Designing the structure of a service-oriented application

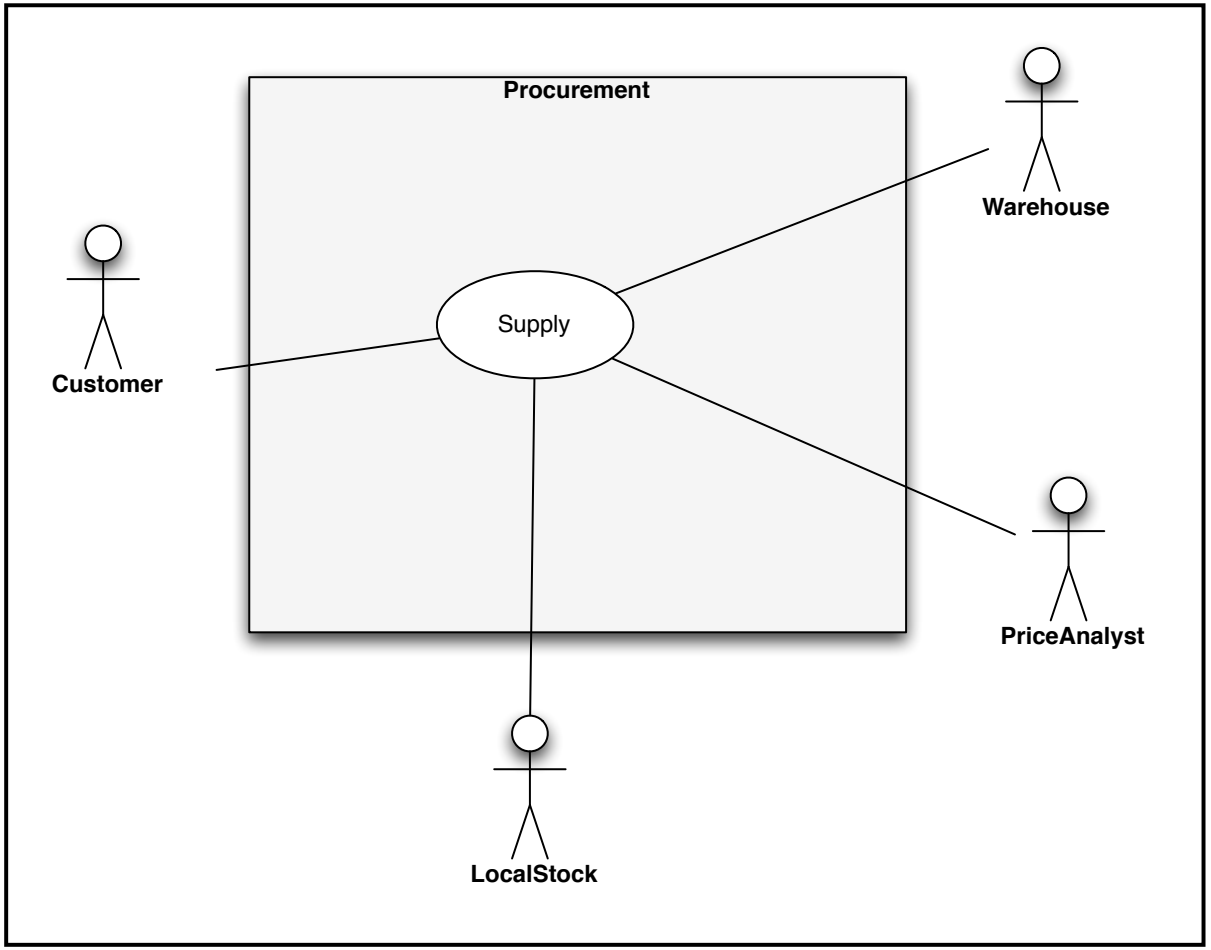
Laura Bocchi
bocchi@mcs.le.ac.uk

Agenda

- UML use case diagrams
- Use case diagrams for service-oriented applications
- SRML: an overview of the module structure

Use Case Diagrams (recall)

- System boundaries
- Actors
- Use cases
- Associations between one actor and one use case
- There are also other aspects such as the associations between use cases (extension, inclusion, generalisation) and generalisation between actors. We do not consider them here.

