AOM at Bellairs

Aspect-Oriented Multi-View Modeling

Wisam Al Abed, Jörg Kienzle
Software Engineering Laboratory
School of Computer Science
McGill University, Montreal, QC, Canada

Jacques Klein
Laboratory for Advanced Software Systems
University of Luxembourg
Luxembourg
Talk Outline

• Motivation
• Reusable Aspect Models
  • Aspect packaging
  • 3 views: structure, state, message
  • Support for elaborate aspect dependency chains
• AspectOPTIMA case study
• Recursive Weaving Algorithm
• Consistency Checks
• Conclusion
Multi-View Modeling

• Allows developers to describe a (software) system from multiple points of view
  • Structural views vs. behavioral views

• Allows developers to use multiple modeling notations / formalisms
  • Makes it possible for a modeler to use the most appropriate formalism to express the facet of the system in focus

• Challenges
  • Scalability
  • Consistency
Aspect-Oriented Modeling

• Aspect-oriented Modeling
  • Current weaving techniques based on one modeling notation only
  • Can address scalability within one modeling notation

• Aspect-oriented Multi-View Modeling
  • Compatible composition techniques and weaving algorithms must be defined to ensure that the consistency of individual aspects is preserved in woven model
Reusable Aspect Models (RAM)

- Aspect-oriented modeling approach integrating class diagrams, state diagrams and sequence diagrams
  - Aspect models can specify structure and behavior
  - Based on existing class diagram weaving technology [France et al.]
  - Based on existing sequence diagram weaving technology [Klein et al.]
  - Template parameters borrowed from Theme-UML [Clarke et al.]
- Reusable
  - Modular packaging
  - Dependencies encapsulated and automatically resolved
RAM Aspect Models

- Aspect package groups structure and behavioral models related to a concern
- One structural view
- One state view for each class defined in structural view
- At least one message view for each public method defined in structural view

![Aspect-Oriented Multi-View Modeling](aspect-oriented-multi-view-modeling.png)
Case Study: AspectOPTIMA

- Aspect-oriented framework implementing (different kinds of) transactions
  - Flat / nested / MTT / OMTT / CAA
  - Pessimistic / optimistic concurrency control
  - Undo / Redo recovery
  - In-place / deferred update
- Implemented in AspectJ
- Currently the functionalities of 18 aspects are modeled in RAM
  - Individually reusable
  - Complex aspect dependencies and interactions
Feature diagram allows users of the framework to choose desired transaction configuration.
The Context Aspect

- A process / thread of computation can create and enter a context
- Once it is inside, the thread is a context participant
- A context participant can leave the context at any time
1st View: Structural View

- One structural view
- Classes (partial or complete) with methods, associations between classes
- Partial classes are mandatory instantiation parameters of the model, shown as UML template parameters to the view
2nd View: State View (1)

- One state view for each class (partial or complete) defined in structural view
- Describes interaction protocol of instances
  - Each method must have at least one corresponding transition
2nd View: State View (2)

- State views of partial classes declare the behavior they assume (pointcut), and how they extend the behavior (advice)
- Placeholder states are mandatory instantiation parameters, and shown as UML template parameters to the view
At least one message view for each public method declared in the structural view

Describes message exchange between objects of the aspect that implements desired behavior
Aspect-Oriented Multi-View Modeling, © 2009 Jörg Kienzle

Base Model: A Banking Application

<table>
<thead>
<tr>
<th>Thread</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Thread create() + destroy()</td>
<td>int balance + Account create() + destroy() + withdraw(int amount) + deposit(int amount)</td>
</tr>
</tbody>
</table>

- **t: Thread**
  - createAndEnterContext()
  - withdraw(100)
  - leaveContext() -> deposit(100)

- **a: Account**
  - createAndEnterContext()
  - withdraw(100)

- **b: Account**
  - createAndEnterContext()

- **sd Thread**
  - createAndEnterContext
  - create
  - destroy

- **sd Account**
  - create
  - withdraw
  - deposit

Outside

Inside

Ready
Reuse by Instantiation / Binding

• To reuse the aspect *Context*, the base model must instantiate it explicitly

• An instantiation directive maps all mandatory instantiation parameters to base model elements

• When a model A instantiates an aspect model B, A reuses the structure and behavior modeled by B.

