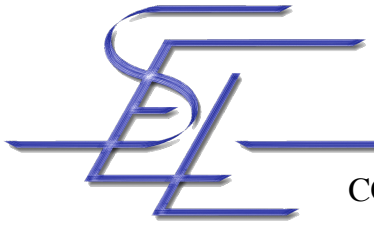
AspectOPTIMA
Part 2

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

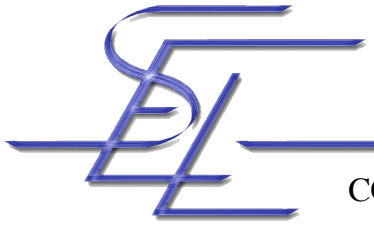
With Contributions From:
Samuel Gélineau, Ekwa Duala-Ekoko,
Güven Bölükbaşi, Barbara Gallina



McGill

Reusable Aspect Models (RAM)

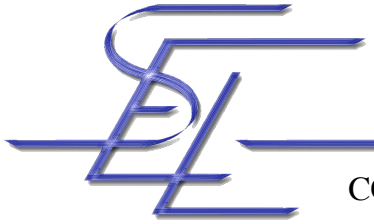
- Aspect-oriented modeling approach integrating class diagrams, state diagrams and sequence diagrams
  - Aspect models can specify structure and behavior
  - Based on existing class diagram weaving technology [France et al.]
  - Based on existing sequence diagram weaving technology [Klein et al.]
  -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



McGill

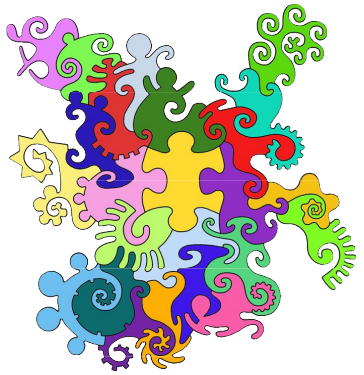
Multi-View Modeling

- 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

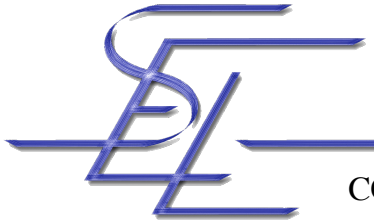


McGill

Aspect-Oriented Modeling



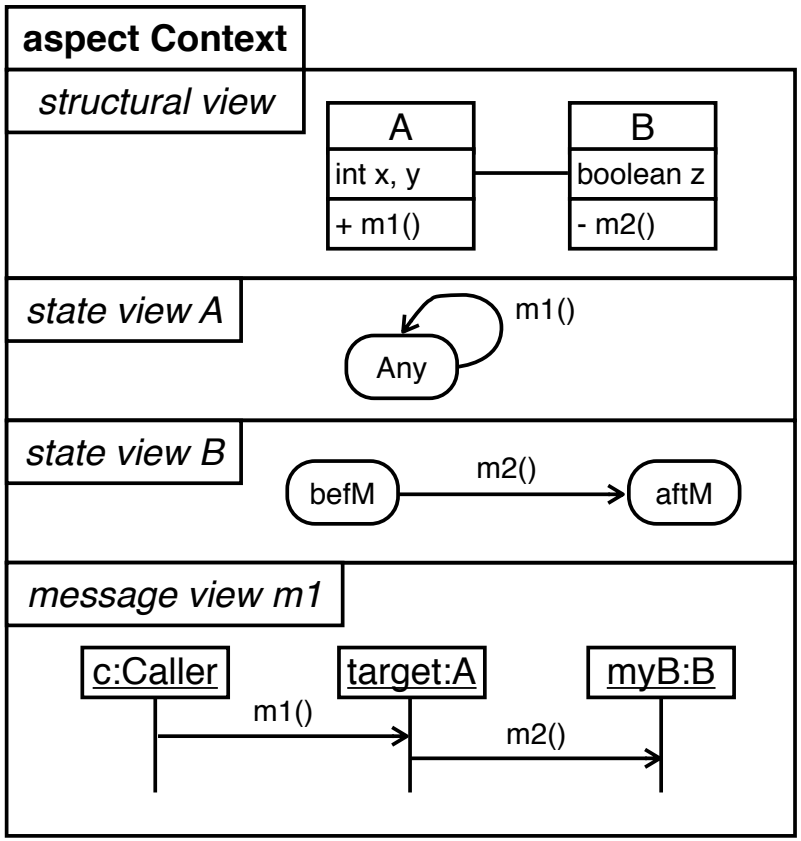
- Aspect-oriented Modeling
  - Current weaving techniques based on *one* modeling notation only
  - Can address scalability within one modeling notation
- Aspect-oriented Multi-View Modeling
  - Compatible composition techniques and weaving algorithms must be defined to ensure that the consistency of individual aspects is preserved in woven model

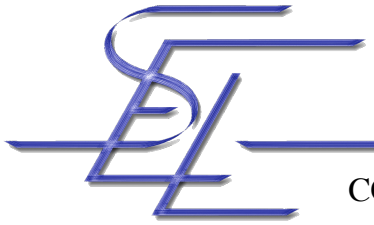


McGill

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

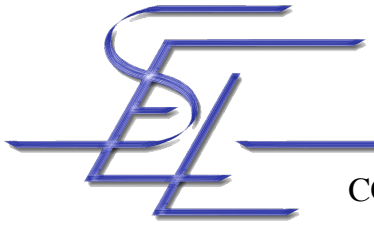




McGill

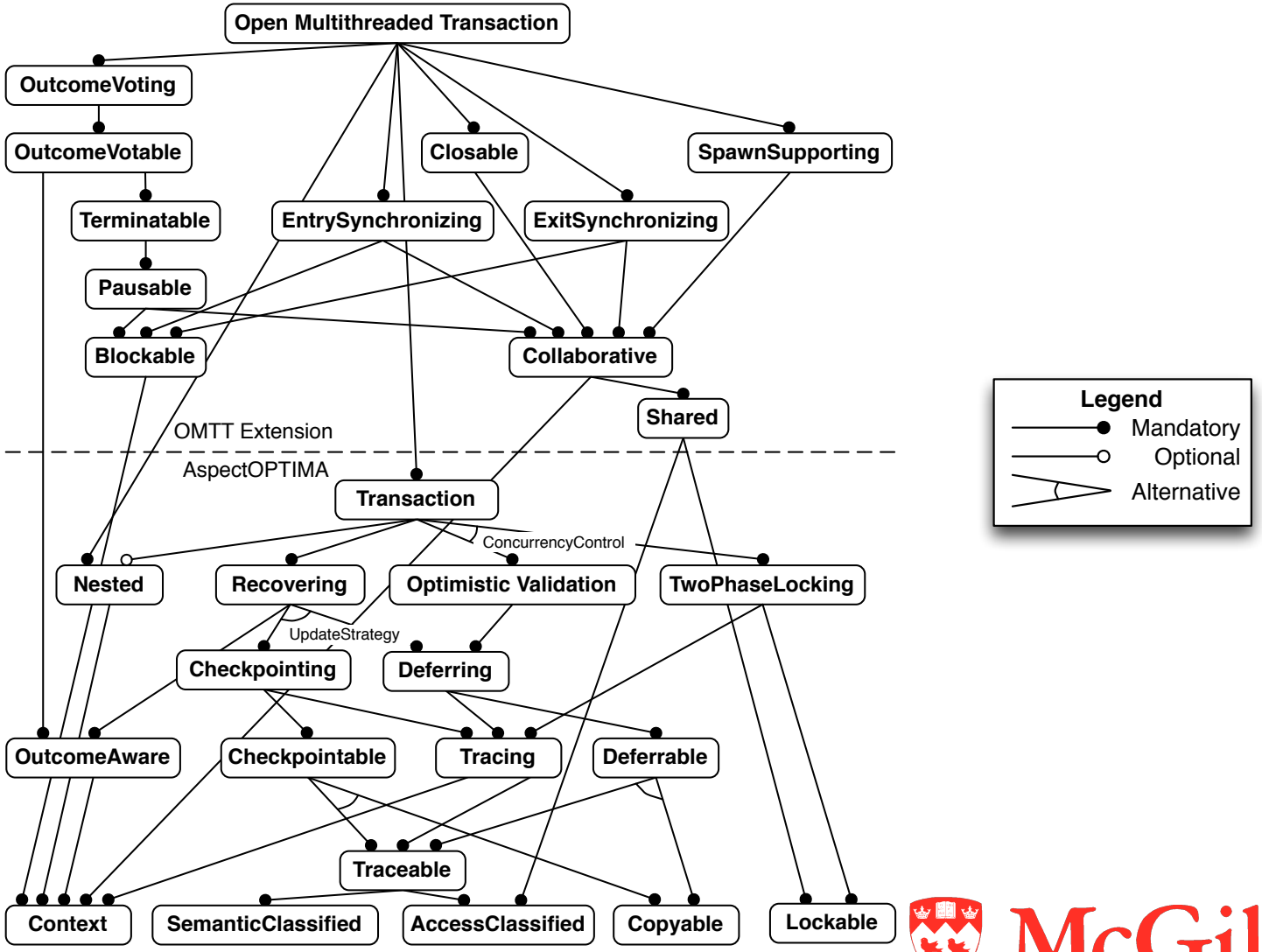
Case Study: AspectOPTIMA

- 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 29 aspects are modelled in RAM
  - Individually reusable
  - Complex aspect dependencies and interactions



