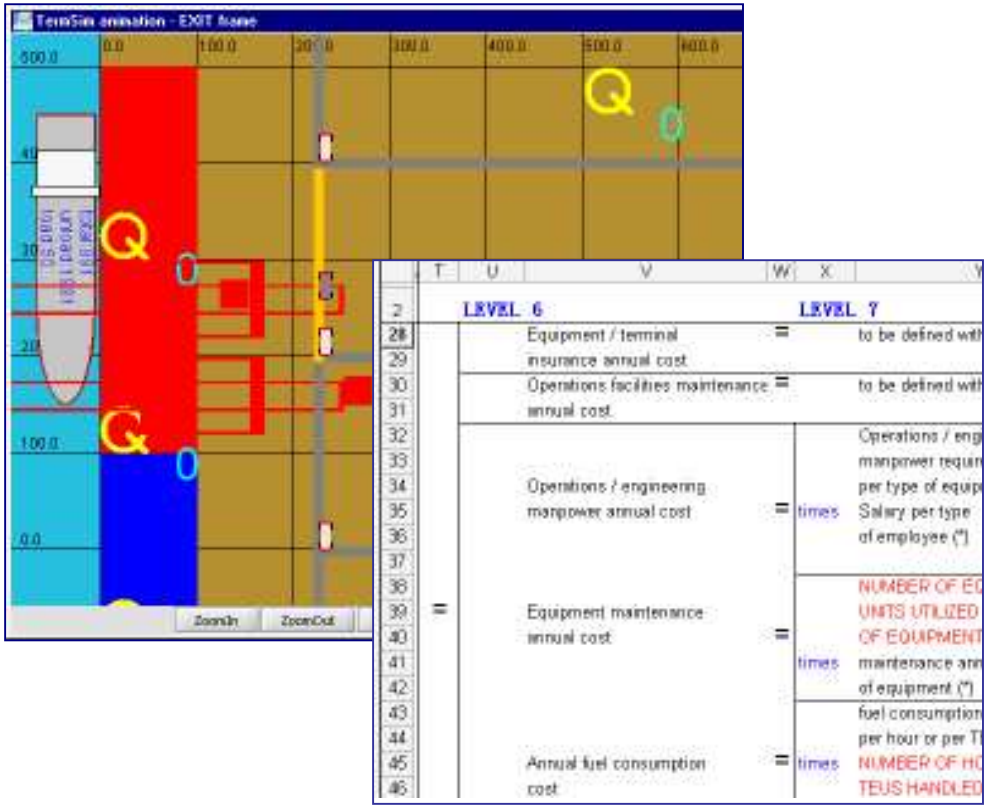
DSOL tutorial part 2 building *TermSim*



Niels Lang

Rotterdam School of Management

Erasmus University Rotterdam

ERASMUS UNIVERSITEIT ROTTERDAM

Contents

- Introduction to TermSim
- A basic modeling framework
  - Identification
  - Logging
  - Context
- Animation
- Basic packages
  - MovingObject
  - Customer / resource / service
- Features

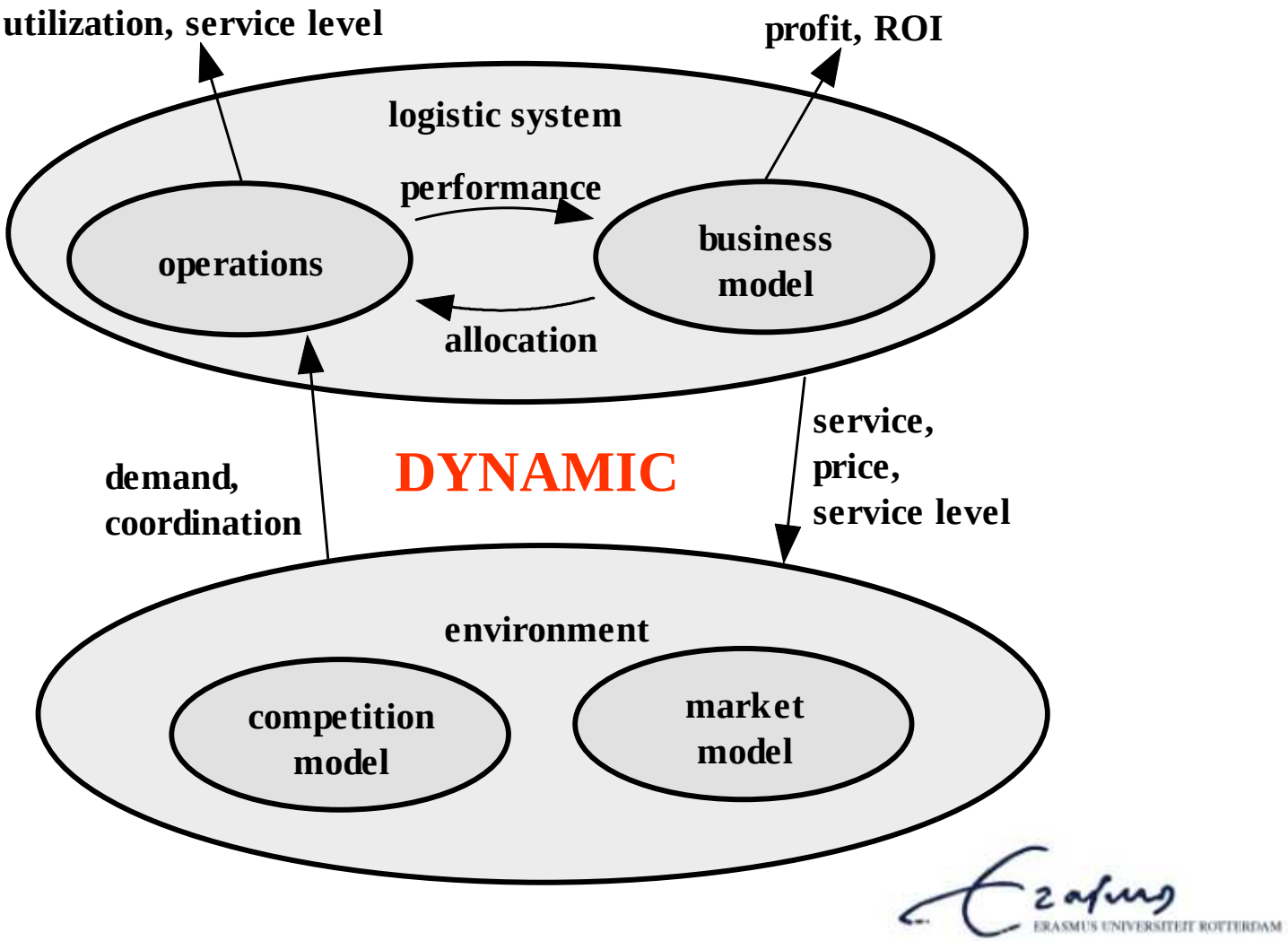
Logistic system design challenges

- **Many actors, many viewpoints, many questions**
  - **Investors (ROI)**
  - **Engineers (feasability)**
  - **Customers (service level)**
- **Economic challenges**
  - **Sunk costs, no room for failure!**
  - **Competition will not sit still: dynamics**
  - **Dynamic market conditions, sensitive to ‘events’**

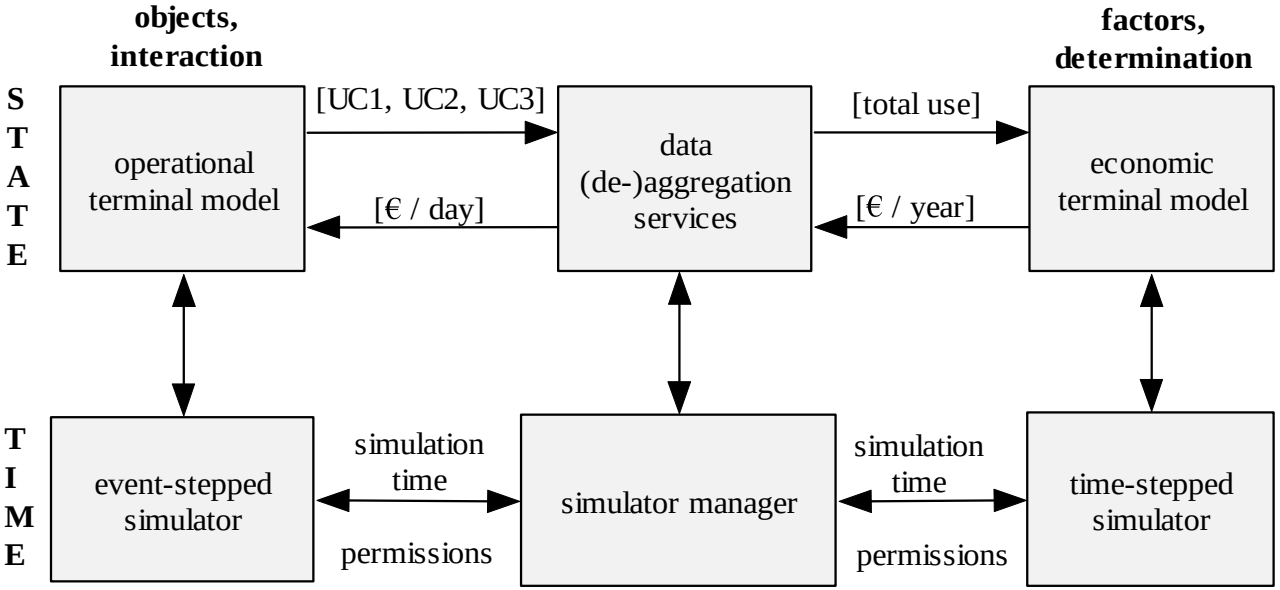
Research questions

- What modeling approach is needed for logistic system design in dynamic environments?
  - Focus on economic performance
  - Operations, business, market and competition should be taken into account
  - All aspects could be dynamically dependent
- What are design requirements for a support environment for such an approach?
  - Computational support is expected to be necessary