• Reuse of B might require A to extend the structure and behavior of B, in which case A must additionally define a binding to B
State View Weaving

Context instantiation

Context: | ContextParticipant | Thread
|-----------------|------------------|
| IContextParticipant.Idle | Outside
| IContextParticipant.IWorking | Inside

Binding

Any → *

Pointcuts

createAndEnterContext

Idle

IWorking

Advice

match

createAndEnterContext

Entering

setContext

Idle

Leaving

setContext

IWorking

leaveContext

Any

createAndEnterContext

Entering

setContext

Outside

Leaving

leaveContext

Inside

setContext

Any

getContext
Message View Weaving

match

caller: Caller

newContext := create()
addParticipant(target)
setContext(newContext)

createAndEnterContext()
withdraw(100)
deposit(100)
leaveContext(o)

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()

createAndEnterContext()
createAndEnterContext()
Aspect Dependencies

• RAM makes it very easy to build aspect frameworks
  • Aspects providing complex functionality can reuse aspects providing lower-level functionality

• Example
  • OutcomeAware depends on Context
  • Adds the notion of successful completion or unsuccessful completion of a context
OutcomeAware Structural View

aspect OutcomeAware depends on Context

structural view depends on Context

IOutcomeControllingParticipant
+ createAndEnterContext()
+ voteAndLeaveContext(Outcome vote)

Outcome myOutcome
+ Outcome getOutcome()
+ setOutcome(Outcome o)

Context instantiation
Context.IContextParticipant \rightarrow IOutcomeControllingParticipant

Context binding
ContextWithOutcome \rightarrow Context.Context

New Field
New Methods
OutcomeAware Message View

message view voteAndLeaveContext depends on Context

Pointcut

**caller: Caller**

**target: IOutcomeControlling Participant**

voteAndLeaveContext(o)

**Context.leaveContext instantiation**

leaveContext.target \rightarrow \text{target}
leaveContext.Caller \rightarrow \text{Caller}
leaveContext.caller \rightarrow \text{caller}

Advice

**caller: Caller**

**target: IOutcomeControlling Participant**

voteAndLeaveContext(o)

myContext :=

getContext()

setOutcome(o)

leaveContext()

**Binding**

caller \rightarrow *

Caller \rightarrow *

target \rightarrow *

leaveContext

leaveContext

leaveContext

leaveContext

leaveContext

leaveContext

leaveContext

leaveContext

leaveContext

leaveContext
Complex Example: Recovering
Base Model: Structure View

<table>
<thead>
<tr>
<th>Thread</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int balance</td>
</tr>
<tr>
<td>+ Thread create()</td>
<td>+ Account create()</td>
</tr>
<tr>
<td>+ destroy()</td>
<td>+ destroy()</td>
</tr>
<tr>
<td></td>
<td>+ withdraw(int amount)</td>
</tr>
<tr>
<td></td>
<td>+ deposit(int amount)</td>
</tr>
</tbody>
</table>

Recovering instantiation
Recovering.|RecoveringParticipant → Thread
Recovering.|Recoverable → Account
Recovering.|RecoveringParticipant instantiation
|RecoveringParticipant.|Idle → Outside
|RecoveringParticipant.|Working → Inside
|Recoverable.|BeforeM → Account.Ready
|Recoverable.|AfterM → Account.Ready
Woven Model: Structure View

