

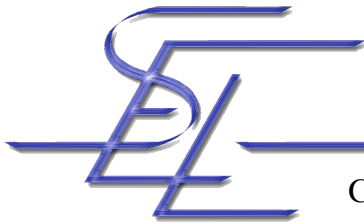
COMP-667 Software Fault Tolerance

AspectOPTIMA Part 1

Jörg Kienzle

School of Computer Science
McGill University, Montreal, QC, Canada

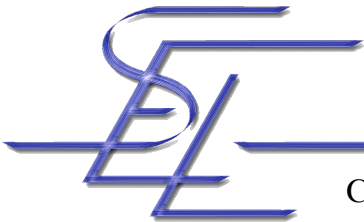
With Contributions From:
Samuel Gélinau, Ekwa Duala-Ekoko,
Güven Bölükbaşı, Barbara Gallina



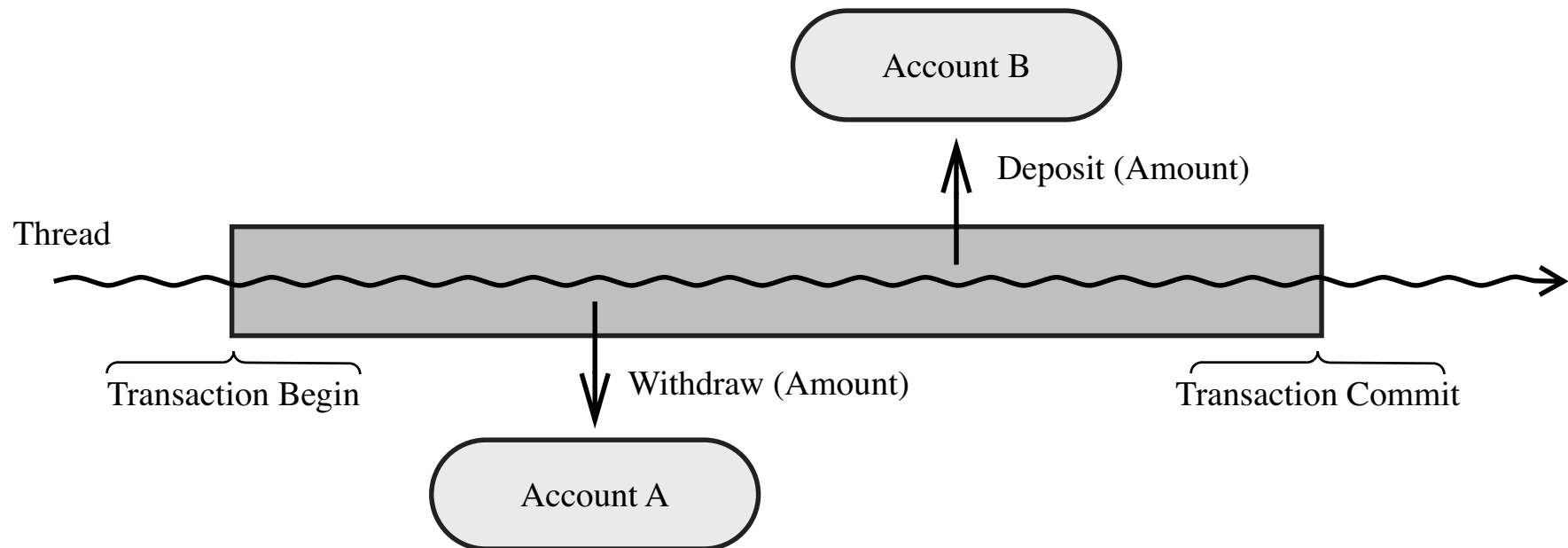
McGill

Talk Outline

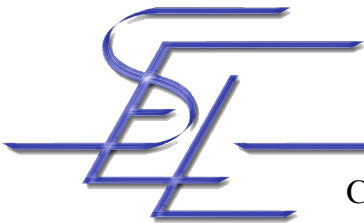
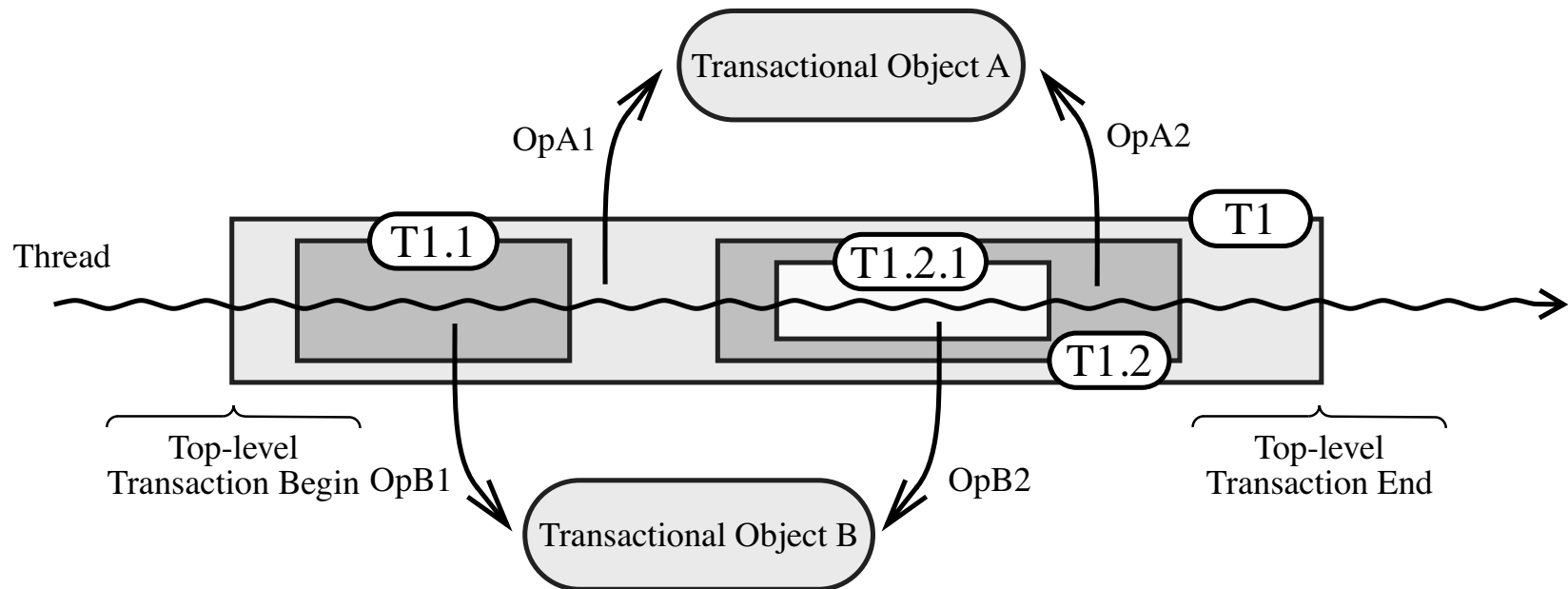
- Motivation
 - Many different transaction models
 - Different ways of doing concurrency control and recovery
- AspectOPTIMA
 - Aspects for
 - Objects
 - Threads
 - Contexts
 - Example Configurations
- Exception Handling Extension