Towards a system of integrated, dynamic models



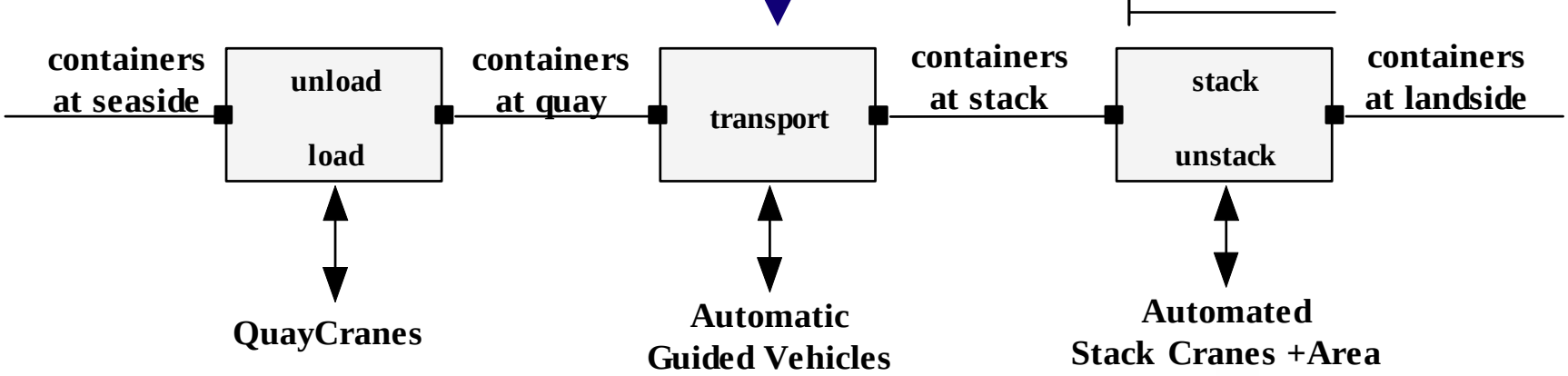
Main integration issues



- **Different dynamic paradigms are translated by a simulator manager**
- **Data between models is communicated using data (de)aggregation and translation services**

Operations: conceptual model

resources & processes

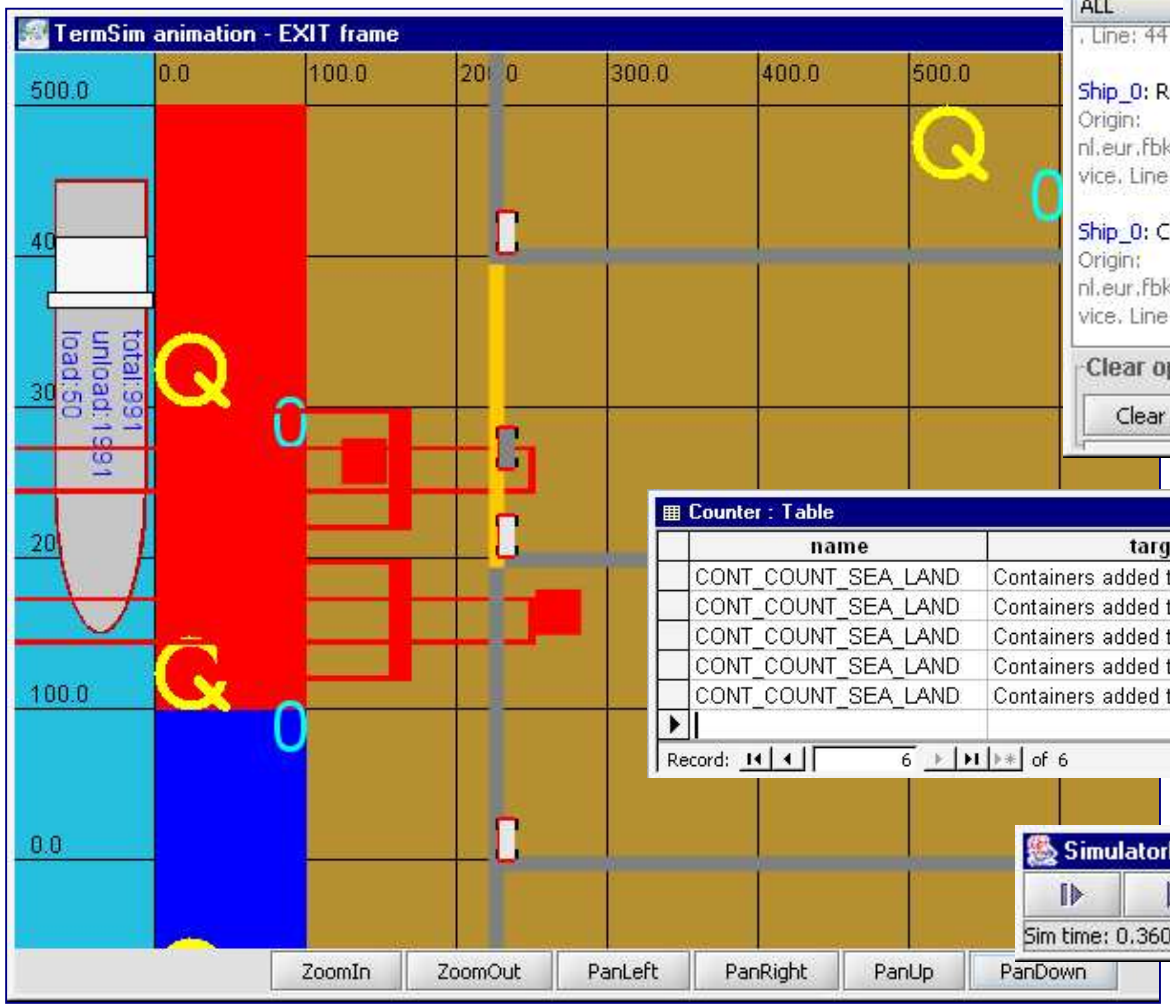


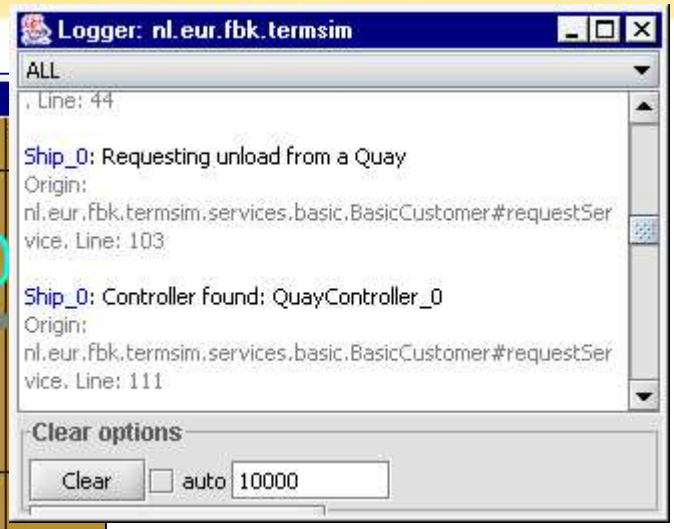
Current status: simulation

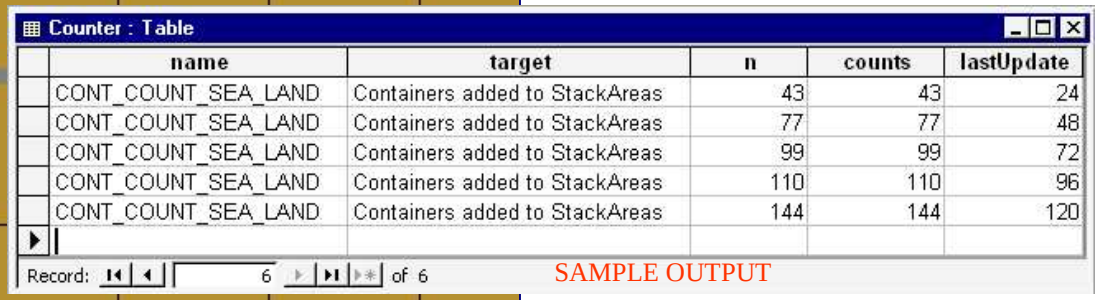
- **Model status:**
  - **Resources and processes are modeled, fine-tuning of allocation mechanisms and layout is needed.**
  - **Specification of process-times and # of equipment is far from complete**
- **Technical status:**
  - **Database connectivity realized, including experiment/scenario database**
  - **Several features are reviewed for adoption in DSOL**