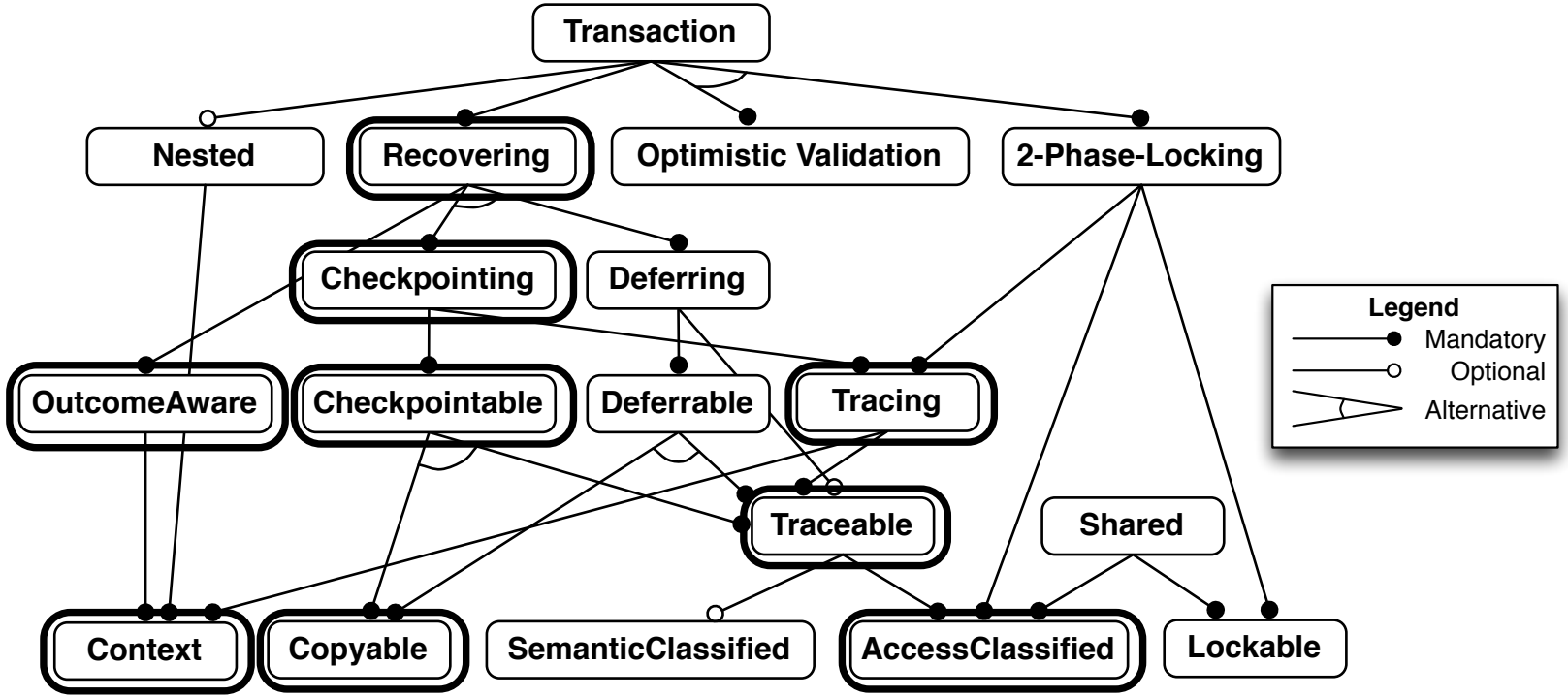
McGill

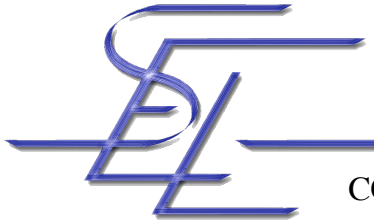
AspectOPTIMA Feature Diagram



AspectOPTIMA Feature Diagram

- Feature diagram allows users of the framework to choose desired transaction configuration

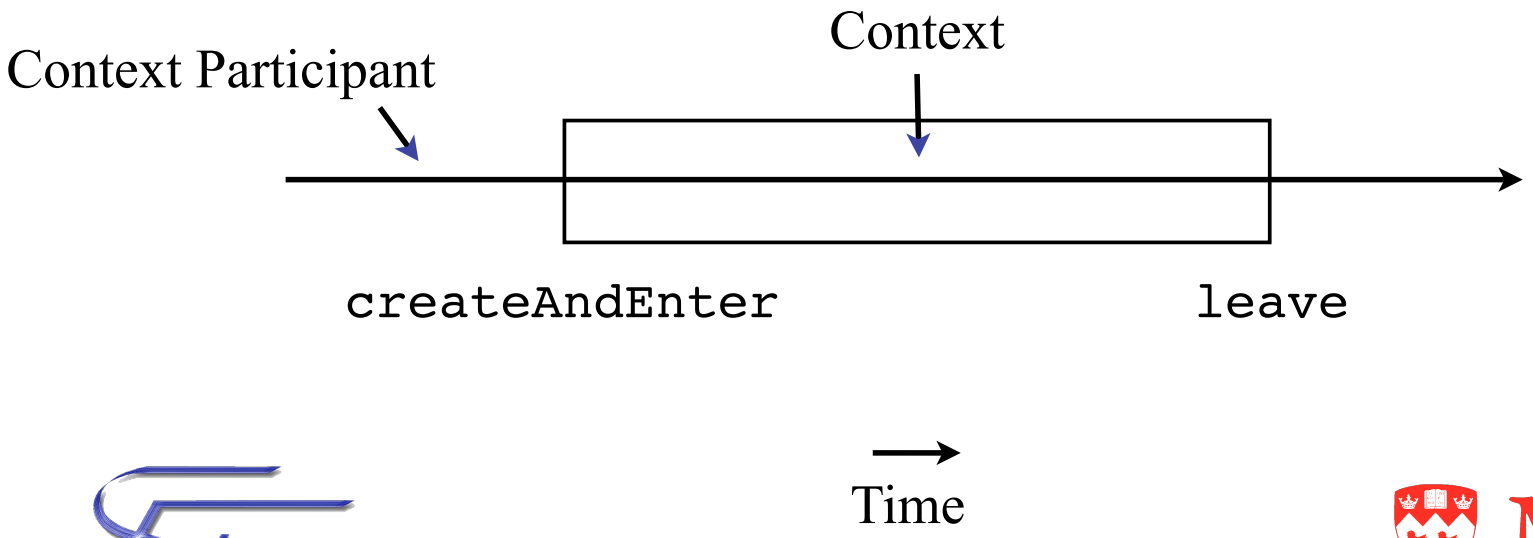


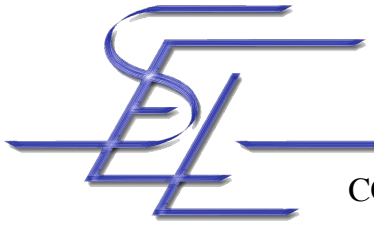


McGill

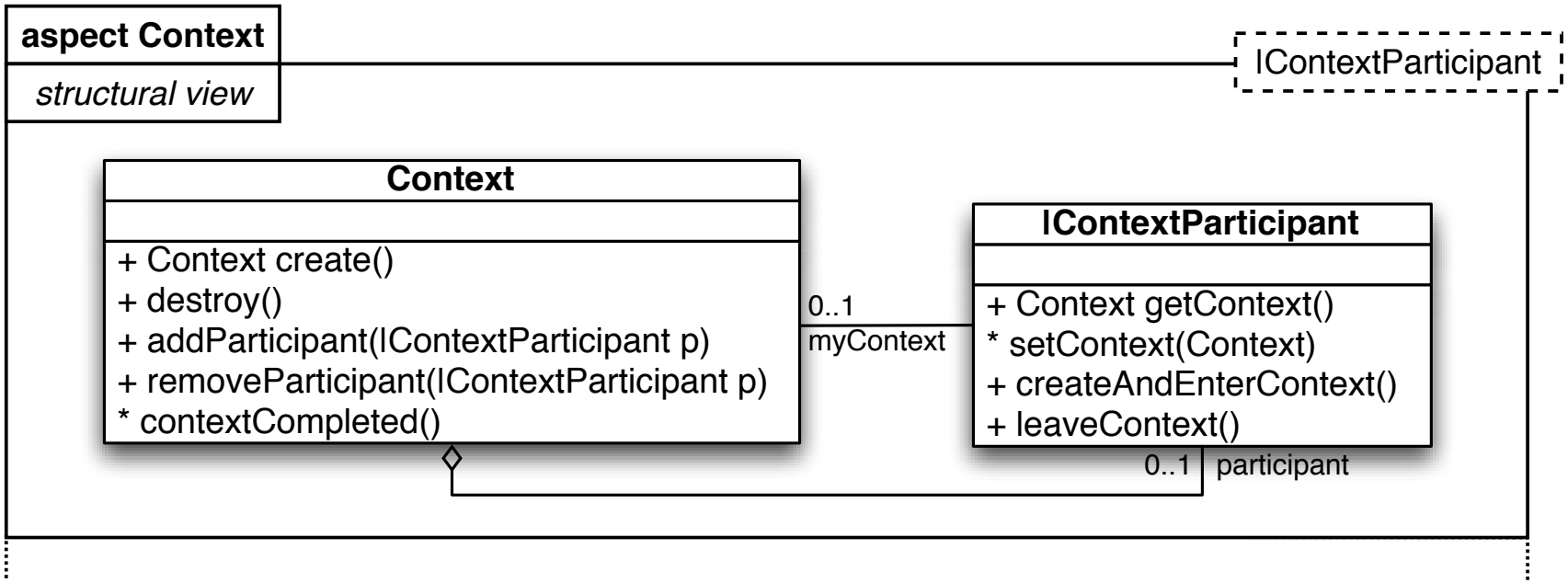
The *Context* Aspect

- 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

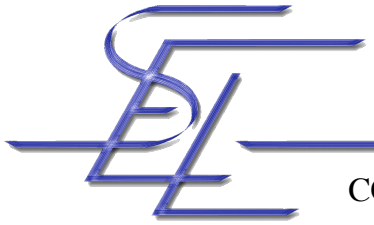




1st View: Structural View

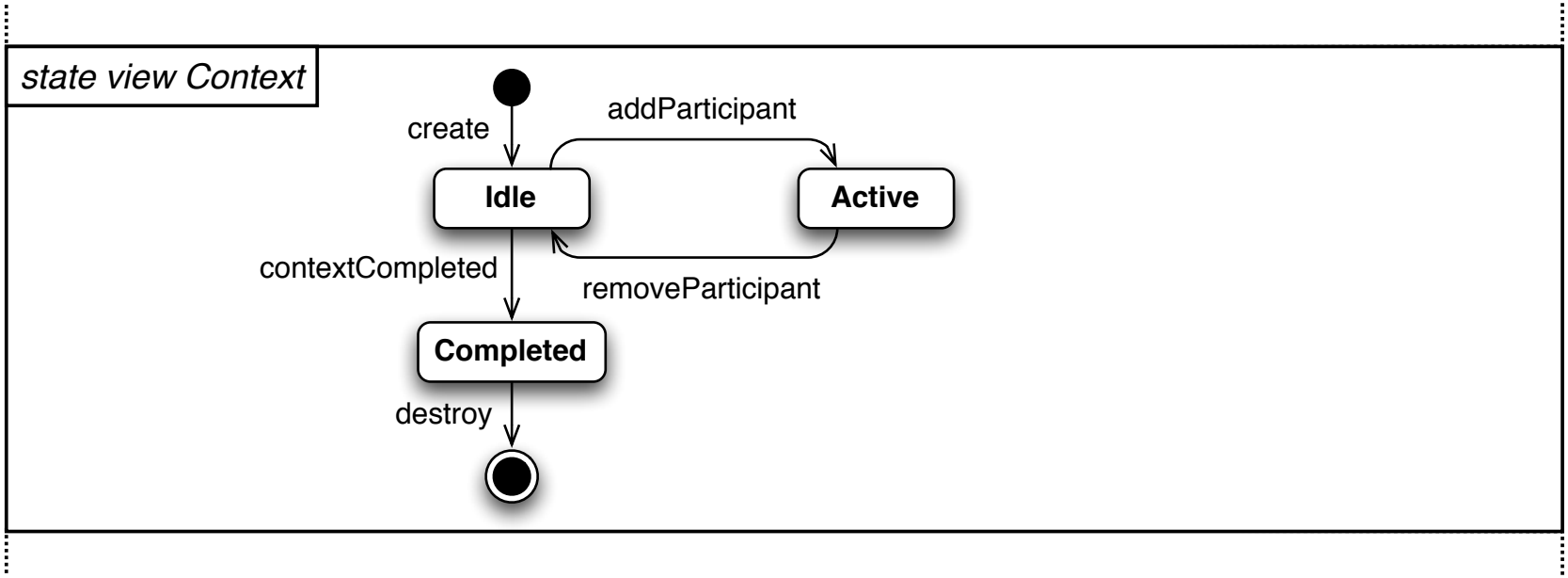


- One structural view
  - Classes (partial or complete) with methods, associations between classes
- Partial classes are mandatory instantiation parameters of the model, shown as UML template parameters to the view