<table>
<thead>
<tr>
<th>Stack</th>
<th>Account</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ push(Account e)</td>
<td>+ Account create()</td>
<td>+ create(Method m, Kind k, Account t)</td>
</tr>
<tr>
<td>+ Account getLast()</td>
<td>+ destroy()</td>
<td>+ boolean wasAppliedTo(Account t)</td>
</tr>
<tr>
<td>+ discardLast()</td>
<td>+ withdraw(int amount)</td>
<td>+ boolean isModify()</td>
</tr>
<tr>
<td></td>
<td>+ deposit(int amount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Kind getAccessKind(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Account clone()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>Account</td>
<td>Trace</td>
</tr>
<tr>
<td>+ push(Account e)</td>
<td>+ Account create()</td>
<td>+ create(Method m, Kind k, Account t)</td>
</tr>
<tr>
<td>+ Account getLast()</td>
<td>+ destroy()</td>
<td>+ boolean wasAppliedTo(Account t)</td>
</tr>
<tr>
<td>+ discardLast()</td>
<td>+ withdraw(int amount)</td>
<td>+ boolean isModify()</td>
</tr>
<tr>
<td></td>
<td>+ deposit(int amount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Kind getAccessKind(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Account clone()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- establish()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- restore()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- discard()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Account clone()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- establish()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- restore()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- discard()</td>
<td></td>
</tr>
</tbody>
</table>

| TraceList   | Context                                      |
|-------------|----------------------------------------------|--------------------|
| + insert(Trace t) | + Context create()                         | + Context create() |
| + Trace[] findTraces(Account o) | + destroy                           | + destroy |
| + Trace[] getTraces() | + outcome                          | - addParticipant(Thread t) |
|             | + Trace[] findTraces(Account o) | - removeParticipant(Thread t) |
|             | + Trace[] getTraces()                  | - Outcome getOutcome() |
|             |                                | - setOutcome(Outcome o) |
|             |                                | - boolean wasModified(Account obj) |
|             |                                | - Account[] getModified() |
|             |                                | - addTrace(Trace t) |
|             |                                | - addTraces(Trace t[]) |
|             |                                | - Trace[] getTraces() |
|             |                                | - removeTrace(Trace t[]) |
|             |                                | - restoreCheckpoints() |
|             |                                |                     |

| Stack       | Account                                      |
|-------------|----------------------------------------------|--------------------|
| + push(Account e) | + Account create()                          | + create(Method m, Kind k, Account t) |
| + Account getLast() | + destroy()                                | + boolean wasAppliedTo(Account t) |
| + discardLast() | + withdraw(int amount)                      | + boolean isModify() |
|             | + deposit(int amount)                       |                     |
|             | - Kind getAccessKind(Method m)             |                     |
|             | - Account clone()                          |                     |
|             | - copyState(Account from)                  |                     |
|             | - Trace createTrace(Method m)              |                     |
|             | - establish()                              |                     |
|             | - restore()                                |                     |
|             | - discard()                                |                     |
|             | + Account clone()                          |                     |
|             | - copyState(Account from)                  |                     |
|             | - Trace createTrace(Method m)              |                     |
|             | - establish()                              |                     |
|             | - restore()                                |                     |
|             | - discard()                                |                     |

<table>
<thead>
<tr>
<th>Account</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Account create()</td>
<td>+ create(Method m, Kind k, Account t)</td>
</tr>
<tr>
<td>+ destroy()</td>
<td>+ boolean wasAppliedTo(Account t)</td>
</tr>
<tr>
<td>+ withdraw(int amount)</td>
<td>+ boolean isModify()</td>
</tr>
<tr>
<td>+ deposit(int amount)</td>
<td></td>
</tr>
<tr>
<td>- Kind getAccessKind(Method m)</td>
<td></td>
</tr>
<tr>
<td>- Account clone()</td>
<td></td>
</tr>
<tr>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td>- establish()</td>
<td></td>
</tr>
<tr>
<td>- restore()</td>
<td></td>
</tr>
<tr>
<td>- discard()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ TransactionContext getContext()</td>
<td>+ push(Account e)</td>
</tr>
<tr>
<td>+ setContext(TransactionContext c)</td>
<td>+ Account getLast()</td>
</tr>
<tr>
<td>+ beginTransaction()</td>
<td>+ discardLast()</td>
</tr>
<tr>
<td>+ commitTransaction()</td>
<td></td>
</tr>
<tr>
<td>+ createContext()</td>
<td></td>
</tr>
<tr>
<td>+ leaveContext()</td>
<td></td>
</tr>
</tbody>
</table>