Current status: simulation

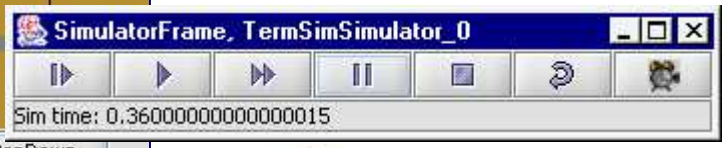
screenshots + demo







SAMPLE OUTPUT

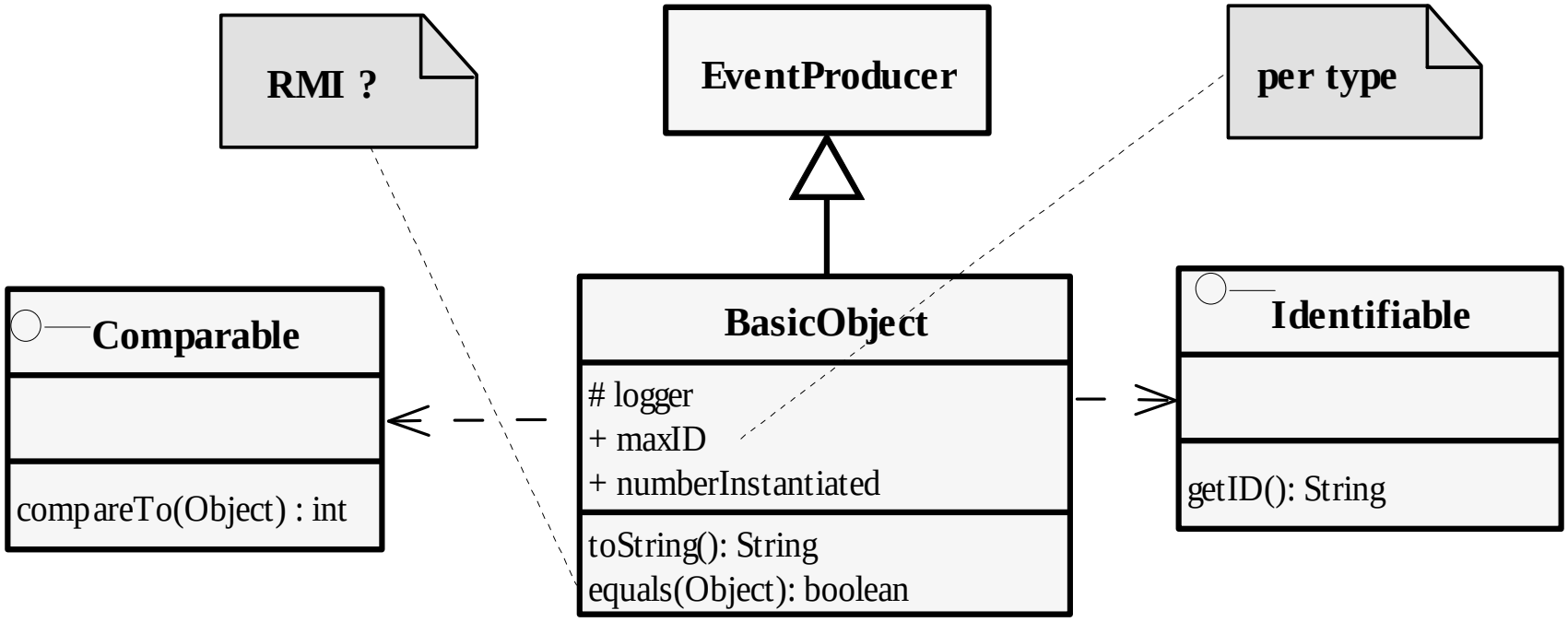


Requirements for a basic framework

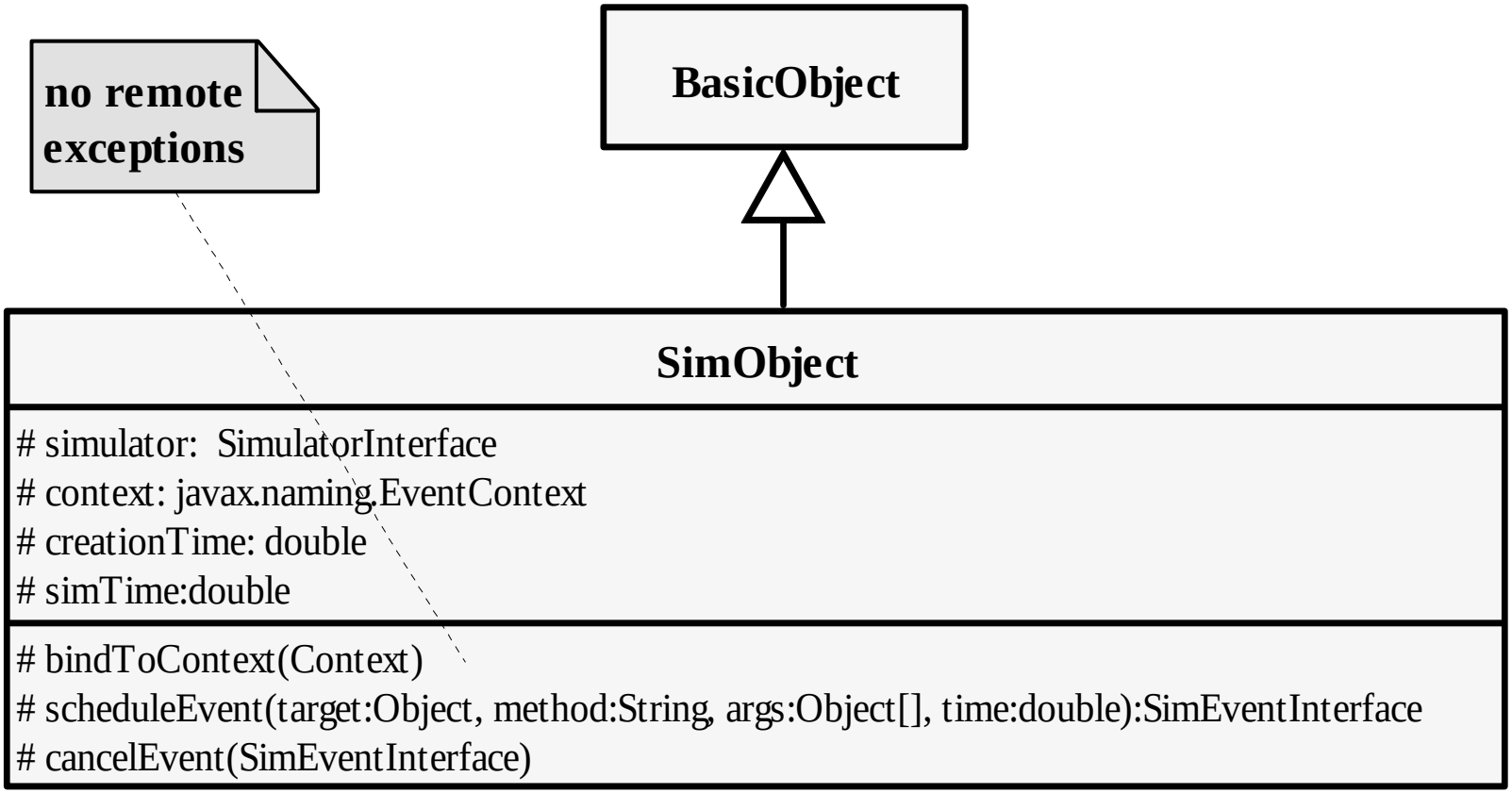
- Many tedious operations should be concentrated in one place
  - Logging, exception handling
  - Scheduling
  - Canceling
- Model objects should all possess a minimum set of characteristics and behaviour
  - Identification, equality operator
  - Presentation (toString), comparisson
  - References to simulator and context

The basics: BasicObject

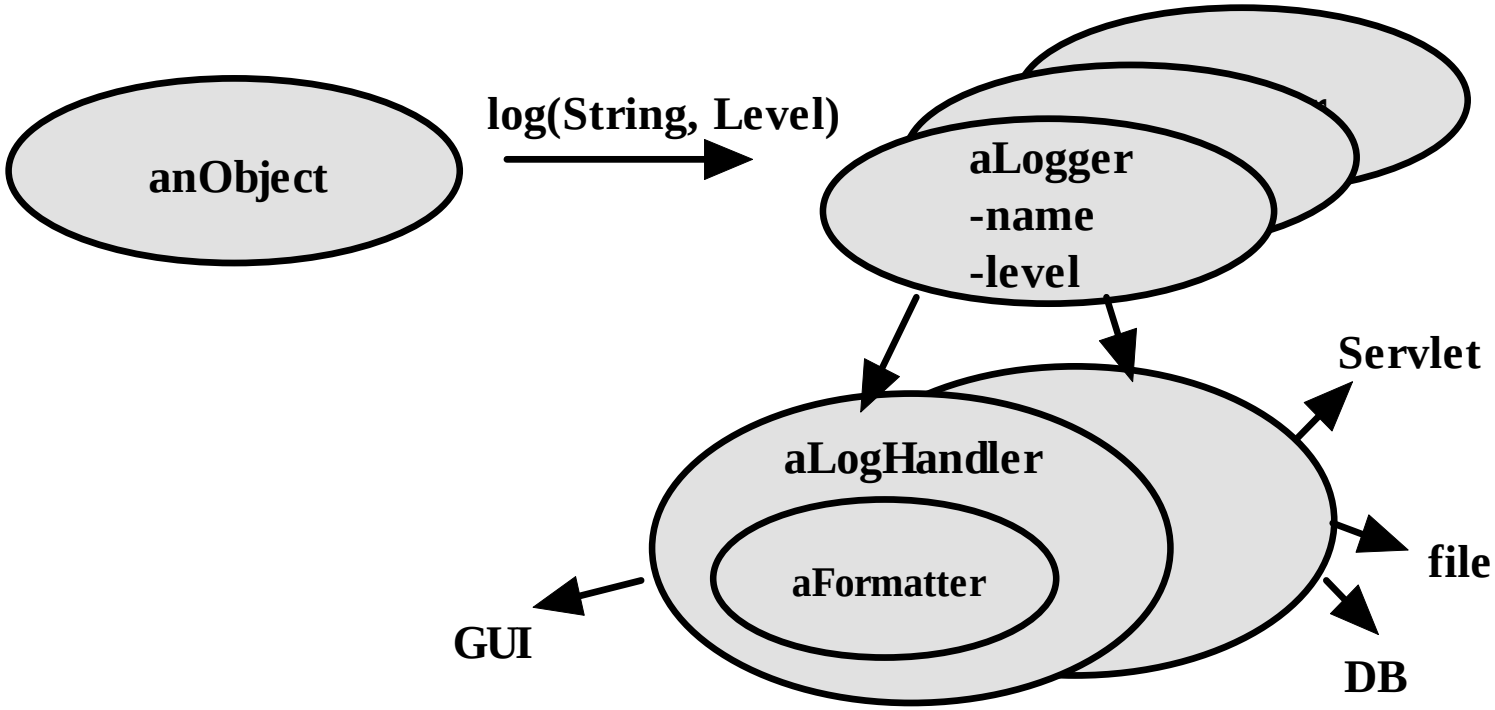
- **TermSim is not designed for a distributed environment**



