Model Composition Contracts

ACM/IEEE 12th Int. Conf.on Model Driven Engineering Languages and Systems

MODELS 2009, Denver, Colorado, USA

Jon Oldevik (1,2) *jonold at ifi.uio.no*

Massimiliano Menarini (2) *mmenarini at ucsd.edu*

Ingolf Krüger (2) *ikrueger at ucsd.edu*

(1) University of Oslo and SINTEF, Oslo, Norway
(2) University of San Diego, California (UCSD), La Jolla, USA

Problem Overview

Aspect-oriented technologies provide flexible and powerful composition mechanisms

Some common characteristics:

- **Modularising cross cutting concerns**

- **Powerful pointcut / query mechanisms**

- **Oblivousness – the base model / code is unaware of any aspects applied.**

Problem Overview(2)

But, they may have unexpected side effects...

Breach of encapsulation assumptions

Base model – or code – may be changed in ways not intended

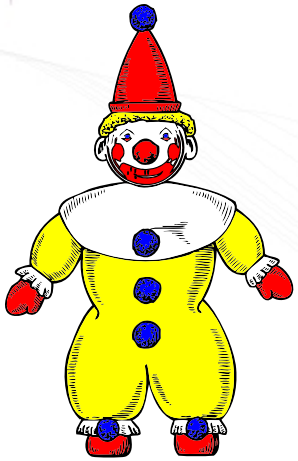
This may result in...

