Advanced System Design

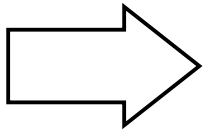
Modelling Service-Oriented Systems with SRML

Laura Bocchi
bocchi@mcs.le.ac.uk

What is this part of the course about?

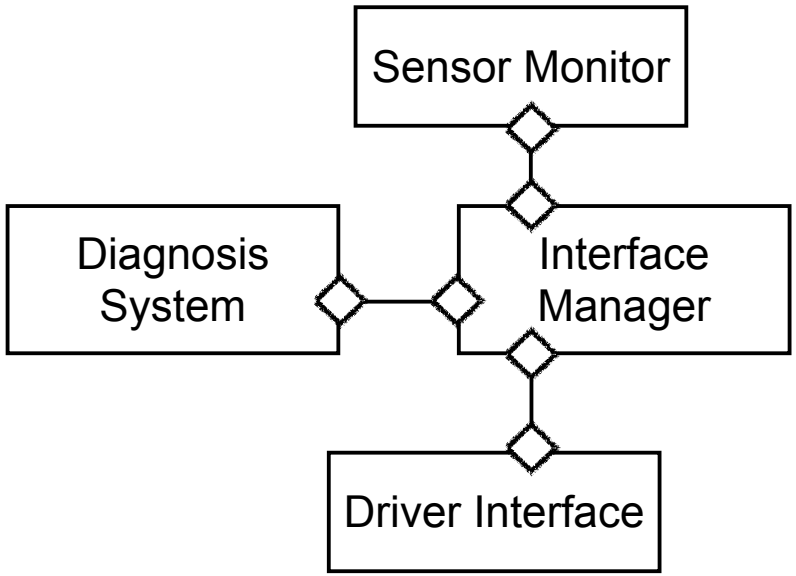
- Software Engineering > Advanced system design
- From Components Based Development to Service-Oriented engineering
- What is a Service-Oriented Architecture?
- Focus on Service-Oriented modelling
  - Modelling languages (UML and SRML)
  - Methodology

Lecture 1: Introduction



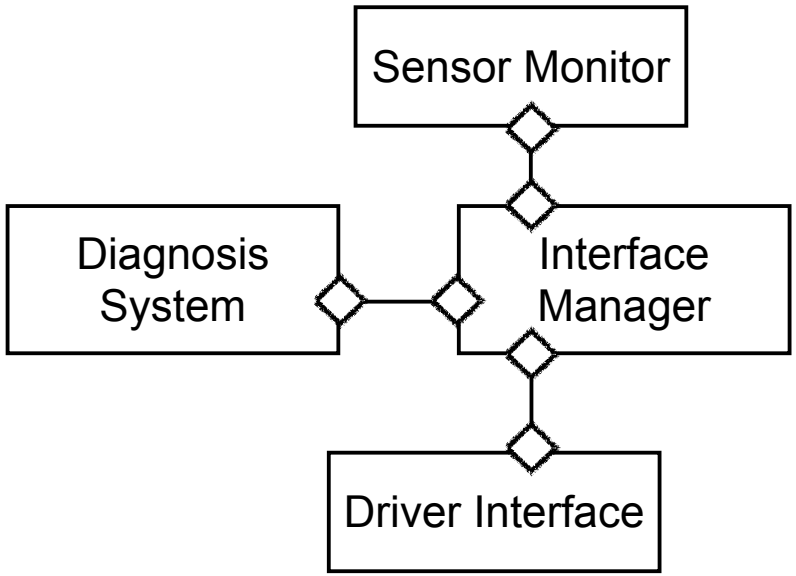
- Component Based Development (CBD)
  - an example of system configuration
  - static vs dynamic reconfiguration
- Service Oriented Architecture
  - What is a SOA?
  - Some motivations for SOAs
  - Services vs Web services
- Software Engineering for SOA
  - Background
  - Sensoria
  - Modelling languages (UML, SRML)

CBD: a simple example



- The example describes a control system in a vehicle
- **Sensor Monitor** detects warnings triggered by in any sensor in the vehicle
- **Diagnosis System** elaborates the data provided by the sensors to suggest a diagnosis
- **Driver Interface** visualises diagnosis and warnings on a screen visible to the driver
- **Interface Manager** coordinates the interactions among the parties

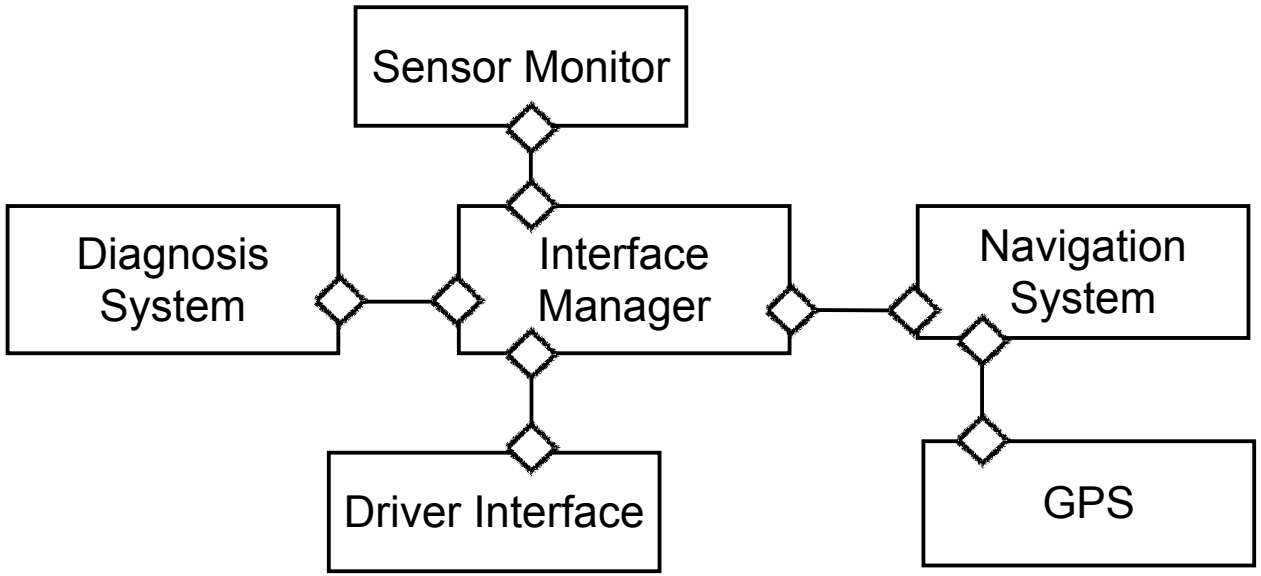
CDB: recall …



- A **component** is a software package that encapsulates a sub-functionality of a system
- A **component interface** (white rectangles in the picture) describes: (1) the supported set of interactions and possibly (2) possibly some behavioural properties
- A **connector** (black lines with diamonds in the picture) describes how components are inter-related
- The description of a model **abstracts from the actual implementation**