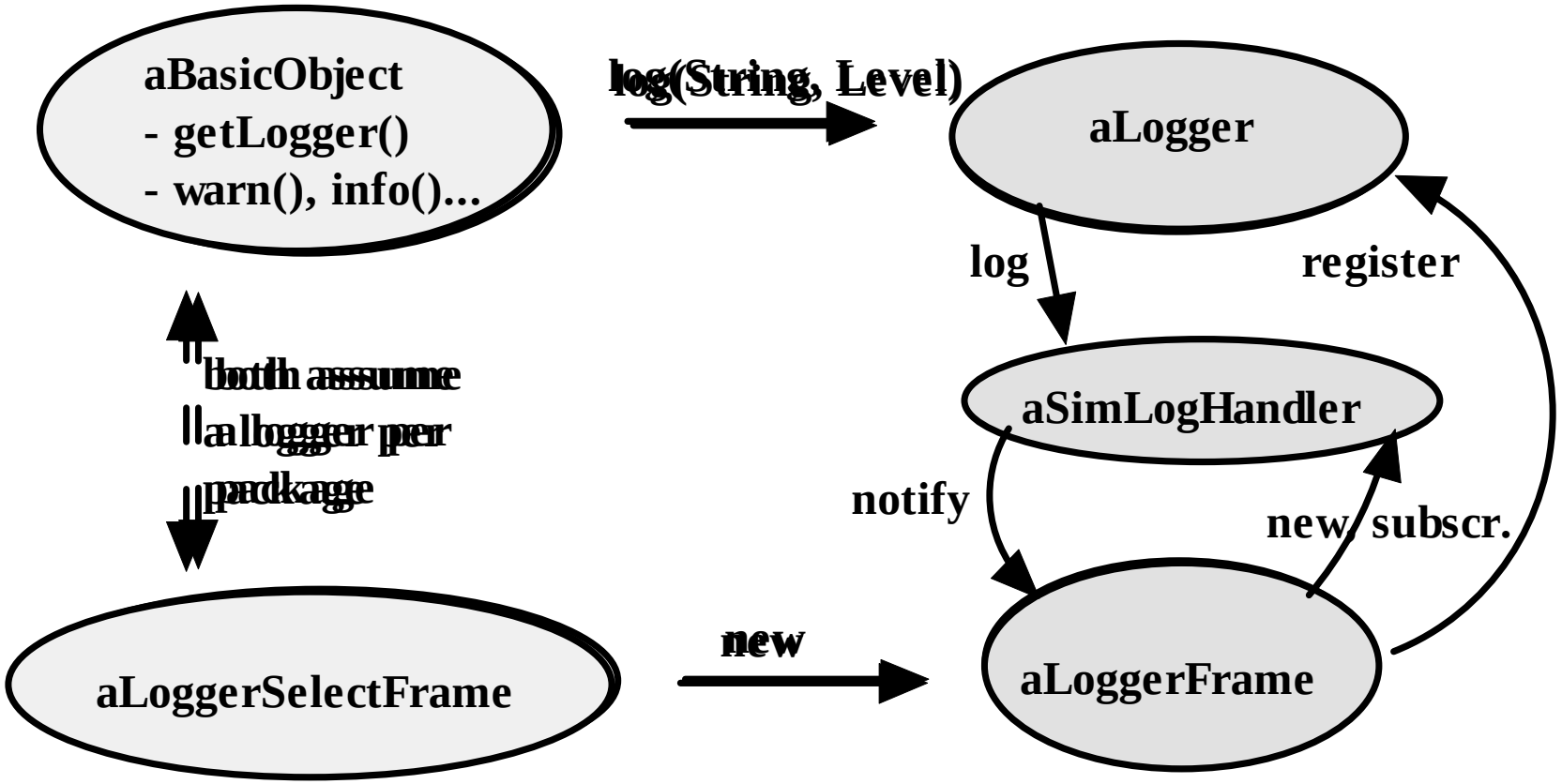
The basics: SimObject



The basics: Logging



The basics: Logging in TermSim



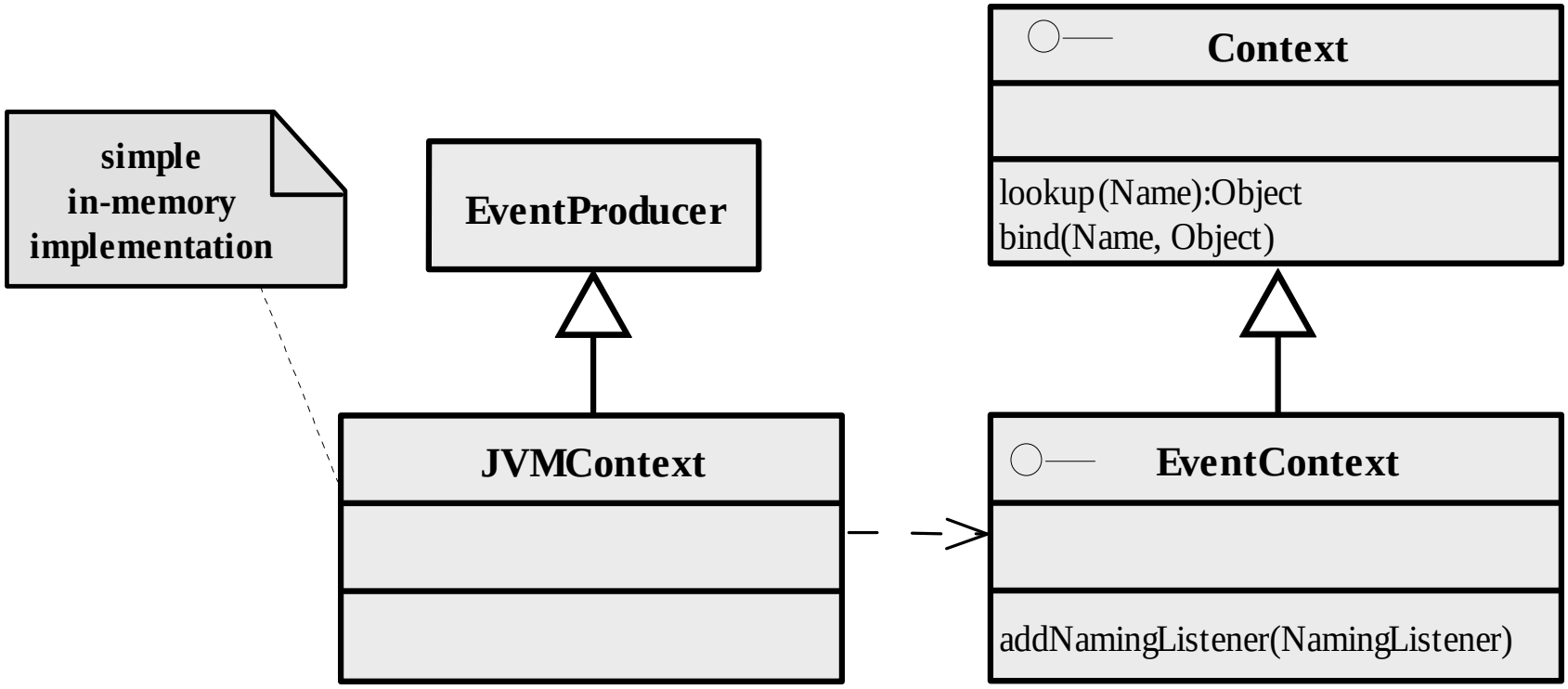
The basics: Context and naming

- Model objects may need to locate each other:
  - Resource controller
  - Count all instantiated Customer objects
  - …
- Model objects may need to be notified when objects are created or removed
- A complex model may be large, so a naming solution should:
  - Scale
  - Be persistent

The Basics: javax.naming

- The Java Naming and Directory Interface (JNDI) is a standard Java extension that suits our needs.
- JNDI consists of a number of naming interfaces:
  - Context (naming only)
  - EventContext (fires events on change)
  - LDAPContext (allows querying)
- All major directory providers can be accessed over JNDI

