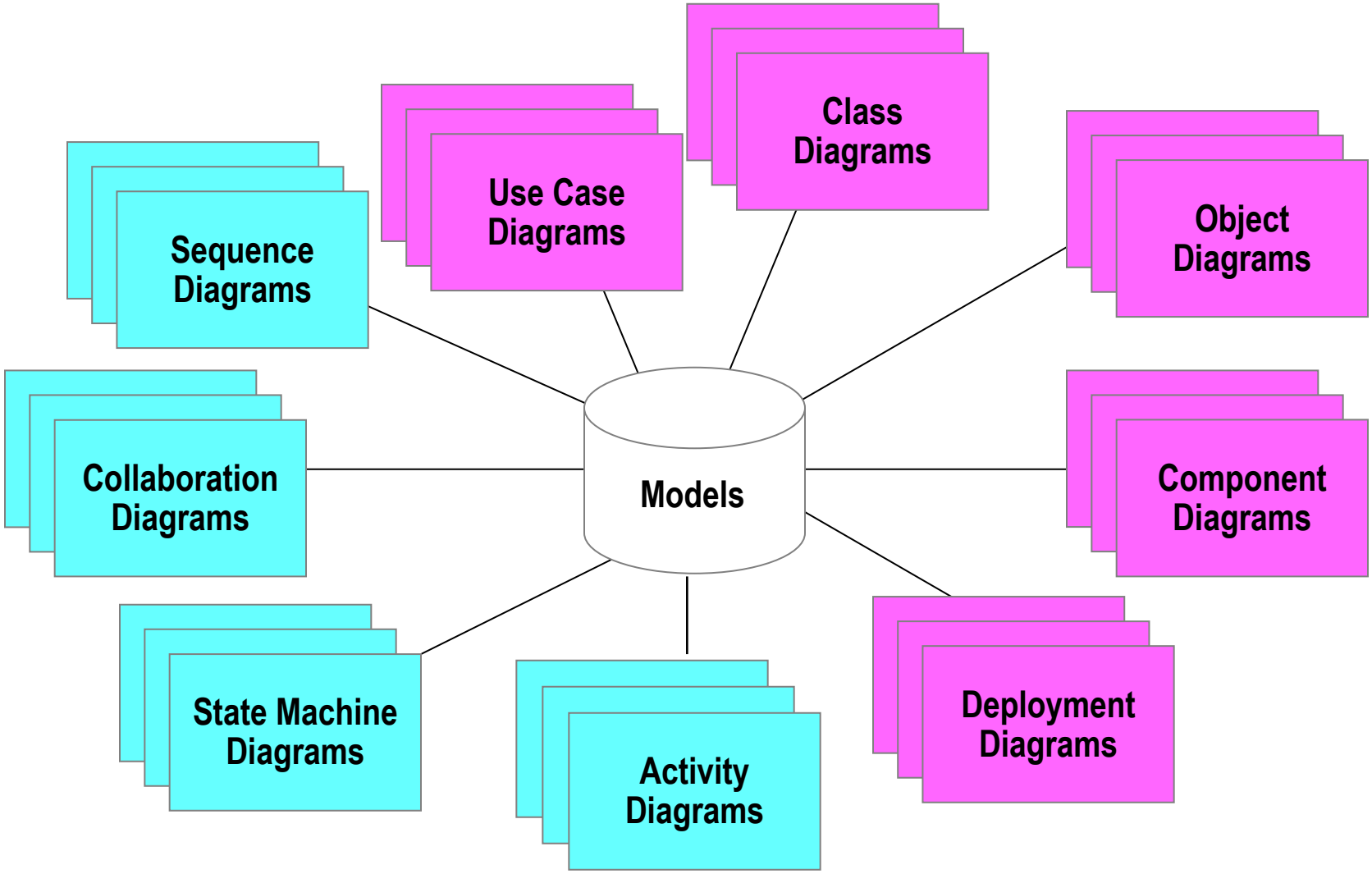
Politecnico
di Milano

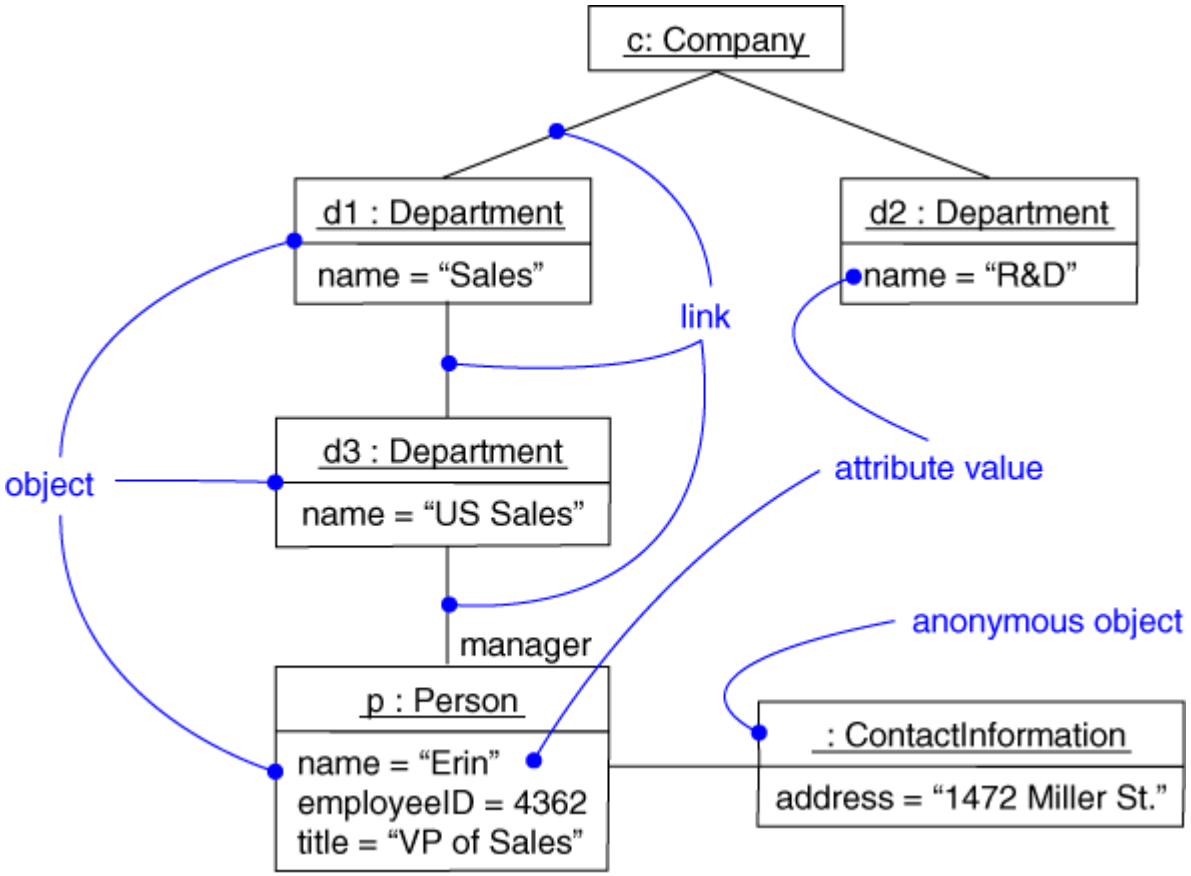
Tips and Tricks for Supporting Architecture Description with UML