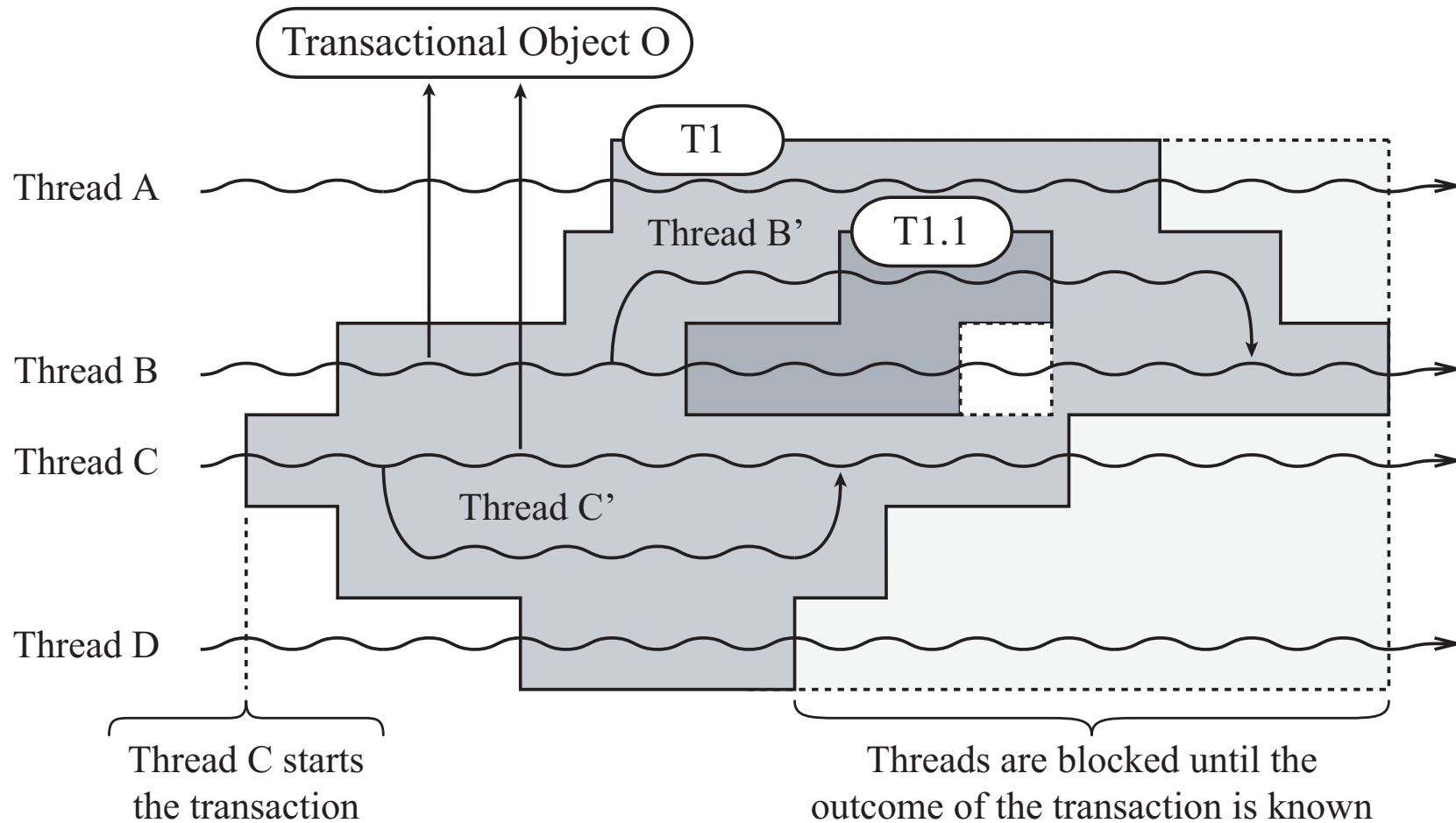
Transaction Models: Flat Transactions



Transaction Models: Nested Transactions



Transaction Models: Open Multithreaded Transactions



Motivation for AspectOPTIMA

- Observations
 - Concurrency control and recovery are separate concerns at a higher level of abstraction
 - At the implementation level, the two concerns are tightly coupled (see lecture on transaction implementations)
 - Most transaction models are related, i.e. they share common concepts
- Challenge
 - Is it possible to define many individually reusable aspects that, when put together in different ways, can implement various transaction models, concurrency control and recovery strategies?

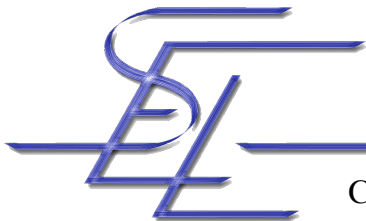
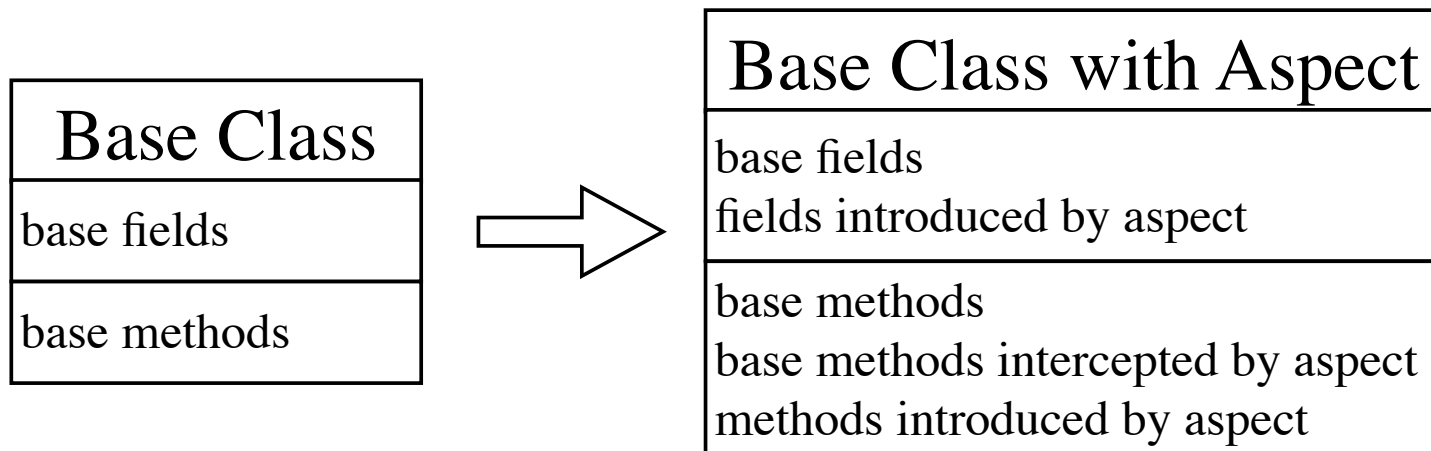


McGill



AspectJ Design of AspectOPTIMA

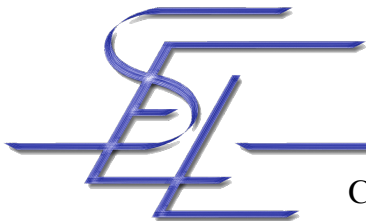
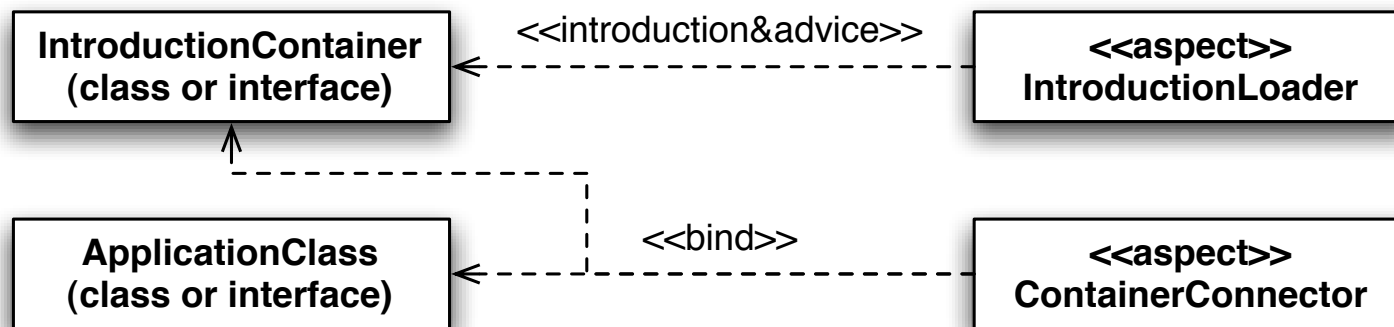
- In our AspectJ implementation of AspectOPTIMA, an aspect encapsulates additional structure and behavior applicable to base classes



McGill

Reusable Bindings in AspectJ

- Abstract Introduction Idiom
 - Each aspect is applied to a dummy interface
 - Bindings are established by making an application class implement the dummy interface
 - Binding can be specified using an aspect as well!



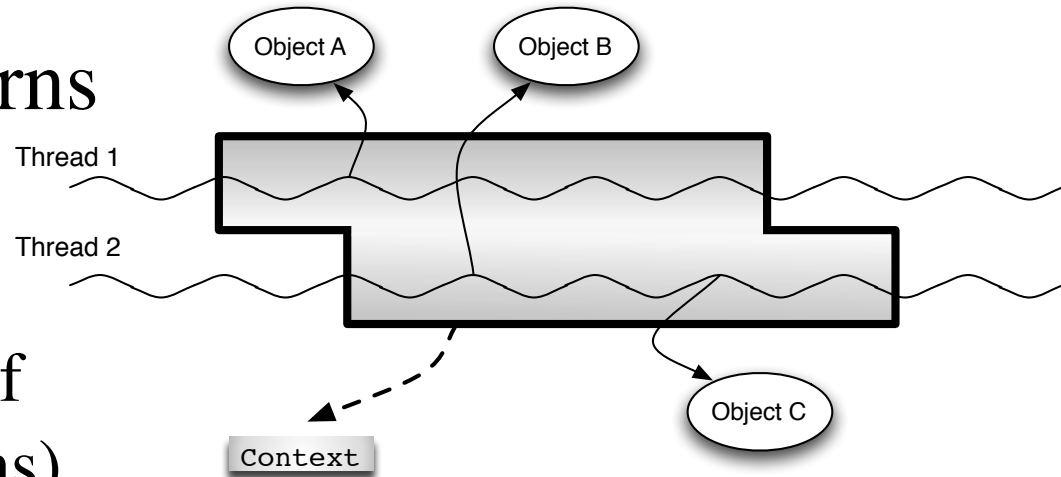
Design of AspectOPTIMA

- 3 high-level concerns

- *Objects*

- *Threads*

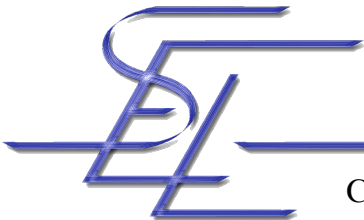
- *Contexts* (or scopes of computations)



- We identified 12 aspects for objects, 3 for threads, and 13 for contexts

- Each aspect has well-defined functionality and is individually reusable

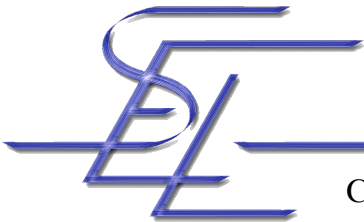
- Subtle dependencies and conflicts between aspects



AccessClassified in AspectJ

```
package library.aspects.object;
import java.lang.reflect.Method;
import library.annotations.*;
import library.interfaces.AccessClassified;
import library.util.AccessTypes;

public aspect AccessClassifiedAspect {
    public String AccessClassified.getAccessTypeOfMethod (String methodName) {
        String accessType = AccessTypes.WRITE;
        for (Method method : this.getClass().getMethods()) {
            if ((methodName.trim()).equalsIgnoreCase(method.getName())) {
                if(method.isAnnotationPresent(ReadAccess.class))
                    accessType = AccessTypes.READ;
                else if(method.isAnnotationPresent(UpdateAccess.class))
                    accessType = AccessTypes.UPDATE;
                break;
            }
            // If there is no annotation, assume the worst case
        }
        return accessType;
    }
}
```