The Basics: Naming in TermSim

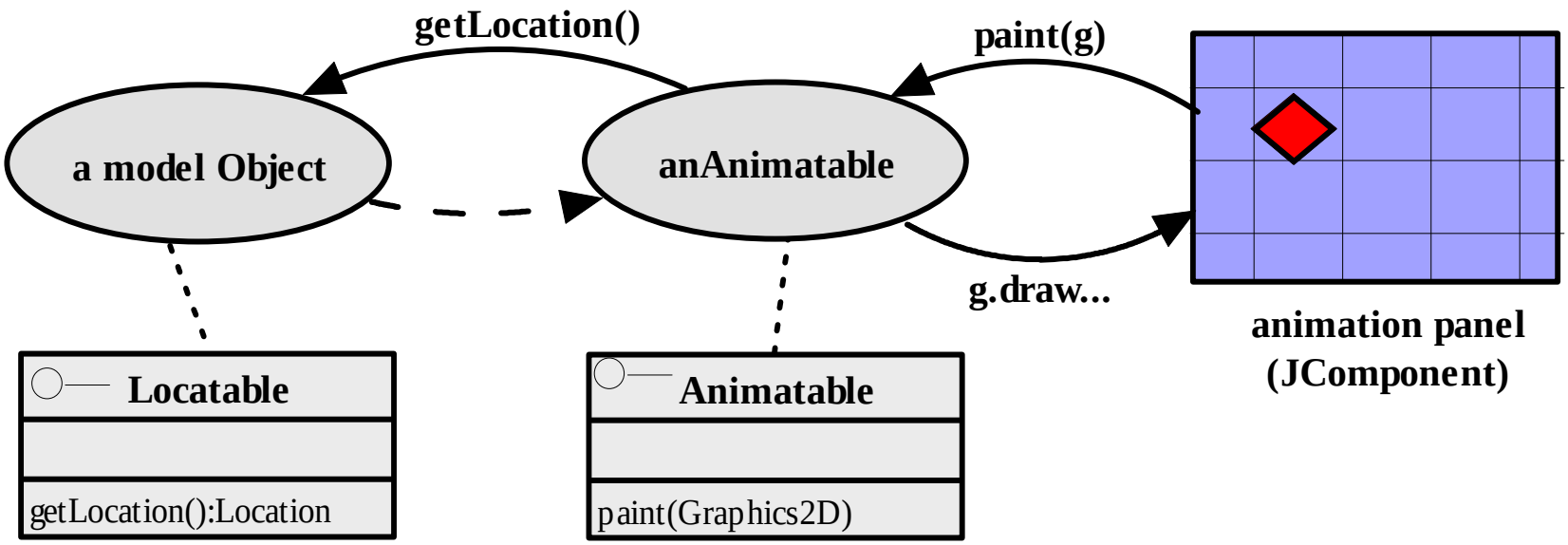


jndi.properties

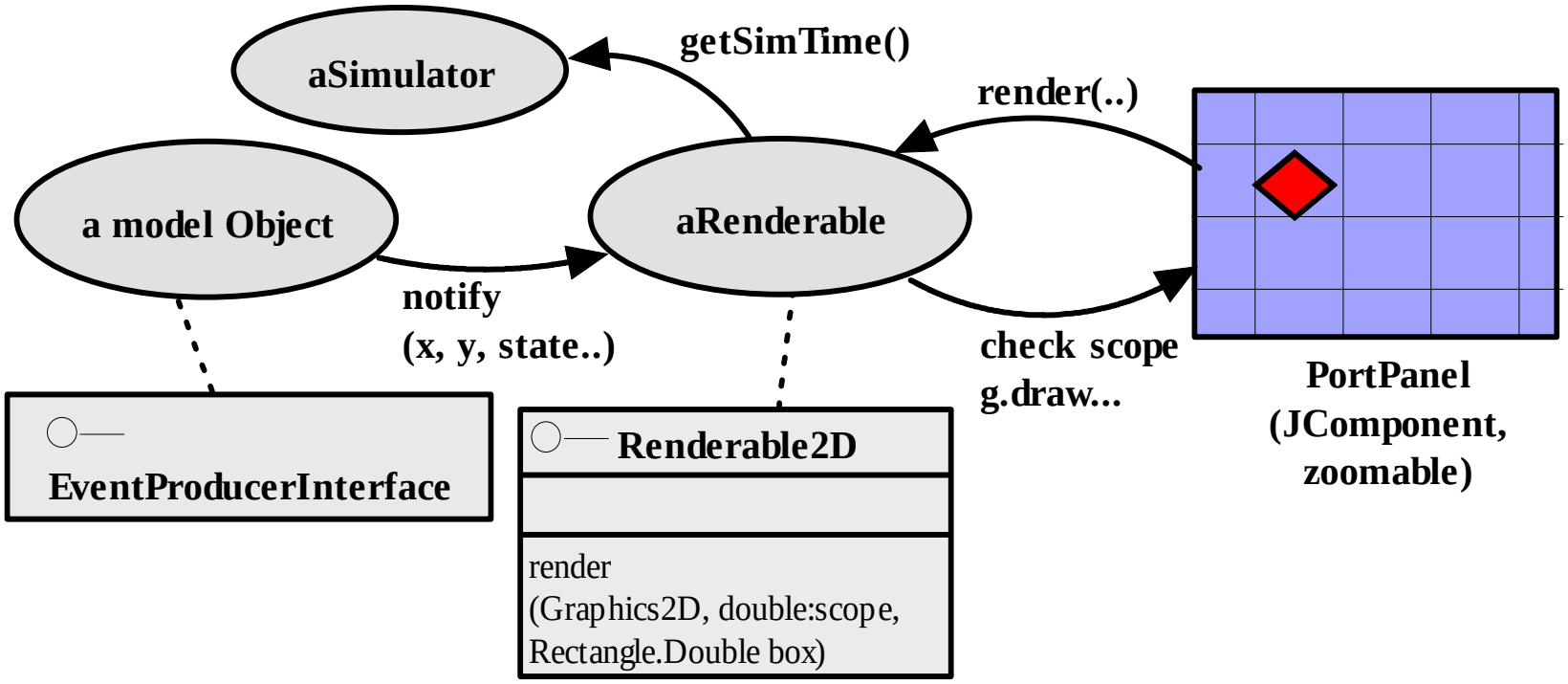
Animation

- Animation in TermSim has been developed separate from DSOL animation
- It is still working in progress, despite nice pictures:
  - Animation is underperforming (TermSim at least)
  - No simple, complete animation paradigm for distributed environments has been developed
- However, ingredients are there, cooking will take a few weeks

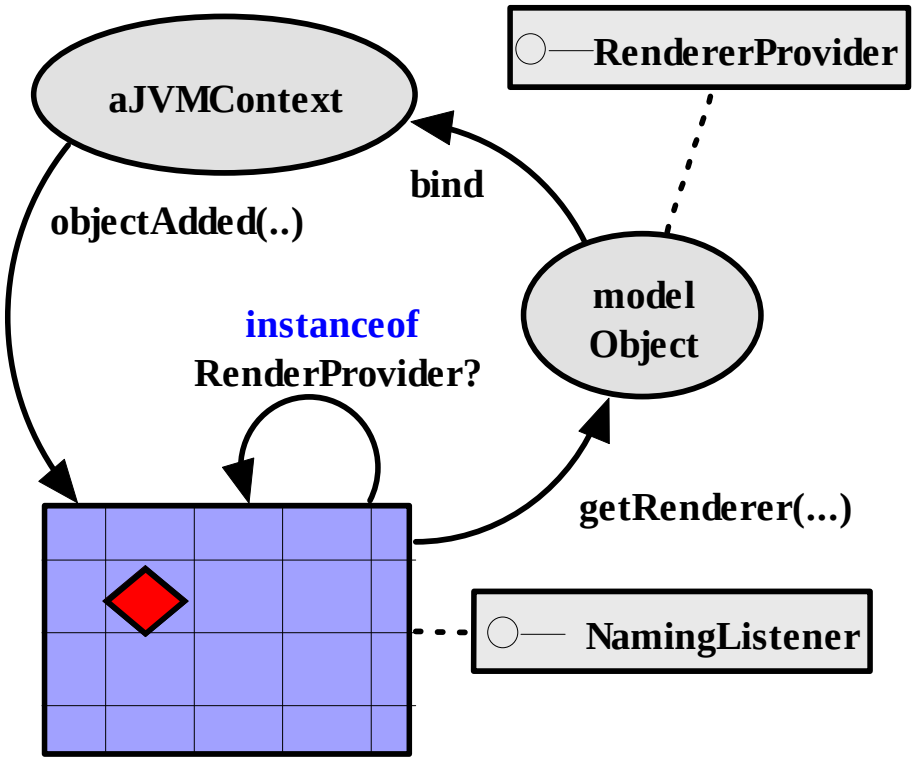
Animation: rendering in DSOL

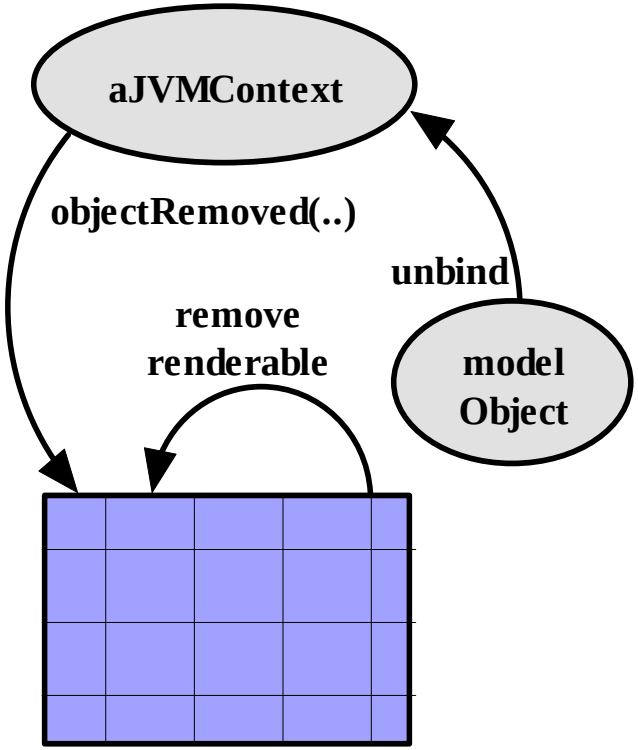


Animation: rendering in TermSim



Animation: initializing in TermSim





Animation: next steps

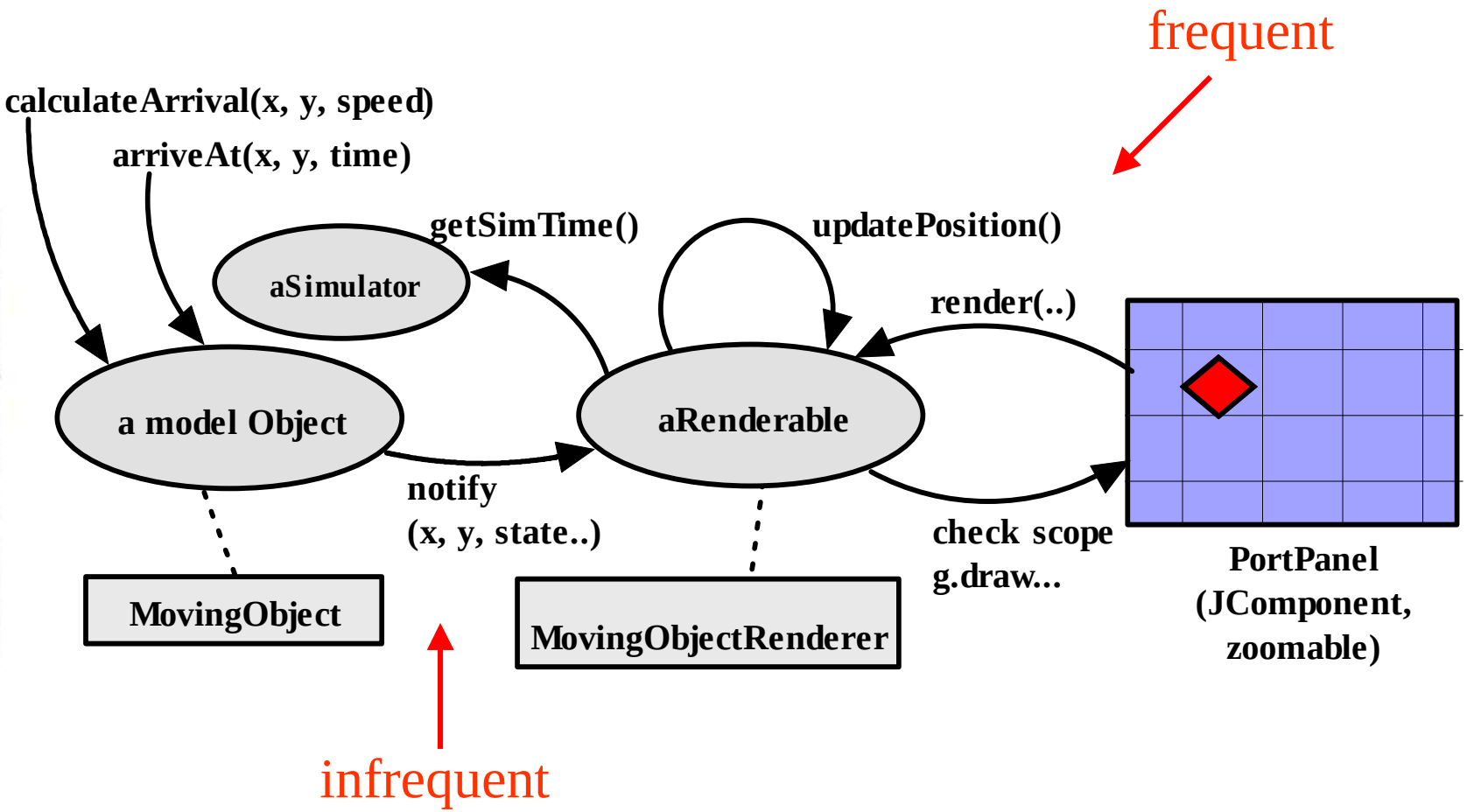
- A Renderable2D object should not contact the simulator: far too much overhead. SimTime should be provided by panel
- Graphics2D object can provide scope and clip information
- A scenegraph approach may be needed to deal with large quantities of renderables:
  - Provides visibility hierarchy
  - Provides visibility ranging