UML Models, Views, and Diagrams



“Object” Diagram
(in UML 2.5 they are really Class Diagrams with only instances)

- Captures instances and links

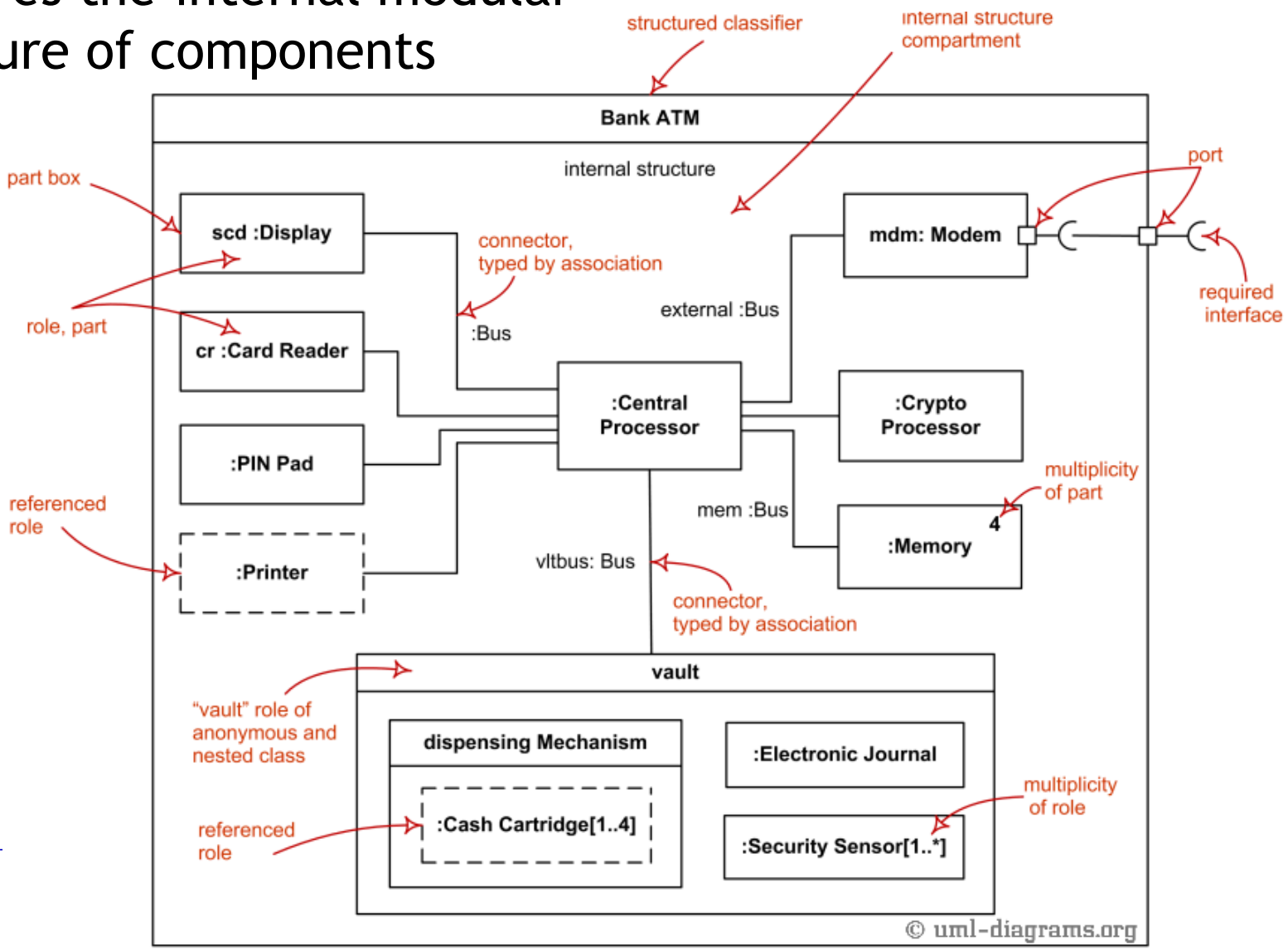


Object Diagram

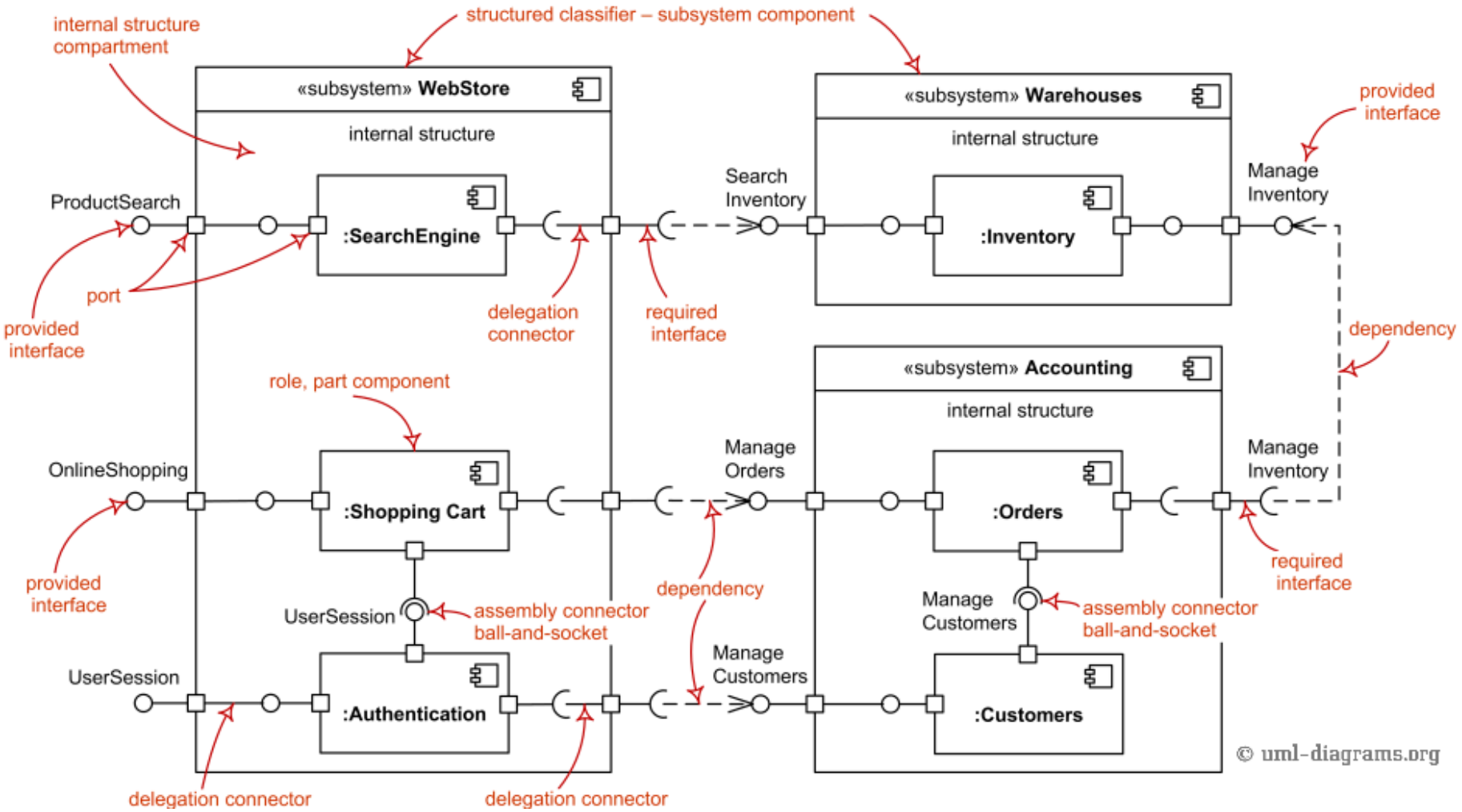
- Shows instances and links that architecture elements maintain across functions
- Built during analysis and design
- Purpose
  - Illustrate data/object structures
  - Specify architecture runtime snapshots
- Developed by analysts, designers, and implementers