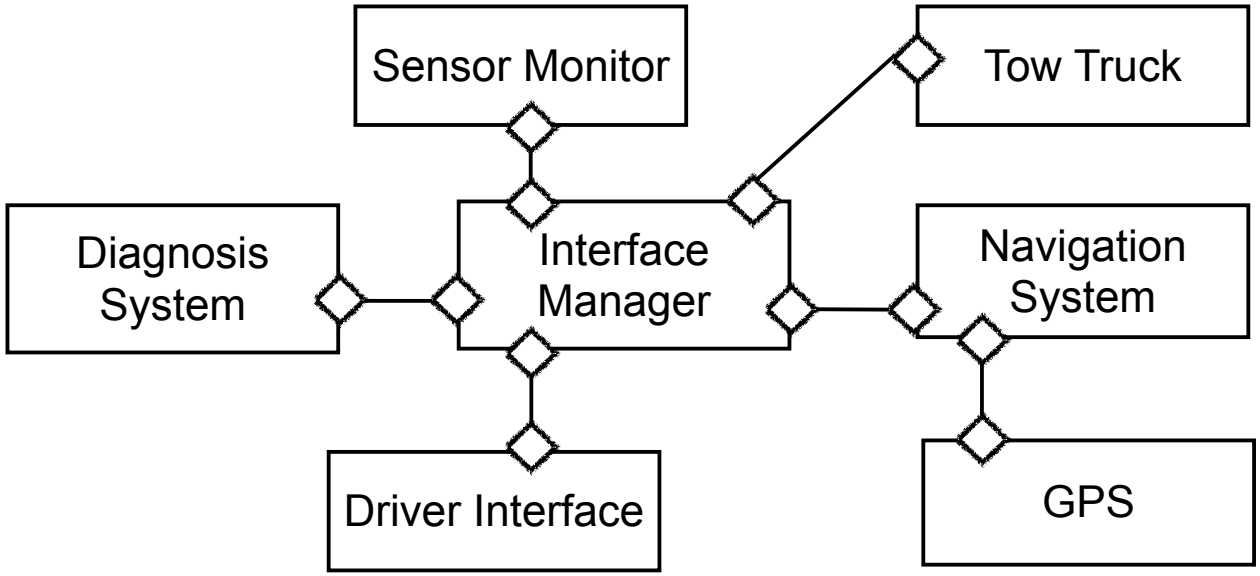
- Why? To incrementally build the system, to prove properties of the model, to reuse existing components, to make substitutions easier, to make changes easier …

CBD: extending the example



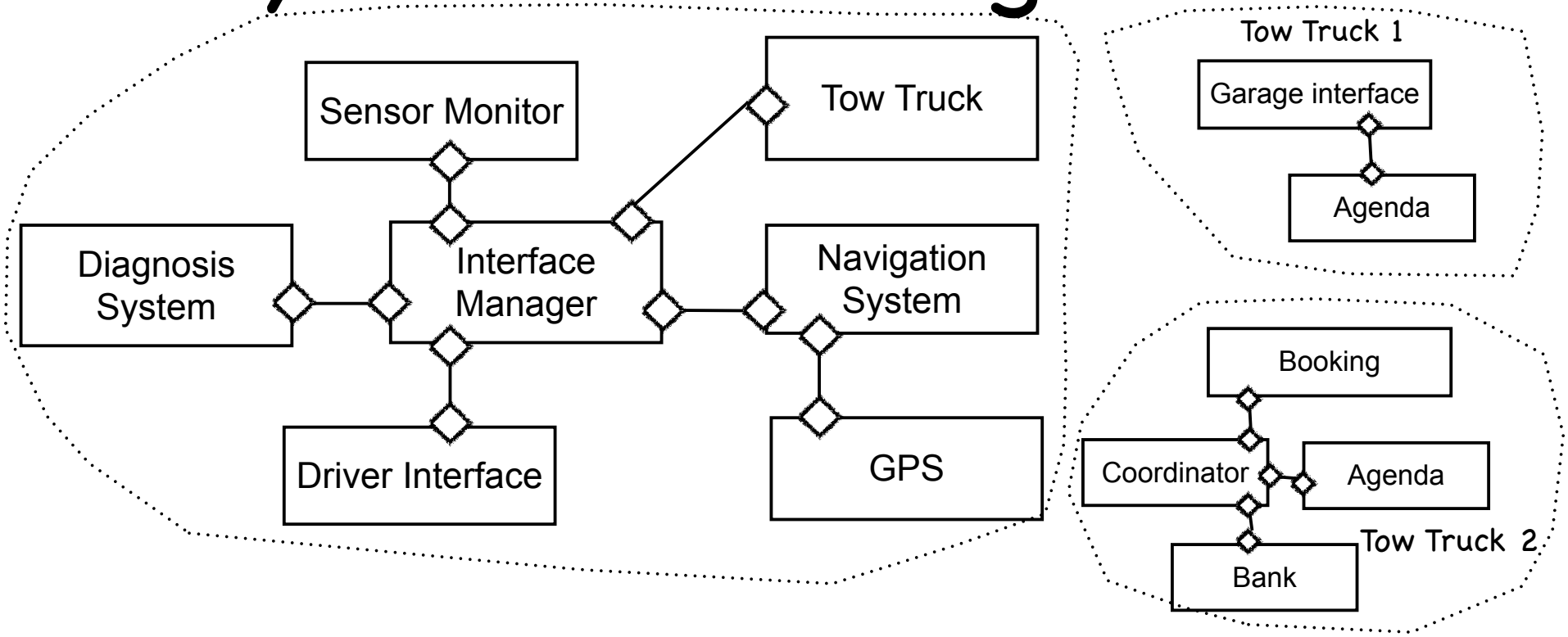
- Assume that we want to include a built in navigation system …
- At design-time we can add two components **GPS** and **Navigation System** and some connectors… (this may require some changes to **Interface Manager**)