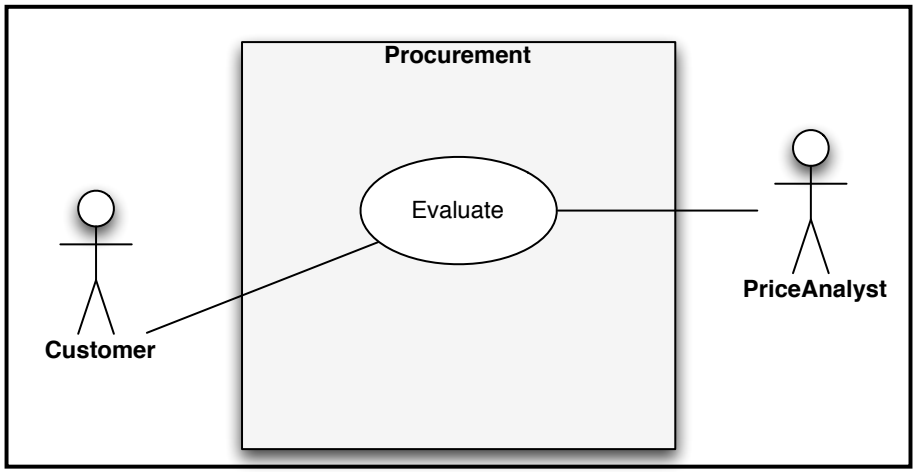


Use Cases and scenarios (1/2)

- The set of functionalities (use cases) of a system can be derived by creating a number of scenarios
- A scenario involves one or more actors and can be described as an interaction between the involved actors and the system
- E.g., Scenario 1: (1) the customer asks for a quote, (2) the system gets a quote using pricing analyst, (3) the system returns the quote
- E.g., Scenario 2: (1) the customer asks for a quote, (2) the system gets a quote using the price analyst, (3) the product is no longer on the market, the system return a warning message to the customer

Use Cases and scenarios (2/2)

- A use case represents a collection of scenarios that fulfill a common goal from the perspective of the user
- E.g., the use case "evaluation" collects Scenario 1 and Scenario 2