| Stack       | Account                                      |
|-------------|----------------------------------------------|--------------------|
| + push(Account e) | + Account create()                          | + create(Method m, Kind k, Account t) |
| + Account getLast() | + destroy()                                | + boolean wasAppliedTo(Account t) |
| + discardLast() | + withdraw(int amount)                      | + boolean isModify() |
|             | + deposit(int amount)                       |                     |
|             | - Kind getAccessKind(Method m)             |                     |
|             | - Account clone()                          |                     |
|             | - copyState(Account from)                  |                     |
|             | - Trace createTrace(Method m)              |                     |
|             | - establish()                              |                     |
|             | - restore()                                |                     |
|             | - discard()                                |                     |

<table>
<thead>
<tr>
<th>Thread</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ TransactionContext getContext()</td>
<td>+ push(Account e)</td>
</tr>
<tr>
<td>+ setContext(TransactionContext c)</td>
<td>+ Account getLast()</td>
</tr>
<tr>
<td>+ beginTransaction()</td>
<td>+ discardLast()</td>
</tr>
<tr>
<td>+ commitTransaction()</td>
<td></td>
</tr>
<tr>
<td>+ createContext()</td>
<td></td>
</tr>
<tr>
<td>+ leaveContext()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Account create()</td>
<td>+ create(Method m, Kind k, Account t)</td>
</tr>
<tr>
<td>+ destroy()</td>
<td>+ boolean wasAppliedTo(Account t)</td>
</tr>
<tr>
<td>+ withdraw(int amount)</td>
<td>+ boolean isModify()</td>
</tr>
<tr>
<td>+ deposit(int amount)</td>
<td></td>
</tr>
<tr>
<td>- Kind getAccessKind(Method m)</td>
<td></td>
</tr>
<tr>
<td>- Account clone()</td>
<td></td>
</tr>
<tr>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td>- establish()</td>
<td></td>
</tr>
<tr>
<td>- restore()</td>
<td></td>
</tr>
<tr>
<td>- discard()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ TransactionContext getContext()</td>
<td>+ push(Account e)</td>
</tr>
<tr>
<td>+ setContext(TransactionContext c)</td>
<td>+ Account getLast()</td>
</tr>
<tr>
<td>+ beginTransaction()</td>
<td>+ discardLast()</td>
</tr>
<tr>
<td>+ commitTransaction()</td>
<td></td>
</tr>
<tr>
<td>+ createContext()</td>
<td></td>
</tr>
<tr>
<td>+ leaveContext()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack</th>
<th>Account</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ push(Account e)</td>
<td>+ Account create()</td>
<td>+ create(Method m, Kind k, Account t)</td>
</tr>
<tr>
<td>+ Account getLast()</td>
<td>+ destroy()</td>
<td>+ boolean wasAppliedTo(Account t)</td>
</tr>
<tr>
<td>+ discardLast()</td>
<td>+ withdraw(int amount)</td>
<td>+ boolean isModify()</td>
</tr>
<tr>
<td></td>
<td>+ deposit(int amount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Kind getAccessKind(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Account clone()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- establish()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- restore()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- discard()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Account clone()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- copyState(Account from)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Trace createTrace(Method m)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- establish()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- restore()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- discard()</td>
<td></td>
</tr>
</tbody>
</table>
Base Model: State View

Ouside
create
createAndEnterContext
destroy
voteAndLeaveContext

Inside

Recovering instantiation
Recovering.IRRecoveringParticipant -> Thread
Recovering.IRRecoverable -> Account

Recovering.IRRecoveringParticipant instantiation
IRecoveringParticipant.Idle -> Outside
IRecoveringParticipant.IWorking -> Inside
Recovering.IRecoverable instantiation
IRecoverable.BeforeM -> Account.Ready
IRecoverable.AfterM -> Account.Ready

sd Thread

sd Account

Thread
create
withdraw
destroy
deposit

Account
create
withdraw
destroy
deposit

Ready

McGill
Base Model: Message View

- **t: Thread**
  - `createAndEnterContext()`
  - `withdraw(100)`
  - `voteAndLeaveContext(o)`