Inconsistent models

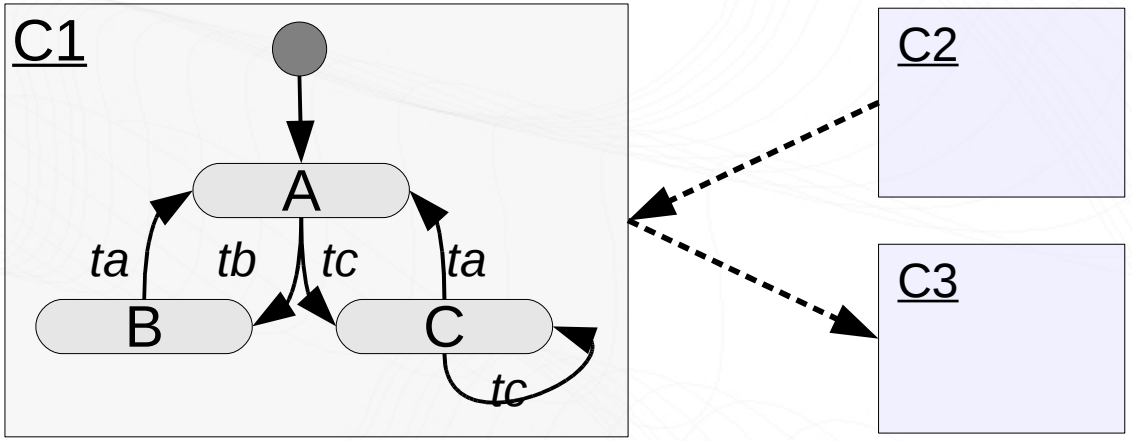
Erronous implementations

System failures

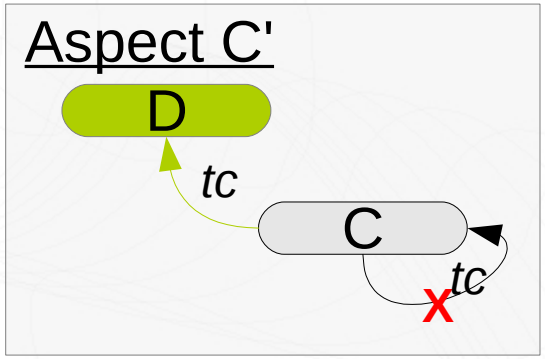
The Problem Illustrated



The Main Developer

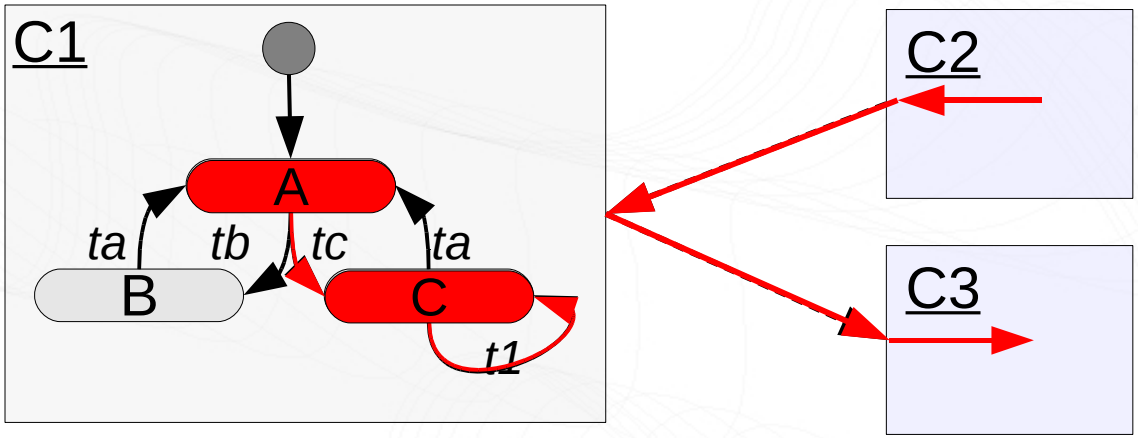


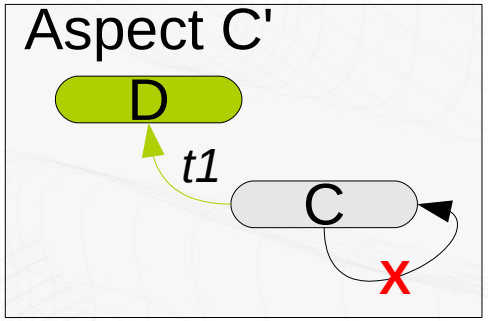
The Aspect Developer

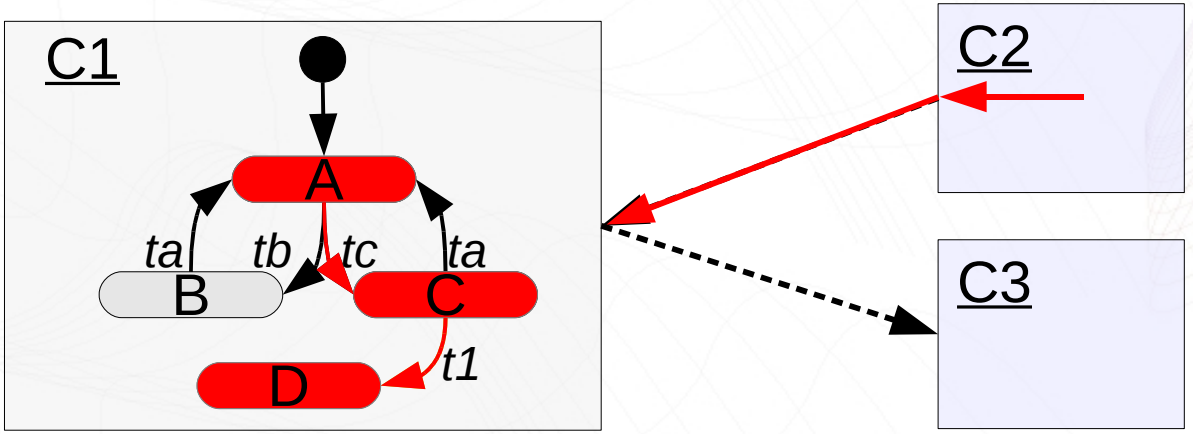


The Problem Illustrated

Base System







Modified System

Proposed Solution

Associate composition contracts with models

A composition contract defines

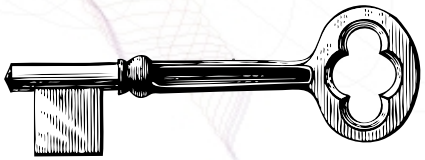
Access rules for the models

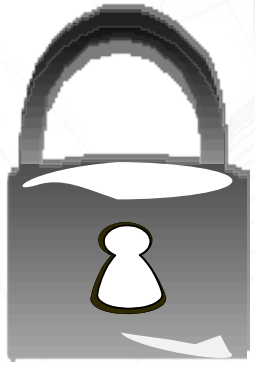
Allowed modifications in terms of

Pre-conditions && **post**-conditions

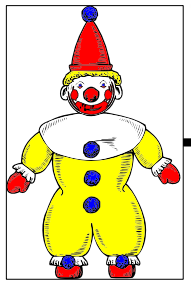
Based on OCL (with some operational extensions)

A composition engine should adhere to the contract in order for a composition to be allowed.