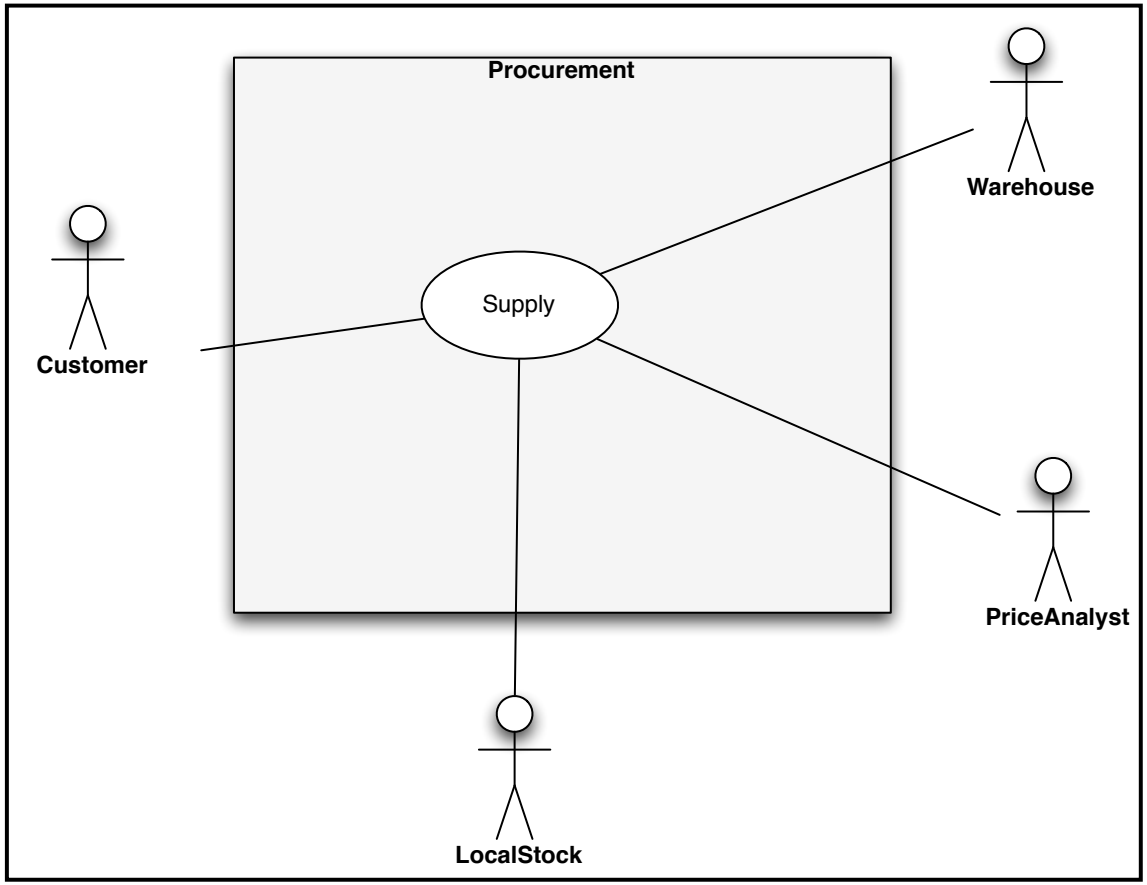
Primary & Secondary Actors

- An actor can be a person, a device, a system, etc.
- An actor can be a primary actor or a secondary actor
- A primary actor
  - acts on the system
  - initiates an interaction with the system
  - uses the system to fulfill his/her goal
- A secondary actor
  - is acted on/invoked/used by the system
  - helps the system to fulfills its goal

Primary & Secondary Actors (example)

Discussion:

which actors are primary and which are secondary?



Agenda

- UML use case diagrams
- Use case diagrams for service-oriented applications
- SRML: an overview of the module structure

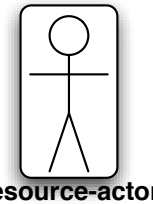
Use-Case for SOA

- We refine the notion of system boundary
- We refine the notion of use case
- We define different types of primary and secondary actors

SRML notes: section 2

Secondary actors in a SOA

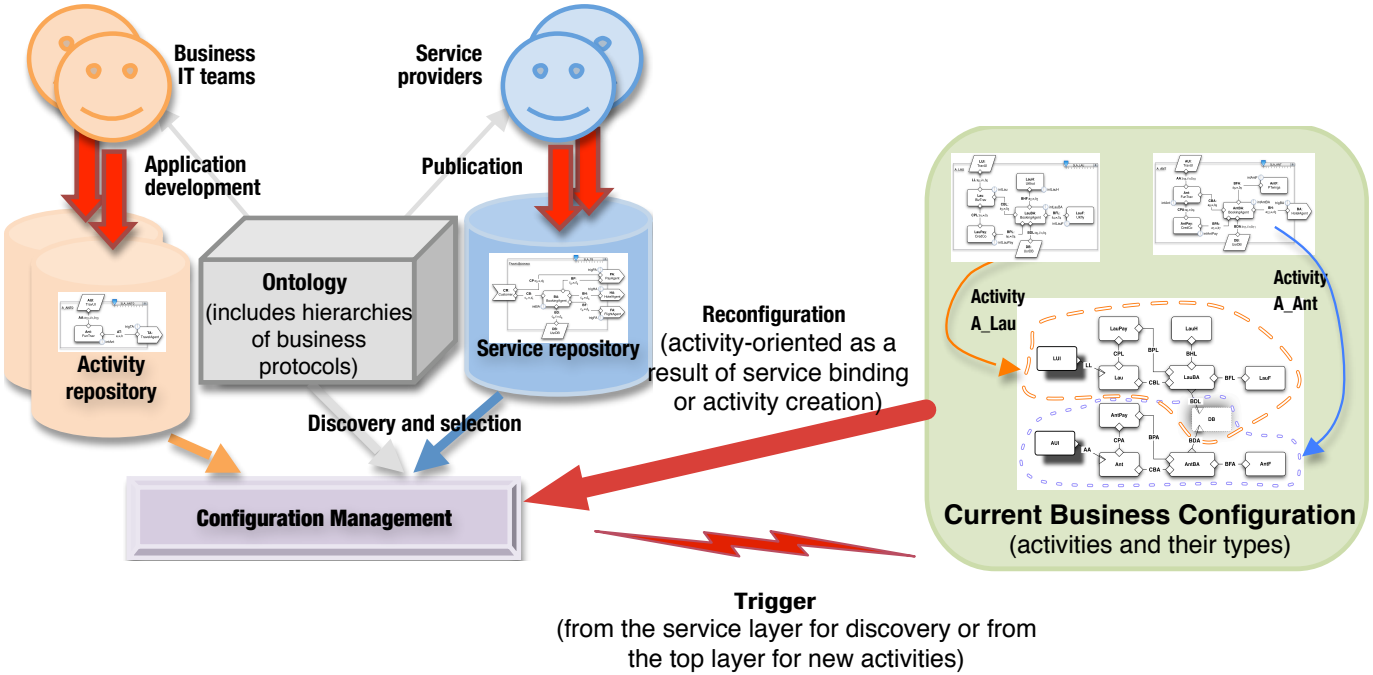
service-actor



resource-actor

- **Secondary Actors**: represent entities to rely on in order to achieve the underlying business goal
  - **Service-actors**: represent a functionality to be provided on the fly (typically change from instance to instance)
  - **Resource-actors**: are statically bound and persistent (they are the same for all the instances)