Composite Structure Diagram

- Captures the internal modular structure of components



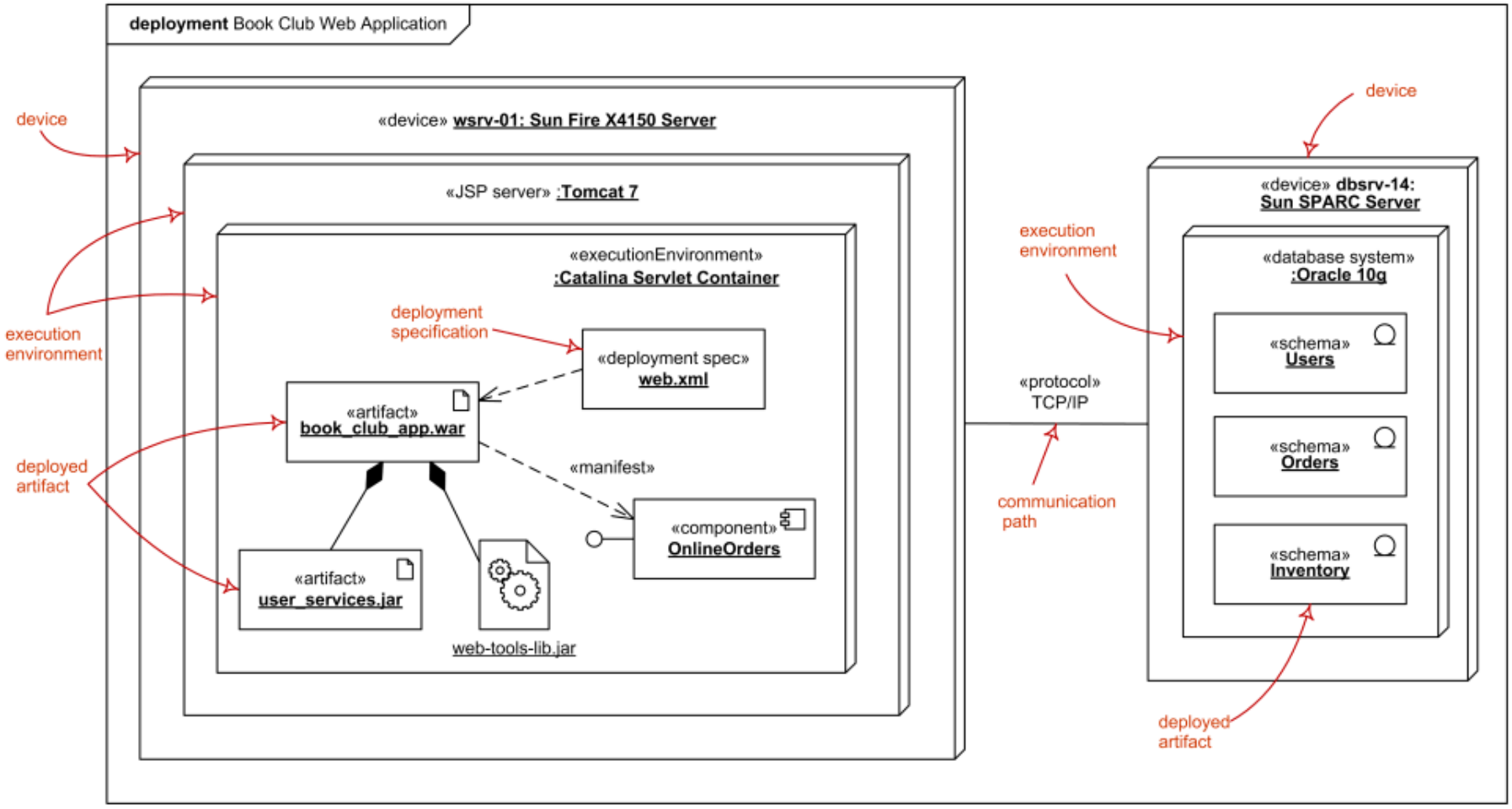
Component Diagram



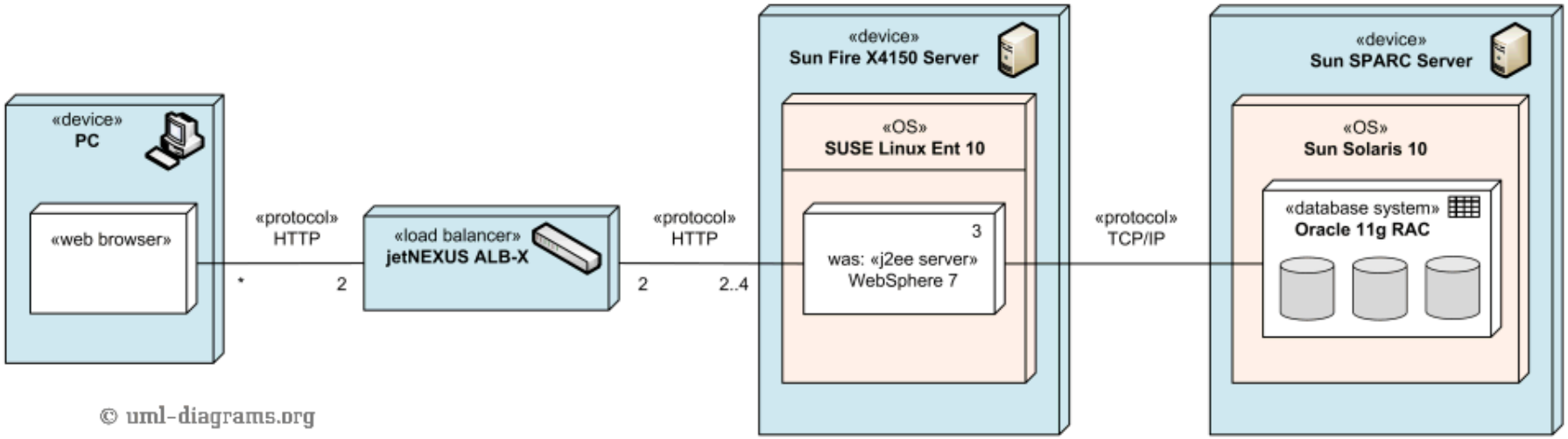
Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers

Deployment diagram



Deployment diagram

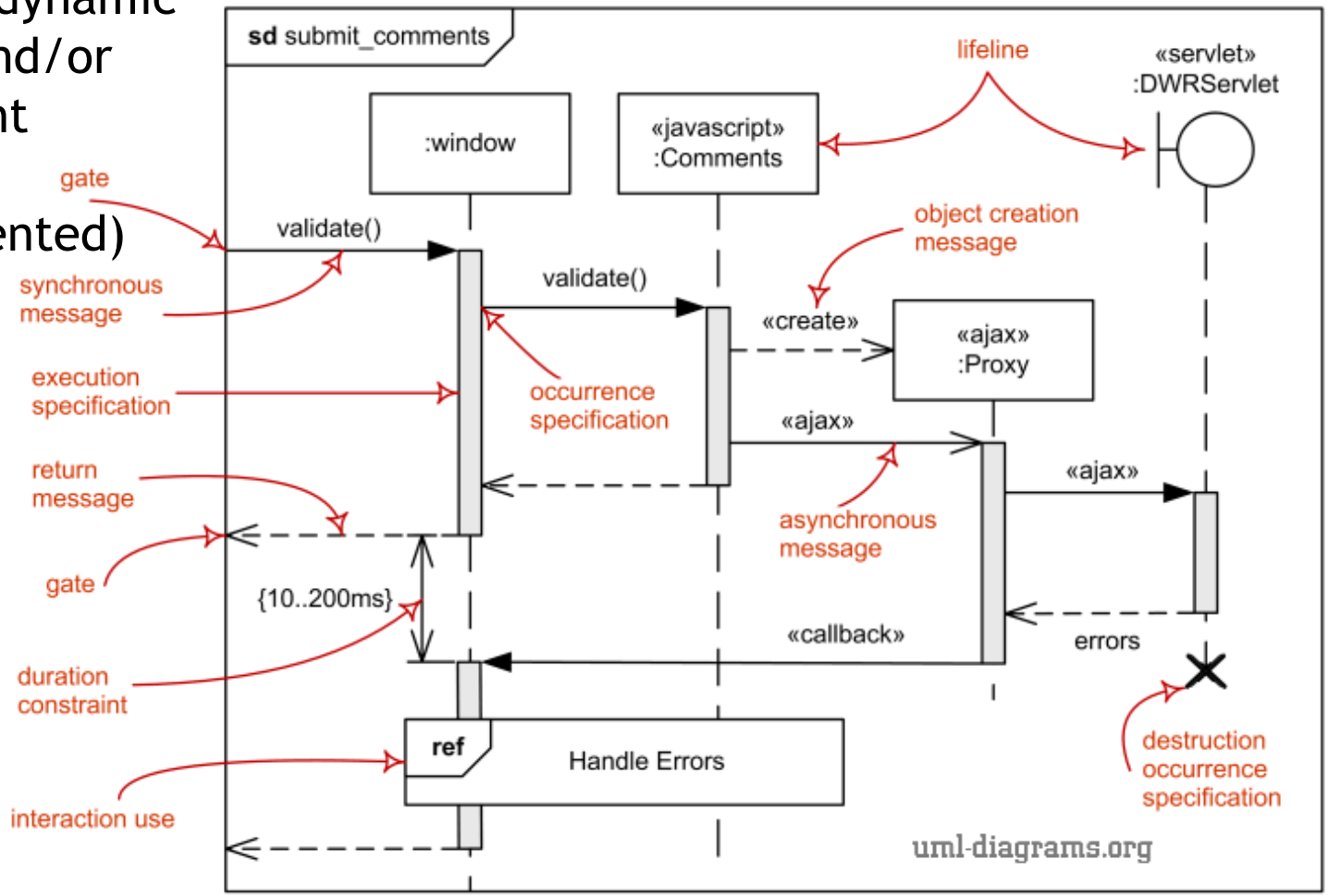


Deployment Diagram

- Captures the topology of a system's hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

“Architectural” Sequence Diagram
(i.e., showing interactions among components)

- Captures dynamic module and/or component behavior (time-oriented)



“Architectural” Sequence Diagram

- Captures dynamic behavior (time-oriented)
- Purpose
  - Model flow of control
  - Illustrate typical scenarios
