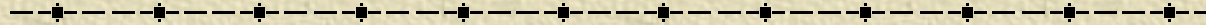
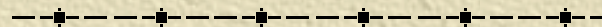


Applying aspect-oriented programming ideas in a component based context: **Composition Adapters**

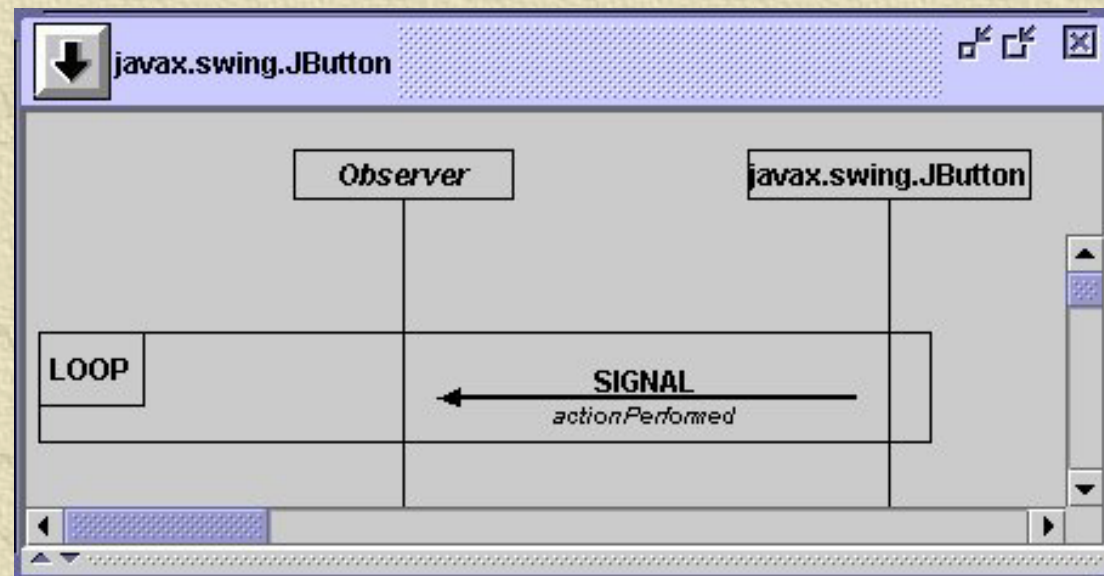


Wim Vanderperren



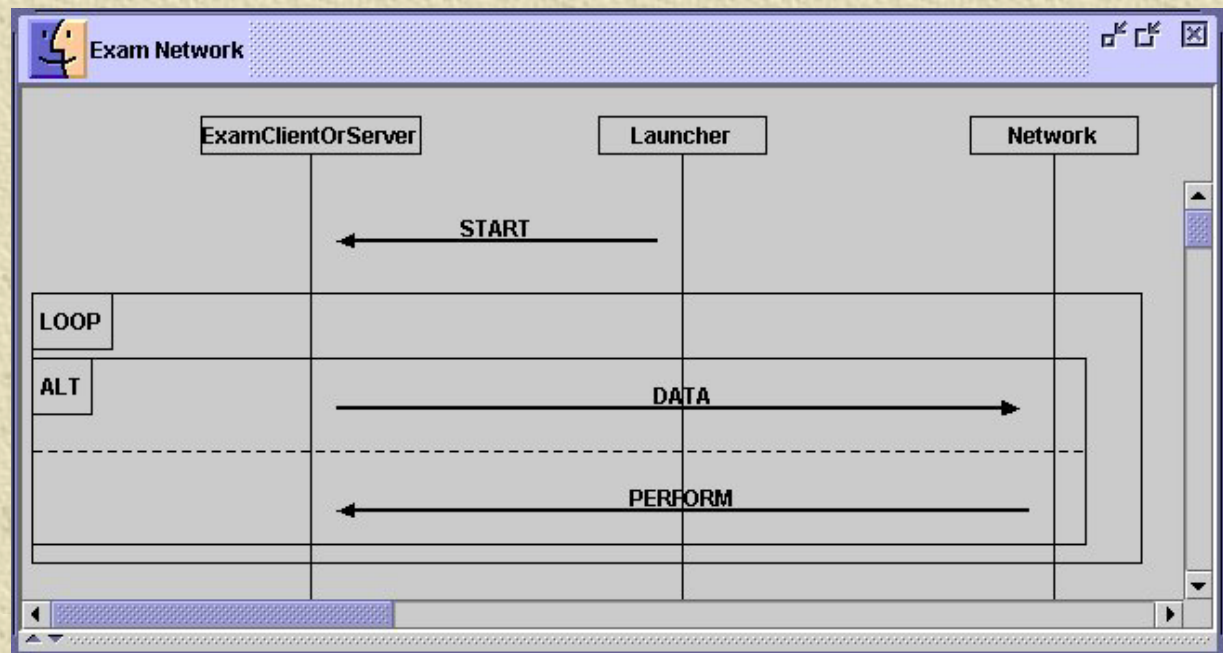
Research Context: PacoSuite (1)

Document components through **usage scenarios**
(MSC's).