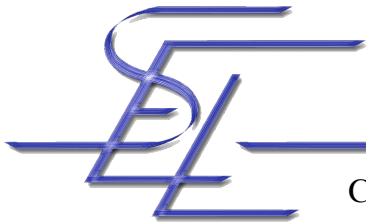
AccessClassified Bank Account

```
import library.annotations.*;
import library.interfaces.AccessClassified;

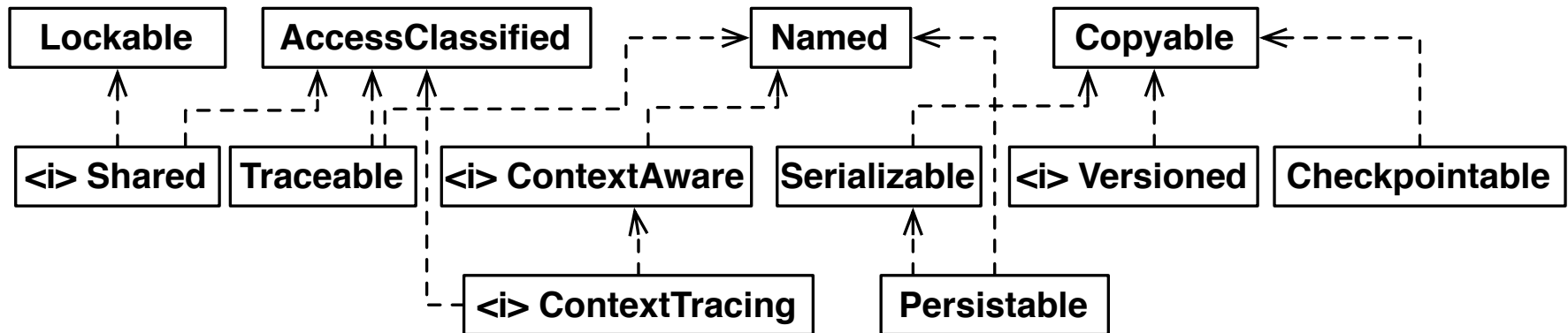
public class Account implements AccessClassified {
    public Account(int startingBalance) {
        balance = startingBalance;
    }

    @ReadAccess
    public int getBalance()
    {
        return balance;
    }

    @WriteAccess
    public void setBalance(int newBalance)
    {
        balance = newBalance;
    }
}
```



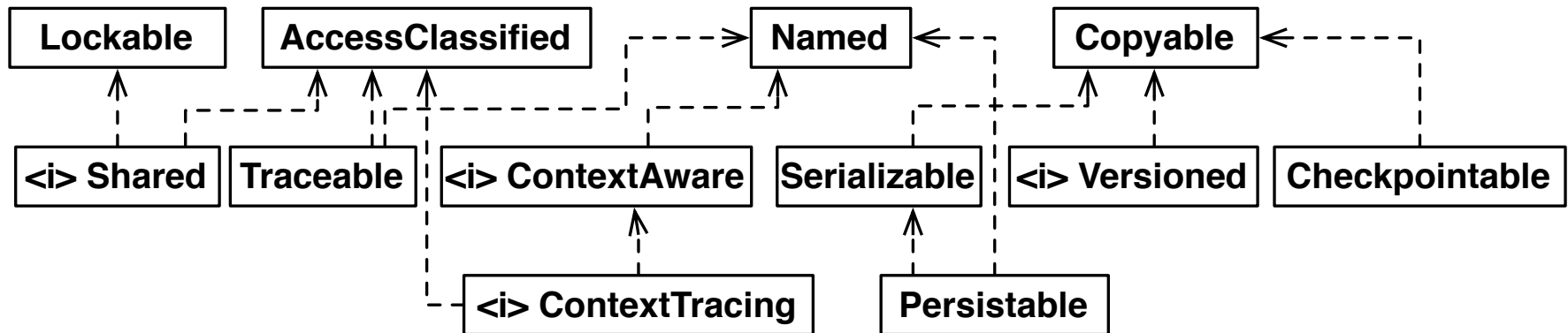
Per Object Aspects



- *Lockable*: Creates lock types, gets and releases locks
- *AccessClassified*: Provides access kind for each method (*read, write, update*)
- *Named*: Associates a name (string) with each application object instance



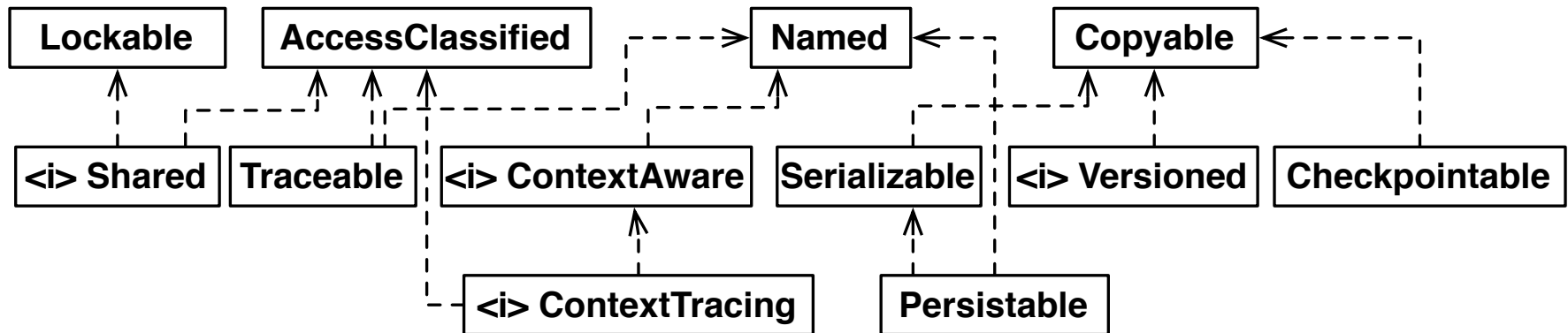
Per Object Aspects



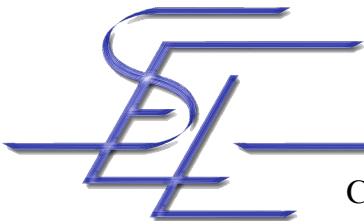
- *Copyable*: Provides cloning and state replacement capabilities
- *Shared*: Enforces multiple reader / single writer
- *Traceable*: Provides operation invocation information



Per Object Aspects



- *ContextAware*: Informs context whenever an operation is invoked
- *Serializable*: Provides streaming capabilities
- *Versioned*: Creates views (separate instances of the *same* application object), associable to threads



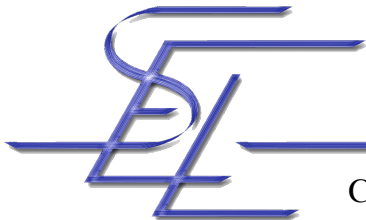
Specifying Dependencies in AspectJ

- Traceable objects have to be AccessClassified and Named as well

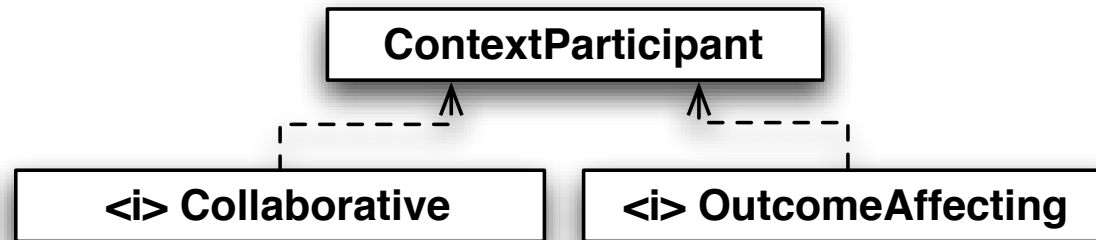
```
package library.aspects.object;
import library.interfaces.Traceable;
import library.interfaces.Named;
import library.interfaces.AccessClassified;
import library.util.ObjectTrace;

public aspect TraceableAspect {
    declare parents: Traceable implements Named, AccessClassified;

    public ObjectTrace Traceable.createMyTrace(String methodName) {
        String accessType = getAccessTypeOfMethod(methodName);
        ObjectTrace trace = new ObjectTrace(this, accessType);
        return trace;
    }
}
```



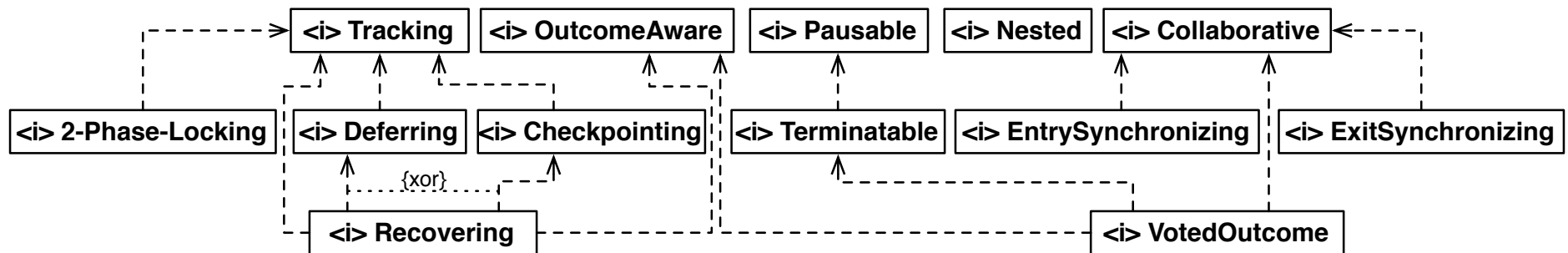
Per Thread Aspects



- *ContextParticipant*: Provides context creation and destruction functionality
- *Collaborative*: Provides joining functionality and control on number of participants
- *OutcomeAffecting*: Provides opinion on outcome of the context