  - Analyse architecture -ilities

Politecnico
di Milano

How to reason about design? An example

The taxi service case

Exercise

- Assume that the Milano municipality asks you to develop a digital taxi service
- The municipality wants to replace the taxi stops available on the streets with the online service
- The Mayor sends you a written description of what they would like to have (see next two slides)
- Define the architecture of the system

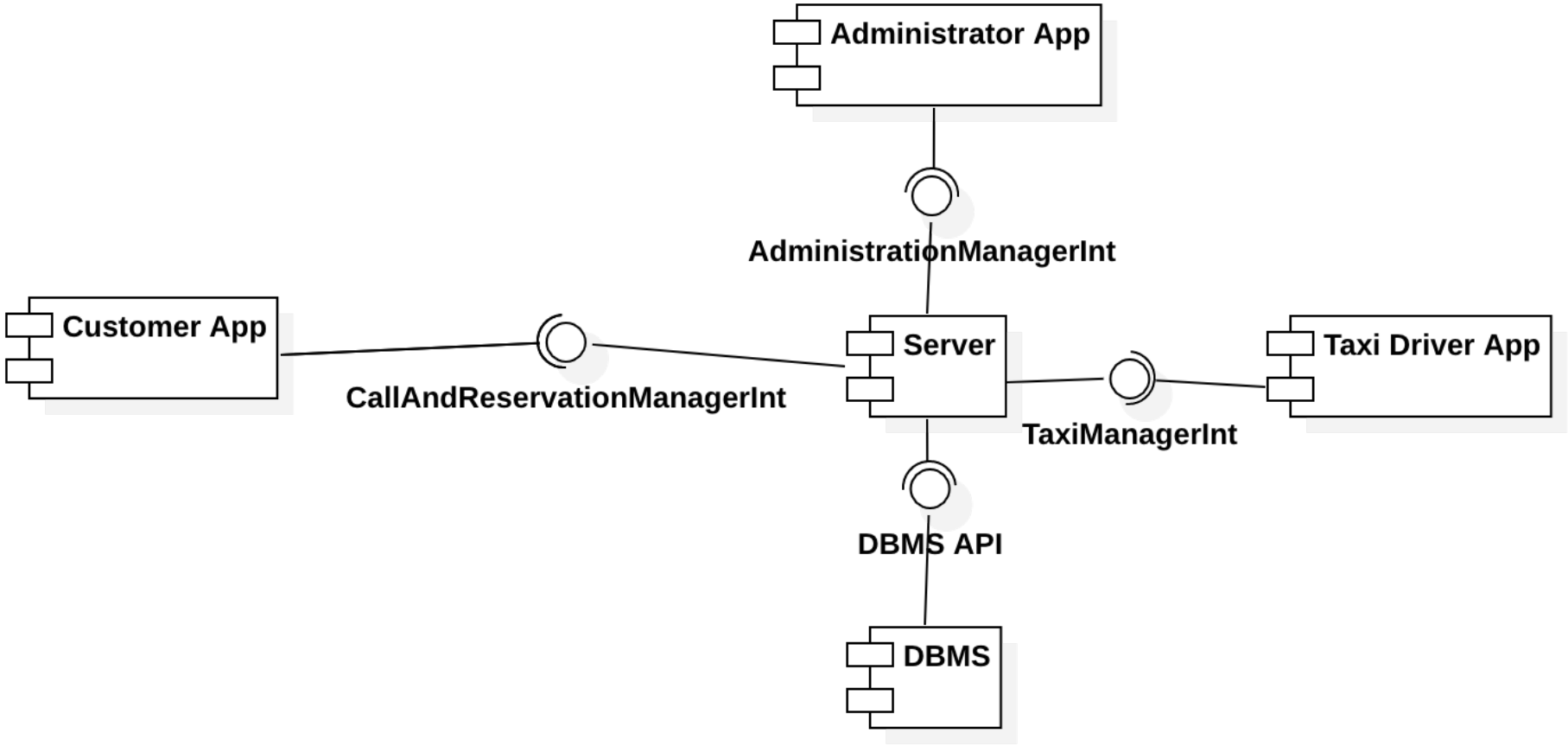
MyTaxi Service (1)

