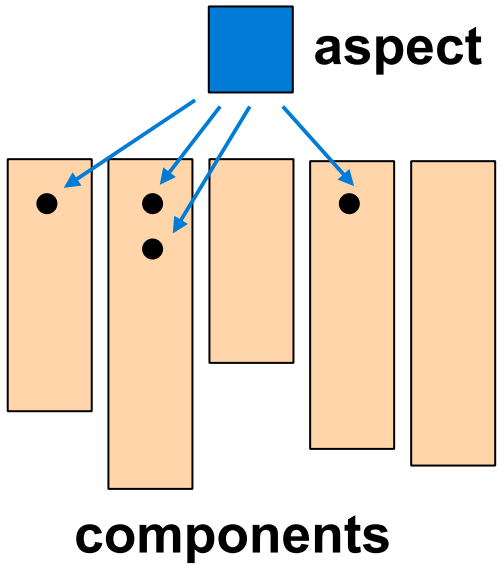
AOP – Short Introduction

- Encapsulation of (cross-cutting) concerns in **aspects**



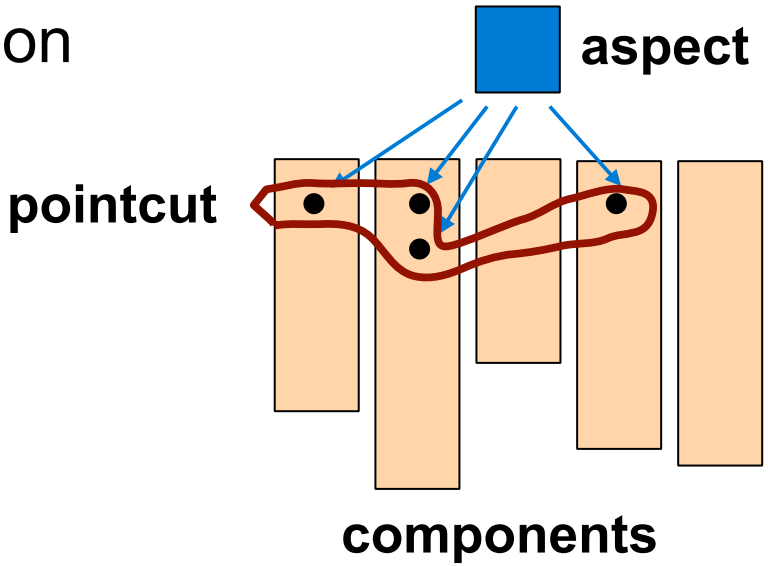
AOP – Short Introduction

- Encapsulation of (cross-cutting) concerns in **aspects**
- Aspects give **advice** to **join points** in the target system



AOP – Short Introduction

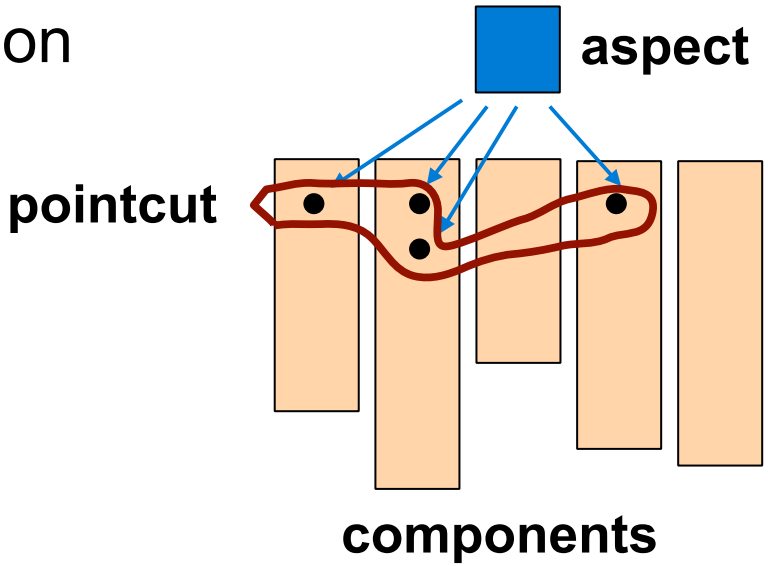
- Encapsulation of (cross-cutting) concerns in **aspects**
- Aspects give **advice** to **join points** in the target system
- Set of join points described by a **pointcut** expression