Activities vs Services



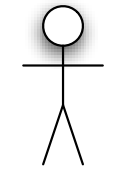
activities:

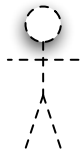
- applications that use but do not provide services
- developed to meet requirements of a specific business organisation

services:

- applications that may use and do provide a service
- developed to be published and discovered at run-time

Primary actors in a SOA



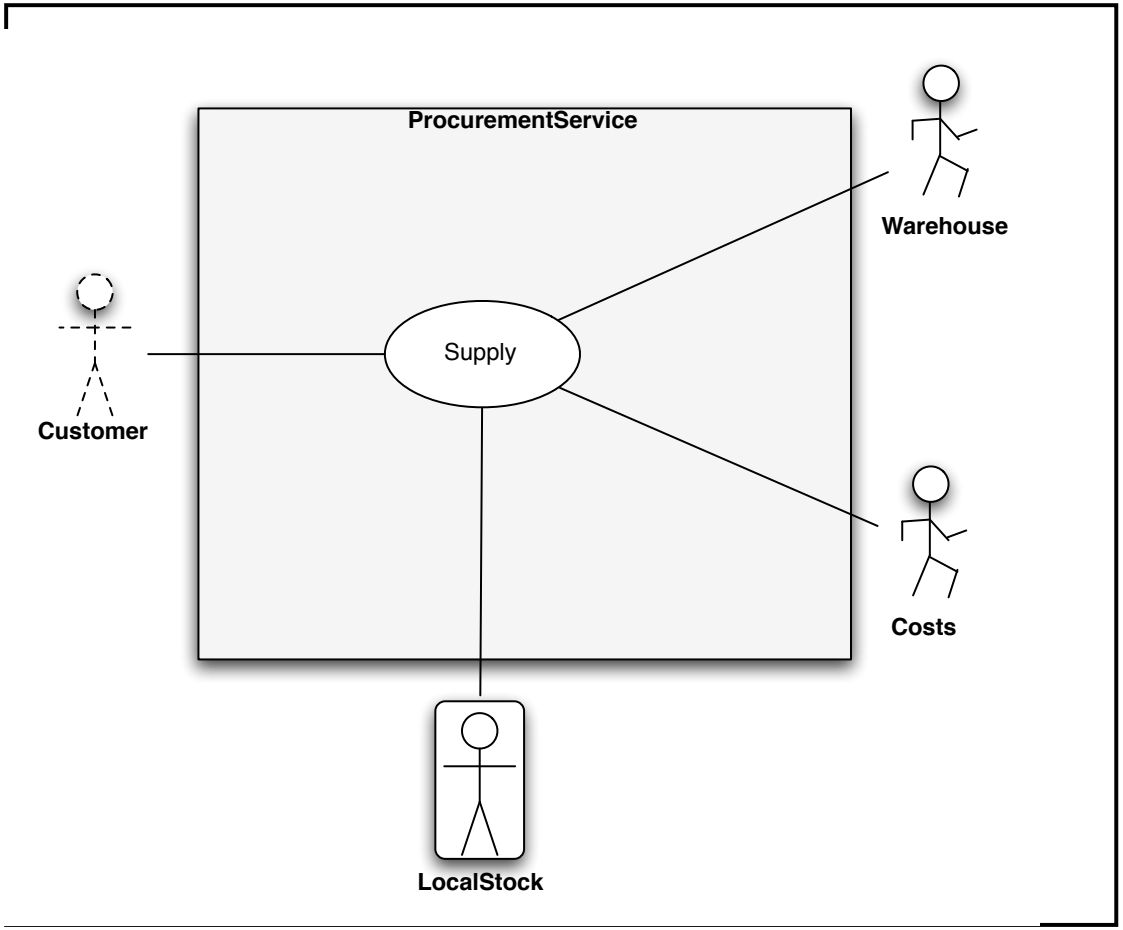


- **Primary Actors**: represent entities that initiate the use case and whose goals are fulfilled through the successful complention of the use case
  - **User-actors**: instantiate an activity
  - **Requester-actors**: are service requester that discovery/instantiate a servicecase diagrams: overview of usage requirements for a system to be built