We have implemented a tool to support the approach



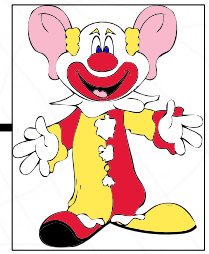
Base model

Contract

exports

Aspect

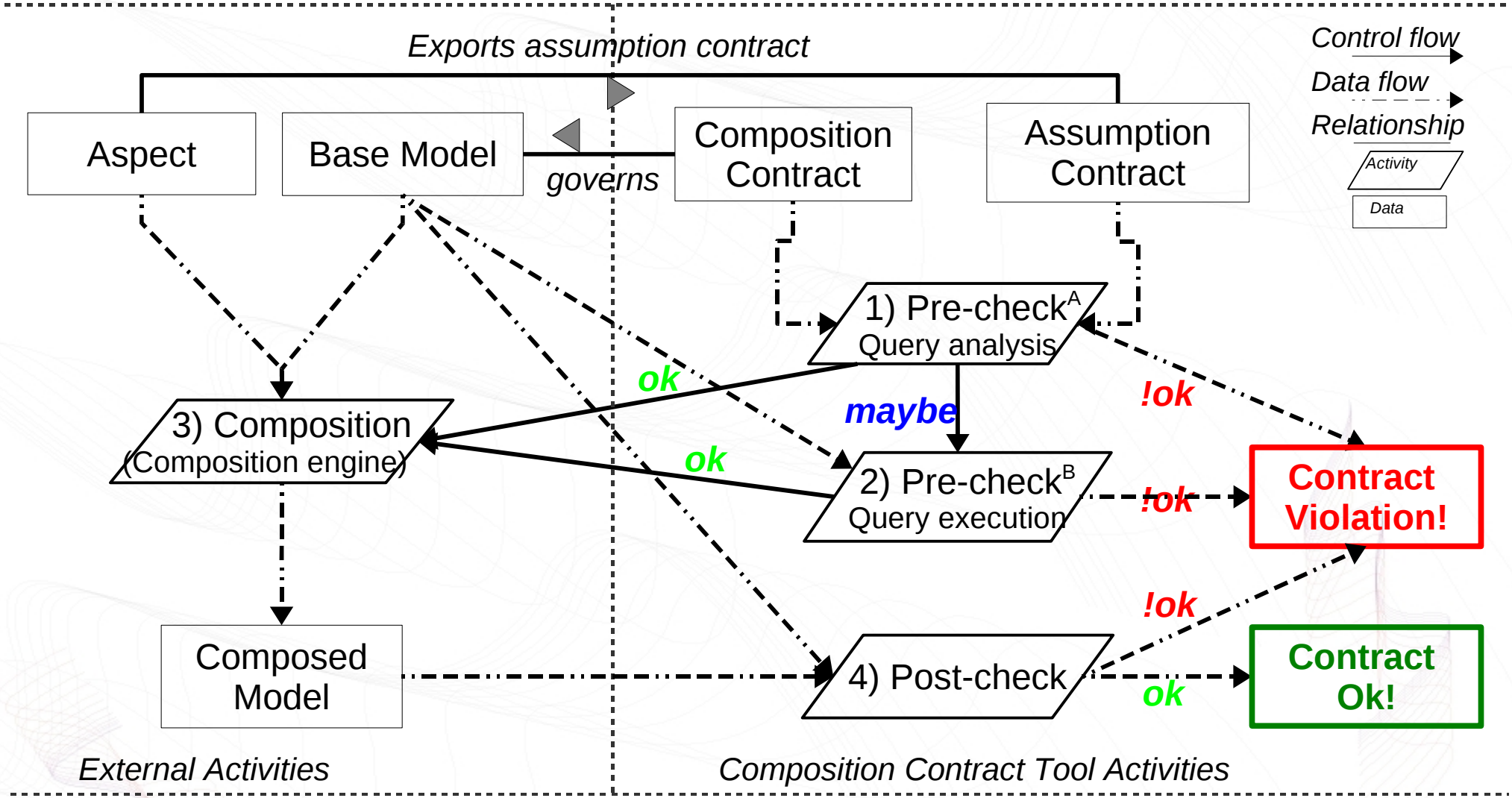