Dynamic reconfiguration



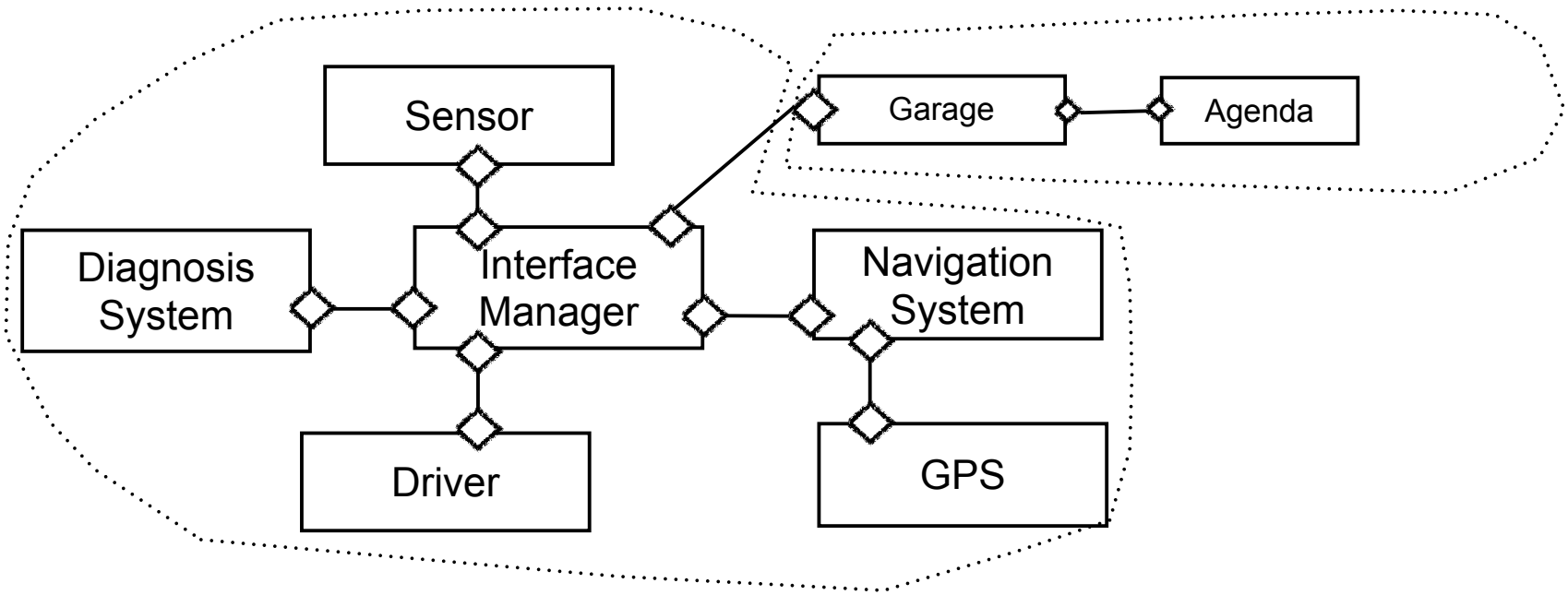
- We now add the possibility of calling a tow truck in case of serious engine damage
- Consider that (1) the functionality is provided by an external party, (2) we do not know **WHERE** the car will break and **WHO** will provide the tow truck functionality in that location
- The system reconfigures at run-time to include new parts of the system (i.e., **Tow Truck**)

Dynamic reconfiguration



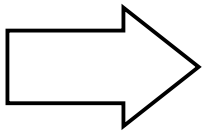
- At design time we do not care HOW the functionalities are modelled (how many components etc.) we care about WHAT is provided and how to use it
- Tow Truck should be a more abstract description with respect to the other component interfaces
- Service Oriented Modelling centres on the notion of service: a functionality that is provided externally and procured at run-time

Dynamic reconfiguration



- A system configuration consists of the components of the system and their interconnections
- Dynamic reconfiguration is a change of configuration that happens at run-time
- Service Oriented Modelling describes how the system should reconfigure when a new service is invoked. Such description is done at design-time.

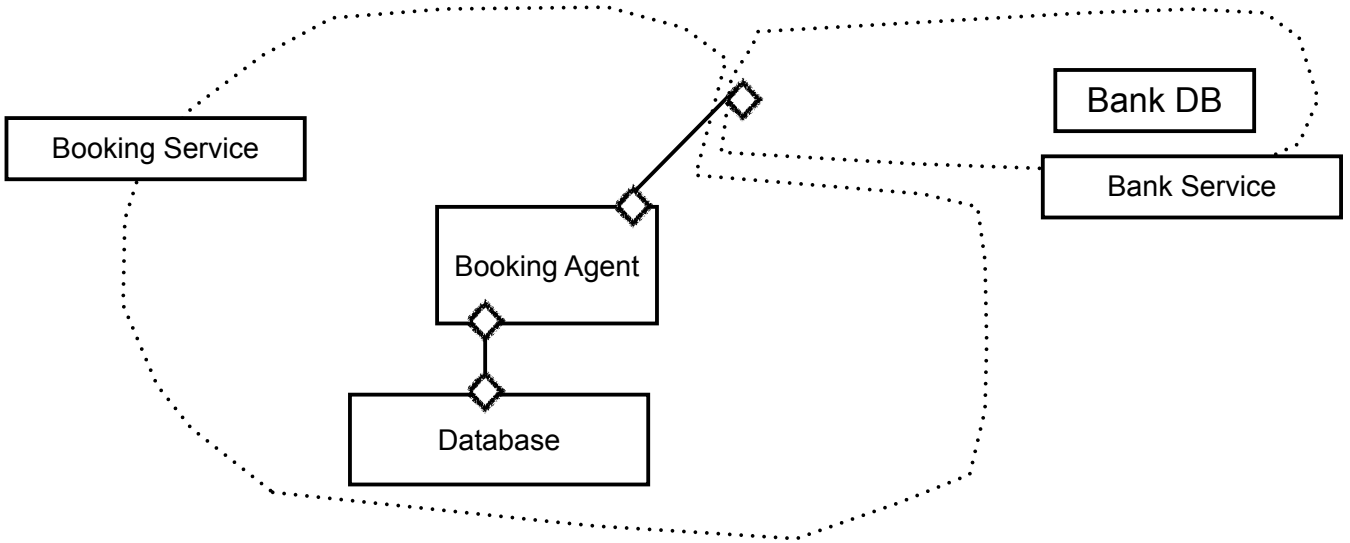
Lecture 1: Introduction



- Component Based Development
  - an example of system configuration
  - static vs dynamic reconfiguration
- Service Oriented Architecture
  - What is a SOA?
  - Some motivations for SOAs
  - Services vs Web services
- Software Engineering for SOA
  - Background
  - Sensoria
  - Modelling languages (UML, SRML)