Research Context: PacoSuite (2)

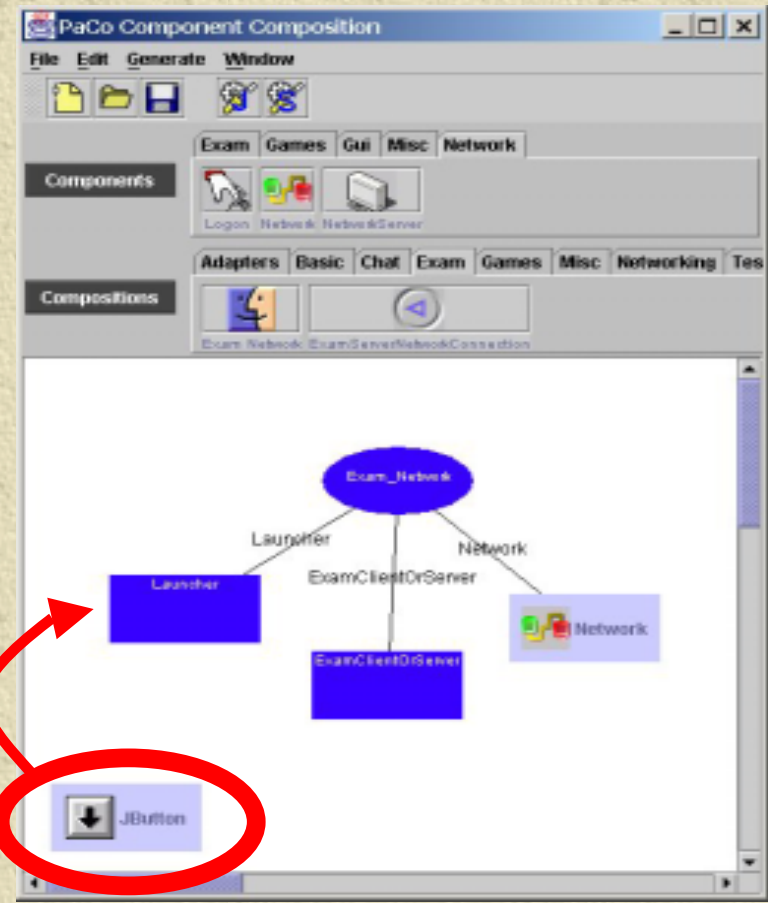
Document generic compositions of interacting participants using **composition patterns (CP)**.



Research Context: PacoSuite (3)

Component Wiring:

- ✦ Automatic Compatibility Checks.
- ✦ Automatic Glue-code generation.



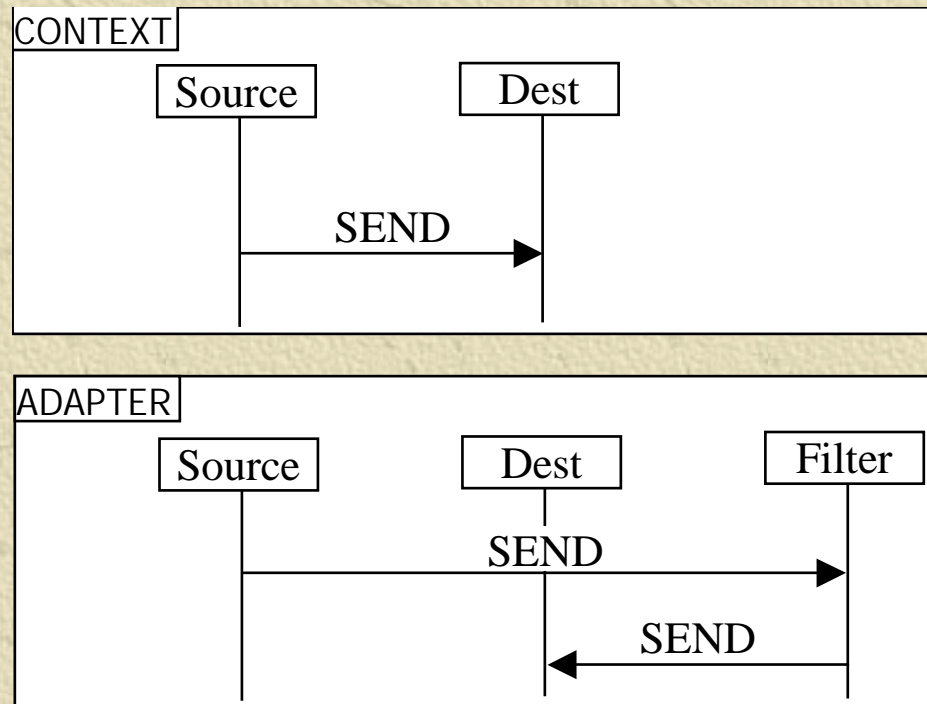
Research Goal

OBSERVATION: Some concerns are not cleanly modularized
E.g: accounting, tracing...