Basic Packages: MovingObject

- MovingObject(Renderer) provides basic animation aspects:
  - Location, rotation
  - Size, shape
  - Speed, movement
- MovingObject keeps own state. State is updated using events.
- Users schedule arrival time and location
  - Arrival time is calculated in advance
  - Arrivals can be replaced by others

Show methods

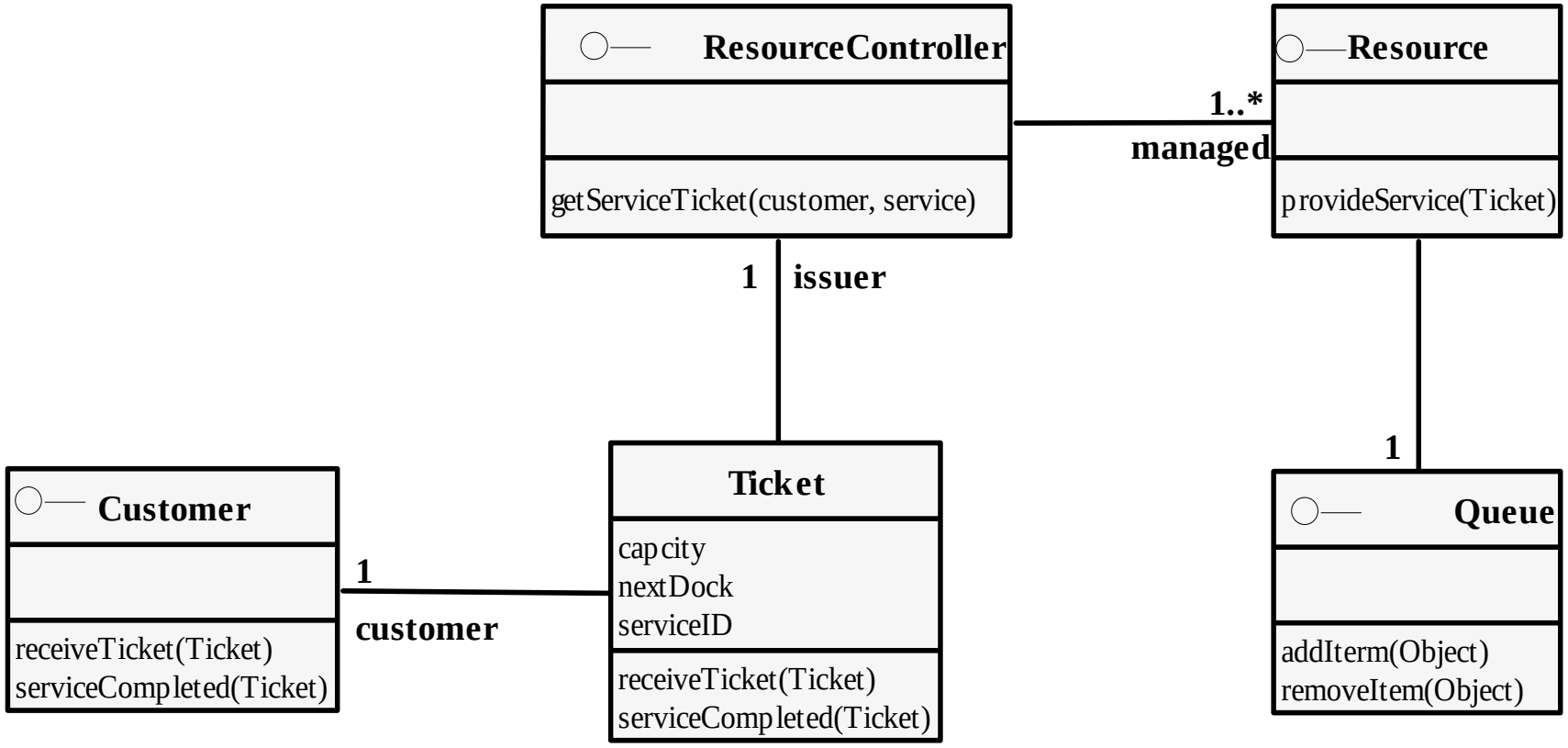
Basic Packages: MovingObject



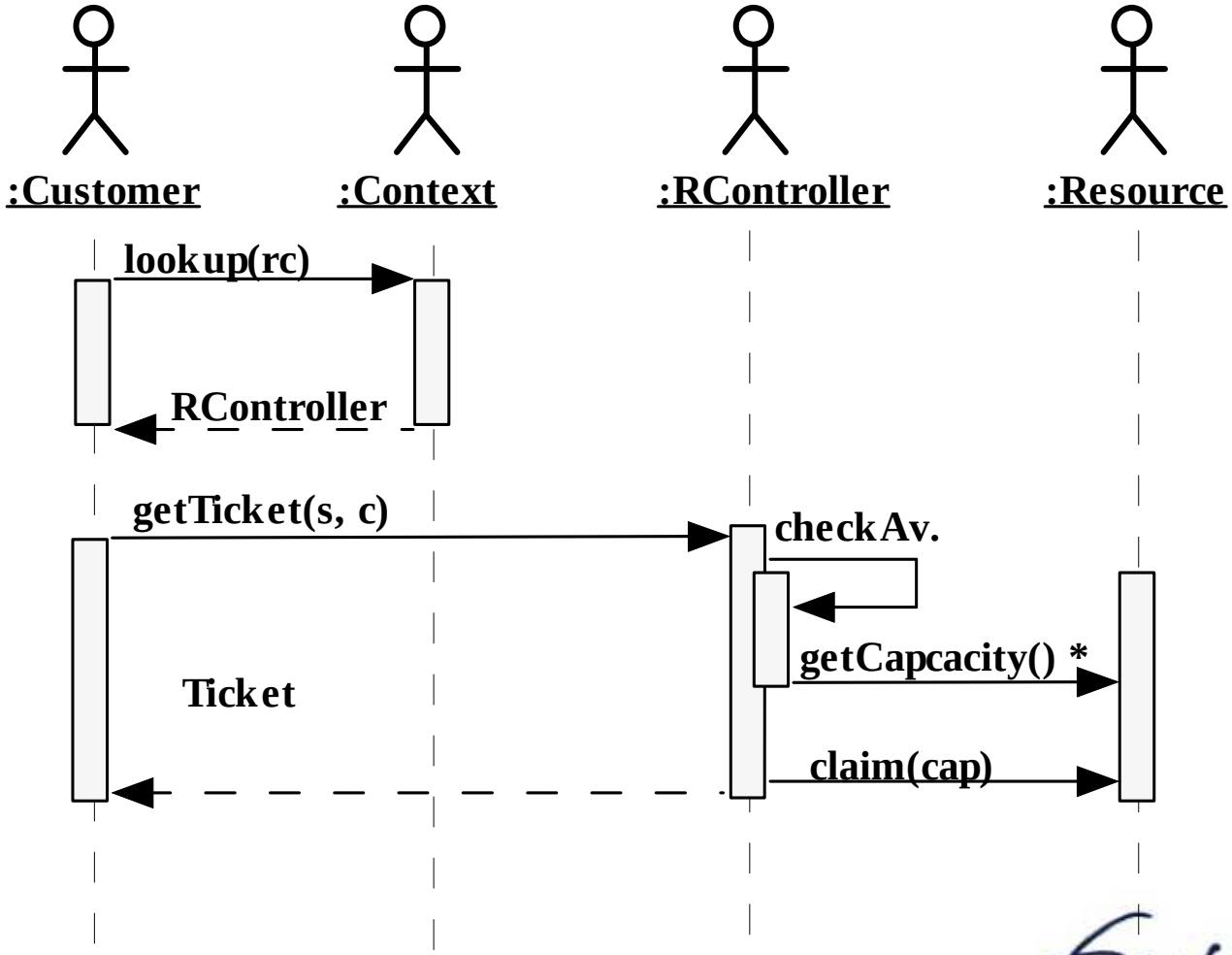
Services

- Many models involve service systems, consisting of:
  - Servers, resources & controllers
  - Customers, entities
  - Queues
- An attempt has been made to implement an OO library for the basic service:
  - Interfaces for Resource, Customer, Controller
  - Separate responsibilities (no flow)