AOP – Short Introduction

- Encapsulation of (cross-cutting) concerns in **aspects**
- Aspects give **advice** to **join points** in the target system
- Set of join points described by a **pointcut** expression

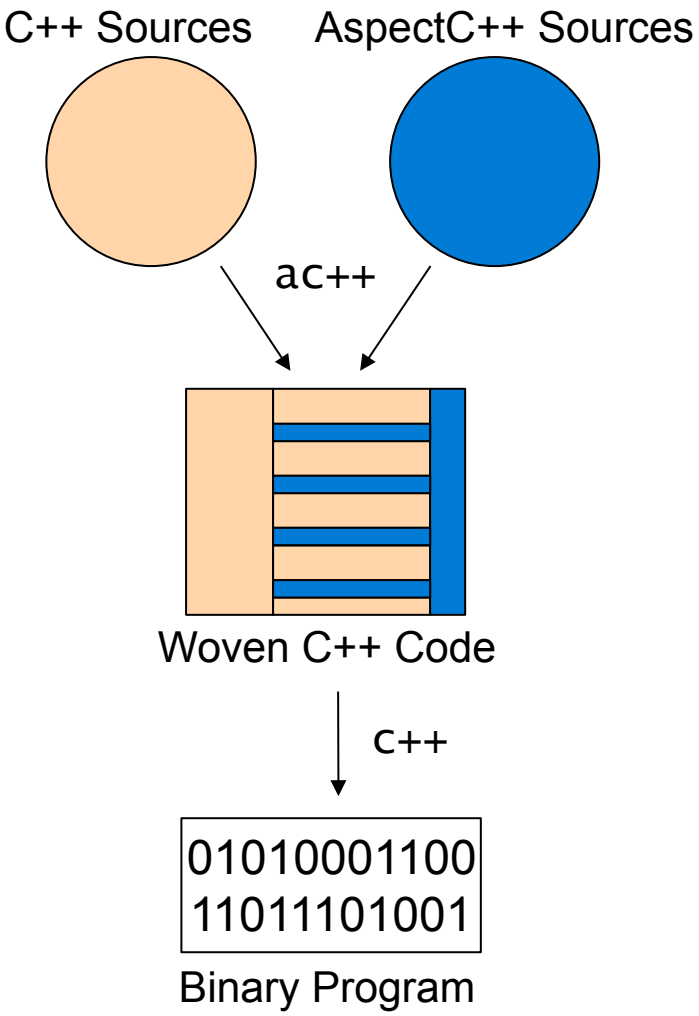
Obliviousness & Quantification



AOP – AspectC++

- Extension to C++
- Source-to-source weaver

http://www.aspectc.org



AOP – Why Can It Help?

- Inversion of caller–callee binding:

```
// traditional implementation

void dingus()
{
        …// do basic stuff
        #ifdef OPT_FEATURE
        thingy();
        #endif
}

void thingy()
{
        …// do optional stuff
}
```

AOP – Why Can It Help?

- Inversion of caller–callee binding:

```
// traditional implementation

void dingus()
{
        …// do basic stuff
        #ifdef OPT_FEATURE
        thingy();
        #endif
}

void thingy()
{
        …// do optional stuff
}
```

```
// AOP implementation

void dingus()
{
        // do basic stuff
}



advice execution("% dingus()") :
after()
{
        // do optional stuff
}
```

AOP – Why Can It Help?

- Quantification over multiple join points:

```
// traditional implementation

void dingus_foo() {
        #ifdef OPT_FEATURE
        thingy();
        #endif
}

void dingus_bar() {
        #ifdef OPT_FEATURE
        thingy();
        #endif
}

void thingy(){
        // do optional stuff
}
```

AOP – Why Can It Help?