Different Types of Actor (example)

Discussion:

determine the user, requester, service and resource actors



System boundary and use cases in SOA

- In a service-oriented context there is no "system" but a number of services and activities
- The system boundary represents the scope as a logic unit developed by the same company
- The scope may encompass entities that are physically distributed but are assembled together at design time

- A service/activity describes a single usage requirement thus results in one use case

Agenda

- UML use case diagrams
- A profile for use case diagrams for service-oriented applications
- SRML: an overview of the module structure

Modelling in SRML

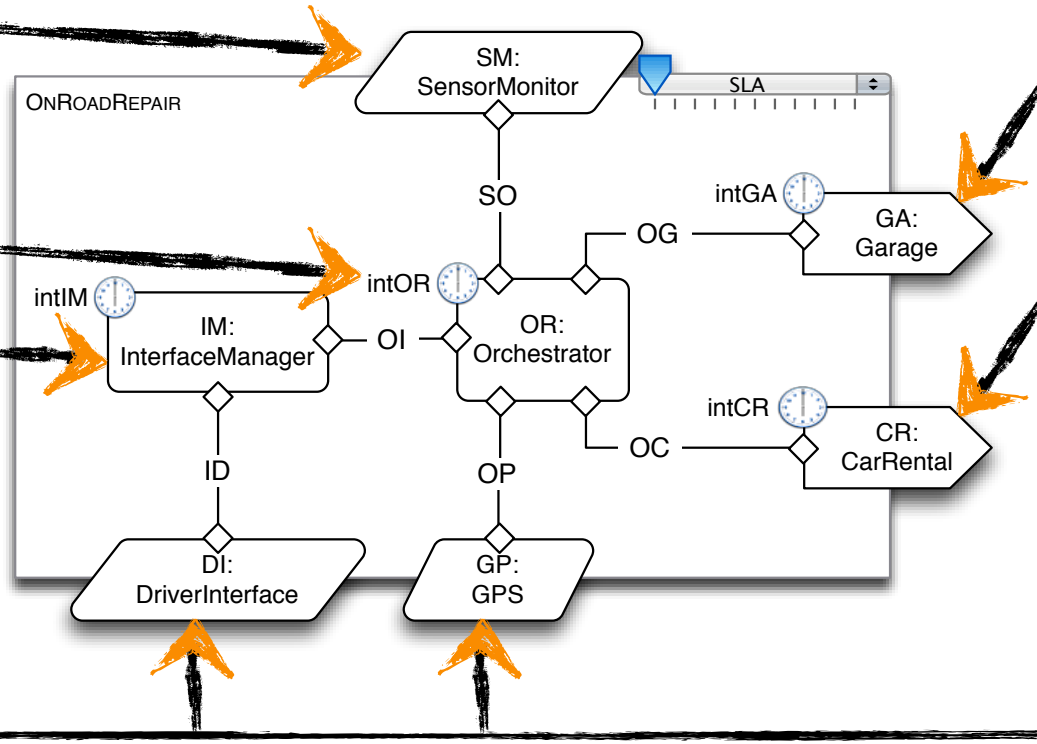
- SRML is a high level modelling language for service-oriented systems with a formal semantics
- SRML provides primitives for modelling composite
  - services
  - activities
- What do we compose?

A SRML activity module

An Activity Module is launched by the top layer in a traditional way (no discovery)