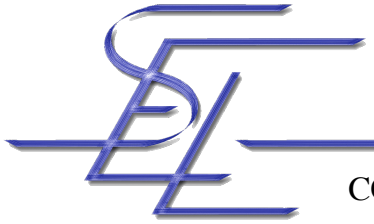


McGill

2nd View: State View (1)

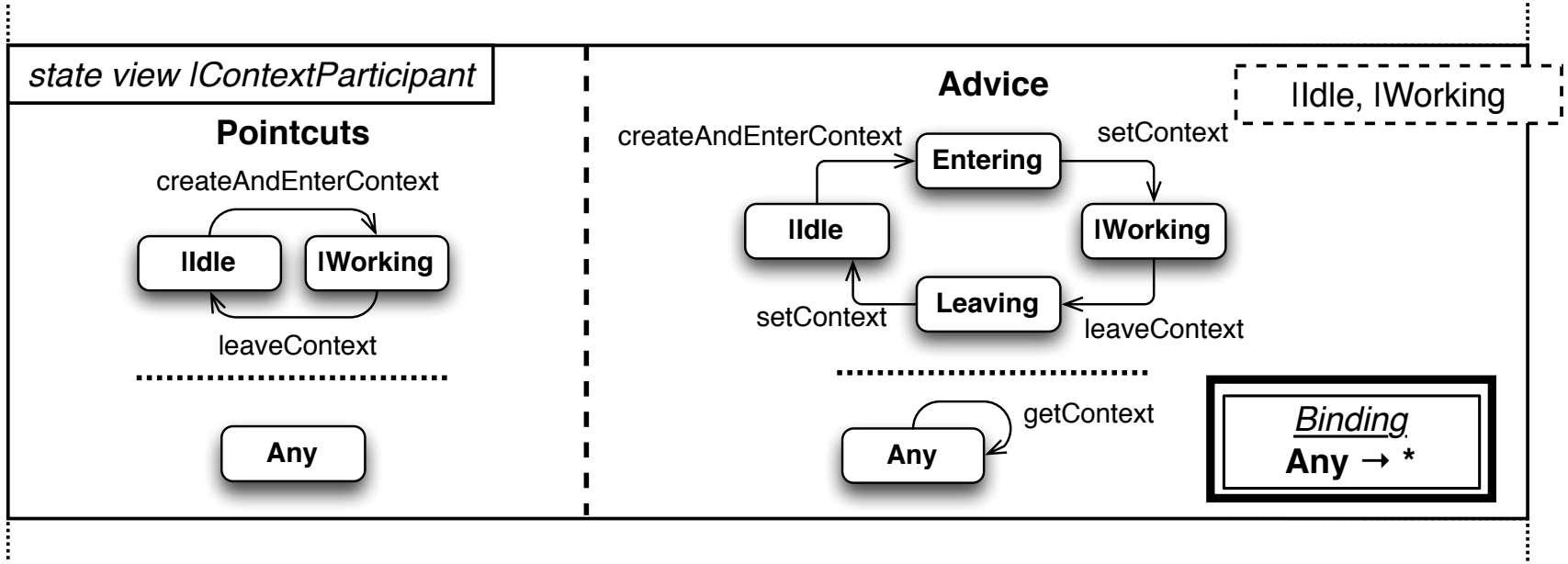


- One state view for each class (partial or complete) defined in structural view
- Describes interaction protocol of instances
  - Each method must have at least one corresponding transition

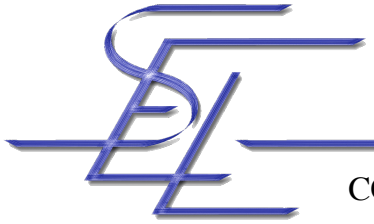


McGill

2nd View: State View (2)

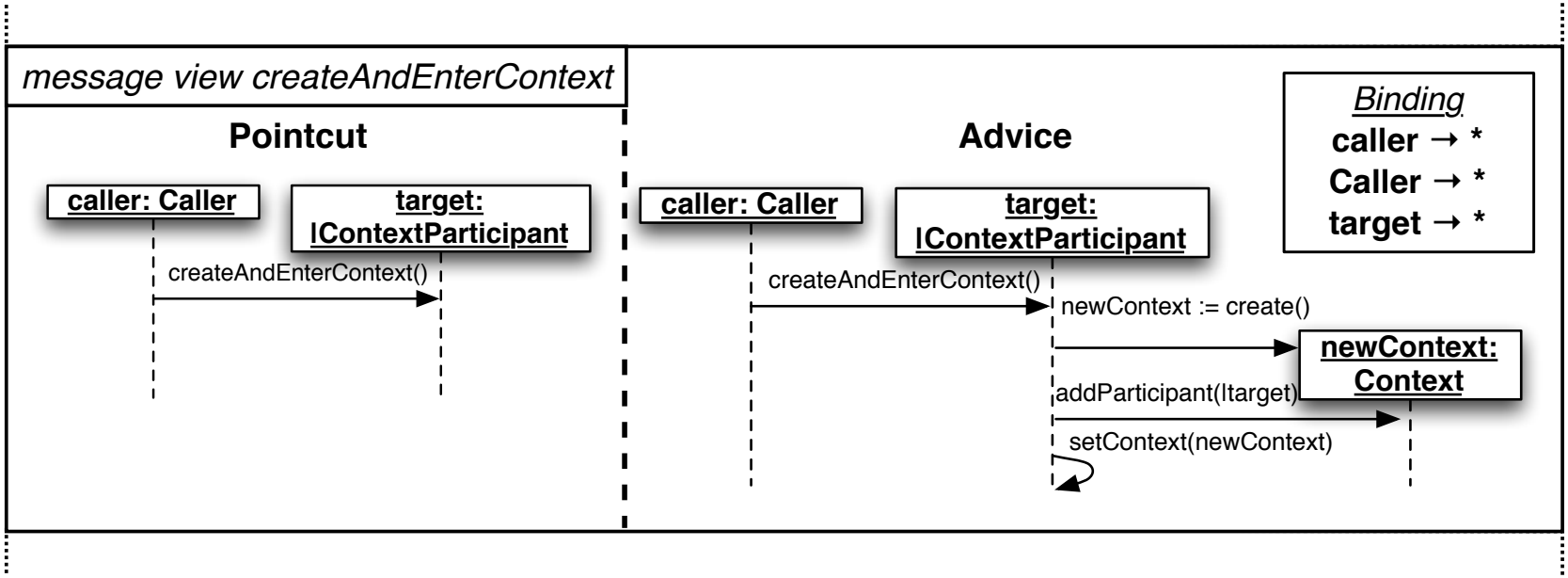


- State views of partial classes declare the behavior they assume (pointcut), and how they extend the behavior (advice)
- Placeholder states are mandatory instantiation parameters, and shown as UML template parameters to the view

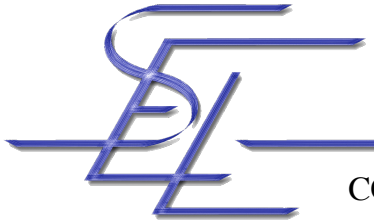


McGill

3rd View: Message View

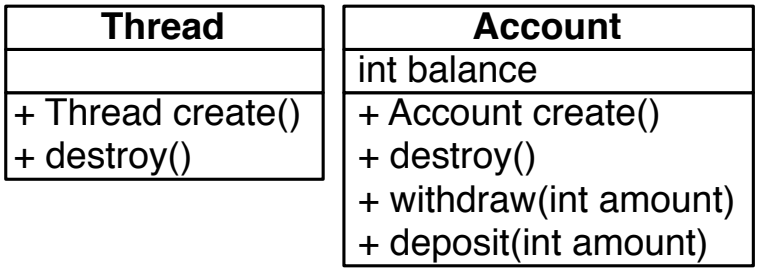


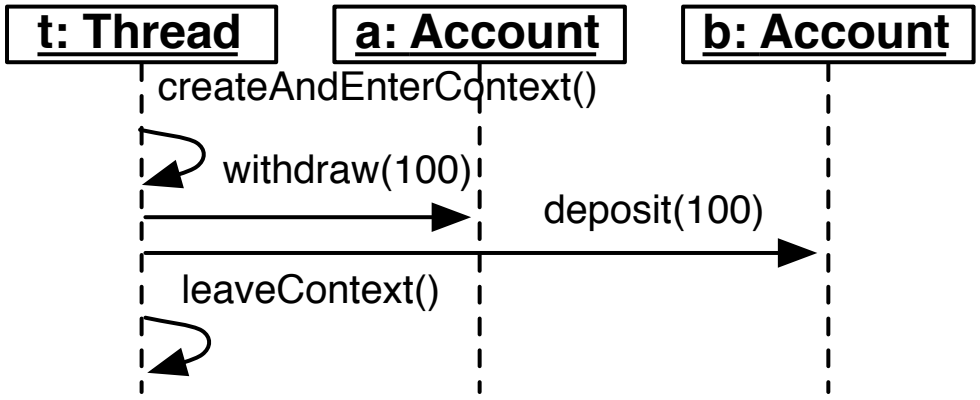
- At least one message view for each public method declared in the structural view
- Describes message exchange between objects of the aspect that implements desired behavior