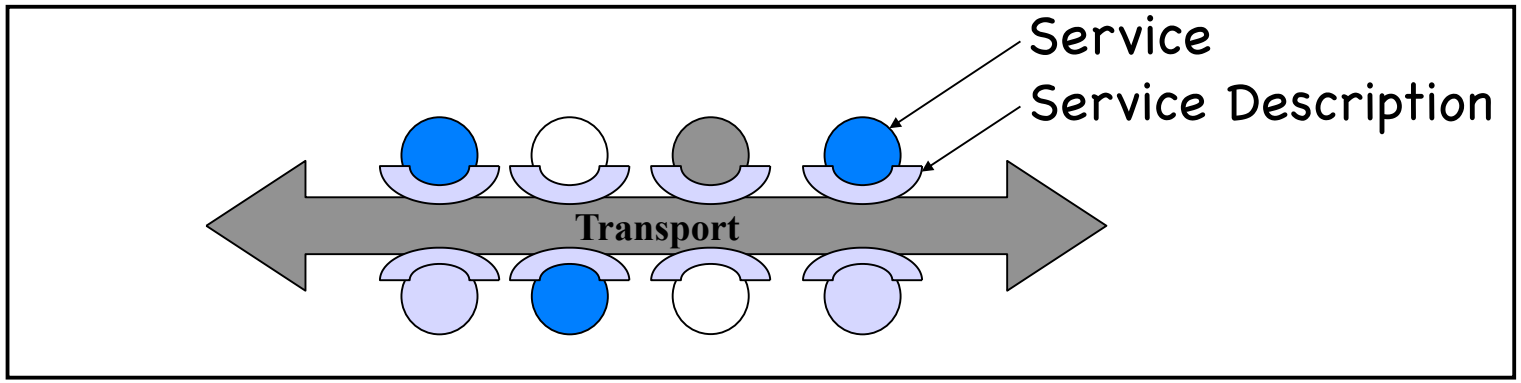
The complexity of evolution

- Service Oriented Computing (SOC) is a way of designing evolving software systems
- Services add a layer of abstraction over the component infrastructure
- Services should be published at a level of abstraction that correspond to a real-world activity or a recognisable business function



Service Oriented Architecture (SOA)

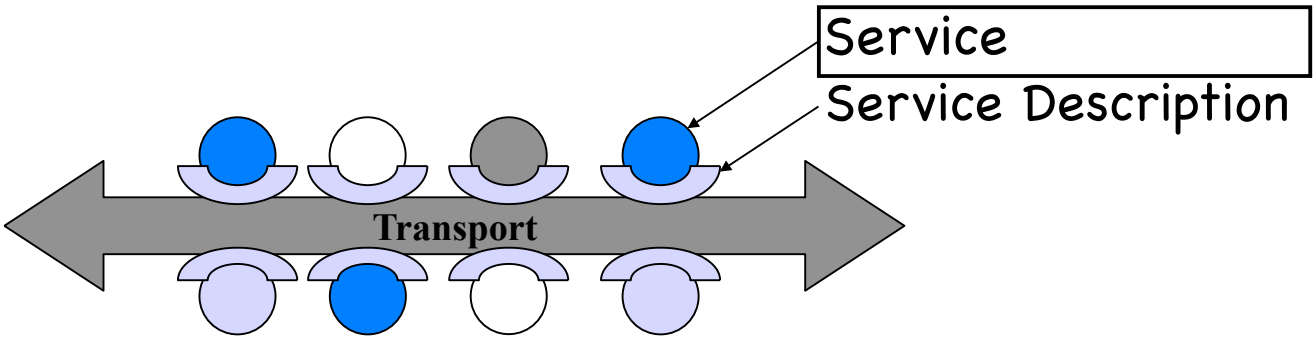
- Type of distributed system where services are network addressable entities that can be published (and revoked) at run time.
- A Service Oriented Architecture (SOA) is "..a set of components which can be invoked and whose interface descriptions can be published and discovered..." [W3C]



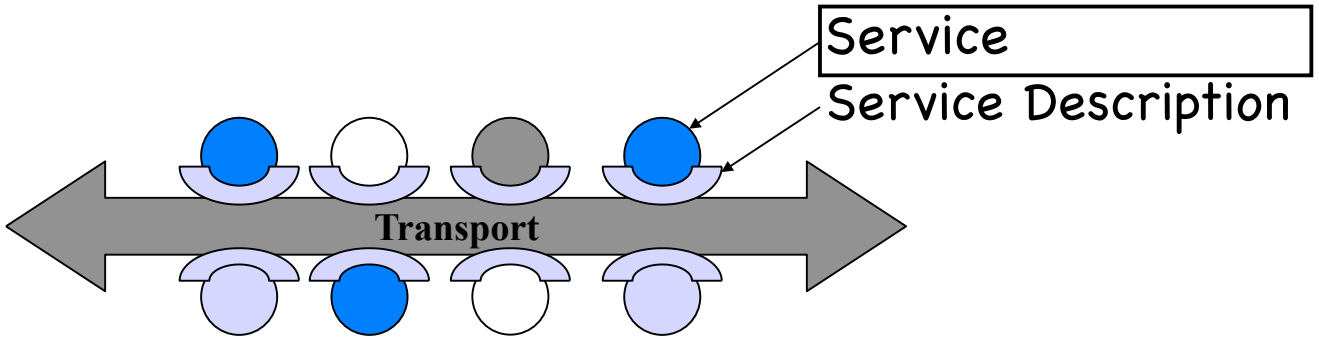
Service

A service (from [W3C Web Service Glossary])

- is an **abstract resource**
- represents a capability of performing tasks that form a **coherent functionality** from the point of view of providers entities and requesters entities
- to be used must be realised by a concrete provider agent

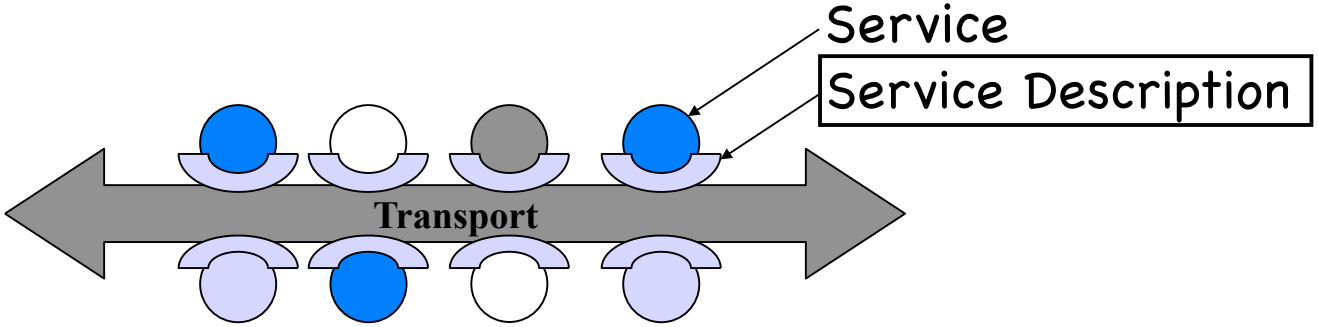


Service (cont.)