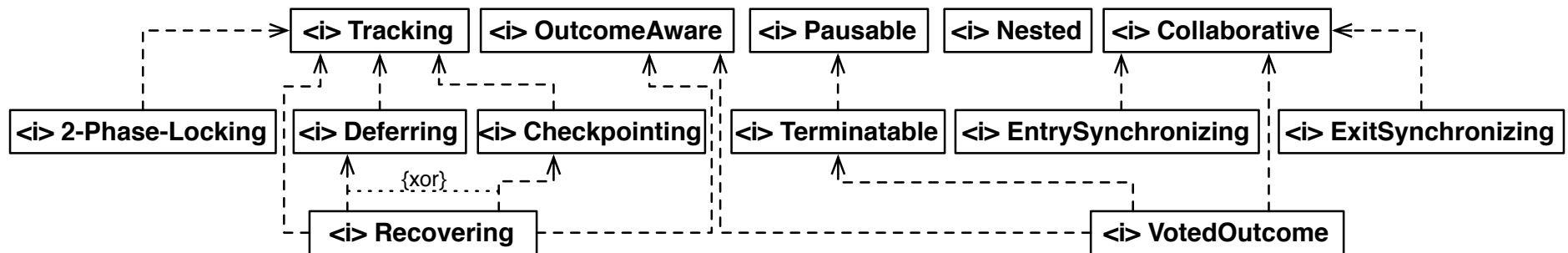
Per Context Aspects



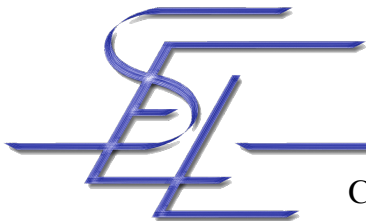
- *Tracking*: Remembers all operation invocations made on behalf of the context
- *OutcomeAware*: Associates success/failure outcome with a context
- *Pausable*: Suspends participant work if needed



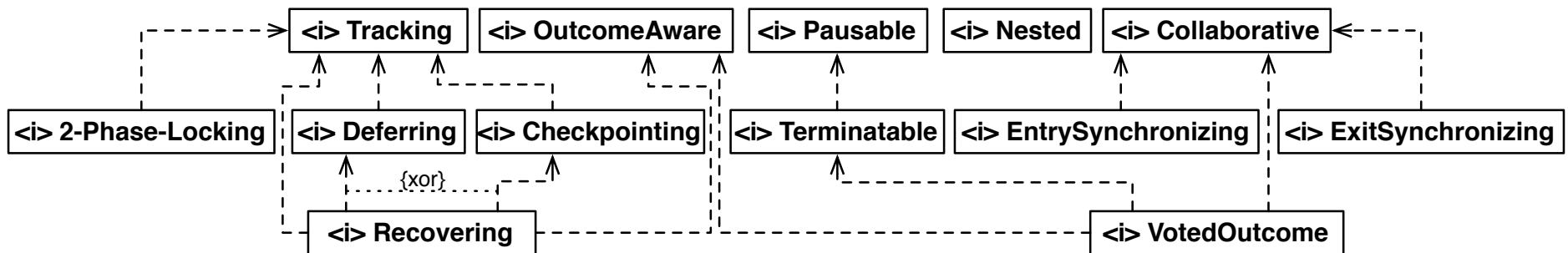
Per Context Aspects



- *Nested*: Allows contexts to be nested
- *Collaborative*: Manages many participants for a context
- *2-Phase-Locking*: Forces participants to acquire read/write/update locks when performing work, and releases all locks when context ends



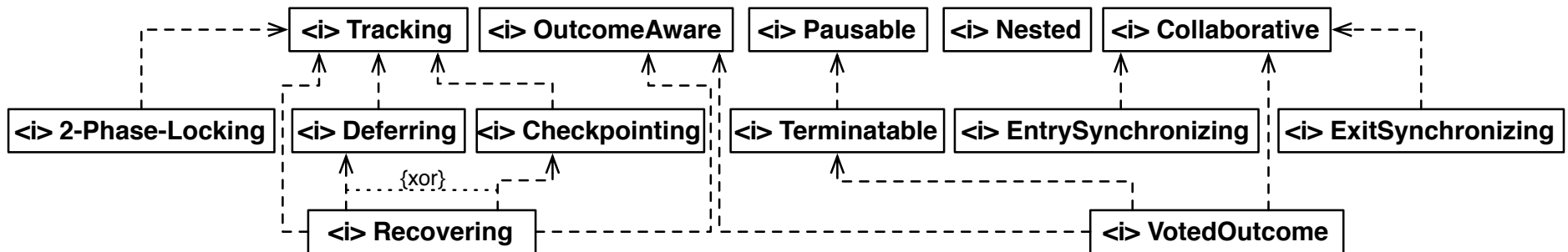
Per Context Aspects



- *Deferring*: Create a context-local version of every object before modification takes place
- *Checkpointing*: Establish a checkpoint before modification takes place
- *Terminatable*: Interrupt participants and end context, if needed



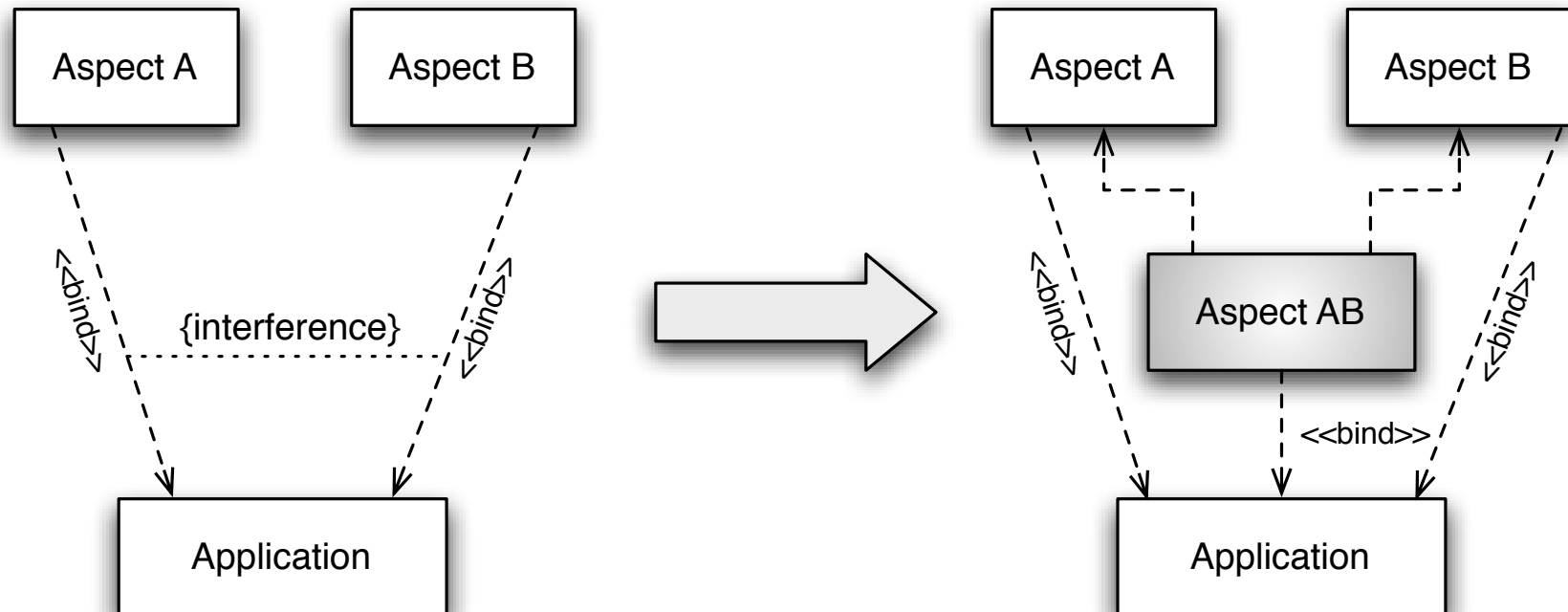
Per Context Aspects



- *EntrySynchronizing / ExitSynchronizing*: Synchronize participants on context entry or context exit
- *Recovering*: Undo all state changes if outcome is unsuccessful
- *VotedOutcome*: Decide on context outcome by applying a voting strategy to the opinions of participants



Dealing with Aspect Conflicts



Conflict Examples

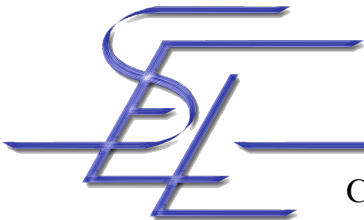
- Copyable \leftrightarrow Lockable
- Copyable \leftrightarrow Shared
- Serializable \leftrightarrow Named
- Versioned \leftrightarrow ContextAware
- Versioned \leftrightarrow Persistable
- Checkpointable \leftrightarrow Persistable
- Nested \leftrightarrow Tracking
- Nested \leftrightarrow Deferring
- Nested \leftrightarrow Checkpointing
- Nested \leftrightarrow 2-Phase-Locking
- Nested \leftrightarrow 2-Phase-Locking \leftrightarrow Recovering