- The government of a large city aims at optimizing its taxi service. In particular, it wants to:
  - i) simplify the access of passengers to the service, and
  - ii) guarantee a fair management of taxi queues.
- Passengers can request a taxi either through a web application or a mobile app. The system answers to the request by informing the passenger about the code of the incoming taxi and the waiting time.
- Taxi drivers use a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call.
- The system guarantees a fair management of taxi queues. In particular, the city is divided in taxi zones (approximately 2 km^2 each). Each zone is associated with a queue of taxis. The system automatically computes the distribution of taxis in the various zones based on the GPS information it receives from each taxi. When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.

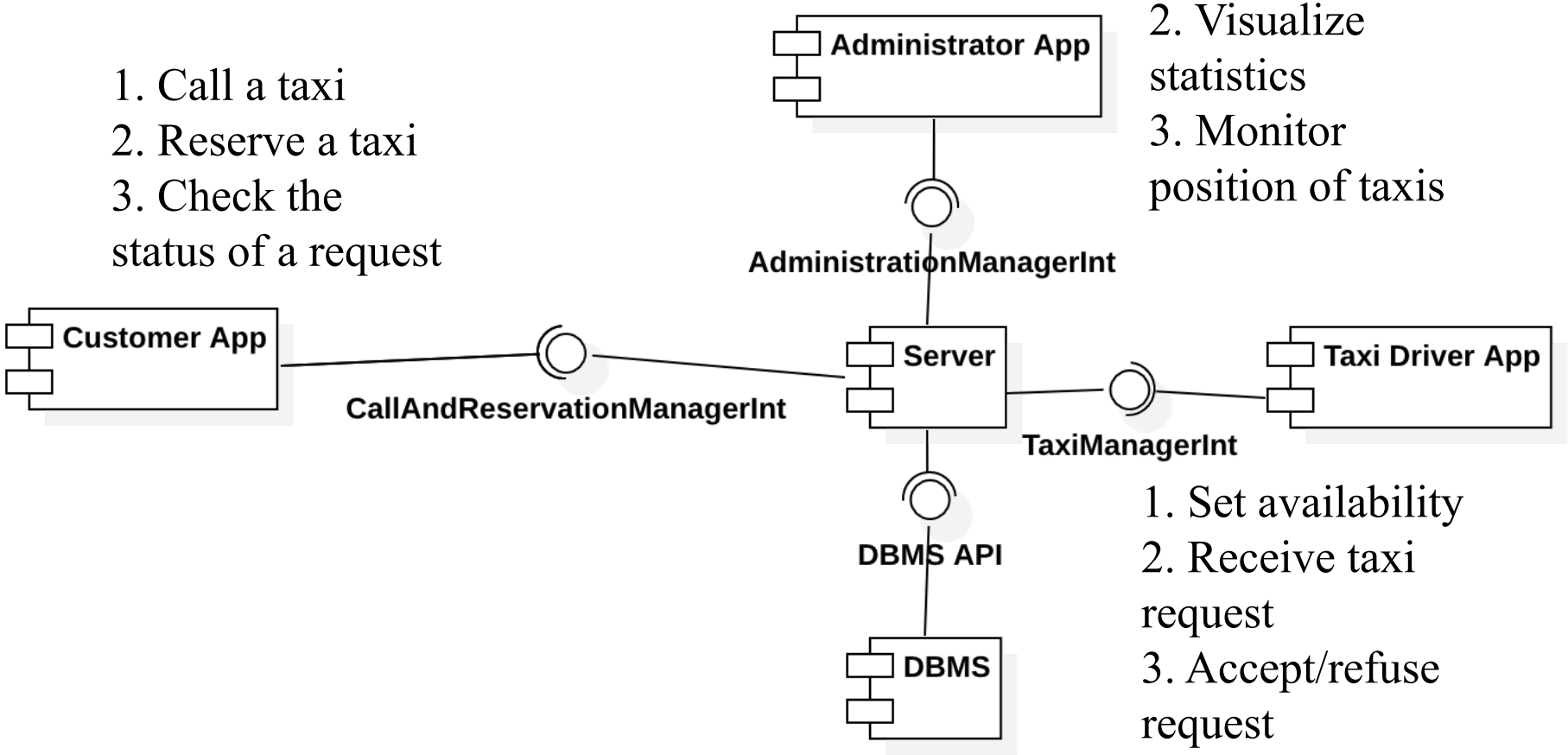
MyTaxi Service (2)

- When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone. If the taxi confirms, then the system will send a confirmation to the passenger. If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.
- A user can also reserve a taxi by specifying the origin and the destination of the ride. The reservation has to occur at least two hours before the ride. In this case, the system confirms the reservation to the user and allocates a taxi to the request 10 minutes before the meeting time with the user.