Assumption Contract



Contract Checking

Verdict

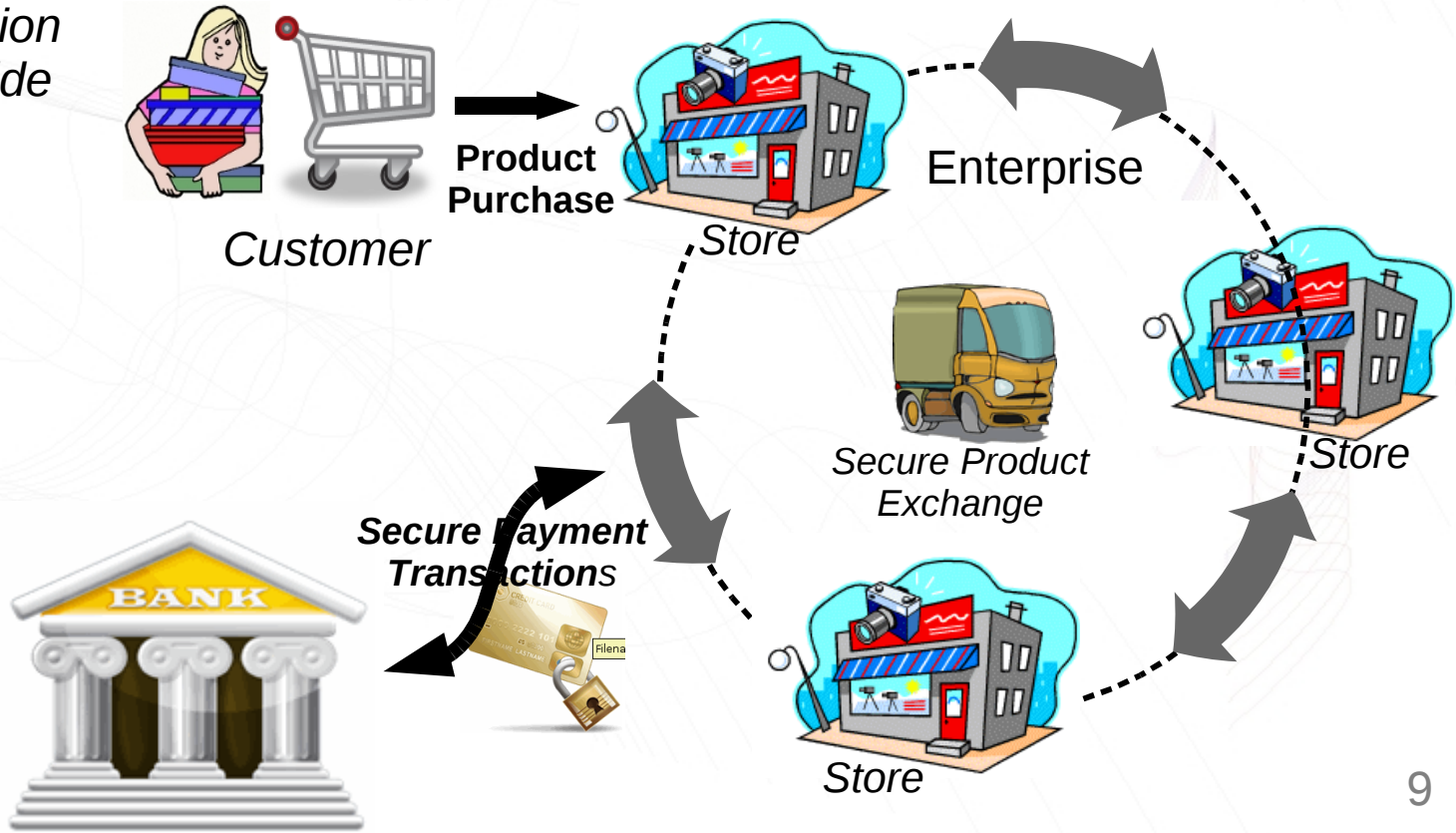
Composition Contract Process Overview