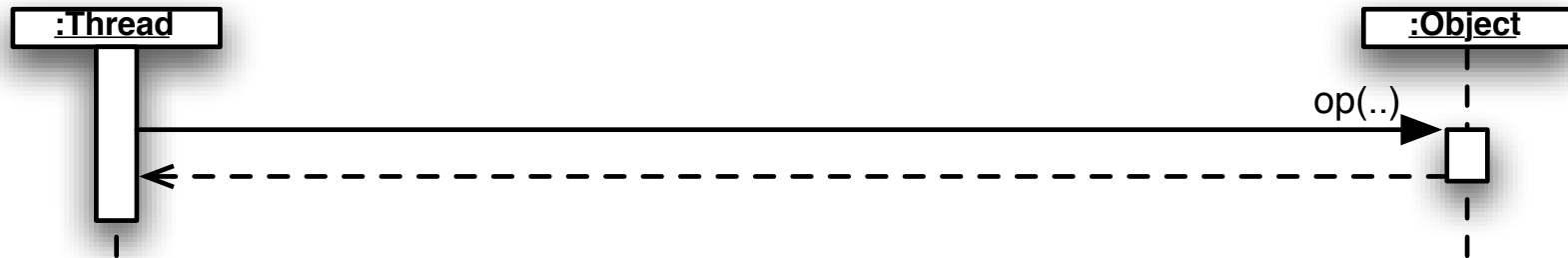
Example Configuration 1

Flat Transactions, Optimistic Concurrency Control, Deferred Update

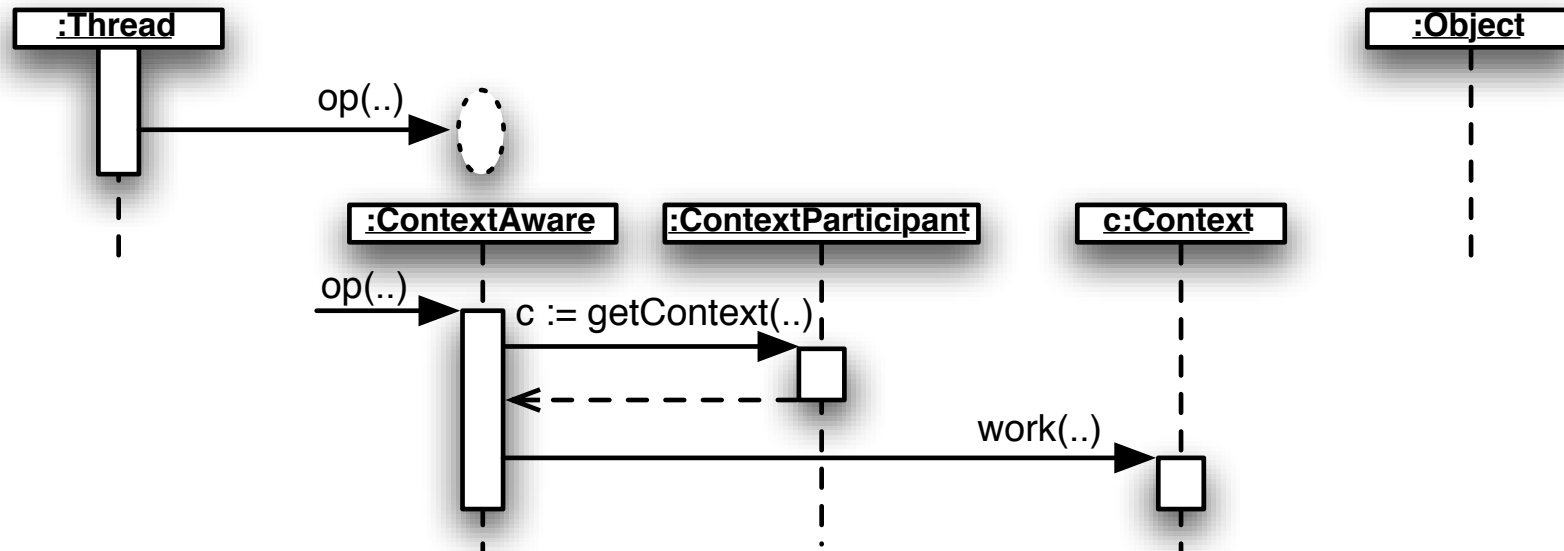
- **Thread:** ContextParticipant, OutcomeAffecting
- **Context:** Tracking, Deferring, OutcomeAware, Recovering
- **Object:** ContextAware, AccessClassified, Named, Trackable, Copyable, Versioned, Persistable, ContextTracking



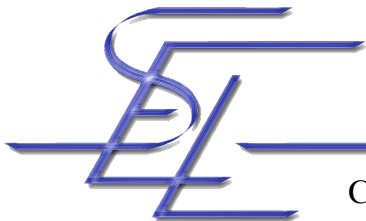
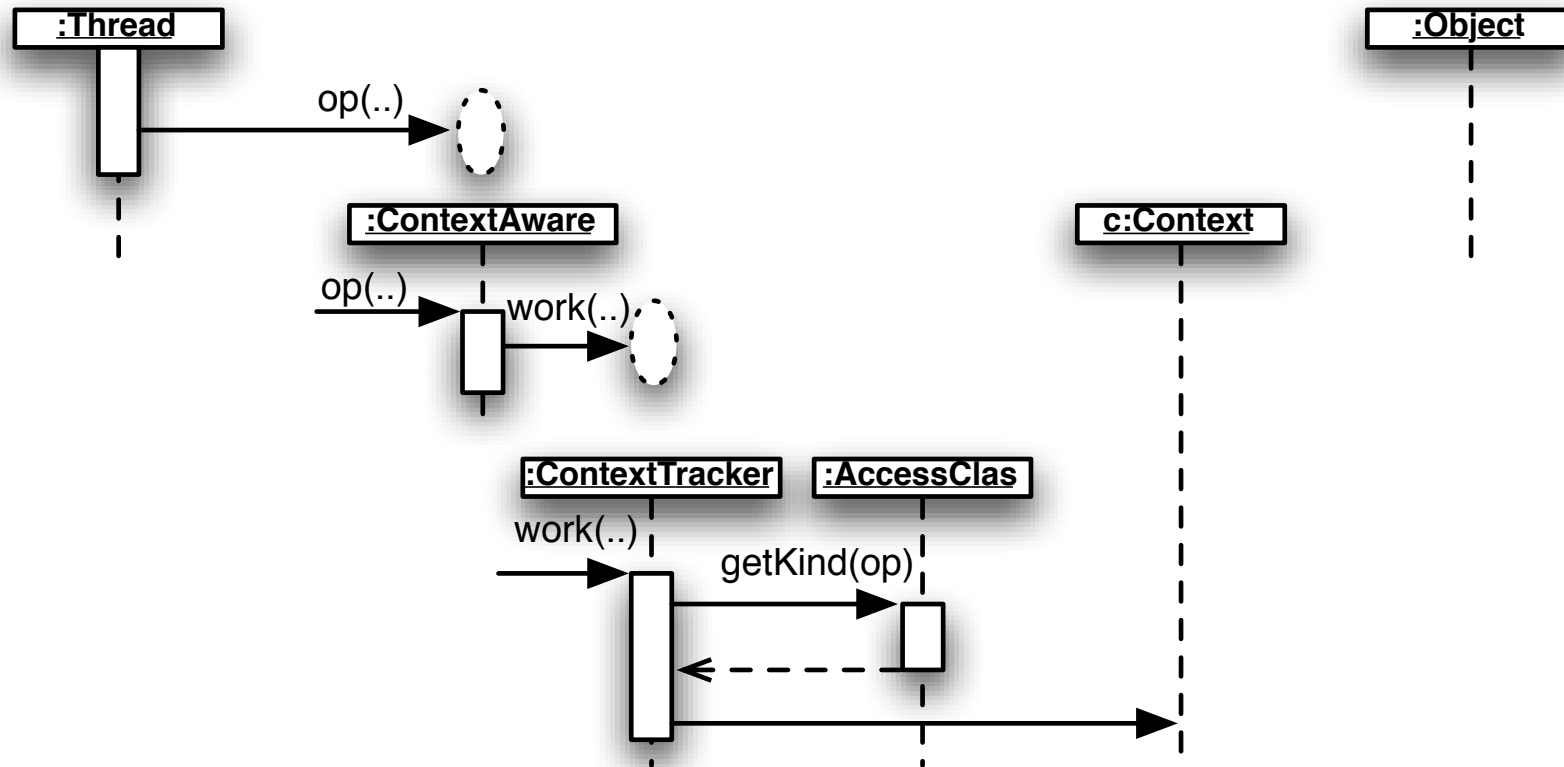
Invoking an Operation



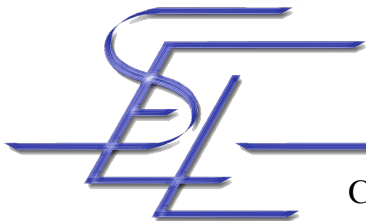
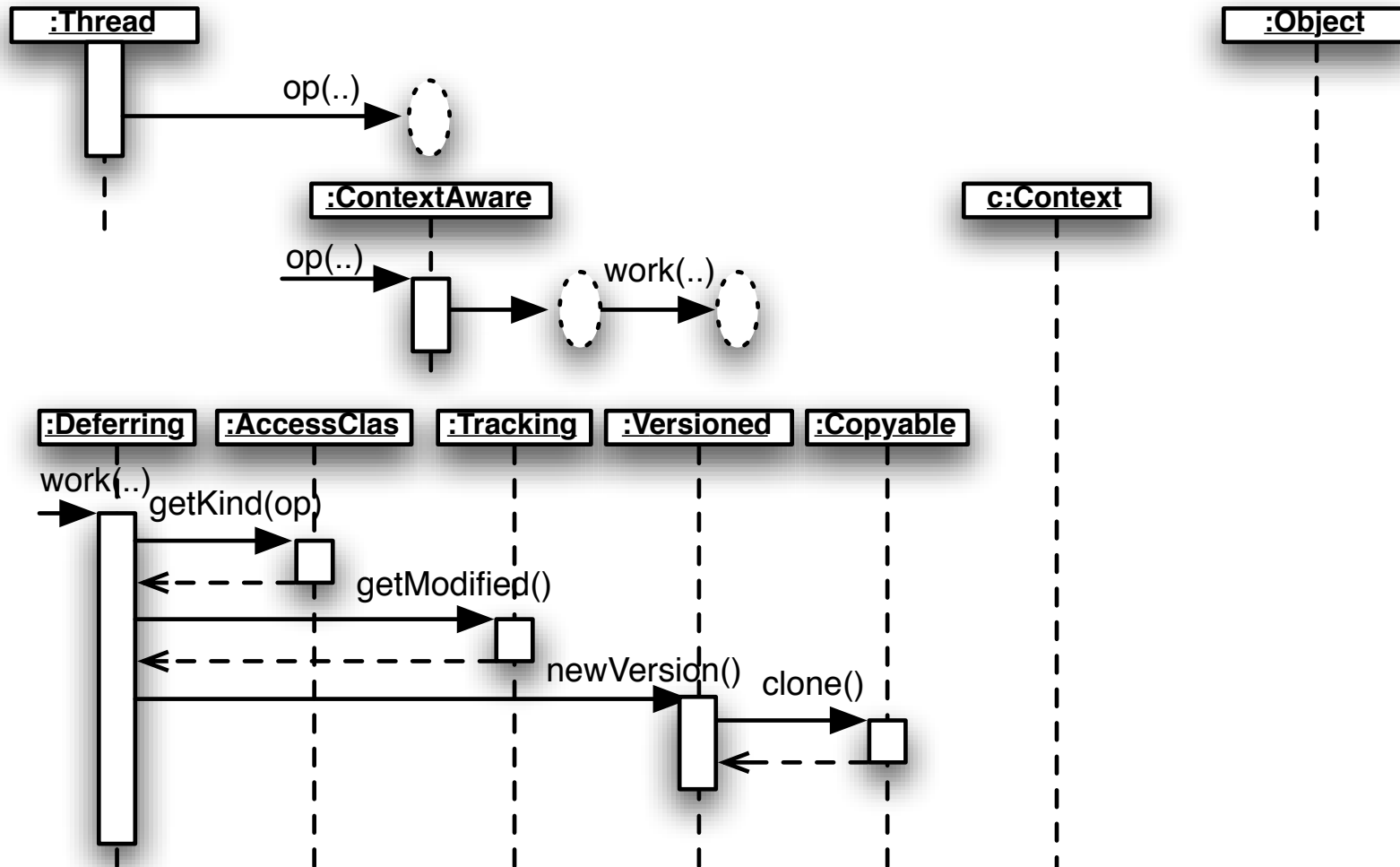
Invoking an Operation