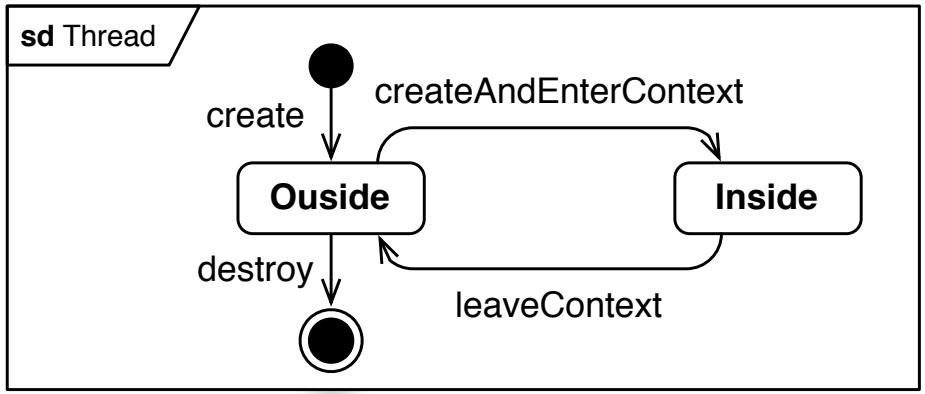


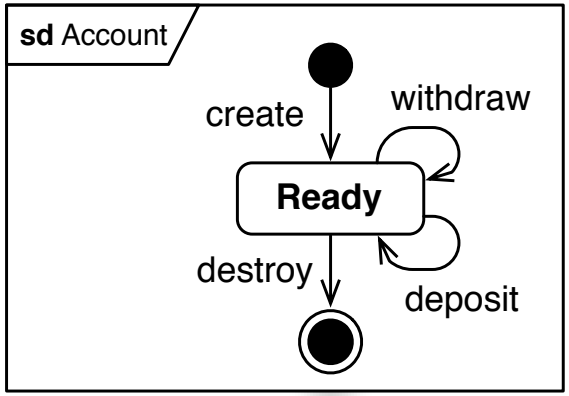
McGill

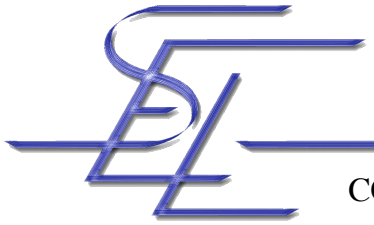
Base Model: A Banking Application











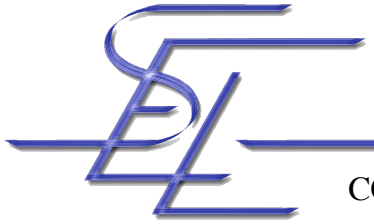
McGill

Reuse by Instantiation / Binding

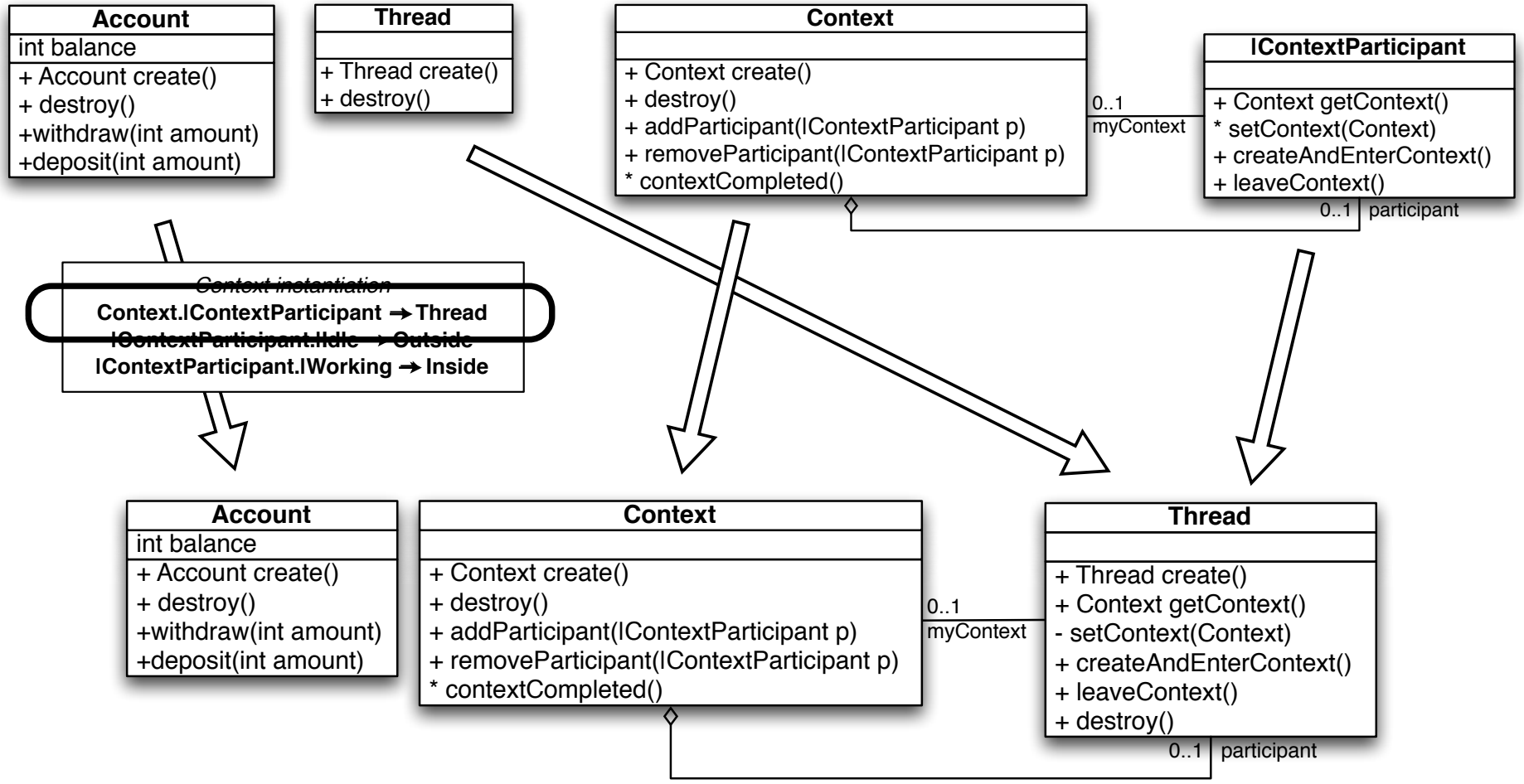
- To reuse the aspect *Context*, the base model must instantiate it explicitly
  - An instantiation directive maps all mandatory instantiation parameters to base model elements

Context instantiation
Context.IContextParticipant ➔ Thread
IContextParticipant.IIdle ➔ Outside
IContextParticipant.IWorking ➔ Inside

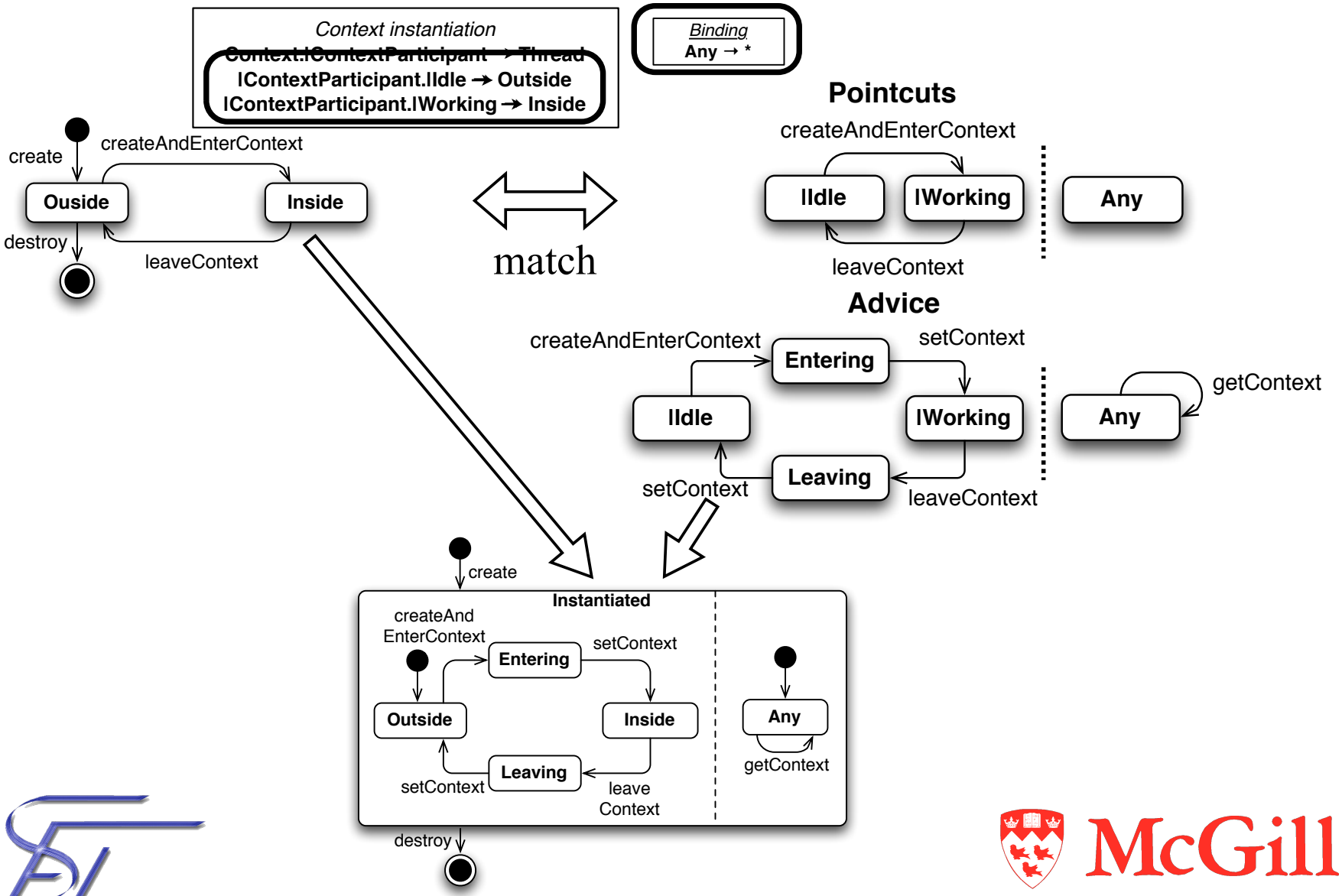
  - When a model A instantiates an aspect model B, A reuses the structure and behavior modeled by B.
  - Reuse of B might require A to extend the structure and behavior of B, in which case A must additionally define a binding to B