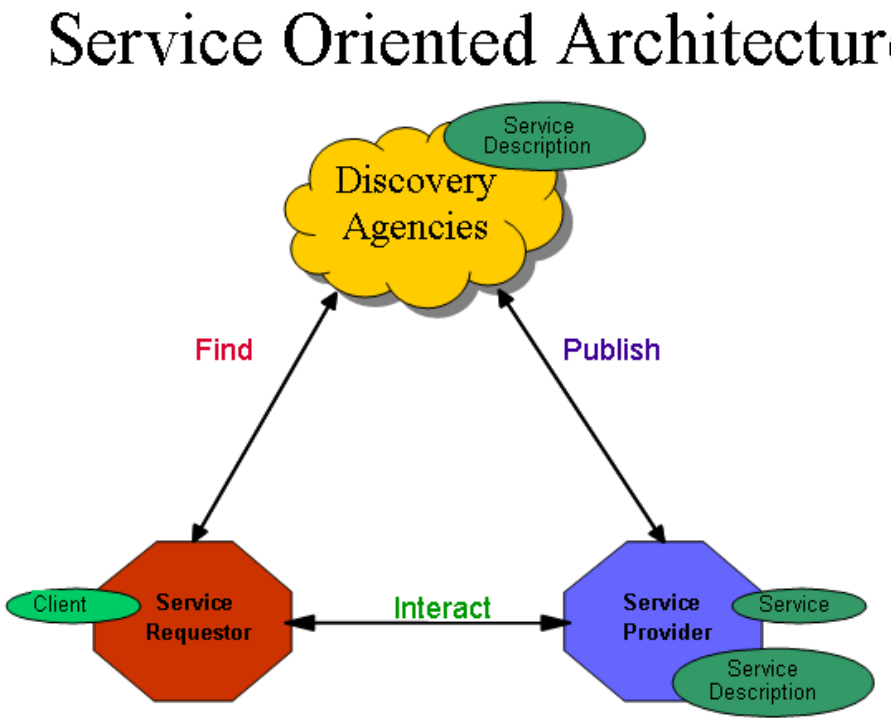
- In general, services are not black boxes that transform a input into an output; a service invocation triggers an **interactive activity**
- A service-oriented application can rely on some **abstract references** (i.e., service descriptions) that are discovered at run time if/when needed
- A service-oriented application can publish a service description to be discovered at run time and provide a functionality for external usage

Service Description



- A service must publish a service description to enable its usage by a user that can either be a human and a machine:
  - **Syntactic interface**: which interaction can we perform? (e.g., ask price, pay, etc.)
  - **Semantic description**: what do we actually need (e.g., "I want to buy a book")
  - **Behavioural interface**: which are the possible conversations? (e.g., confirmation always occurs after payment)
  - **Non-functional description**: attributes specifying quality of service (e.g., "I want a chap shipment", "I want a fast shipment", etc.)

Relationship between provider and requester



http://www.w3.org/TR/ws-arch/

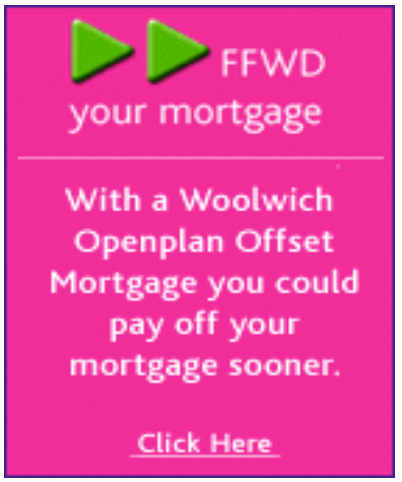
The same entity could have both the roles of service provider and service requester (i.e., provides a service by using external services)

Why SOAs?
An example: find the right loan

- Need to solve the problem of choosing among too many products
  - Which is the best mortgage for me NOW?
  - How to choose the right mortgage?
- The product is just a component of a larger "system"
  - what about insurance policies that protect the lender?
  - what about complementing repayment with a savings scheme?
- The system is dynamic
  - interest rates will change; my status will change: will i have to go all over the search process again? how many times?

Why SOAs?
An example: find the right loan

Lowest



Why SOAs?
An example: find the right loan

- Need to solve the problem of choosing among too many products
  - Which is the best mortgage for me NOW?
  - How to choose the right mortgage?
- The product is just a component of a larger "system"
  - what about insurance policies that protect the lender?
  - what about complementing repayment with a savings scheme?
- The system is dynamic
  - interest rates will change; my status will change: will i have to go all over the search process again? how many times?

From Products to Services

- Abstracts away the identity of the components(s) out of which the service is provided

- Provides an explicit representation for the role under which the service was procured, and which led to the choice of specific components

- The choice of the configuration of components that provide the required service is performed by experts in a more restricted domain

- Service providers have to abide to rules that ensure certain levels of quality

E-business and Outsourcing

- Outsourcing: contracting workers from outside of a company to perform specific tasks instead of using company employees
- The aim is to save money
- The IT infrastructure of an enterprise can involve
  - external networks
  - external resources
  - external services

http://ist-socrates.berkeley.edu/~fmb/articles/outsourcingtrends.html

When a SOA is useful?

- When the development of components is loosely coupled
- When the components are deployed and run on different platforms
- When we want to make a functionality accessible through a network
- Some examples:
  - e-business (Web Service Architecture)
  - e-science (Open Grid Service Architecture)

SOA vs WSA

- The Web Service Architecture (WSA) is an instance of Service Oriented Architecture (SOA)
- Web services are PLATFORM INDEPENDENT
- Web services are LANGUAGE INDEPENDENT
- Web services are not TECHNOLOGY AGNOSTIC (HTTP, SOAP, XML, WSDL, ...)

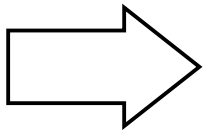
Platform:

some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming language and their runtime libraries.

- Services are technology agnostic
- We focus on modeling service-oriented systems in a technology agnostic way

Lecture 1: Introduction

- Component Based Development
  - an example of system configuration
  - static vs dynamic reconfiguration
- Service Oriented Architecture
  - What is a SOA?
  - Some motivations for SOAs
  - Services vs Web services
- Software Engineering for SOA
  - Background
  - Sensoria
  - Modelling languages (UML, SRML)



Language vs Method

- Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.
  [IEEE Standard Glossary of Software Engineering]
- Method = Language + Process
- The language is the notation for expressing the characteristics of the project
- The process consists of a number of steps to perform for producing the project

do not confuse the **engineering process** with the **business process** (composite activity aimed at the achievement of a business goal)

Modelling vs Programming

- **A model** is a pattern, plan, representation, or description designed to show the structure or workings of an object, system, or concept [Wikipedia]
- The model can abstract from a number of issues such as the details on the platform on which the system will be implemented or the technologies involved
- Why do we use model? To reuse the same model on different systems, to incrementally increase the level of detail or to increase productivity and quality...