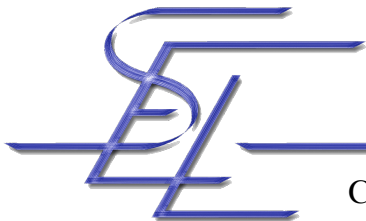
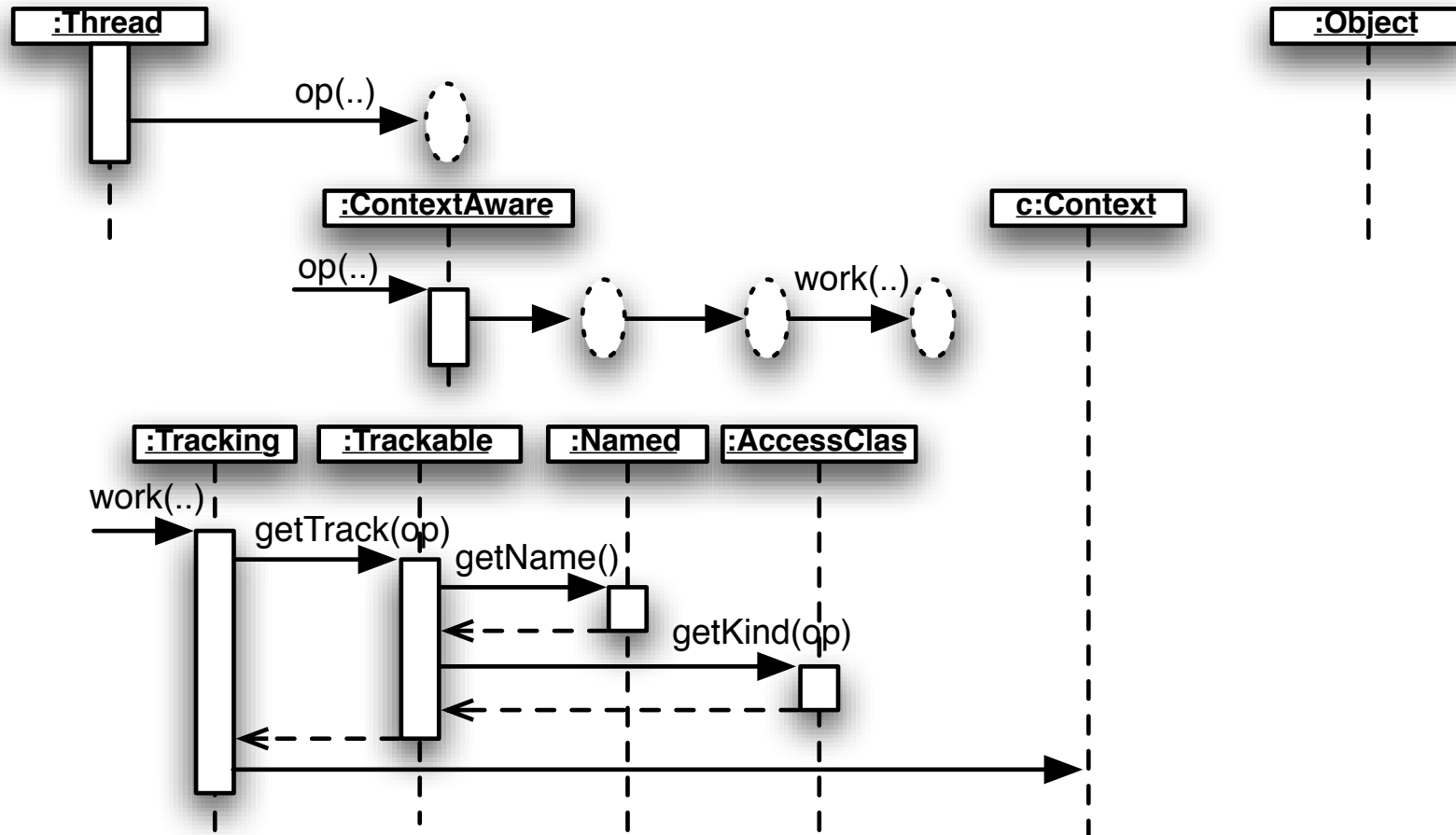
Invoking an Operation