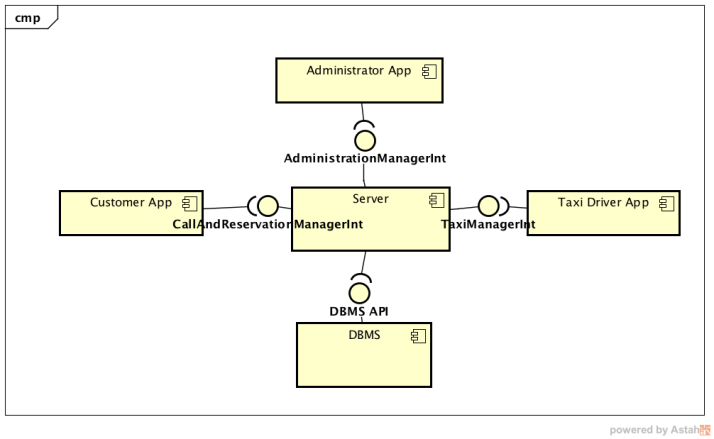
Main components

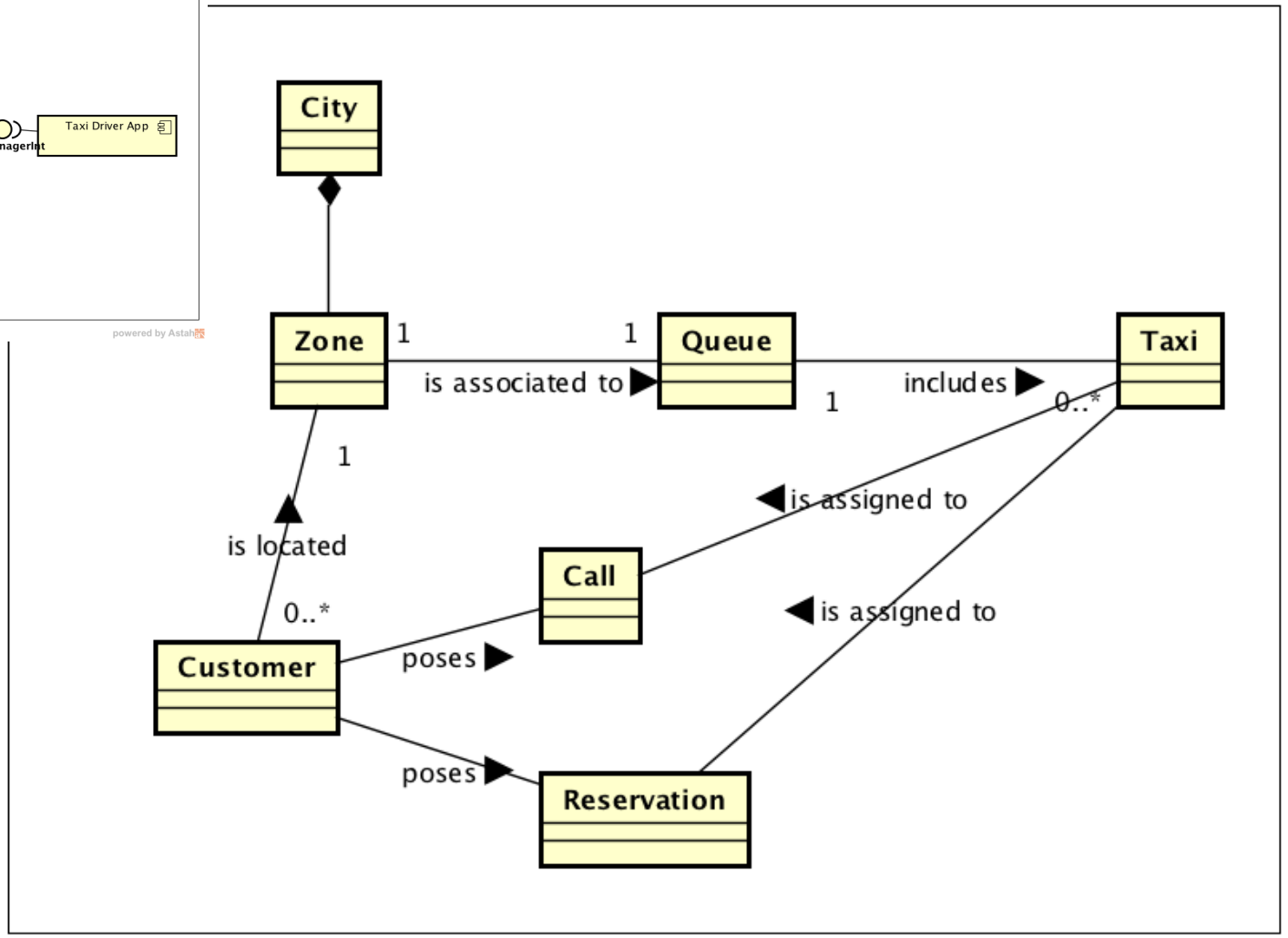


Allocation of functionalities to the external components