Model Driven Architecture™

Model Driven Architecture is about using modelling languages as programming languages rather than merely as design languages. Programming with modelling languages can improve the productivity, quality, and longevity outlook.
"Model Driven Architecture, Applying MDA to Enterprise Computing" by David Frankel, OMG Press

- Model Driven Engineering involves the creation of a platform-independent model (PIM) and its (possibly automated) transformation into one or more platform specific models (PSMs)

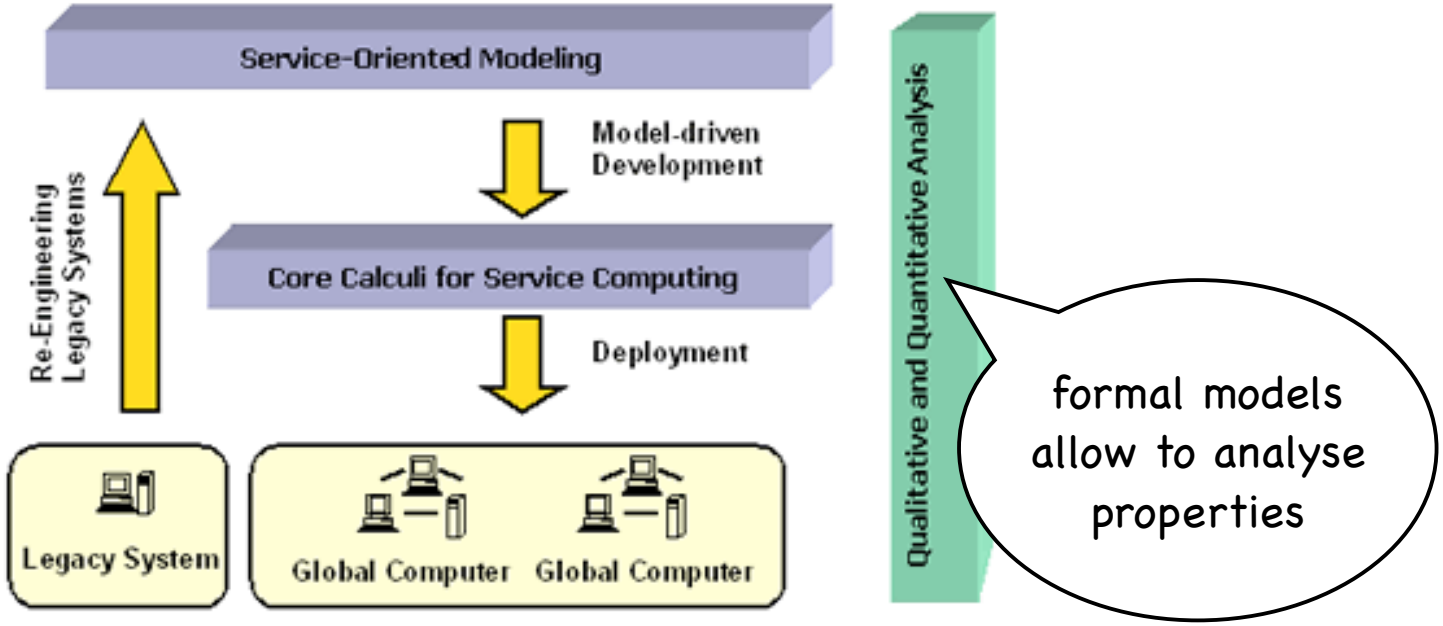
- The process may involve different modelling languages and tools to create a model, transform one type of model into another type of model, analyse models, etc.

SENSORIA

Software Engineering for Service-Oriented Overlay Computers

The aim of SENSORIA is to develop a novel comprehensive approach to the engineering of software systems for service-oriented overlay computers where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach.

Sensoria: overview



- WP1 Provide support for service-oriented modelling at high levels of abstraction, i.e. independently of the hosting middleware and hardware platforms, and the languages in which services are programmed.

- T1.1 **Develop a prototype language for service description and composition including syntax and mathematical semantics: SRML!!!**

Some modelling languages

- Unified Modelling Language (UML)
  - is general purpose
  - is semi-formal (although a number of formal semantics have been defined for the different types of diagrams) and human-oriented
  - Defined by Object Management Consortium (OMG)
- Sensoria Reference Modelling Language (SRML)
  - is specific for services
  - comes with a formal semantics
  - joint work: J.Abreu, L. Bocchi, J.L. Fiadeiro, A. Lopes.

Elements of UML

- Entities: classes, use cases, etc.
- Relationships: associations, generalisations, dependencies, etc.
- Diagrams:

Structure

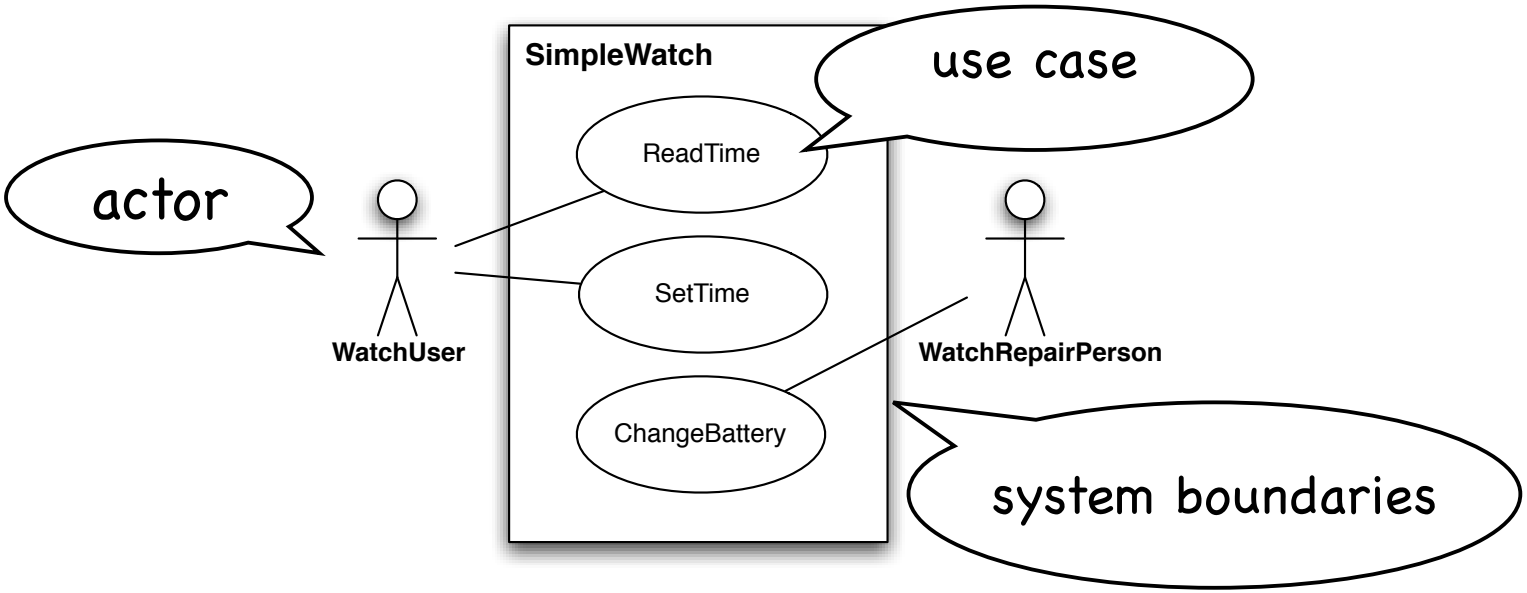
Class/Object/Package Diagrams
Component Diagrams
Composite Structure Diagrams
Deployment Diagrams