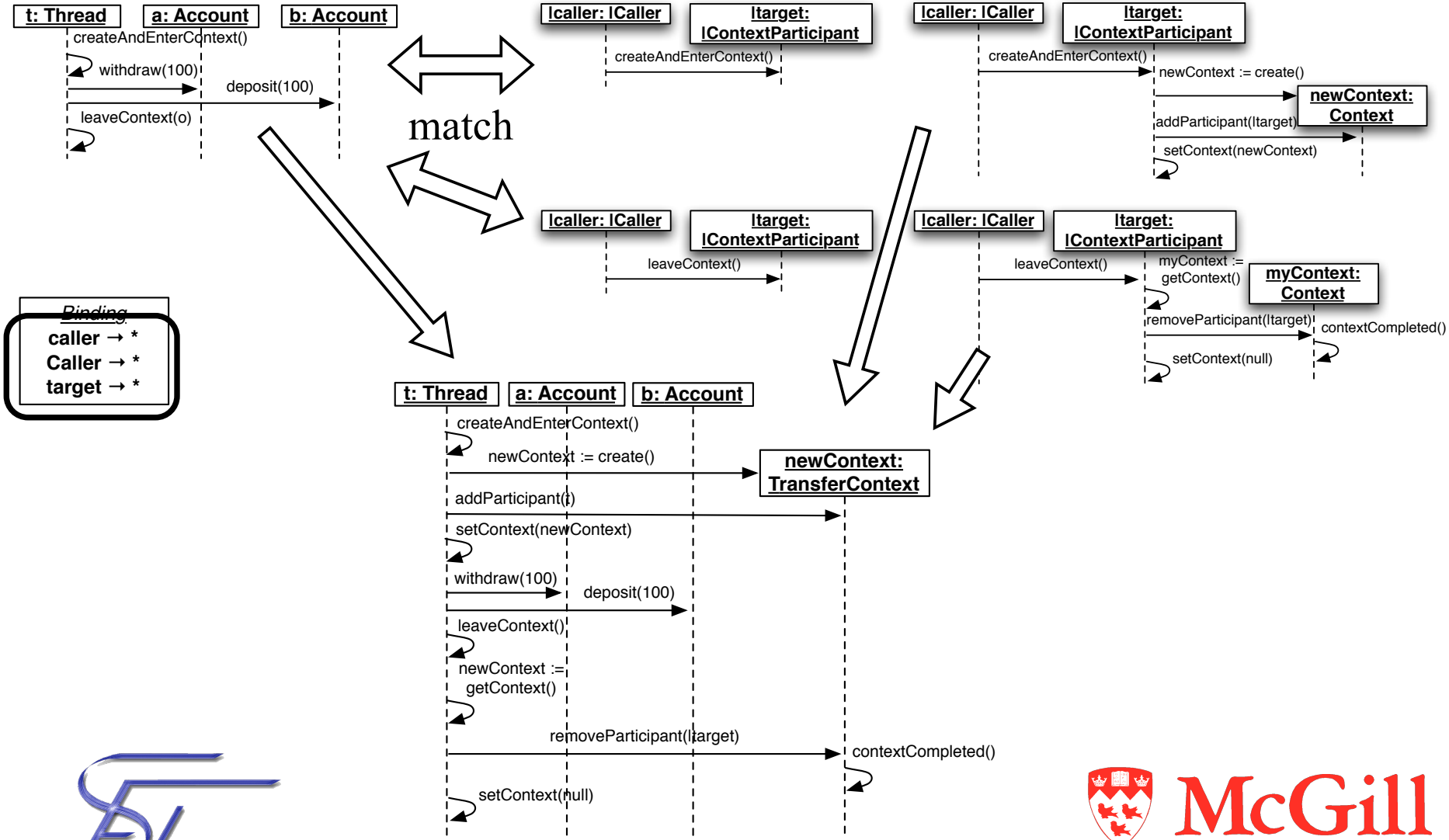
Structural View Weaving



State View Weaving

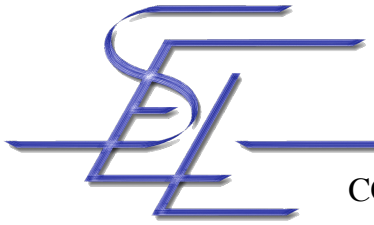


Message View Weaving

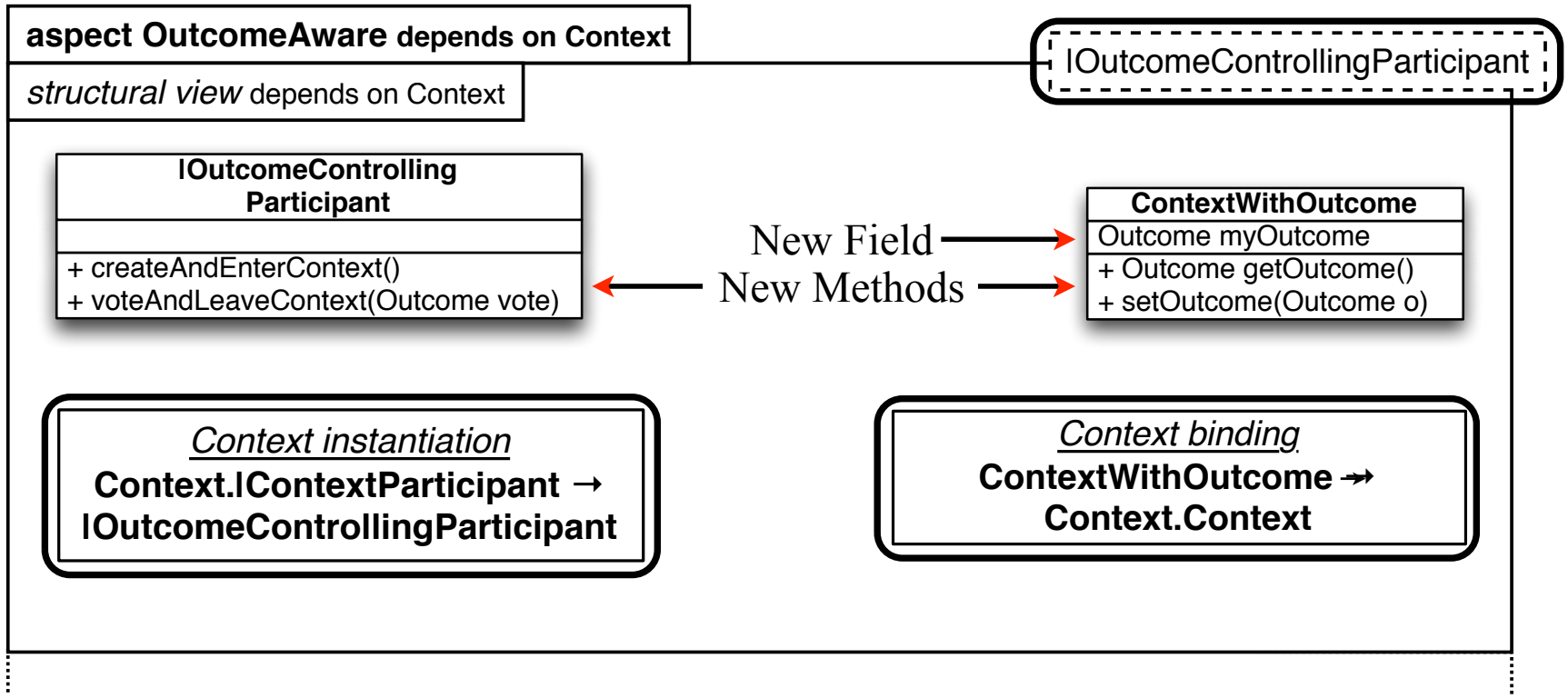


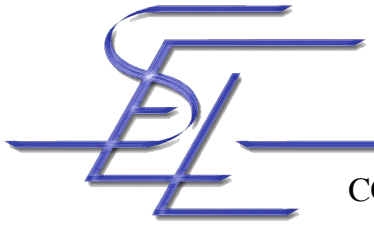
Aspect Dependencies

- 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



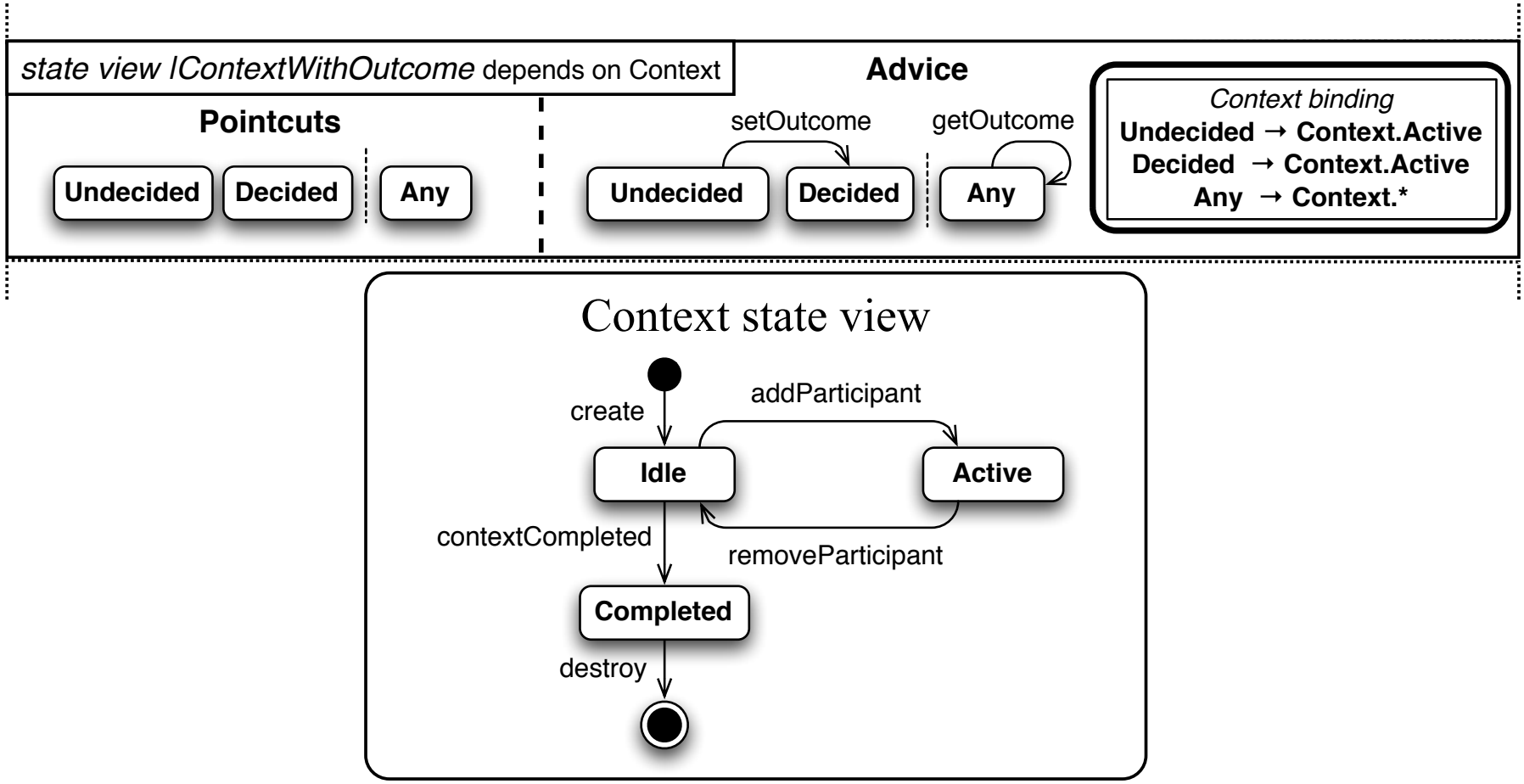
OutcomeAware Structural View



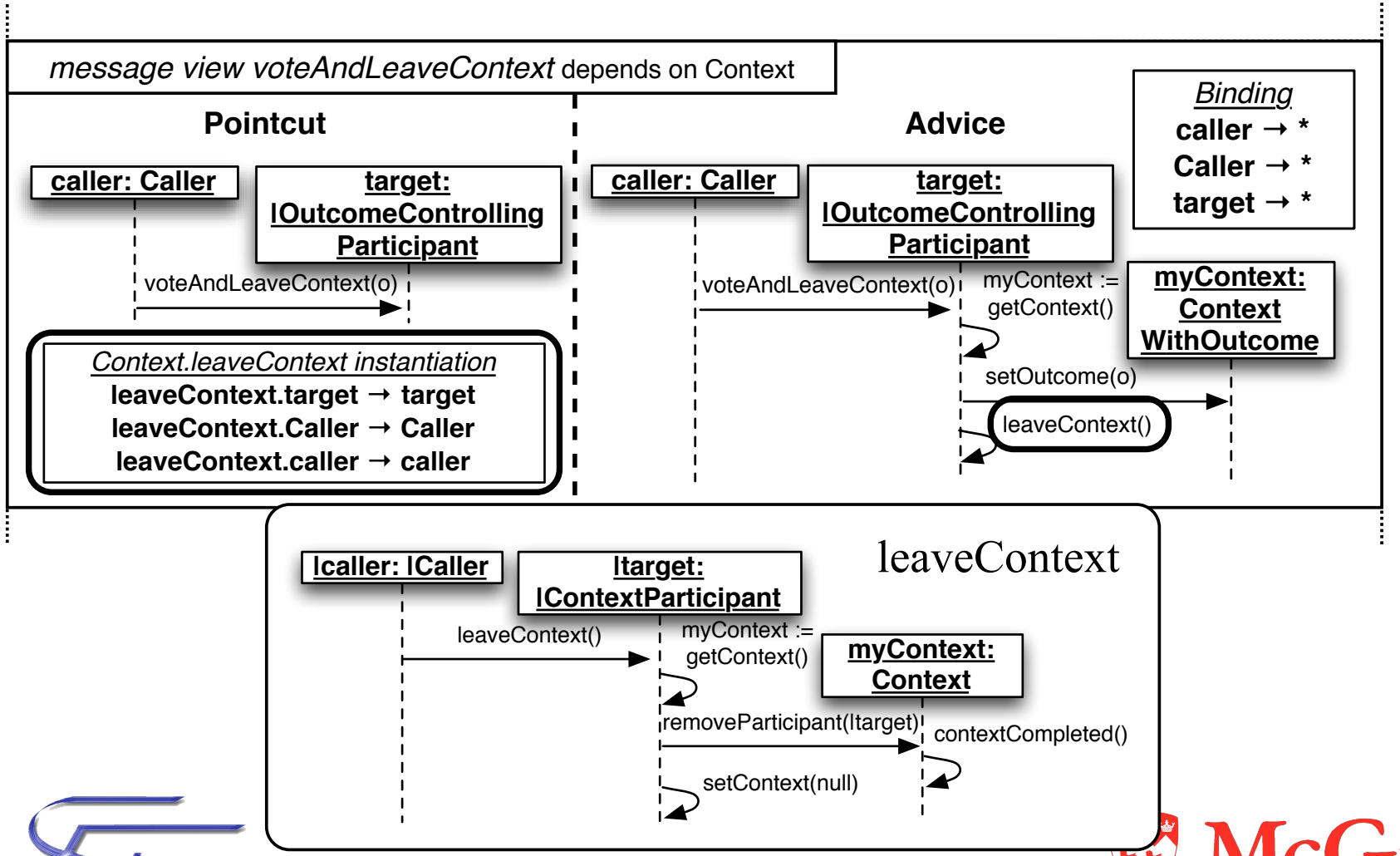


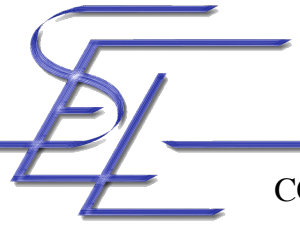
McGill

OutcomeAware State Views



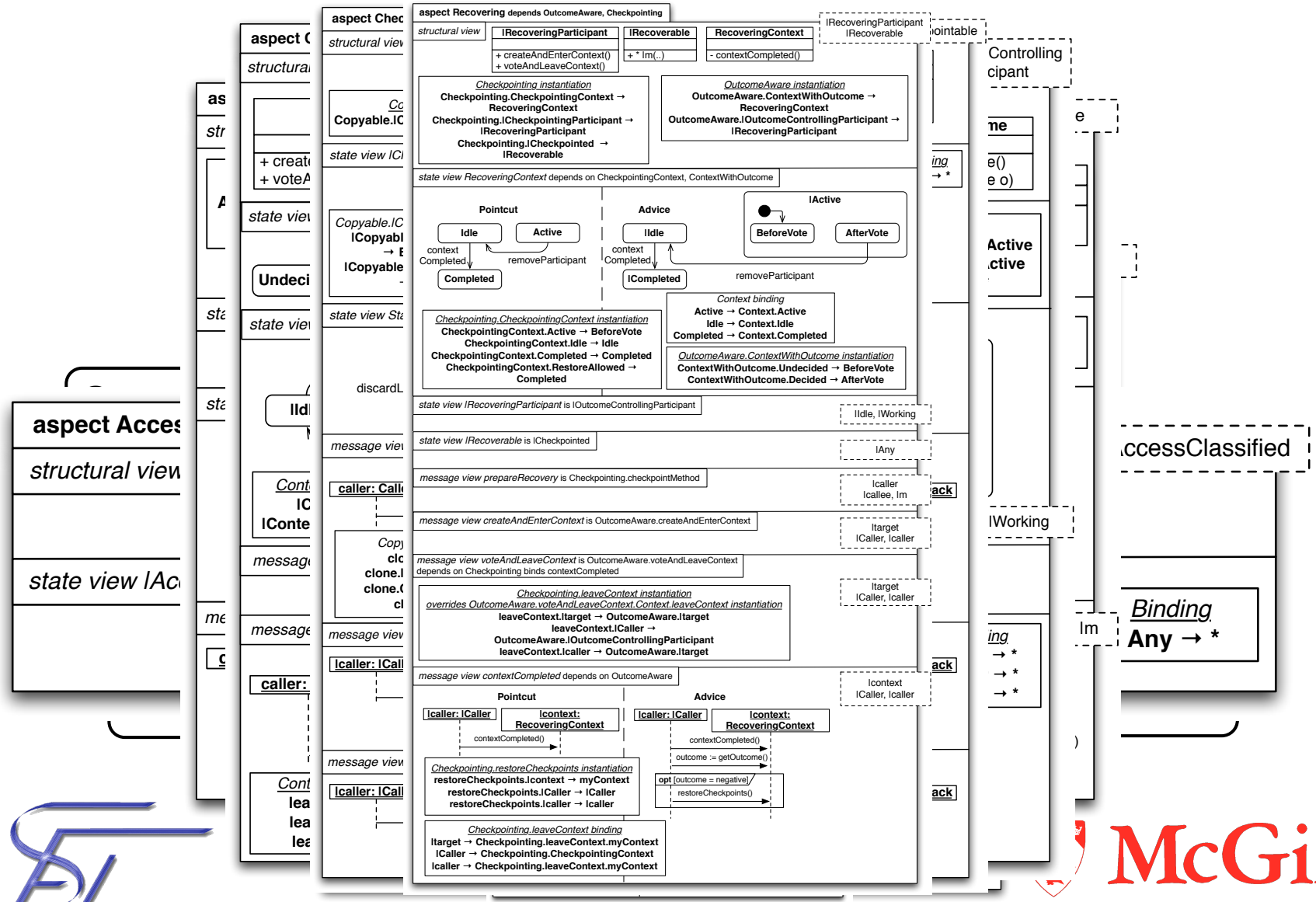
OutcomeAware Message View



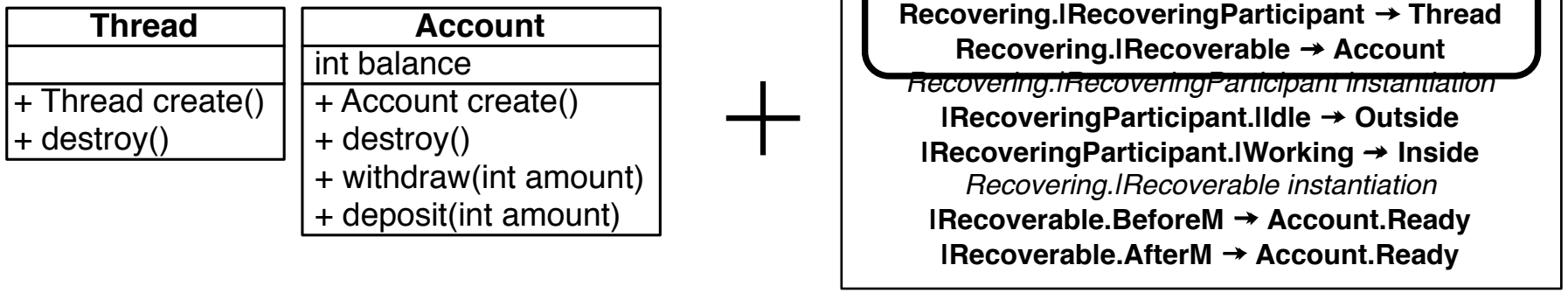


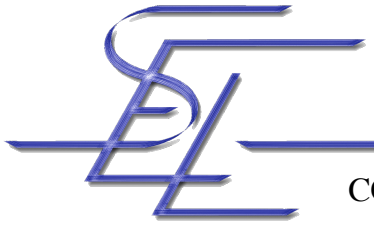
McGill

Complex Example: Recovering



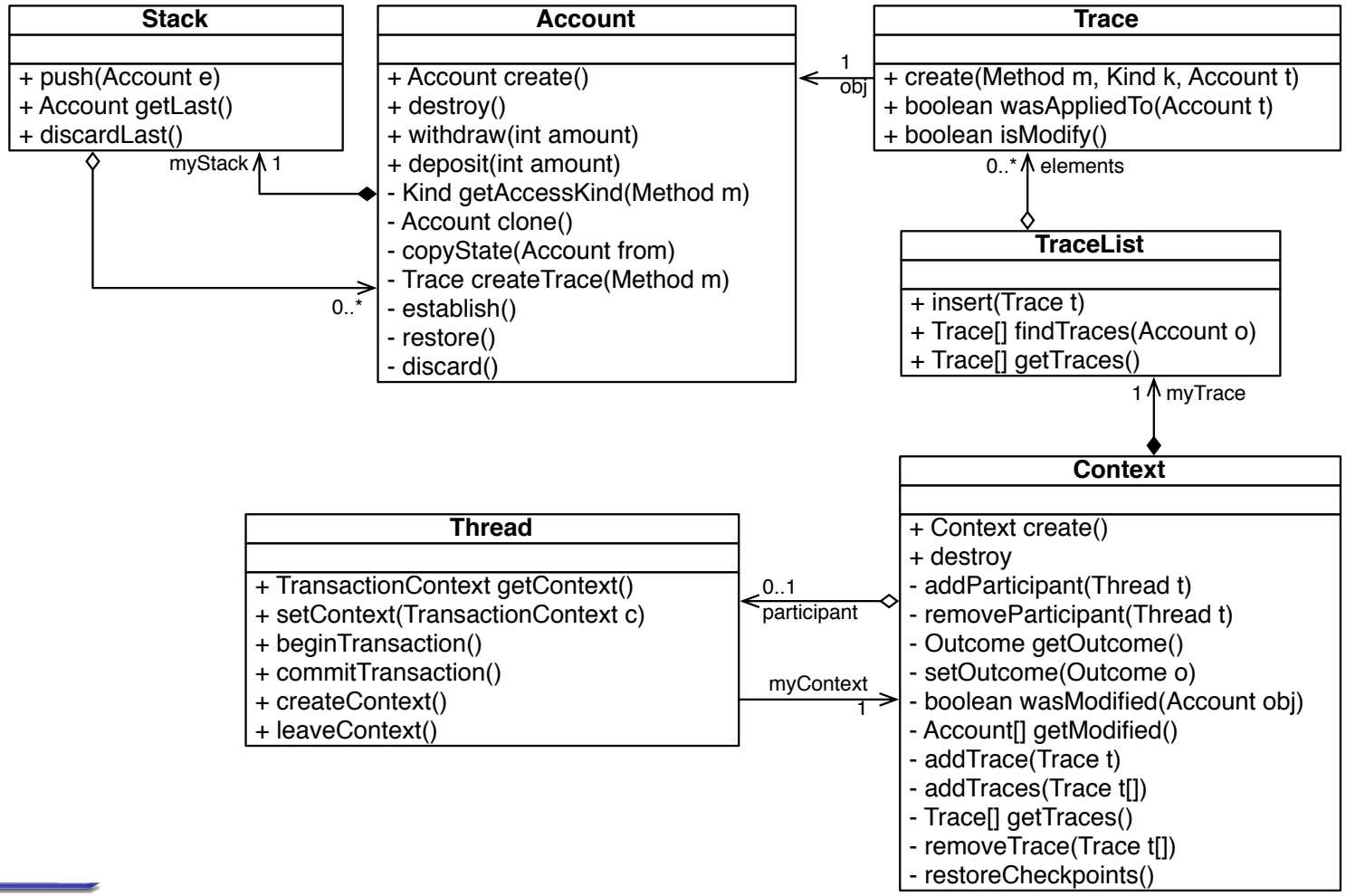
Base Model: Structure View

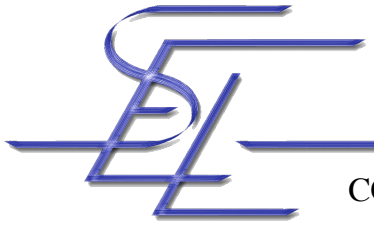




McGill

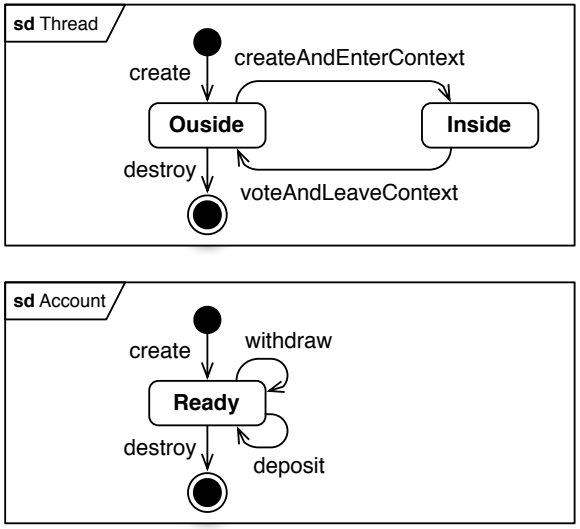
Woven Model: Structure View