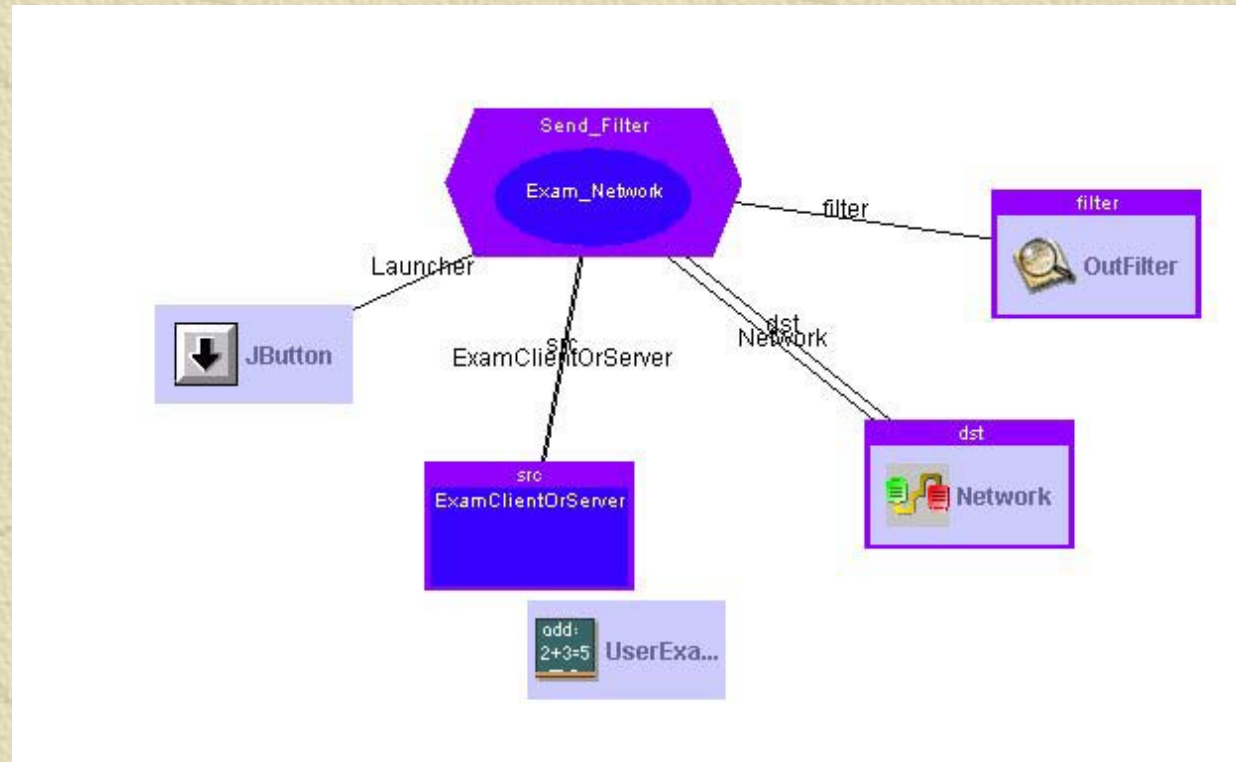
GOAL: To be able to cleanly modularize these crosscutting concerns without lowering the abstraction level.

Composition Adapters (CA)

Filtering CA:



Applying a CA (1)



Applying a CA (2)