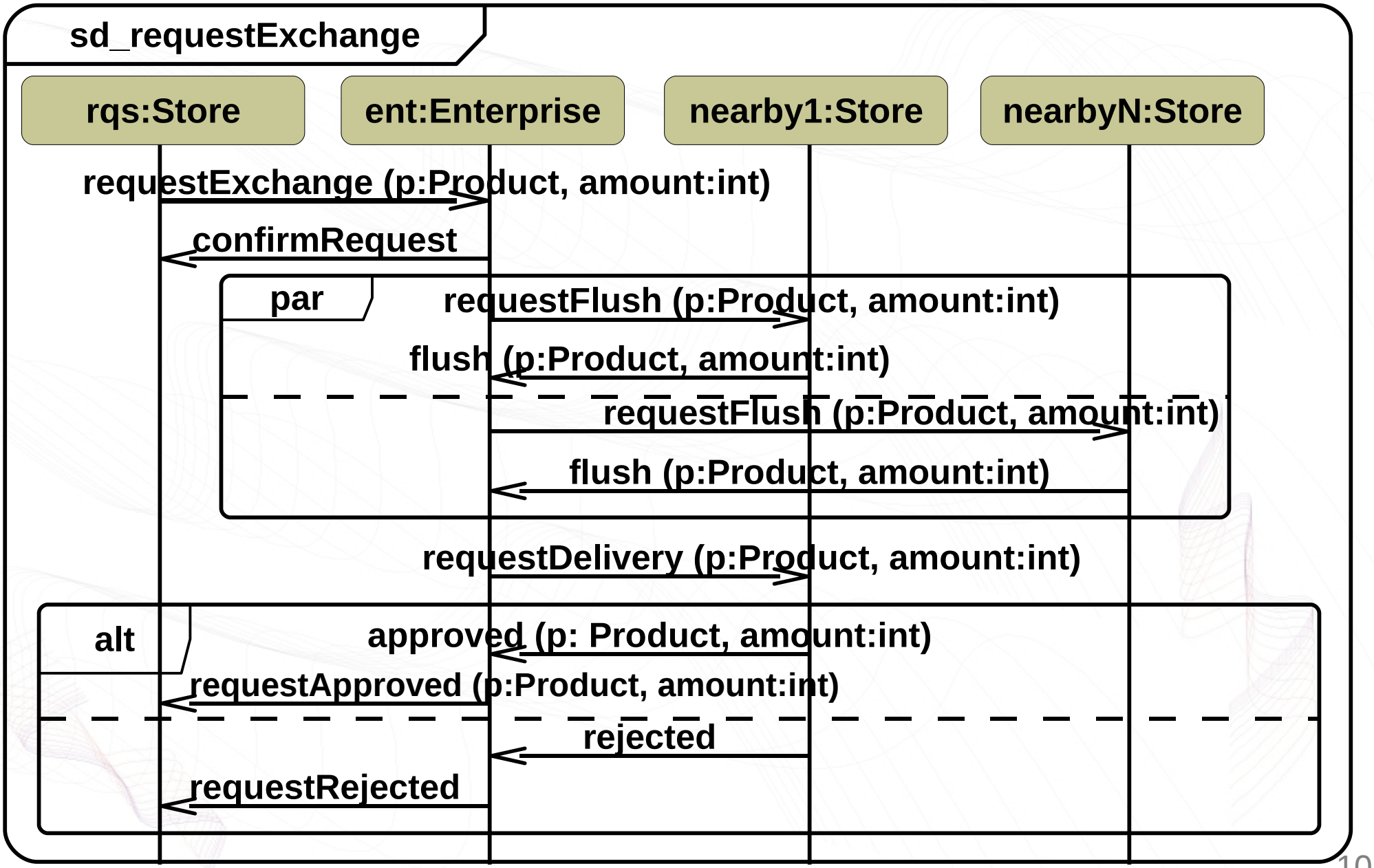
Example: Store Collaboration

Manages product ordering from suppliers, product exchange between stores, and sales to customers.