One serves-interface: interface to the user that triggers the activity

A number of component-interfaces describing the orchestration

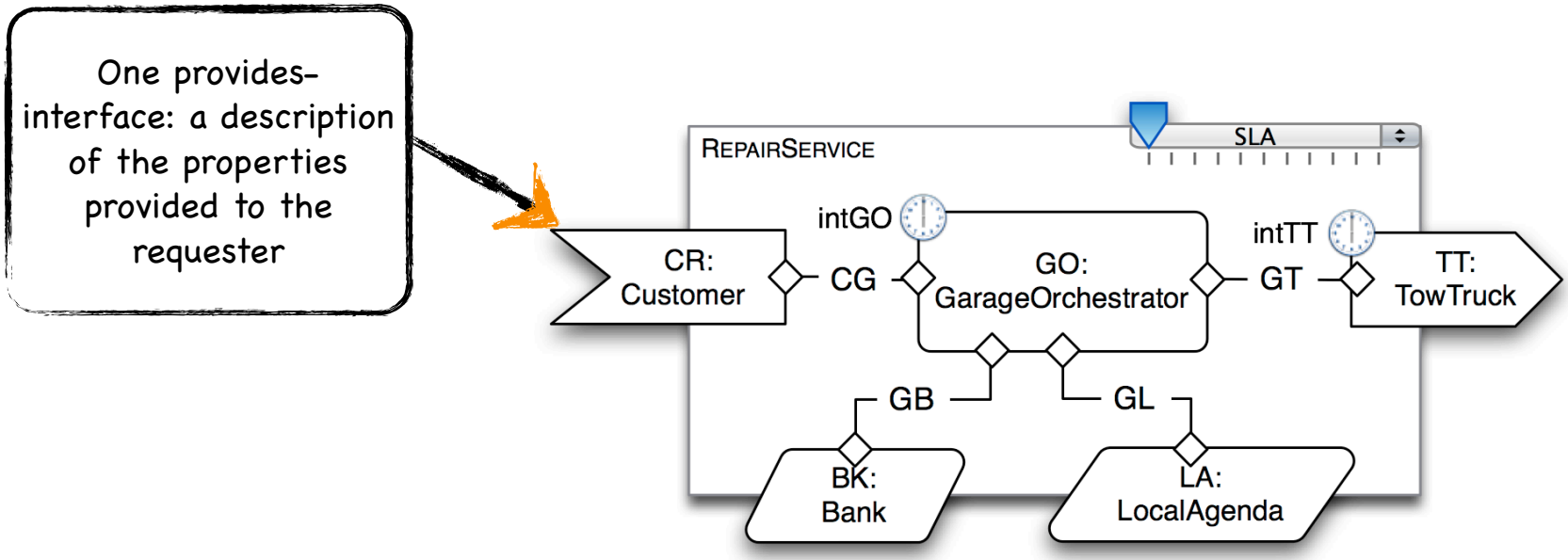


A number of requires-interfaces describing the properties expected by external services discovered at run-time

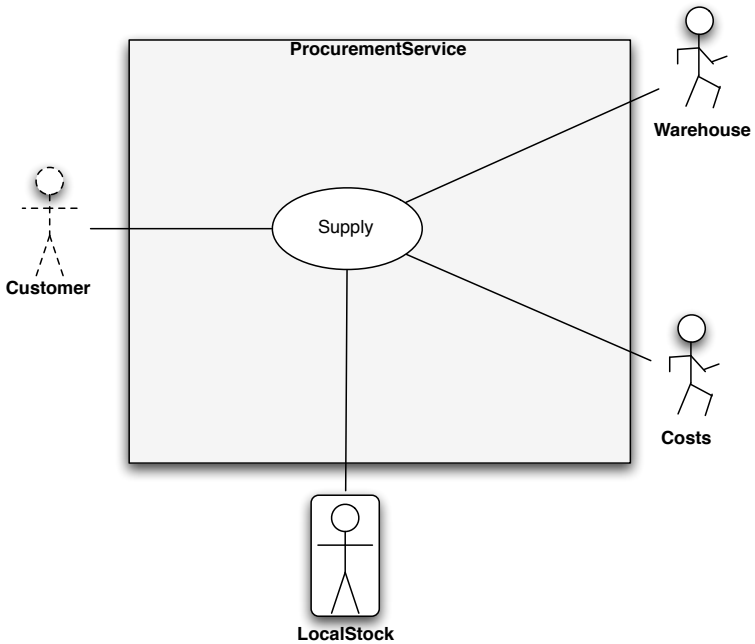
A number of uses-interfaces (e.g., GP, DI) representing persistent resources (no discovery)