McGill

Base Model: State View



+

Recovering instantiation

Recovering.IRecoveringParticipant ➝ Thread

Recovering.IRecoverable ➝ Account

Recovering.IRecoveringParticipant instantiation

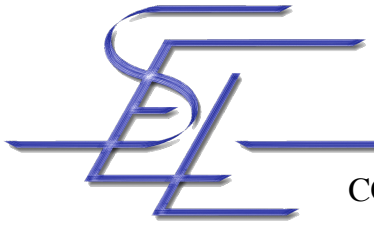
IRecoveringParticipant.IIdle ➝ Outside

IRecoveringParticipant.IWorking ➝ Inside

Recovering.IRecoverable instantiation

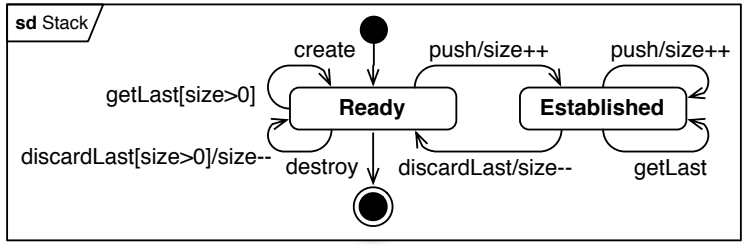
IRecoverable.BeforeM ➝ Account.Ready

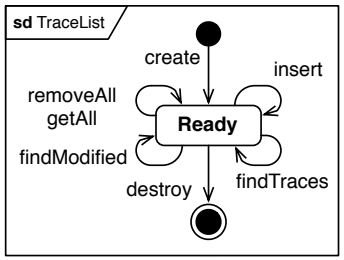
IRecoverable.AfterM ➝ Account.Ready

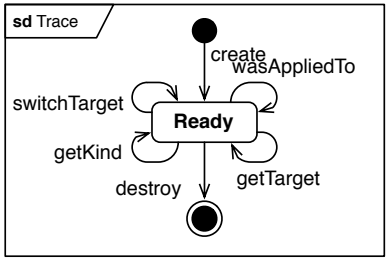


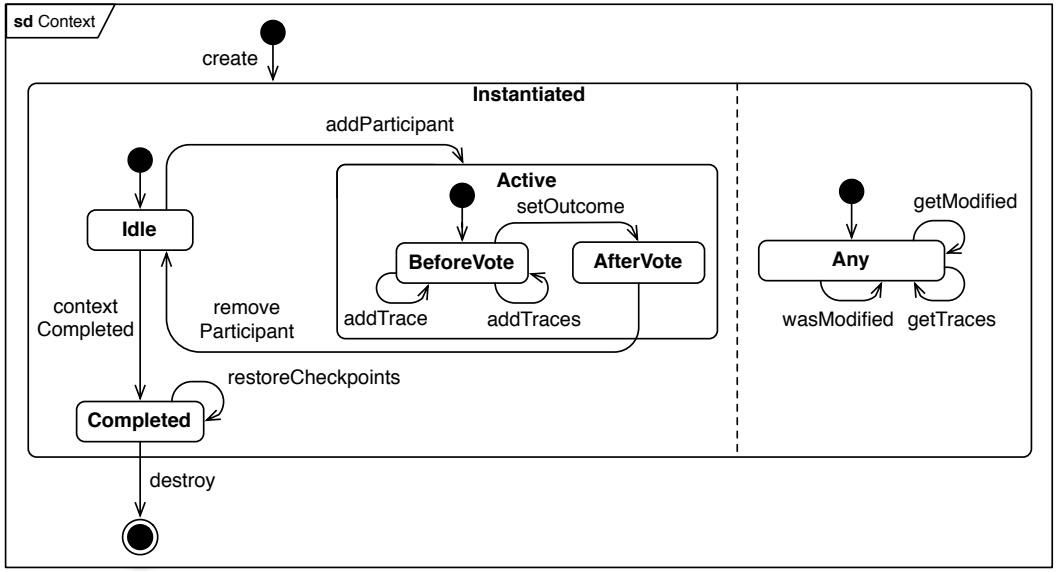
McGill

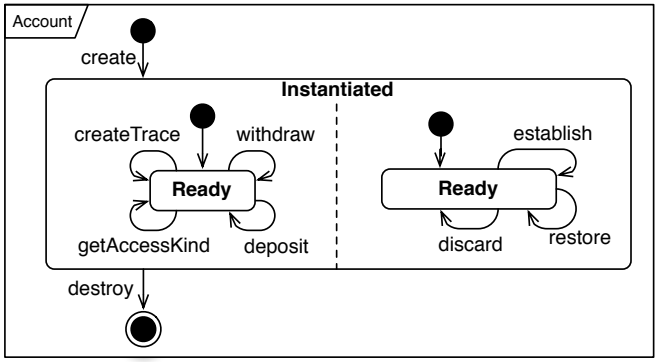
Woven Model: State View

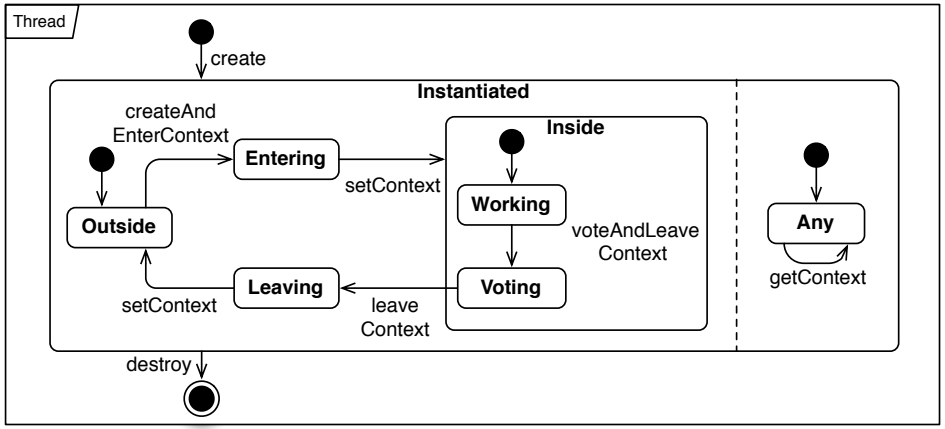


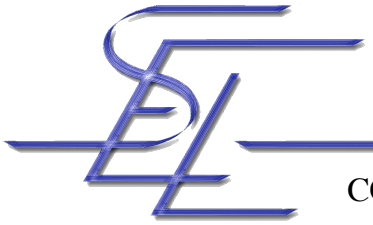






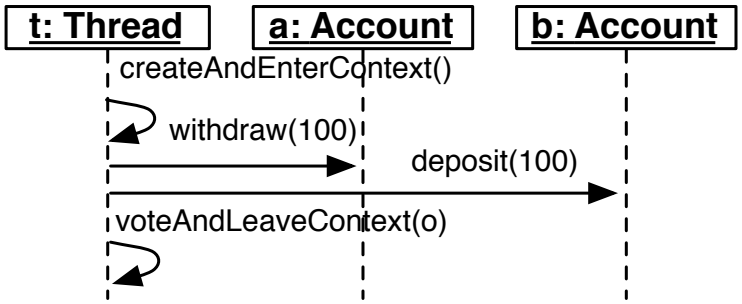






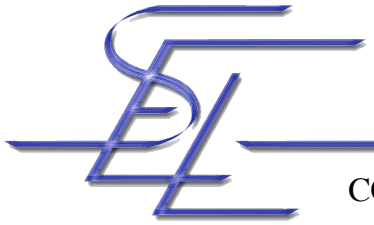
McGill

Base Model: Message View