- ***If a store runs low on a stocked item, it trades with collaborating stores***
- ***Different stores have different requirements for product information***
  - *Parameter type transformation*
- ***Payment and product exchange transactions should be secured with encryption***
  - *No sensitive information must be accessed outside the endpoints.*



Interaction – product exchange



The contract governing the example model

- It is allowed to **modify messages** going from the *Store* to the *Enterprise* role
- The **message parameters** can be **modified**
- It is allowed to **add new lifelines,** e.g. to intercept messages
- **Messages can be replaced** by messages to/from interceptors, but **events must be preserved**.
- No **sensitive information** can be accessed outside the endpoints
  - *I.e. not by the infrastructure*
- **Message parameters can be modified** in existing messages
  - Type transformations / type refinements

Details of the Approach – Contract Definition

```
contract Contract_For_My_System {
  elements Interaction, Lifeline, Message ;

  accessor Interaction[*] interacts : true;
  accessor Messages[*] msgs : let m:Message = self in
      m.sendRole().type()='Store' and m.receiveRole().type()='Enterprise'
      and m.name.matches('.*');

  introduction interacts::newLifeline() {feature lifelines, message}
  modification msgs::argumentChange() {feature argument;}

  query Message::sendRole() : self.sendEvent.oclAsType(Message
     OccurrenceSpecification).covered->asOrderedSet()->first();

  context Lifeline post: self.preval() <> null implies self.coveredBy->size()
      = self.preval().oclAsType(Lifeline).coveredBy->size();
  context Lifeline post: self.type()='Encryption' implies not(self.preval() = null);

}
```