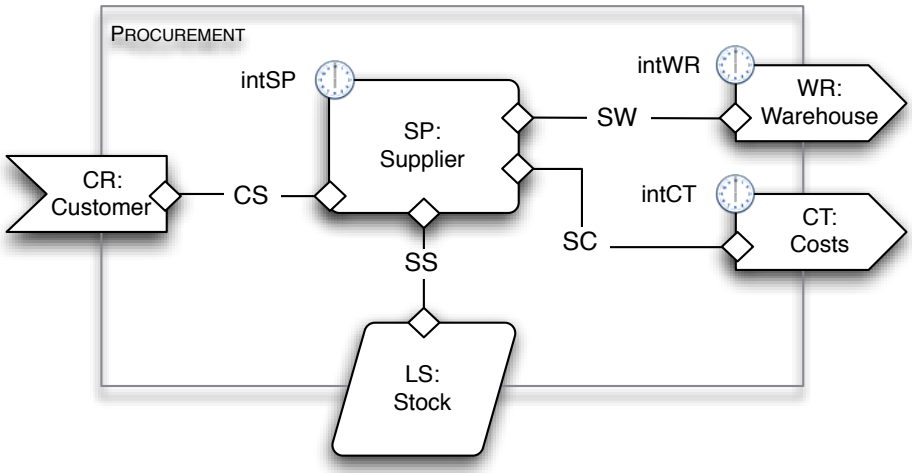
A SRML service module

A Service Module is published, discovered and invoked by a service requester



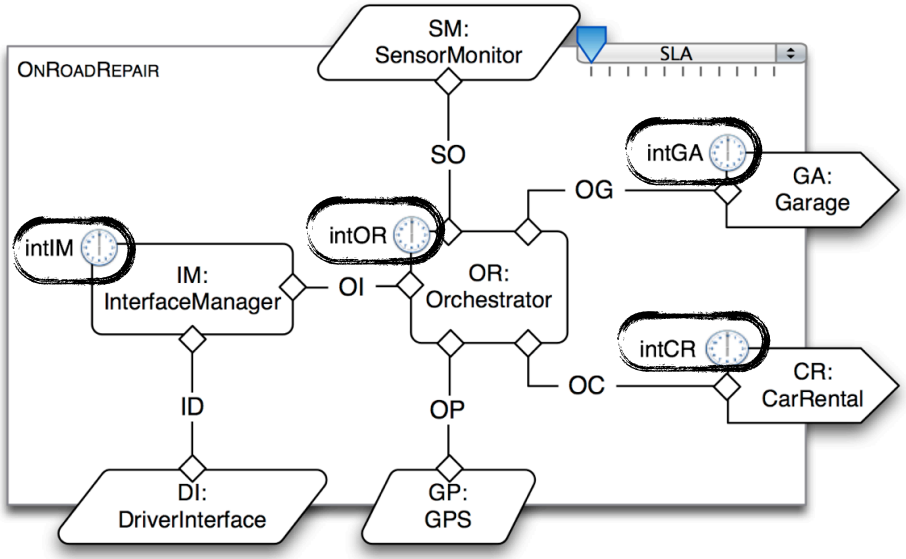
Use-Case for SOA





The internal structure, in terms of components, of the module derived from the Use-Case diagram depends on the components we have already available

Textual representation of modules



The module can have parameters

```
MODULE OnRoadRepair (carID:vehicleId)

SERVES
  SM: SensorMonitor
COMPONENTS
  OR: Orchestrator
        intOR init: s=INIT
        intOR term: s=FINAL
  IM: InterfaceManager
USES
  GP: GPS
  DI: DriverInterface
REQUIRES
  GA: Garage
        intGA trigger: default
  CR: CarRental
        intCR trigger: bookGarage!
WIRES

  ...
```

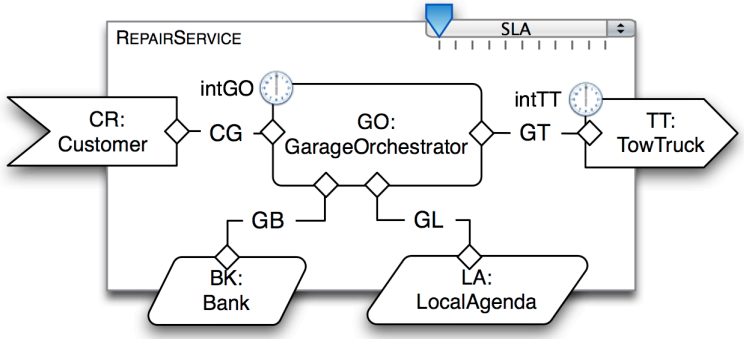
- A module defines some internal reconfiguration policies
  - Triggering event for the discovery of each requires-interface
  - Initialization: assignments and state
  - Termination condition

Internal policies

```
MODULE OnRoadRepair (carID:vehicleId)
═══════════════════════════════════════
  SERVES
  ──────────────────────────────────
    SM: SensorMonitor
  COMPONENTS
  ──────────────────────────────────
    OR: Orchestrator
          intOR init: s=INIT
          intOR term: s=FINAL
    IM: InterfaceManager
  USES
  ──────────────────────────────────
    GP: GPS
    DI: DriverInterface
  REQUIRES
  ──────────────────────────────────
    GA: Garage
          intGA trigger: default
    CR: CarRental
          intCR trigger: bookGarage!
  WIRES
  ──────────────────────────────────
    ...
```