- Quantification over multiple join points:

```
// traditional implementation

void dingus_foo() {
        #ifdef OPT_FEATURE
        thingy();
        #endif
}

void dingus_bar() {
        #ifdef OPT_FEATURE
        thingy();
        #endif
}

void thingy(){
        // do optional stuff
}
```

```
// AOP implementation

void dingus_foo() {
}




void dingus_bar() {
}




advice execution("% dingus%()") :
after() {
        // do optional stuff
}
```

Talk Outline

- Cross-cutting in Configurable System Software
- Aspect-Oriented Programming (AOP) to the Rescue?
- CiAO: Aspect-Aware OS Design
- Evaluation: Suitability for Embedded Systems

CiAO: Aspect-Aware OS Design

- Goal of the CiAO project:

  Evaluate if AOP is suitable for the design of configurable embedded system software

  - Can AOP help to avoid #ifdef hell?
  - Is AOP efficient enough for the domain of embedded systems?

CiAO: Aspect-Aware OS Design

- Goal of the CiAO project:

  Evaluate if AOP is suitable for the design of configurable embedded system software

  - Can AOP help to avoid #ifdef hell?
  - Is AOP efficient enough for the domain of embedded systems?

- **Yes**, if the system is designed in an **aspect-aware** manner

Obliviousness? No!

Quantification? Yes!

CiAO: Aspect-Aware OS Design

- Idea of aspect awareness:

  Provide **unambiguous** and **statically evaluable** join points.

- **Unambiguity**:

  Important system state transitions can be captured by a pointcut expression.

- **Static evaluation**:

  Avoid necessity for dynamic pointcut functions, which bear an overhead.

CiAO: Aspect-Aware OS Design

Loose coupling
Visible transitions
Minimal extensions

- Result:
  Sparse base system designed with classes,
  most functionality provided by (optional) aspects

- Three main aspect purposes:
  - **Extension aspects**: add features
  - **Policy aspects**: glue components together
  - **Upcall aspects**: bind behavior to lower-level events

Extension Aspects

- Example: task scheduling

```
// base implementation

struct Task {
    Pri pri_;
    State state_;
    …
};

class Sched {
    Tasklist ready_;
    Task::Id running_;
  public:
    void activate(Task::Id t);
    void reschedule();
    …
};
```

Extension Aspects

- Example: task scheduling extended (resource control)

```
// base implementation

struct Task {
    Pri pri_;
    State state_;
    …
};

class Sched {
    Tasklist ready_;
    Task::Id running_;
  public:
    void activate(Task::Id t);
    void reschedule();
    …
};
```

```
aspect ResourceSupport {

advice “Task” : slice struct {
    ResourceMask occupied_;
    Pri originalPri_;
};

advice “Sched” : slice class {
  public:
    void getRes(Res::Id r) {
      // lock mutex
    }
    void relRes(Res::Id r) {
      // unlock mutex
    }
};
};
```

ResourceSupport.ah

Policy Aspects

- Example: specification of preemption points

```
aspect FullPreemption {

    // points where another task may get a higher prio
    pointcut pcPreemptionPoints =
        "% Sched::activate(…)" ||
        "% Sched::setEvent(…)" ||
        "% Sched::relRes(…)";

    advice execution(pcPreemptionPoints()) : after() {
        tjp->that()->reschedule();
    }
};
```

Preemption.ah

Upcall Aspects

- Example: binding of an interrupt handler

```
aspect TimerBinding {

    advice execution("% irq::Timer::handler(…)") : after() {
        // handle IRQ
    }

};
```

TimerBinding.ah

Talk Outline

- Cross-cutting in Configurable System Software
- Aspect-Oriented Programming (AOP) to the Rescue?
- CiAO: Aspect-Aware OS Design
- Evaluation: Suitability for Embedded Systems

Evaluation

- Suitability of AOP for **configurable** systems
  - Increased modularization?
  - Scalability?
- Suitability of AOP for **resource-constrained** systems
  - Resource efficiency?
  - Scalability?