Allowed access

Allowed changes

Helper queries

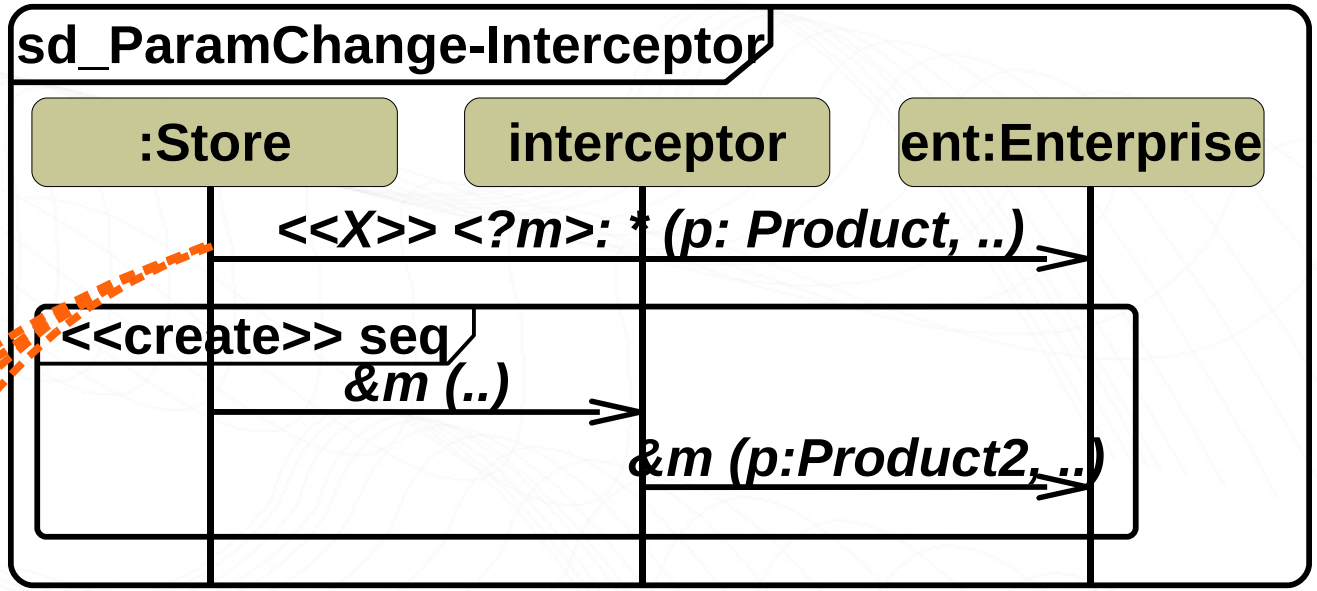
Post-conditions

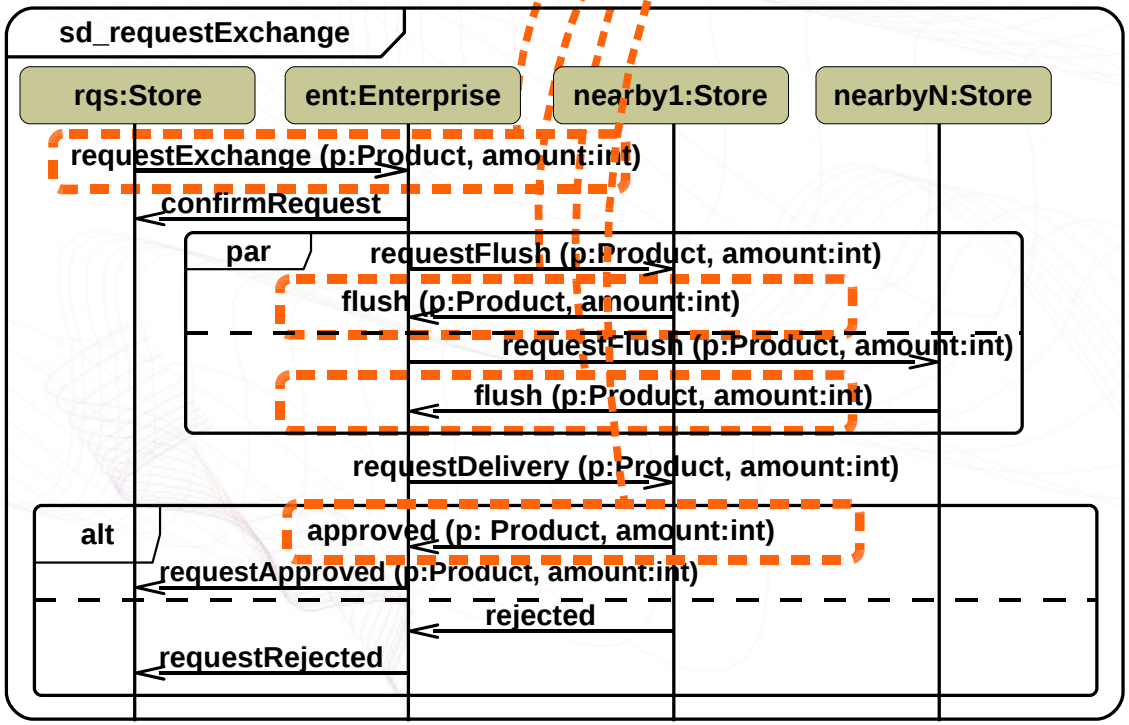
Contract – cont...

Post condition allowing parameter type modifications

```
context Message post: let m:Message=self, life:Lifeline = self.receiveRole()
    in
  life.preval() <> null implies
     life.preval().oclAsType(Lifeline).coveredBy->exists (prefrag:
       InteractionFragment |
     prefrag.oclIsKindOf(MessageOccurrenceSpecification)
        and
     m.refinementOf (prefrag.oclAsType (MessageOccurrenceSpecification).
         message));
```