The discovery of GA is triggered by the first interaction with GA

The discovery of CR is triggered by the request to book the garage

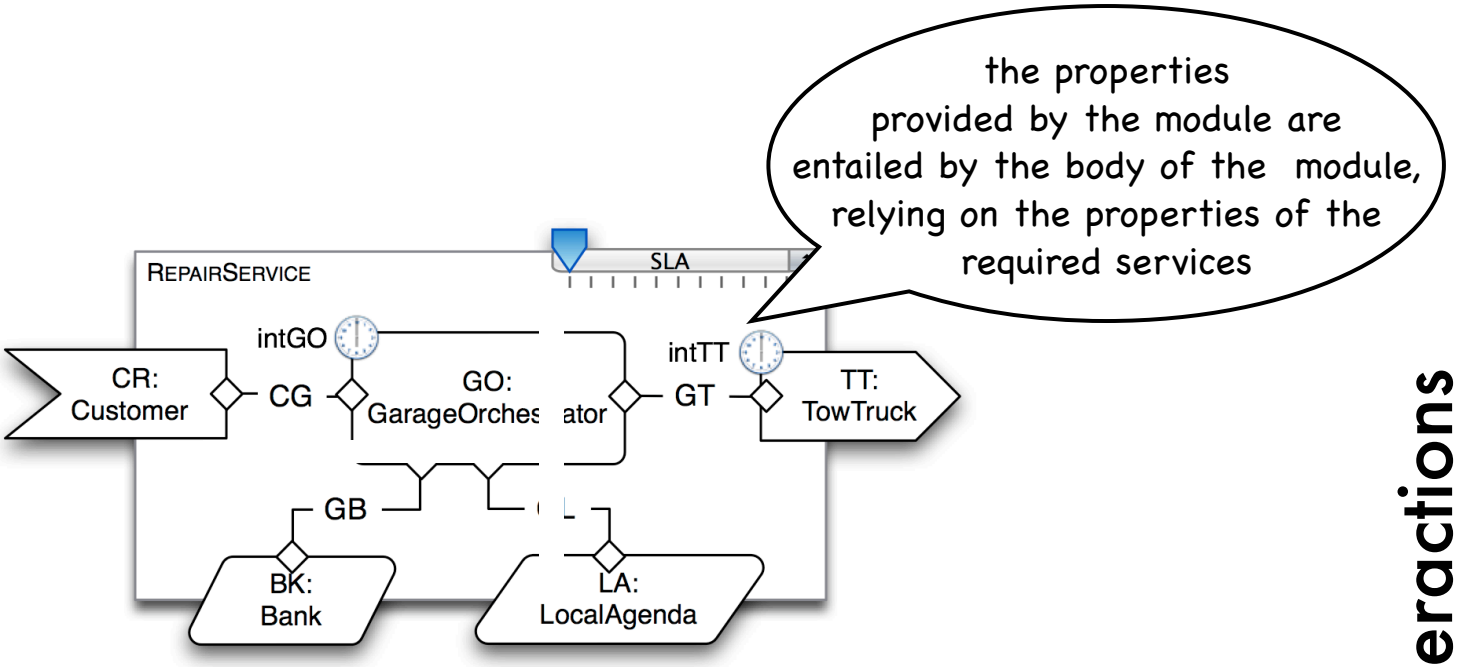


Discussion:

Why we do not define initialisation and termination condition for uses-interfaces?

Why we do not define triggering conditions for provides-interfaces?

Specification Languages 1/2



logics of interactions

Business Roles
(Garage, Orchestrator)

e.g., the component **GO** is an instance of the business role **GarageOrchestrator**

Business Protocols
(Customer, TowTruck)

e.g., the provides-interface **CR** is an instance of the business protocol **Customer**

Layer Protocols
(Bank, LocalAgenda)

e.g., the uses-interface **BK** is an instance of the layer protocol **Bank**

Interaction Protocols
(used by the wires CG, GT, GB, GL)

Summary

- Use cases and scenarios
- Primary and secondary actors
- Services vs activities
- System boundaries, use cases and actors in SOA
  - User, requester, resource and service actors
- Activities and services in SRML
  - Graphical and textual notation, internal policies
- From use cases to SRML module structure