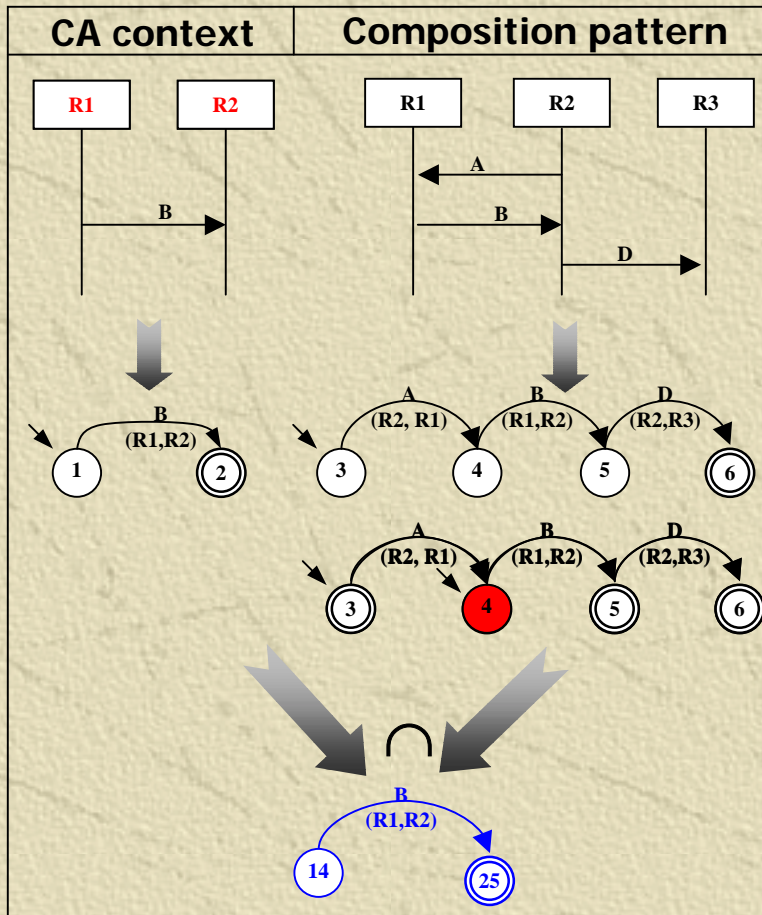
✦ Checking CA

- ◆ Finding all paths corresponding to CA context.

✦ Inserting adaptations into CP

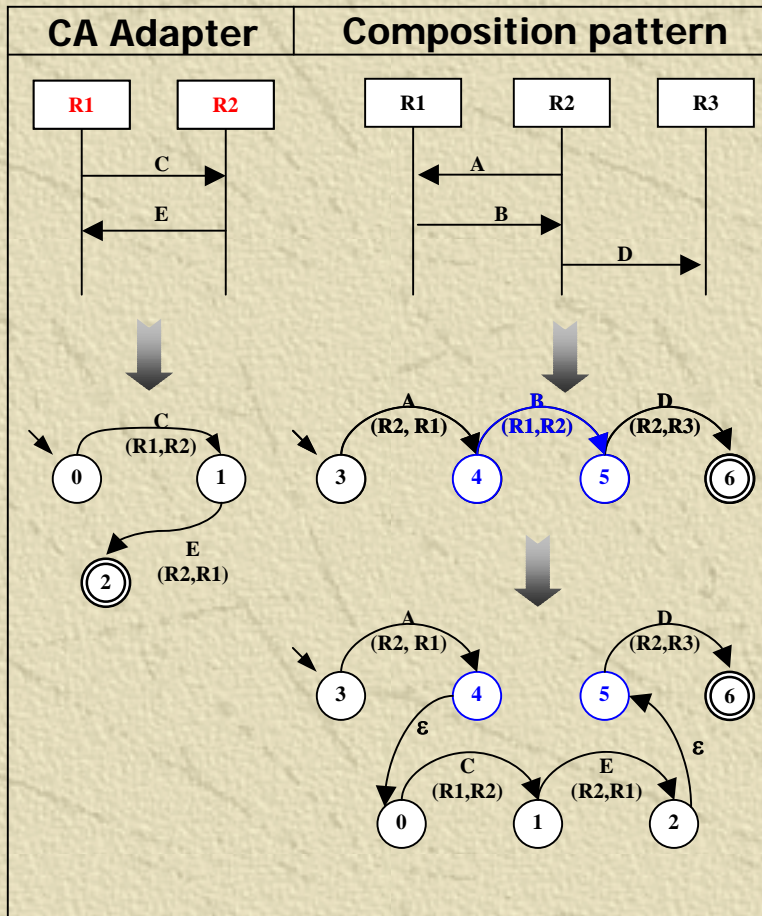
- ◆ Replace all matching paths with CA adapter.

Checking CA



- ✦ Translate CA context to CP.
- ✦ Translate both MSC's to DFA's.
- ✦ For each state in DFA of CP:
 - ✦ Transform CP-DFA.
 - ✦ Calculate Intersection with context DFA.
 - ✦ **If not Empty: Matches with adaptation context!**

Inserting adaptations in CP



- ✦ Translate CA Adapter to CP.
- ✦ Translate both MSC's to DFA's.
- ✦ For each matching path in DFA of CP:
 - ✦ Remove path.
 - ✦ Insert copy of adapter DFA.
- ✦ Transform CP NDFFA to DFA.

Conclusions

- ✦ CA cleanly modularizes crosscutting concerns in CBD systems.
- ✦ Still preserves a high abstraction level.
- ✦ Supports evolutionary changes.
- ✦ Can only adapt component's exterior behavior.