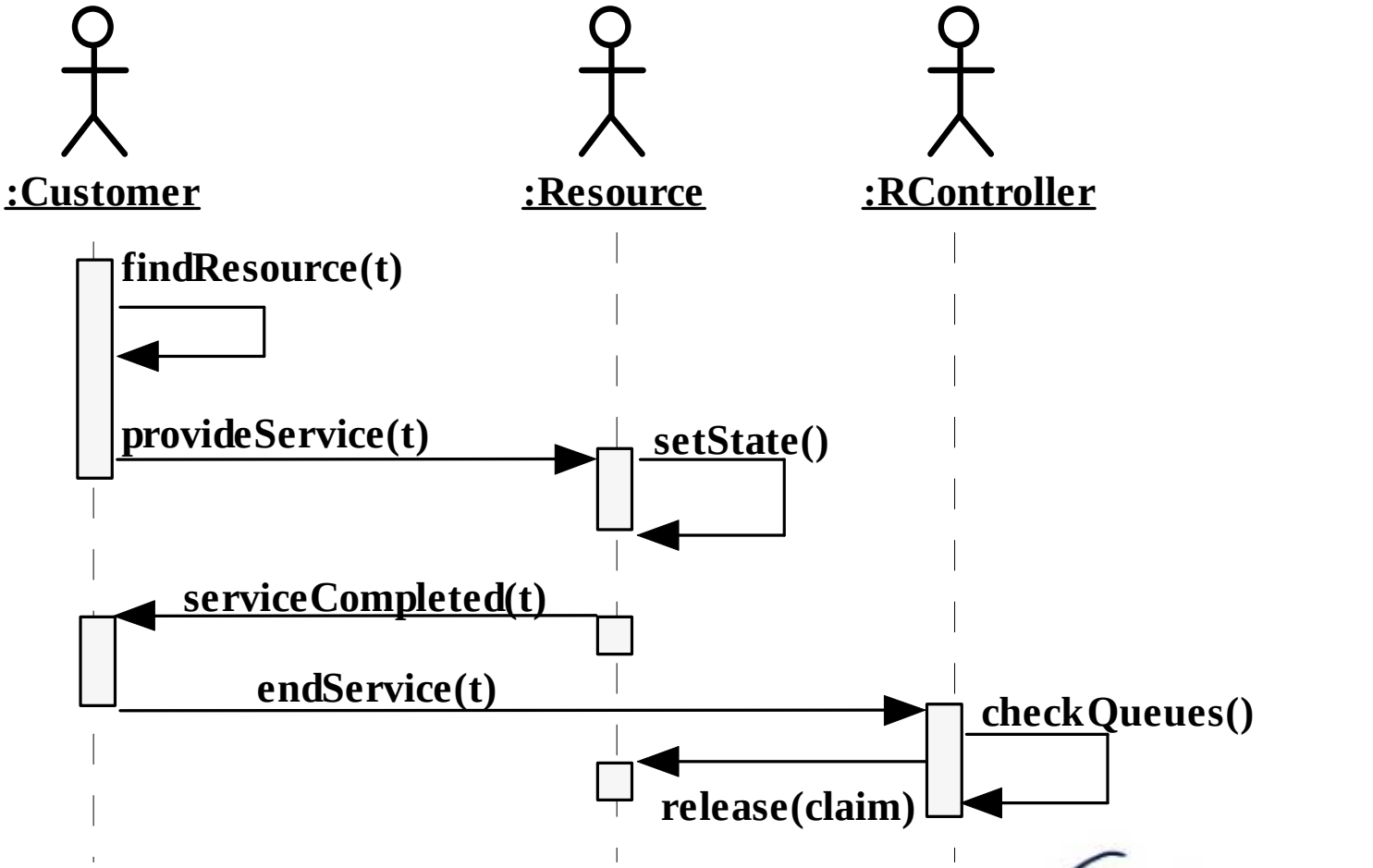
Services: UML



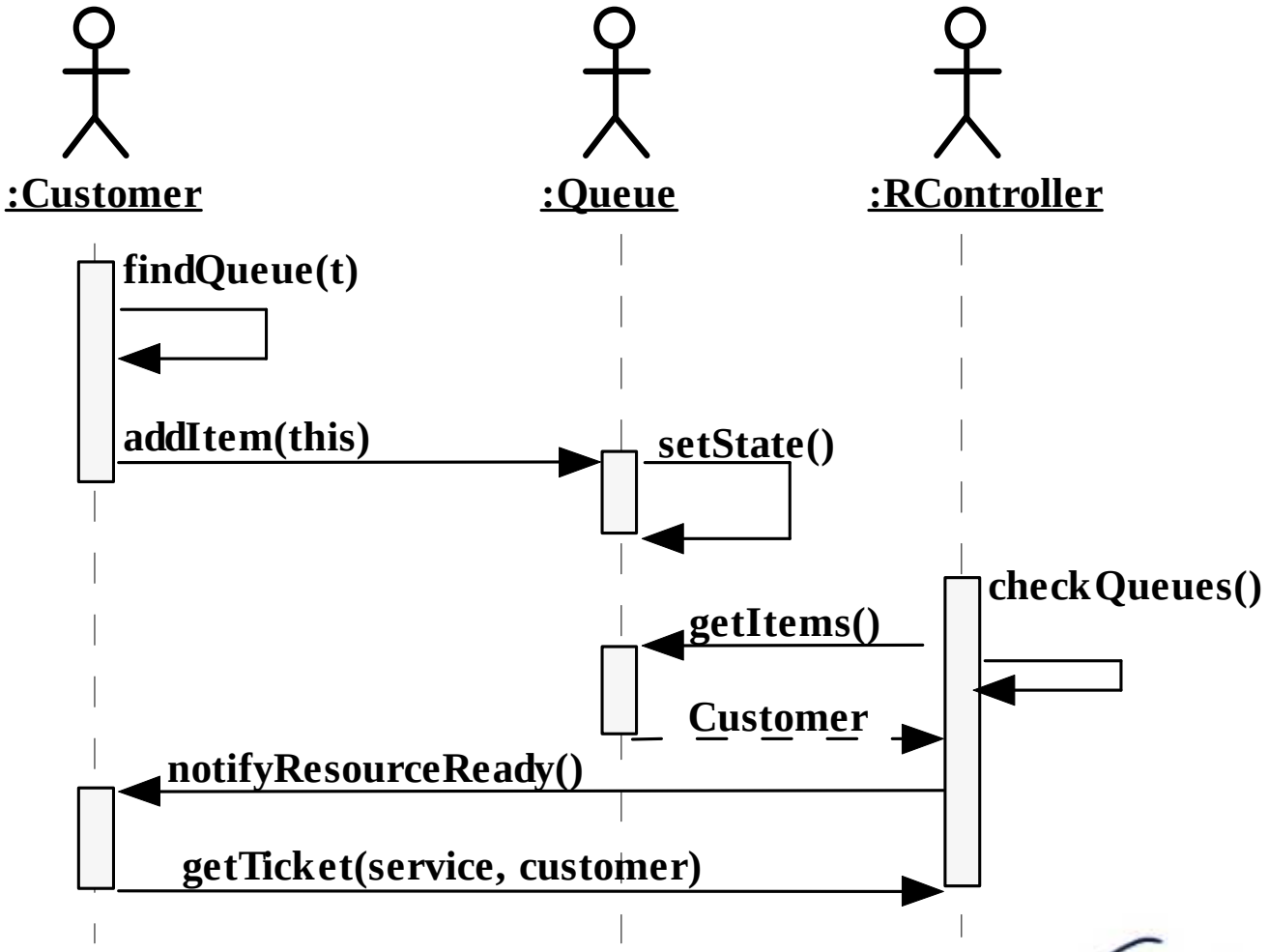
A successful service request



Service provision

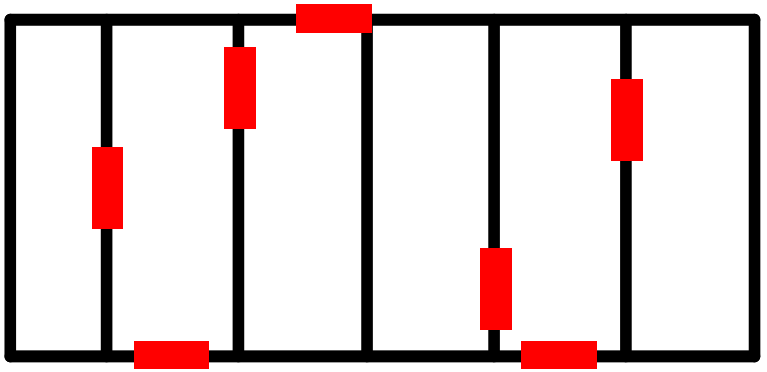


Service queueing

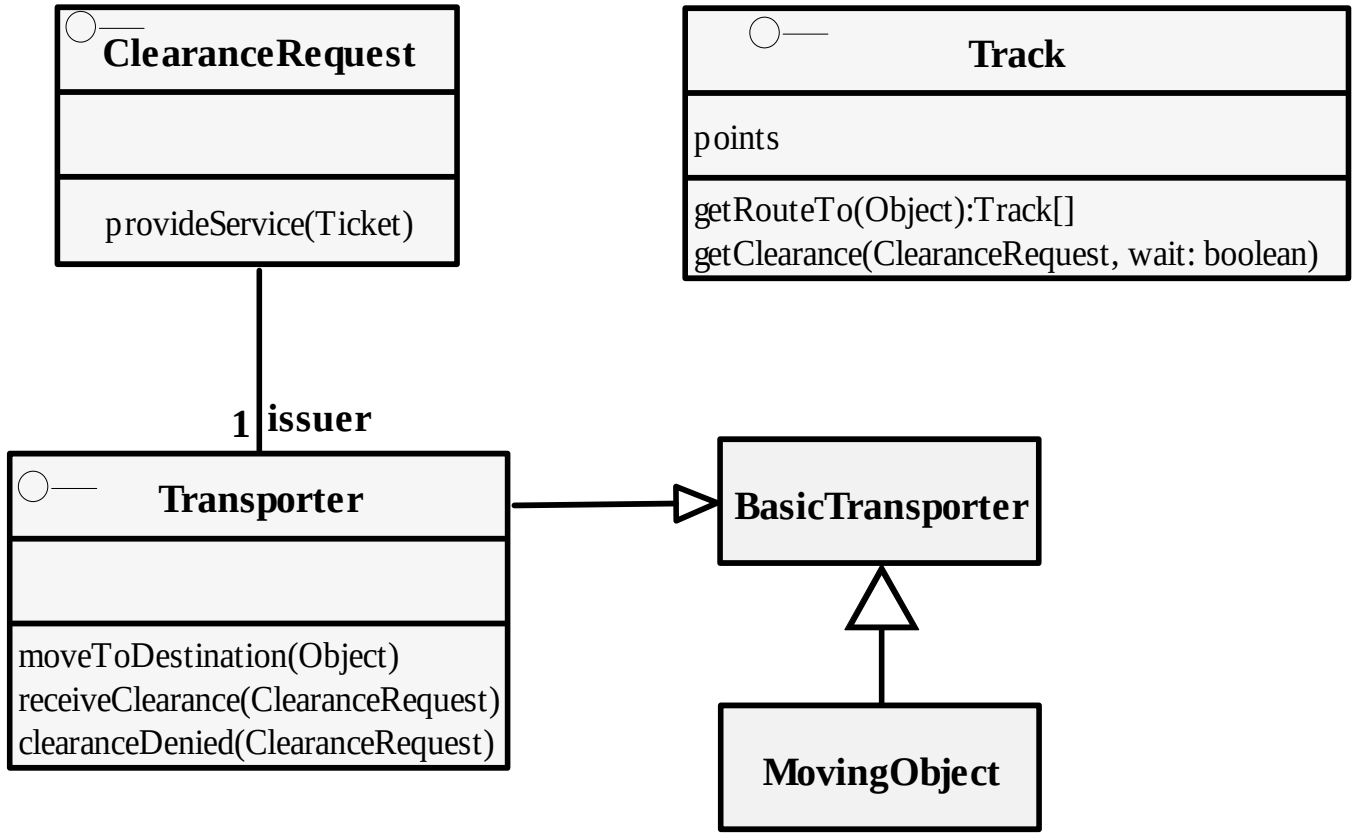


Tracks: AGV semaphores

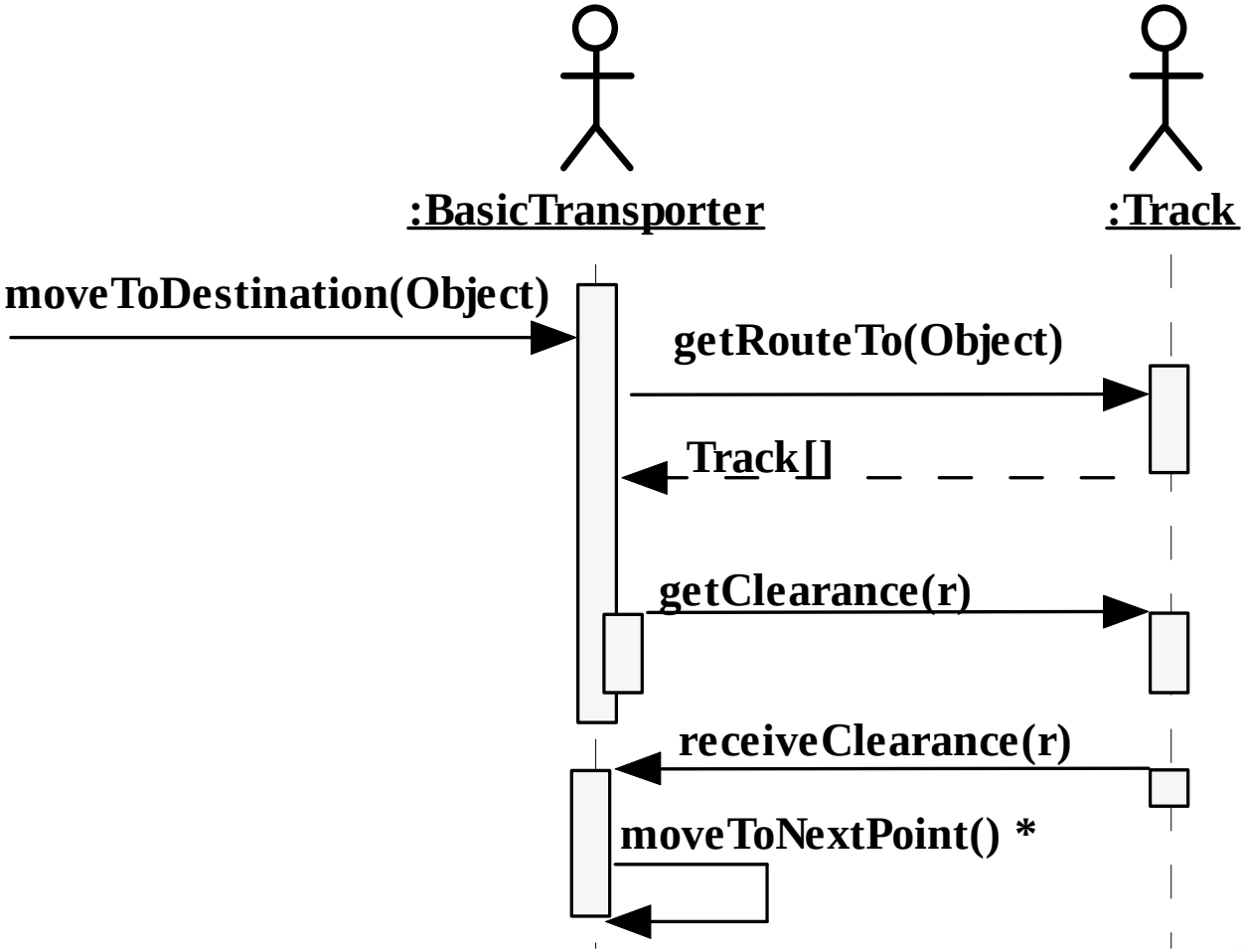
- MovingObjects assume road is clear. This is however not always the case
- Let's assume that non-moving objects are not in line of moving objects (emergency lane principle)
- In a track system, only crossings are now dangerous spots: a clearance system is needed



Transporter & track



Transporter in action



Features: DB experiment

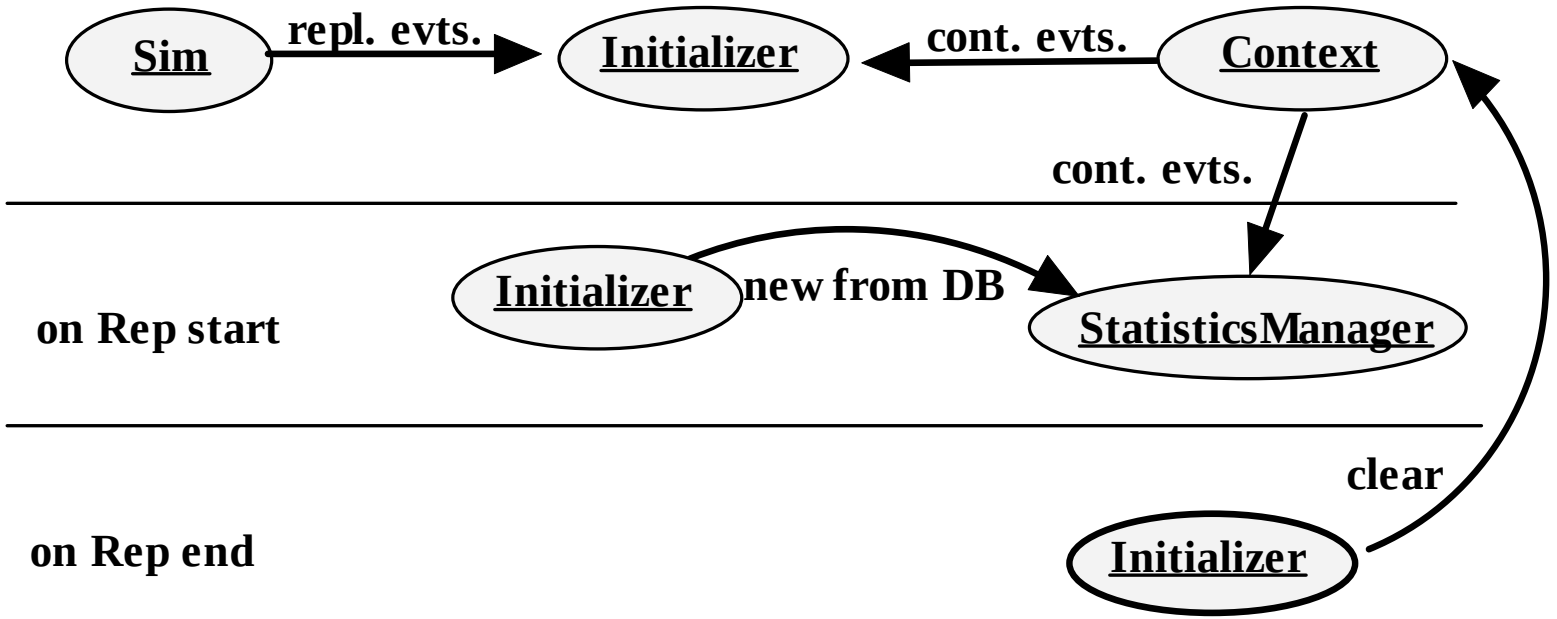
- Enables retrievement of experiments from a database over JDBC
- Enables retrievement of a model's environment (properties)
- Enables persistency of statistic output
- May allow model definition (cmp. XML)
- A DB util package has been developed to support DB experiment

DB Util

- The DBConnection interface defines a variety of DB utility methods
- JDBCConnection is general implementation
- Extensions to ease specific protocols (ODBC, PostgreSQL,…)
- Let's have a look!
  - PrintProperties
  - Show tables

Features: Initialization

- After each replication, statistics need to be refreshed, context needs to be cleared



Features: Listener management

- Java know constructors but no destructors
- Instead, Java removes objects by garbage collection
- However:
  - As long as (non-weak) references exist to an object, this object won't be deleted
  - Garbage piling can easily go unnoticed when memory just keeps gently rising
- So: beware of garbage!

Listener Management: common trap

```
public class SimController
{
    private EventListenerInterface listener = new MyListener();

    public SimController(EventProducerInterface sim)
    {
        sim.addListener(listener, sim.EVT_TIME_CHANGED);
    }

    public void end(SimulatorInterface sim)
    {
        sim.removeListener(listener);
    }

    private class MyListener implements EventListenerInterface
    {
        public void notify(EventInterface evt)
        {
            System.out.println("Received evt: " + evt);
        }
    }
}
```

Questions