Aspect – Changing Parameter Type for Message



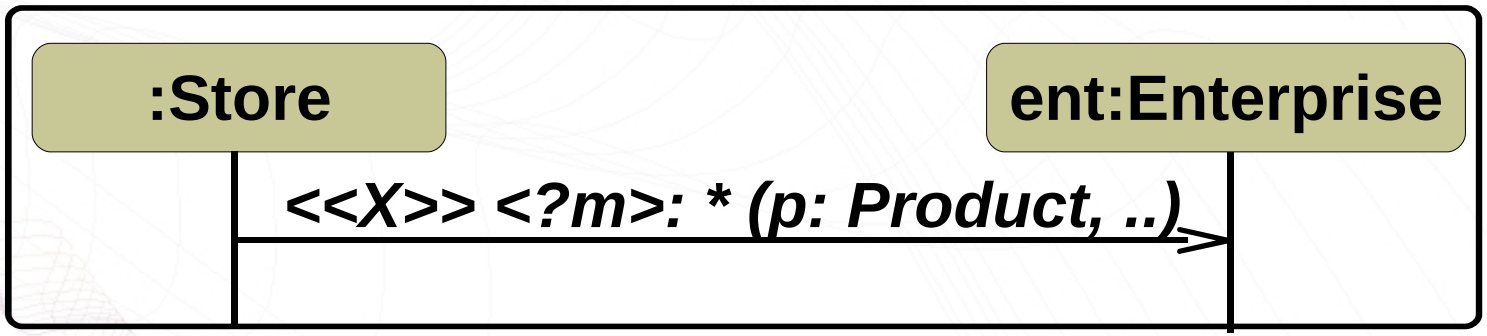


Assumption Contract – Generated / exported from Aspect

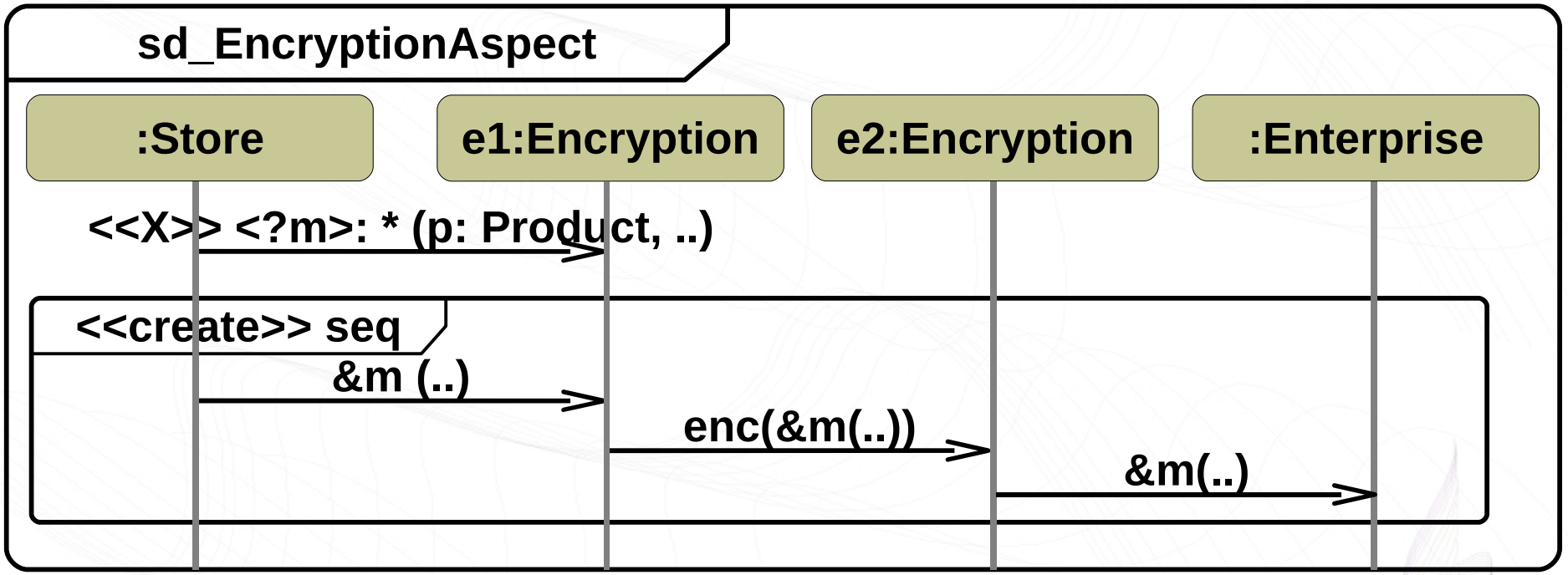
- The assumption contract is used for comparison with the base model contract
  - Pre-checks: query analysis & query execution

---

elements Interaction, Lifeline, Message;
accessor Message[*] messages : self.sendRole().type().name='Store' **and**
self.receiveRole().type().name='Enterprise' **and**
self.name.matches('.*') and self.hasArgType('Product');



Example of Contract Violation : the Encryption Aspect



context Lifeline *post*:

self.type()='Encryption' implies *not* (self.**preval()** = null);

[Lifelines of type Encryption must exist prior to composition]

Summary

- We have defined an approach for associating contracts with models
  - That controls eligibility for composition
  - By specifying rules for allowed access and modificiation
  - Pre- and post-conditions for composition
  - We have developed a prototype implementation for specifying and checking contracts
- Related work
  - Crosscutting programming interfaces (XPI) [Griswold et.al]
  - Confirmed join points [Ossher]
  - Harmless Advice [Dantas and Walker]
- Future work
  - Extend expressiveness; investigate supplement with model invariants
  - Increased usability; look at graphical integration
  - Mapping to and validation of implementation-level contracts