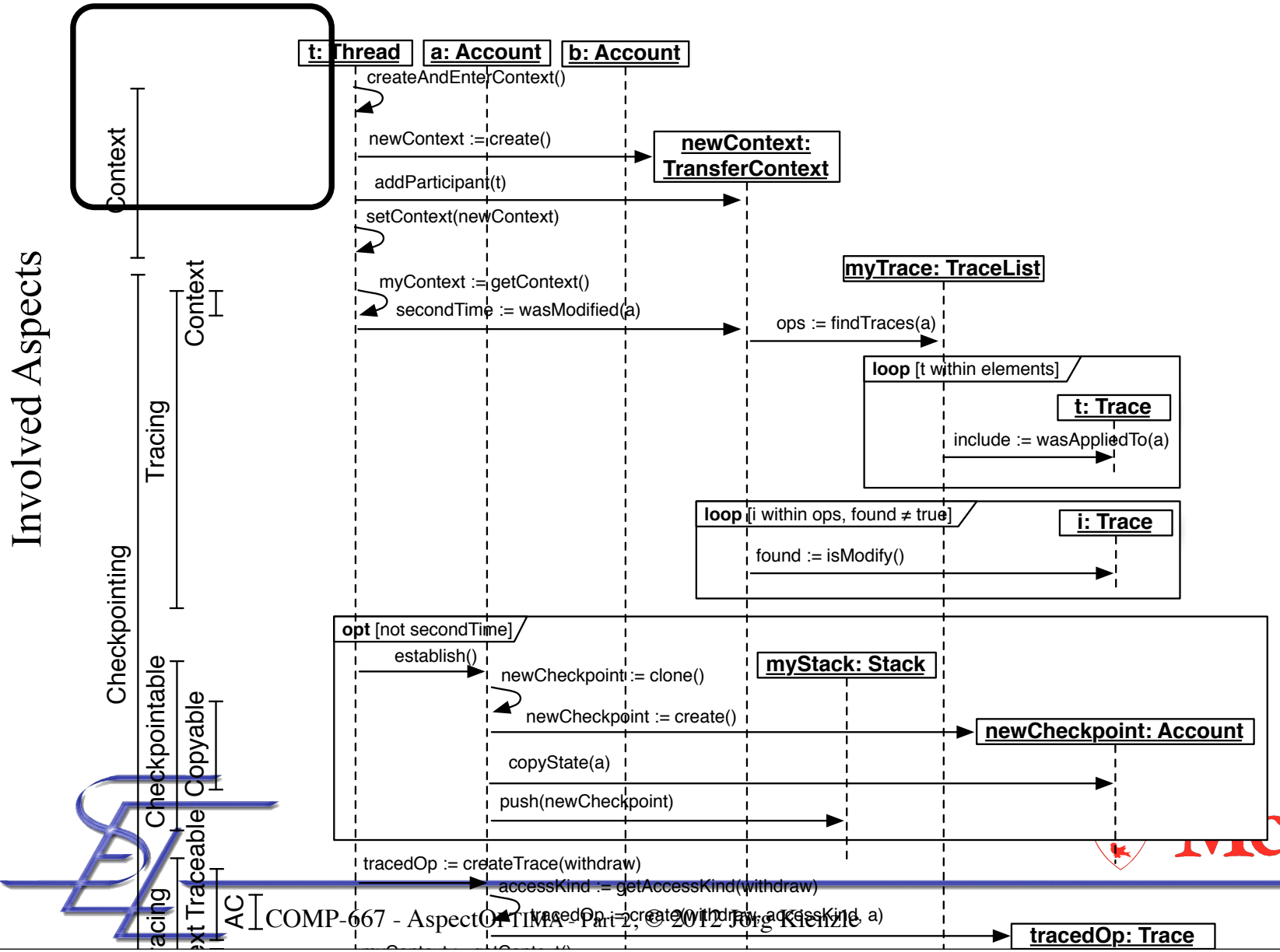
+

Recovering instantiation
Recovering.IRecoveringParticipant ➝ Thread
Recovering.IRecoverable ➝ Account
Recovering.IRecoveringParticipant instantiation
IRecoveringParticipant.IIdle ➝ Outside
IRecoveringParticipant.IWorking ➝ Inside
Recovering.IRecoverable instantiation
IRecoverable.BeforeM ➝ Account.Ready
IRecoverable.AfterM ➝ Account.Ready



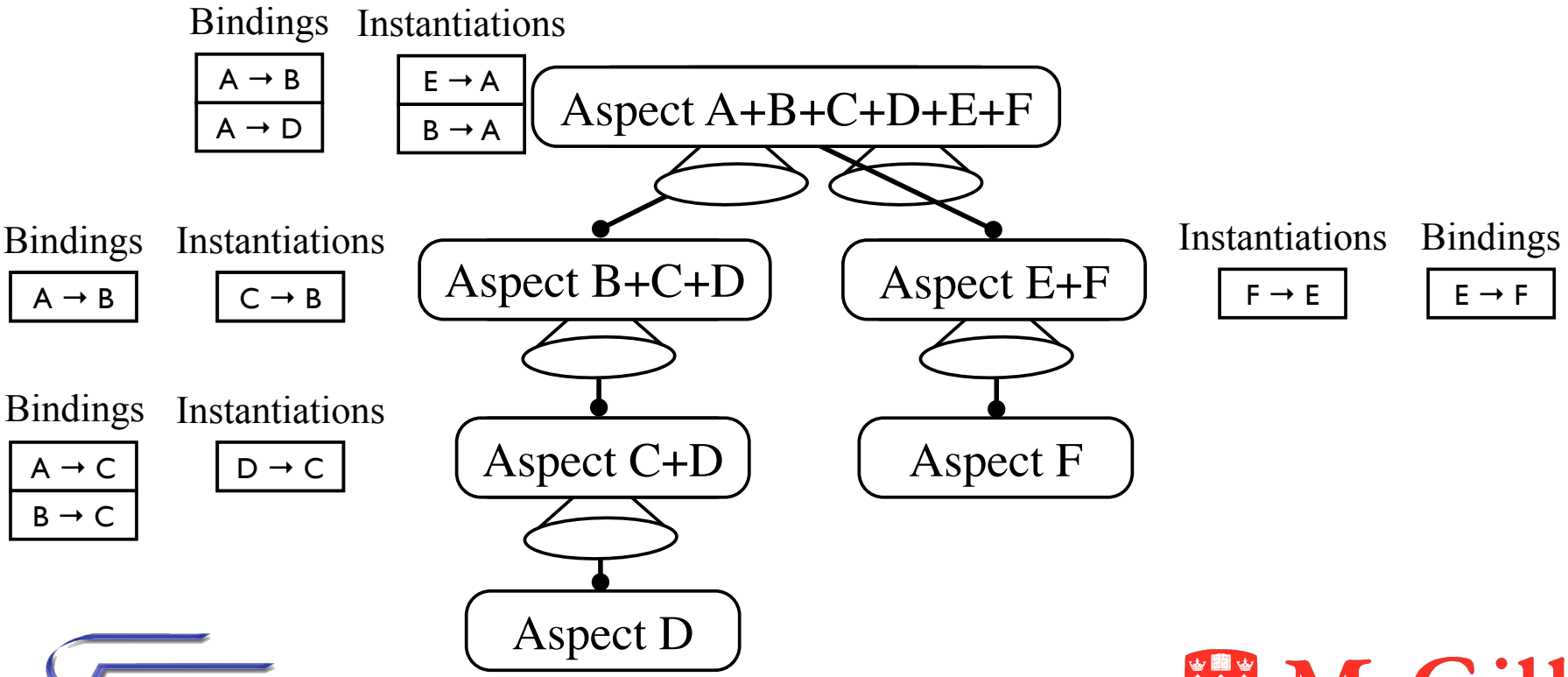
McGill

Woven Model: Message View



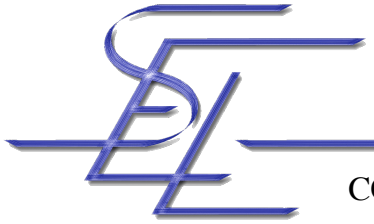
Recursive Weaving Algorithm

- Depth-first recursion through aspect tree, weaving lower-level aspects into higher-level aspects according to instantiation and binding directives
- Details in one of our papers



Consistency Checks

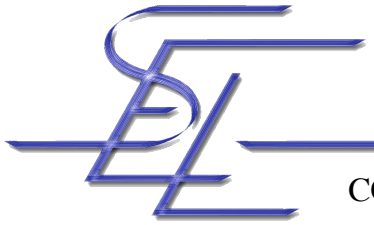
- When creating the aspect model, consistency checks are performed among views
  - Standard consistency checks
  - Conformance between the message views and the state views (model checking).
- Before (during) the weaving, consistency checks are performed among the directives defined at different levels
  - 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)



McGill

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



McGill

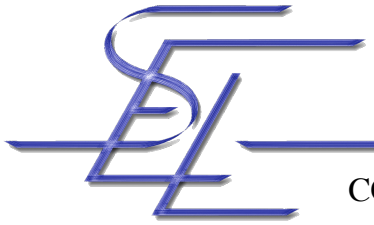
Other Work

- RAM also provides
  - Conflict resolution aspects
  - Feature diagrams to select variations
- Prototype tool
  - Implemented in Eclipse/Kermeta

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)



McGill

AspectOPTIMA References

Aspect-Orientation

[1] J. Kienzle, Ekwa Duala-Ekoko and S. Gélineau, "AspectOPTIMA:A Case Study on Aspect Dependencies and Interactions", Transactions on Aspect-Oriented Software Development, in press.

[2] J. Kienzle and S. Gélineau, " 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.

McGill