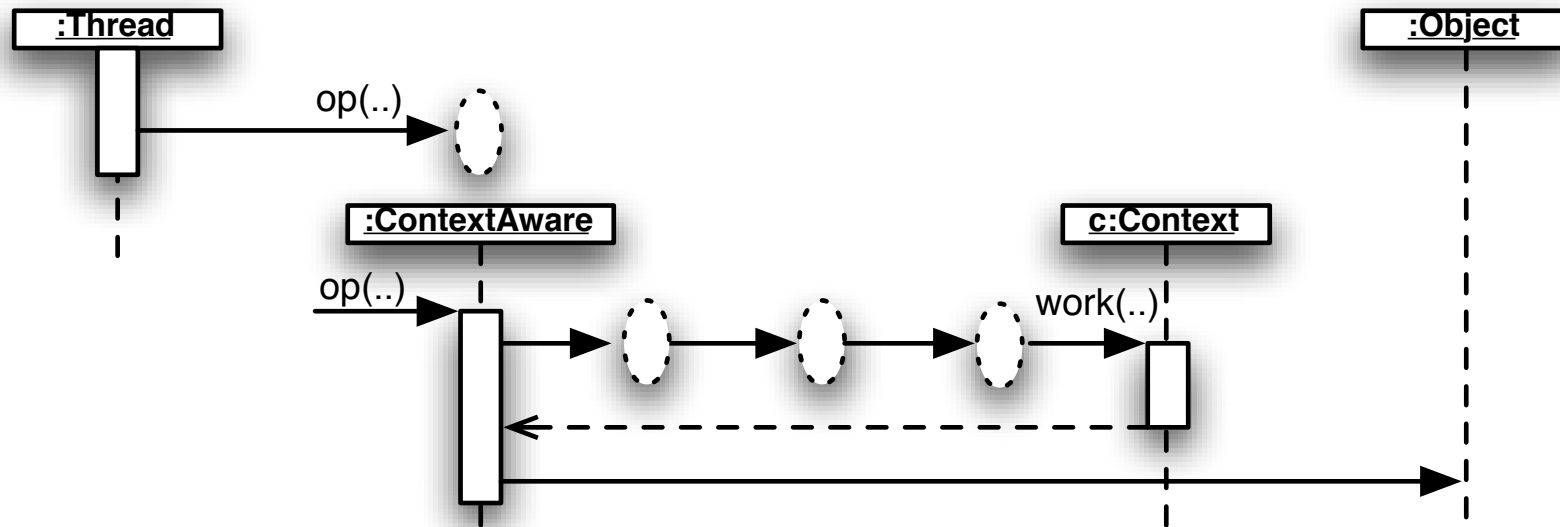
Invoking an Operation



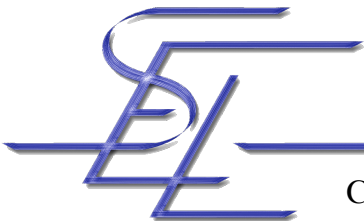
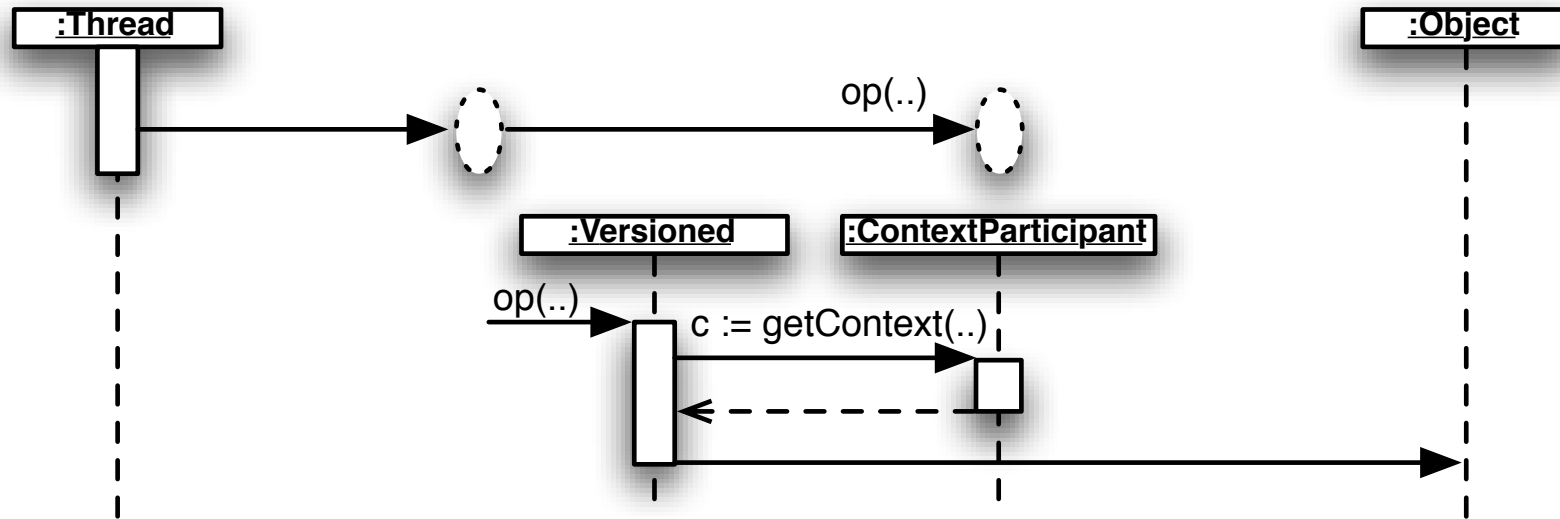
Invoking an Operation



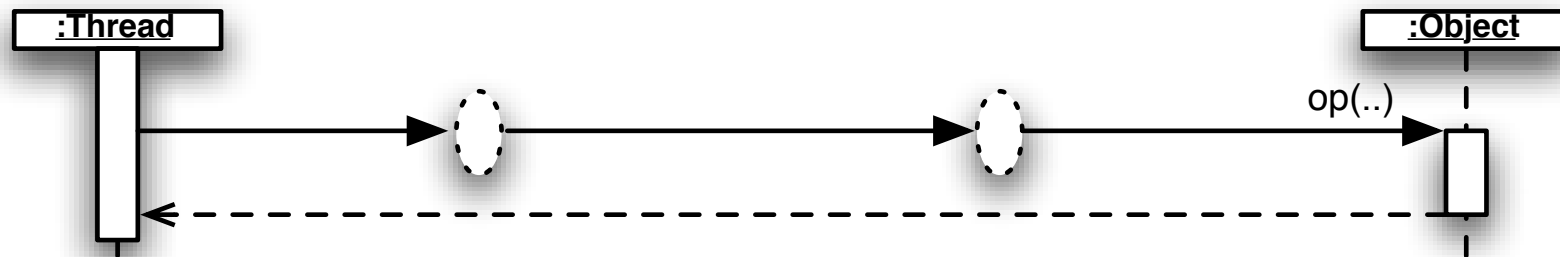
Invoking an Operation



Invoking an Operation



Invoking an Operation



- 5 Interceptions
 - 2 Interceptions of the actual method invocation
 - 3 Interceptions of the *work* operation of the context
- Collaboration of 11 Aspects



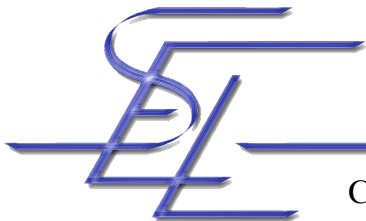
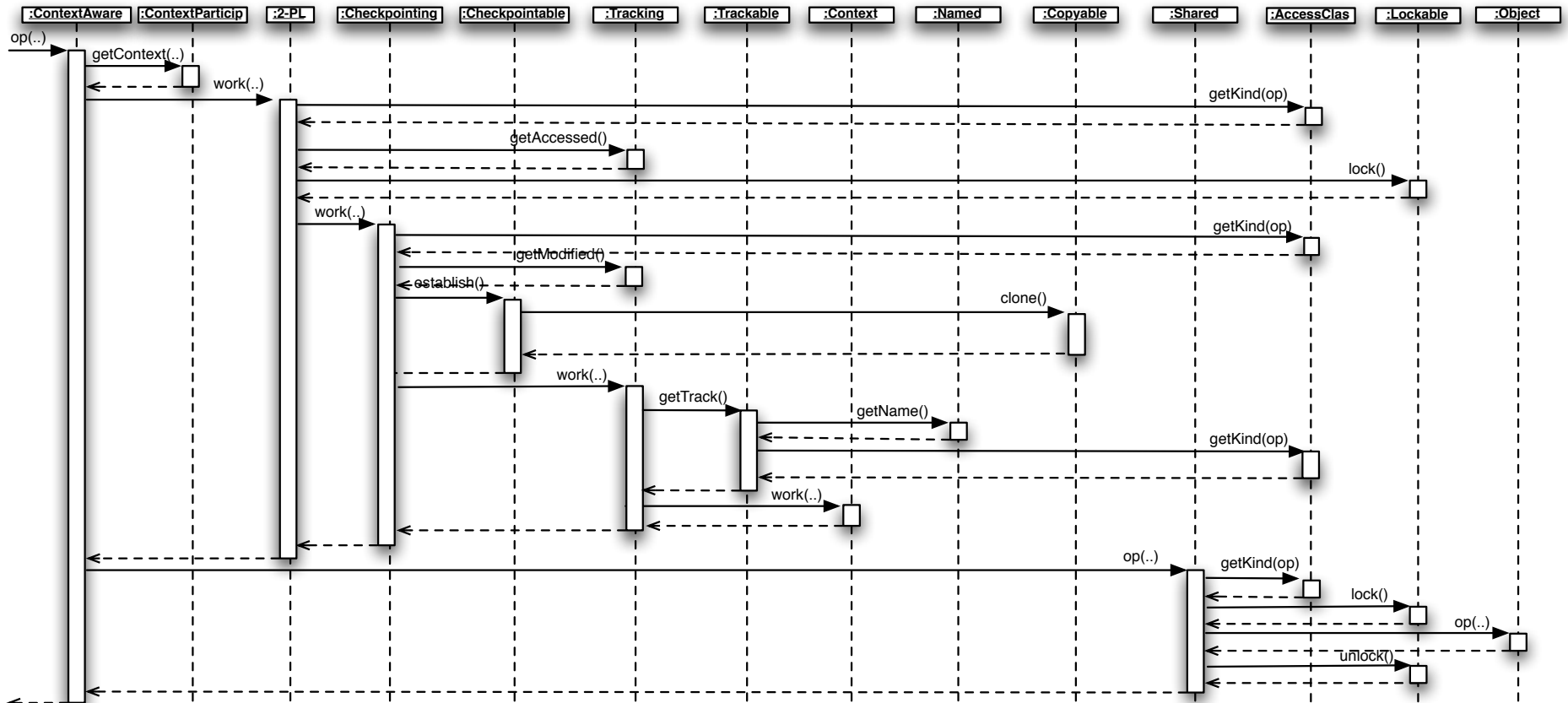
Example Configuration 2

Open Multithreaded Transactions, Pessimistic Lock-Based Concurrency Control, Inplace Update

- **Thread:** ContextParticipant, OutcomeAffecting, Collaborating
- **Context:** Tracking, 2-Phase-Locking, Checkpointing, OutcomeAware, Recovering, Nested, Collaborative, ExitSynchronizing, OutcomeVoted
- **Object:** ContextAware, AccessClassified, Named, Lockable, Trackable, Copyable, Checkpointable, Shared, Persistable



An Operation Invocation



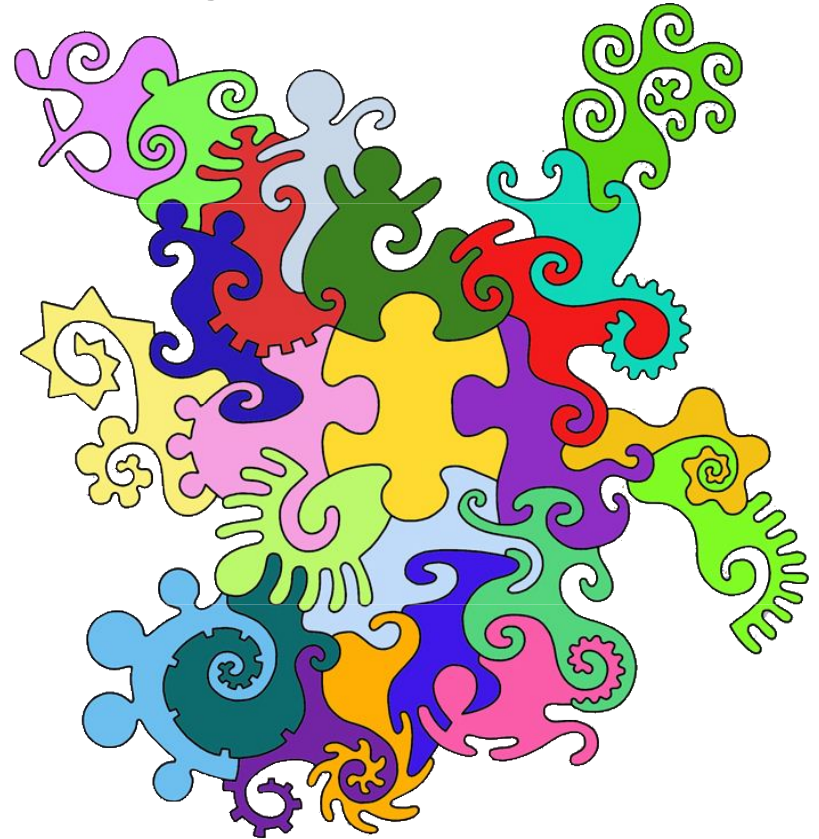
Aspect Frameworks and AO Languages

- Properties of the AspectOPTIMA Design

- Clear separation of concerns
- High reusability
- Complex aspect dependencies
- Complex aspect interference