Evaluation

concern	extension	policy	upcall	advice	join points	extension of | advice-based binding to
ISR cat. 1 support	1		m	$2+m$	$2+m$	API, OS control | m ISR bindings
ISR cat. 2 support	1		n	$5+n$	$5+n$	API, OS control, scheduler | n ISR bindings
Resource support	1	1		3	5	scheduler, API, task | PCP policy implementation
Resource tracking		1		3	4	task, ISR | monitoring of Get/ReleaseResource
Event support	1			5	5	scheduler, API, task, alarm | trigger action JP
Full preemption		1		2	6	| 3 points of rescheduling
Mixed preemption		1		3	7	task | 3 points of rescheduling for task / ISR
Wrong context check		1		1	s	| s service calls
Interrupts disabled check		1		1	30	| all services except interrupt services
Invalid parameters check		1		1	25	| services with an OS object parameter
Error hook			1	2	30	scheduler | 29 services
Protection hook	1	1		2	2	API | default policy implementation
Startup / shutdown hook			1	2	2	| explicit hooks
Pre-task / post-task hook			1	2	2	| explicit hooks

Evaluation

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

Evaluation

feature	with feature or instance	text	data	bss
Base system (OS control and tasks)				
	per task	*+ func*	*+ 20*	*+ 16 + stack*
	per application mode	*0*	*+ 4*	*0*
ISR cat. 1 support		*0*	*0*	*0*
	per ISR	*+func*	*0*	*0*
	per disable–enable	*+ 4*	*0*	*0*
Resource support		*+ 128*	*0*	*0*
	per resource	*0*	*+ 4*	*0*
	per task	*0*	*+ 8*	*0*
Event support		*+ 280*	*0*	*0*
	per task	*0*	*+ 8*	*0*
	per alarm	*0*	*+ 12*	*0*
Full preemption		*0*	*0*	*0*
	per join point	*+ 12*	*0*	*0*
Mixed preemption		*0*	*0*	*0*
	per join point	*+ 44*	*0*	*0*
	per task	*0*	*+ 4*	*0*
Wrong context check		*0*	*0*	*0*
	per void join point	*0*	*0*	*0*
	per StatusType join point	*+ 8*	*0*	*0*
Interrupts disabled check		*0*	*0*	*0*
	per join point	*+ 64*	*0*	*0*
Invalid parameters check		*0*	*0*	*0*
	per join point	*+ 36*	*0*	*0*
Error hook		*0*	*0*	*+ 4*
	per join point	*+ 54*	*0*	*0*
Startup hook or shutdown hook		*0*	*0*	*0*
Pre-task hook or post-task hook		*0*	*0*	*0*

OSEK min
218
280
274
399
54
41
294
47
224
201
1216

Evaluation

- Suitability of AOP for **configurable** systems
  - Increased modularization? **Yes!**
  - Scalability? **Good!**
- Suitability of AOP for **resource-constrained** systems
  - Resource efficiency? **Good!**
  - Scalability? **Good!**

Evaluation: Issues

- Aspects for low-level code:
  - Transformations can break fragile join points (e.g., context switch)
  - Explicit join points with empty functions
- Aspect–aspect interdependencies:
  - Order advice, if several aspects affect same join point
- Join-point traceability:
  - Visualization of join-point deltas between revisions
  - Further tool support needed (acdt.aspectc.org)
- Granularity:
  - Advices apply to procedures (i.e. calls), not statements

Summary

- AOP is very well suited for the implementation of highly configurable system software, **avoiding #ifdef hell**
- With aspect-awareness design principles in mind, the **increased modularization** comes at **no costs**
- CiAO is the first aspect-oriented operating system

Summary

- AOP is very well suited for the implementation of highly configurable system software, **avoiding #ifdef hell**
- With aspect-awareness design principles in mind, the **increased modularization** comes at **no costs**
- CiAO is the first aspect-oriented operating system

Thanks for your attention!