Behaviour

Use Case Diagrams
Interaction
  Sequence Diagrams
  Collaboration Diagrams
  Timing Diagrams
  Interaction Overview
Statechart Diagrams
Activity Diagrams

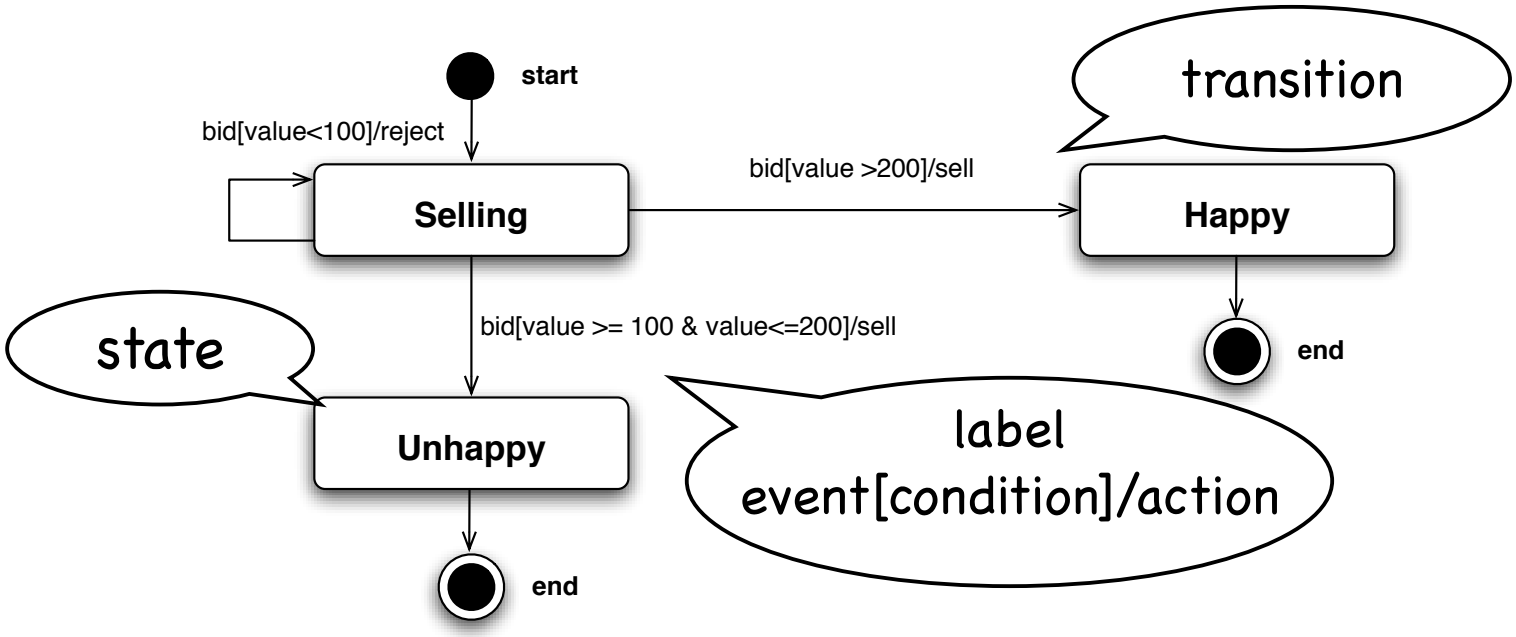
UML 2.0: Use Case Diagrams

- Used to capture requirements
- Focuses on the behaviour of the system as perceived by an external user
- The functionalities of the system are represented as use cases, useful for a specific class of users, represented as an actor