- Essential Language Features

- Separate Aspect Binding
- Inter-Aspect Configurability
- Inter-Aspect Ordering
- Per-Object (per instance) Aspects
- Dynamic Aspects
- Thread-Aware Aspects



McGill

Case Study Target Audience

- Aspect-Orientation
 - AOSD Processes
 - AO Modeling Notations
 - AO Validation and Verification
 - AO Language Features
 - AO Programming Environments
- Fault Tolerance
 - Formalization of Fault Tolerance Models
 - Generation of Fault Tolerance Models



Validation by Implementation [1]

- AspectJ prototype implementation
 - Theoretical implementation in CaesarJ
- Encountered language limitations
 - Weak Aspect-To-Class Binding
 - Reflection/Superclass Execution Dilemma
 - No Explicit Inter-Aspect Configurability
 - No Per-Object Aspects
 - No Dynamic Aspects
- Work-arounds exist
- Language Improvements Suggested
- Initial performance evaluation

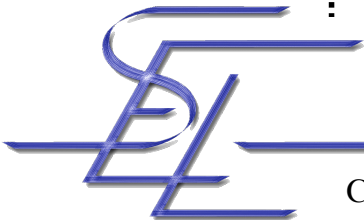
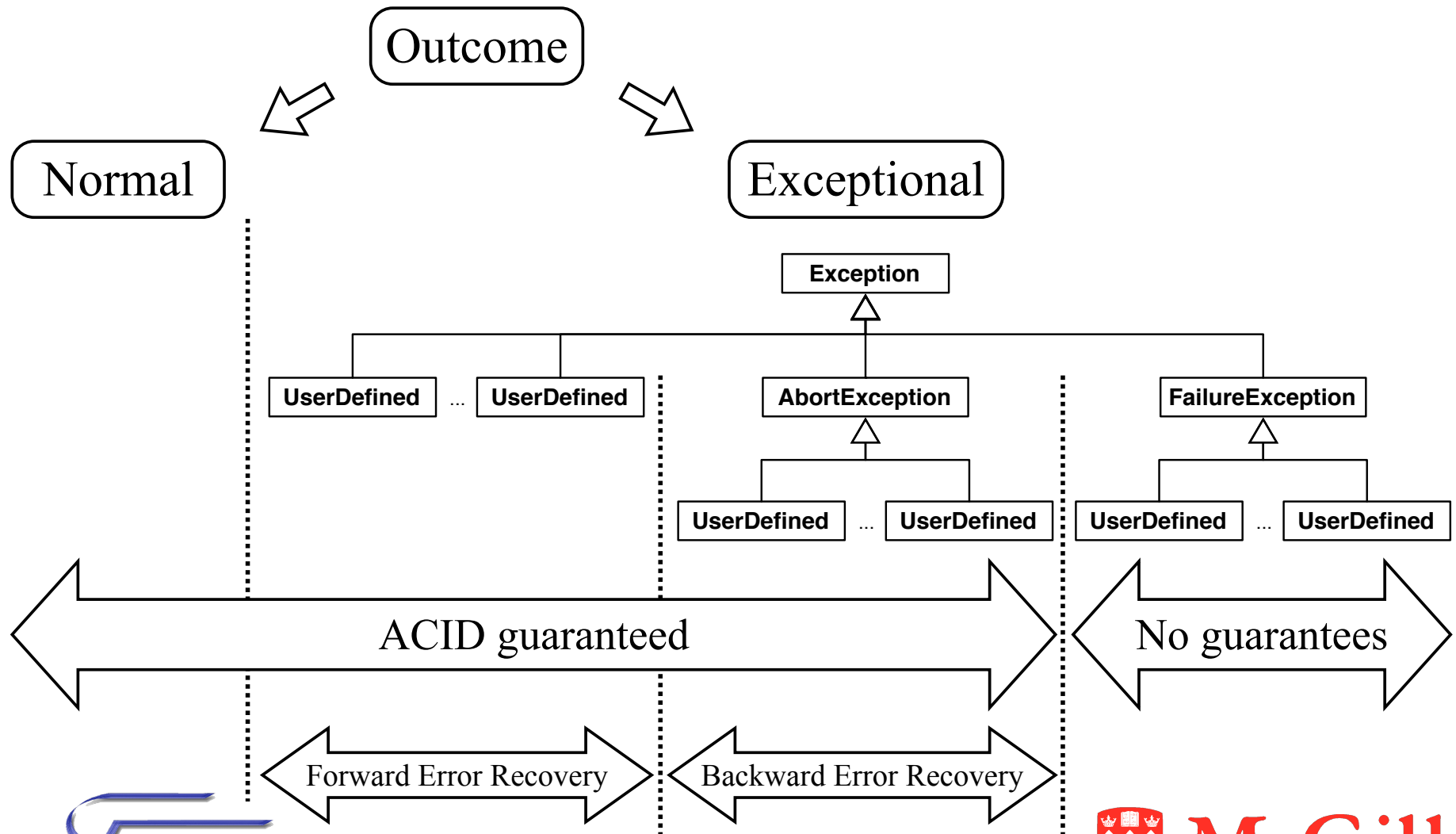


Future Work

- Define Benchmarks and Evaluate Different Compilers
- Implement AspectOPTIMA in other AO languages and compare language expressiveness
- Extend AspectOPTIMA
 - Concurrency Control and Recovery
 - Semantic concurrency control
 - Recovery based on intention lists
 - Provide weaker forms of Isolation, relaxed Atomicity
 - Transaction Models
 - Exception Handling
 - Inter-Transaction Dependencies (Look-Ahead Transactions, SAGAS)
 - Support Other Fault Tolerance Models (N-Version Programming)

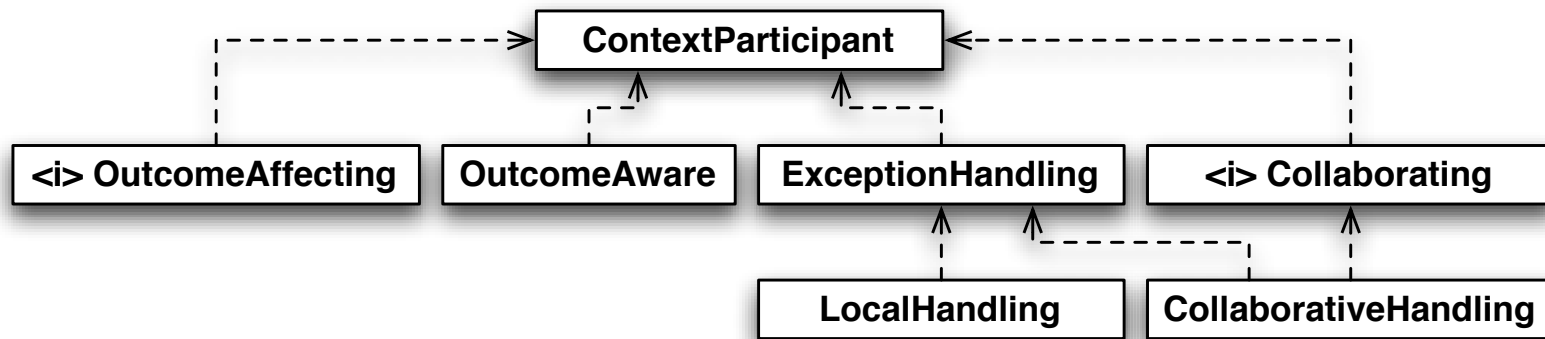


Notion of Outcome

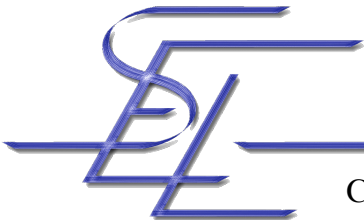


McGill

Exception Handling Participants



- *ExceptionHandling*: Capable of handling internal exceptions
- *Local Handling*: First attempts to handle internal exceptions locally
- *CollaborativeHandling*: Participates in collaborative handling of internal resolved exceptions
- *OutcomeAware*: Is notified of external exception



Exception Handling Objects and Context

- Context
 - BackwardRecovering and ForwardRecovering
 - ExceptionResolving
- Objects
 - Self-Checking

Exact functionality still to be determined



AspectOPTIMA References

Aspect-Orientation

- [1] J. Kienzle, Ekwa Duala-Ekoko and S. G lineau, “AspectOPTIMA:A Case Study on Aspect Dependencies and Interactions”, Transactions on Aspect-Oriented Software Development, in press.
- [2] J. Kienzle and S. G lineau, “ AO Challenge: Implementing the ACID Properties for Transactional Objects”, in Proceedings of the 5th International Conference on Aspect-Oriented Software Development - AOSD 2006, March 20 - 24, 2006, pp. 202 – 213, ACM Press, March 2006.
- [3] J. Kienzle and R. Guerraoui, “ AOP - Does It Make Sense? The Case of Concurrency and Failures”, in 16th European Conference on Object-Oriented Programming (ECOOP’2002), Lecture Notes in Computer Science 2374, (Malaga, Spain), pp. 37 – 61, Springer Verlag, 2002.

Open Multithreaded Transactions

- [4] M. Monod, J. Kienzle, and A. Romanovsky, “Looking Ahead in Open Multithreaded Transactions”, in Proceedings of the 9th International Symposium on Object and Component-Oriented Real-Time Distributed Computing, pp. 53 – 63, IEEE Press, April 2006.
- [5] J. Kienzle, Open Multithreaded Transactions — A Transaction Model for Concurrent Object-Oriented Programming. Kluwer Academic Publishers, 2003.