- **a: Account**
  - `deposit(100)`

- **b: Account**

**Recovering instantiation**
- `Recovering.IRecoveringParticipant → Thread`
- `Recovering.IRecoverable → Account`
- `Recovering.IRecoveringParticipant instantiation`
- `IRecoveringParticipant.Idle → Outside`
- `IRecoveringParticipant.IWorking → Inside`
- `Recovering.IRecoverable instantiation`
- `IRecoverable.BeforeM → Account.Ready`
- `IRecoverable.AfterM → Account.Ready`
## Involved Aspects

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Ware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>Checkpointing</td>
</tr>
</tbody>
</table>

### Checkpointing

1. `newContext := create()`
2. `addParticipant(t)`
3. `setContext(newContext)`
4. `myContext := getContext()`
5. `secondTime := wasModified(a)`
6. `myContext := getContext()`

### Tracing

1. `tracedOp := create(withdraw)`
2. `accessKind := getAccessKind(withdraw)`
3. `trace(tracedOp)`
4. `removeParticipant(t)`
5. `leaveContext()`
6. `opt` (optionally)

### Checkpointable

1. `newCheckpoint := clone()`
2. `copyState(a)`
3. `push(newCheckpoint)`
4. `newCheckpoint := create()`
5. `setOutcome(o)`
6. `newCheckpoint := create()`

### Copyable

1. `newCheckpoint := create()`
2. `copyState(b)`
3. `push(newCheckpoint)`
4. `newCheckpoint := create()`

### Access Kind

1. `accessKind := getAccessKind(d)`
2. `trace(a)`
3. `createAndEnterContext()`
4. `setContext(null)`

### Recovery

1. `restoreCheckpoints()`
2. `contextCompleted()`
3. `obj := getModifications()`
4. `ops := fi` (find indices)
5. `found := isModify()`
6. `discard()`

### Accounting

1. `newCheckpoint := create()`
2. `race(tracedOp)`
3. `race(withdraw)`
4. `race(deposit)`
5. `race(tracedOp)`
6. `race(withdraw)`

---

The diagram illustrates the process of creating and entering a context, adding participants, setting the context, and handling modifications. It includes steps for creating checkpoints, copying states, and pushing new checkpoints for recovery purposes. The diagram also shows how access kinds are handled and trace operations are created and raced.
Recursive Weaving Algorithm

- Depth-first recursion through aspect tree, weaving lower-level aspects into higher-level aspects according to instantiation and binding directives
- Details in the paper

![Diagram of Recursive Weaving Algorithm]
Consistency Checks

• When creating the aspect model, consistency checks are performed among views
  • Standard consistency checks
  • Conformance between the message views and the state views (model checking).

• Before (during) the weaving, consistency checks are performed among the directives defined at different levels
  • For structural views, bindings and instantiations defining mappings of the same structural entity must be consistent.
  • For state views, bindings and instantiations at a “higher” level must designate the same state or substates of the bindings of the “lower” level.
  • For message views, bindings at a “higher” level must match a subset of messages of the bindings of the “lower” level.

• After executing the weaving, consistency checks are performed among the woven views
  • Conformance of the partial sequences of messages defined for each object instance in the woven sequence diagram with the protocol defined in the state view of the corresponding class (model checking)
Conclusion

• Reusable Aspect Models (RAM)
  • Aspect-oriented modeling approach integrating class diagrams, state diagrams and sequence diagrams
  • Reuse is safe (mandatory instantiation parameters) and flexible (optional bindings)
  • Support for elaborate dependency chains
• Weaving algorithm recursively resolves dependencies to create independent final model
• Elaborate consistency checks
  • Model-checking of conformance of sequence diagram and state diagrams
Other Work

• RAM also provides
  • Conflict resolution aspects
  • Feature diagrams to select variations
• Prototype tool
  • Implemented in Eclipse/Kermeta
  • Based on Kompose [Fleury et al.] and GeKo [Morin et al.]

To download papers, the AspectJ implementation or the RAM models of AspectOPTIMA
http://www.cs.mcgill.ca/~joerg/AspectOPTIMA/AspectOPTIMA.html
(or just google AspectOPTIMA)
Questions?