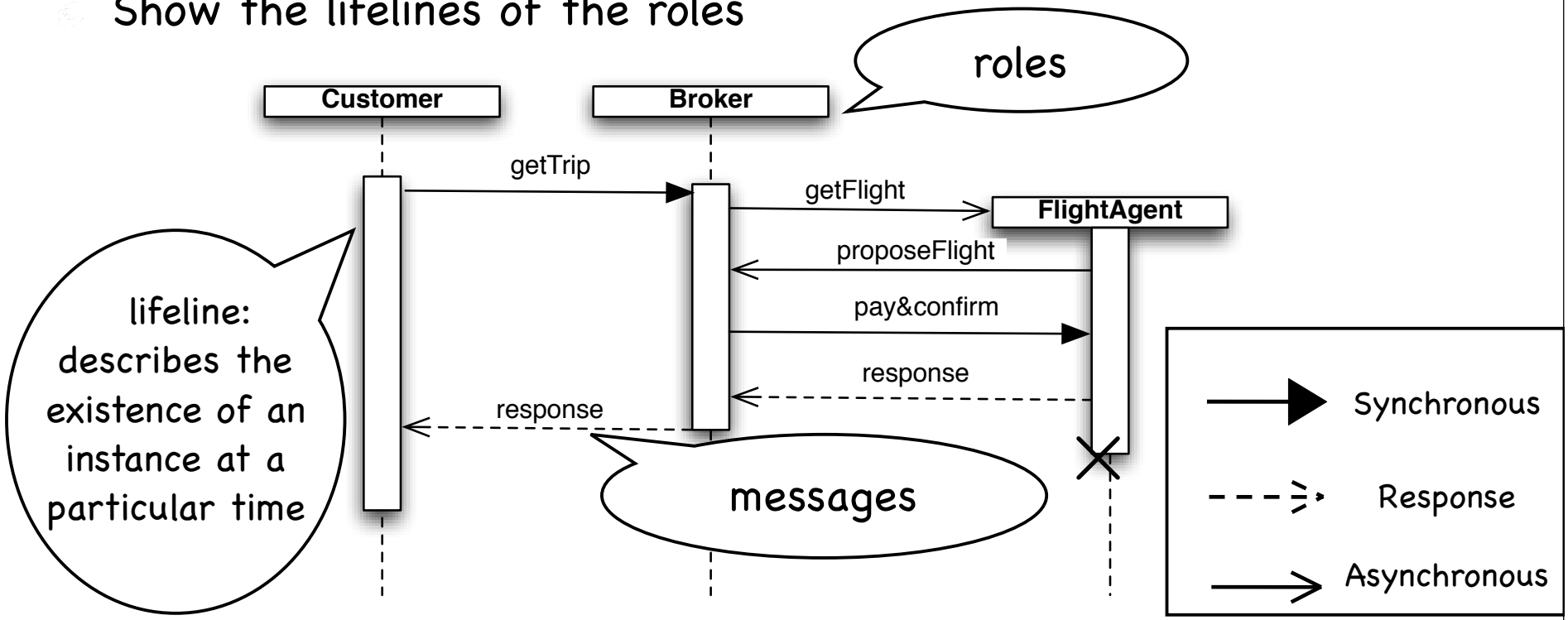
UML 2.0: Statecharts

"State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and track the different states of its objects through the system" [Fowler, Scott: "UML Distilled", Addison-Wesley]



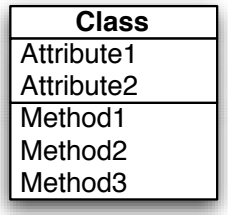
UML 2.0: Sequence Diagrams

- Represent the evolution of a system through the sequence of communications among a number of roles
- Show the temporal order of the messages
- Show the lifelines of the roles



Defining a language

- Syntax: rules by which the elements of the language are grouped together into expressions
- Semantics: rules that assign a meaning to syntactic expressions
- Syntactic rule for UML: "A class is represented as a rectangle divided in three parts by two horizontal lines"



- Semantic rule for UML: "If a class is not abstract, all its operations have to be associated to a method defined in the class"

Modelling in SRML

- SRML is a high level modelling language for service-oriented systems
- SRML provides primitives for modelling composite services
- SRML comes with a formal semantics

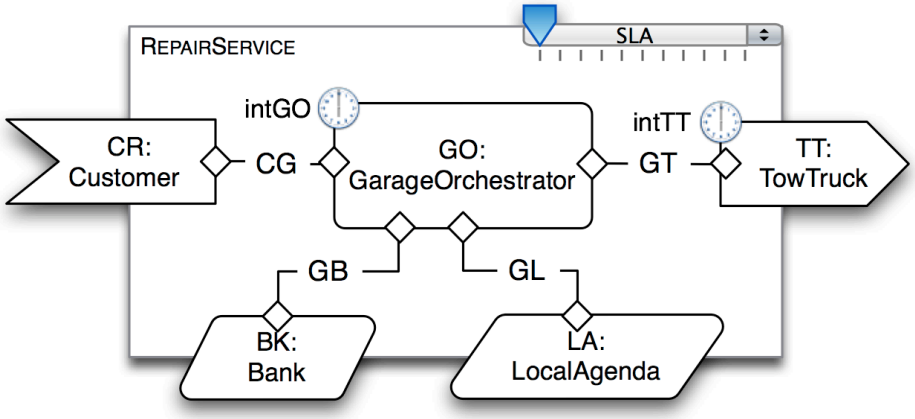
Formal aspects of SRML

- Declarative semantics of service composition and reconfiguration
- declarative in the sense that it relies on mathematical domains (configuration graphs and state transition systems) to make precise the meaning of the different constructs

J. L. Fiadeiro, A. Lopes, L. Bocchi (2008) **Semantics of Service-Oriented System Configuration**

J. L. Fiadeiro, A. Lopes, L. Bocchi (2007) **Algebraic semantics of service component modules**. In: J. L. Fiadeiro, P. Y. Schobbens (eds) Algebraic Development Techniques. LNCS, vol 4409. Springer, Berlin Heidelberg, pp 37–55

Formal aspects of SRML (cont.)



Business Roles (Garage, Orchestrator)

Business Protocols (Customer,TowTruck)

Layer Protocols (Bank, LocalAgenda)

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

logics of interactions

J. Abreu, J. Fiadeiro (2008) **A coordination model for service-oriented interactions**. In: D Lea, G. Zavattaro (eds) Coordination Languages and Models. LNCS, vol 5052. Springer, Berlin Heidelberg New York, pp 1–16

Summary (1/2)

- CBD: an example of system configuration
- Static and dynamic reconfiguration
- SOC needs to model how systems evolve dynamically
- What is a SOA, a service, a service description
- Motivations for SOA
- Examples of SOA

Summary (2/2)

- Software Engineering
- Method = laguage + process
- Modelling vs programing
- MDA, PIM, PSM
- Modelling languages (UML, SRML)
- UML use case diagrams, statechart diagrams, sequence diagrams
- SRML: what it is, in which sense it is formal, the editor

Next class...

- The different entities in a service oriented model
- Capturing the requirements of a service-oriented
- An extension of UML use-case diagrams for SOA

Next lab...

- Introduction on the Eclipse environment
- The SRML Editor, installation - bring your laptop if you like

Problems

- Provide, with your own words, a definition for the terms:
  - **component**,
  - **component interface**,
  - **service**,
  - **service description**,
  - **Service-Oriented Architecture**.

Problems

- What is the difference between a component and a component interface?
- How do you think a service description should differ from a component interface?
- What do we mean by dynamic reconfiguration? Explain the meaning of "static", "dynamic", "design-time" and "run-time"
- What is the difference between SOA and the Web Service Architecture (WSA)?

Problems

- Let us model a university registration system. The system must support the following functionalities:
  - (a) the secretary to add new students,
  - (b) the teacher to add new courses,
  - (c) the student to subscribe to a course,
  - (d) the teacher to see which students are registered in a course
- Sketch the requirements of the system as a use case diagram

Remember to include (1) system boundaries, (2) roles, (3) use cases and (4) associations between roles and use cases.

Problems

- Let us model a fragment of the state machine for the registration of a student to a course.
- The statechart has the folowing states:
  - INITIAL
  - CHECKDATA
  - FINAL
  - FAILED
- The statechart has the following transitions:
  - when event insertCourse occurs we change from INITIAL to CHECKDATA
  - if the course already exists (i.e., existingCourse) then move to FAILED and generate event notifyFailure
  - otherwise (i.e., notExistingCourse) then move to FINAL

event[condition]/action