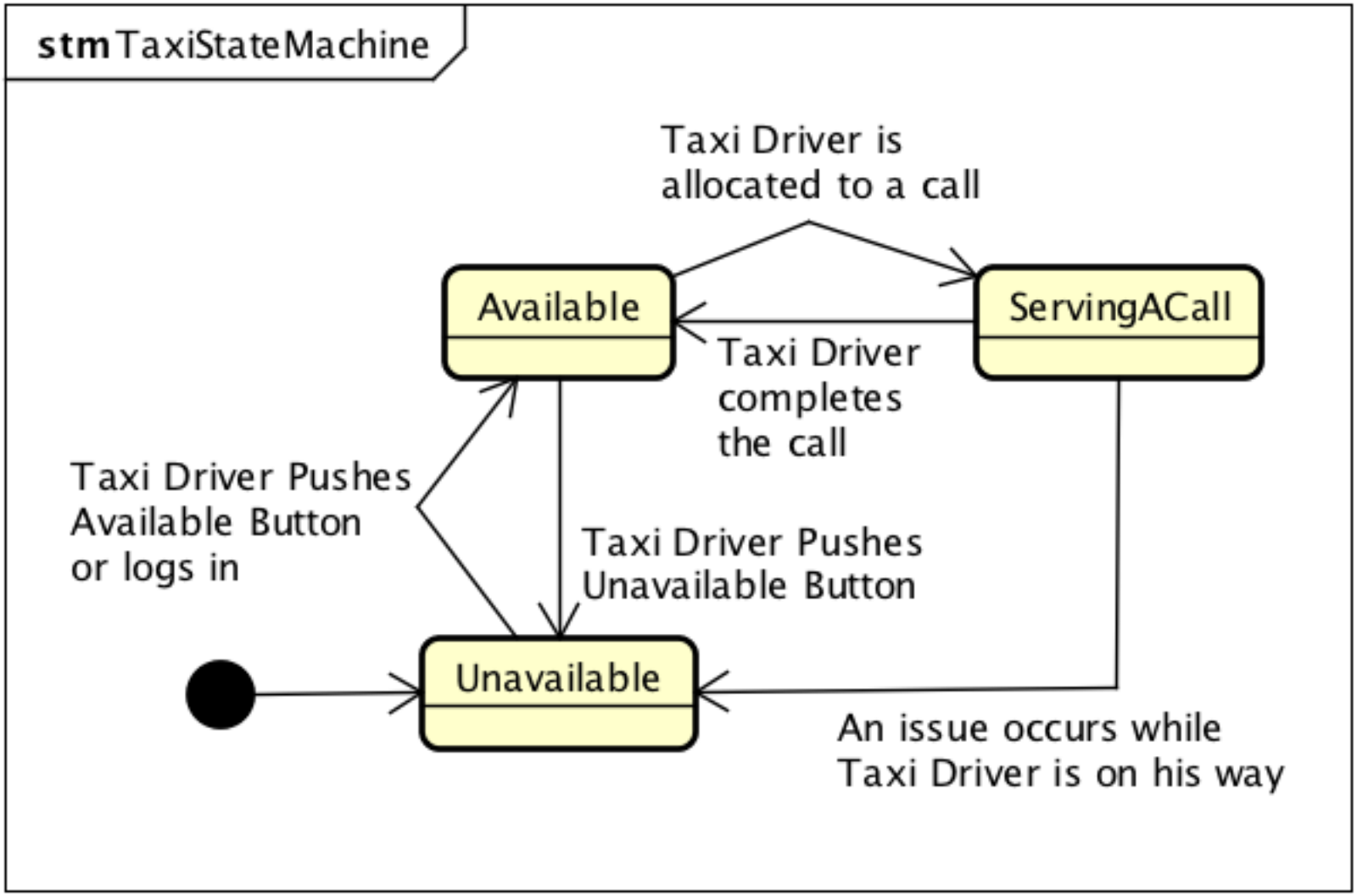


Server: objects wrapping data structures

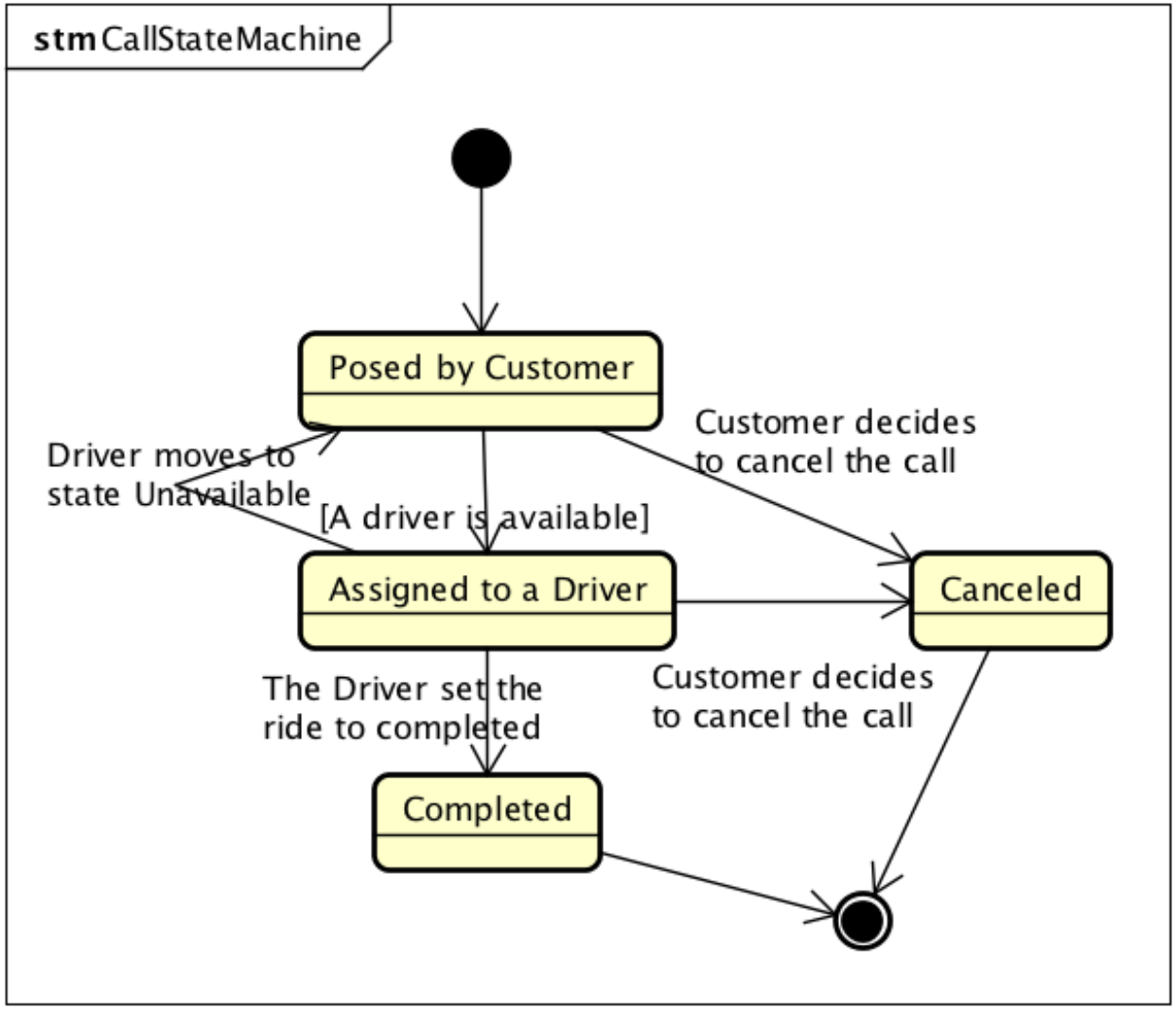




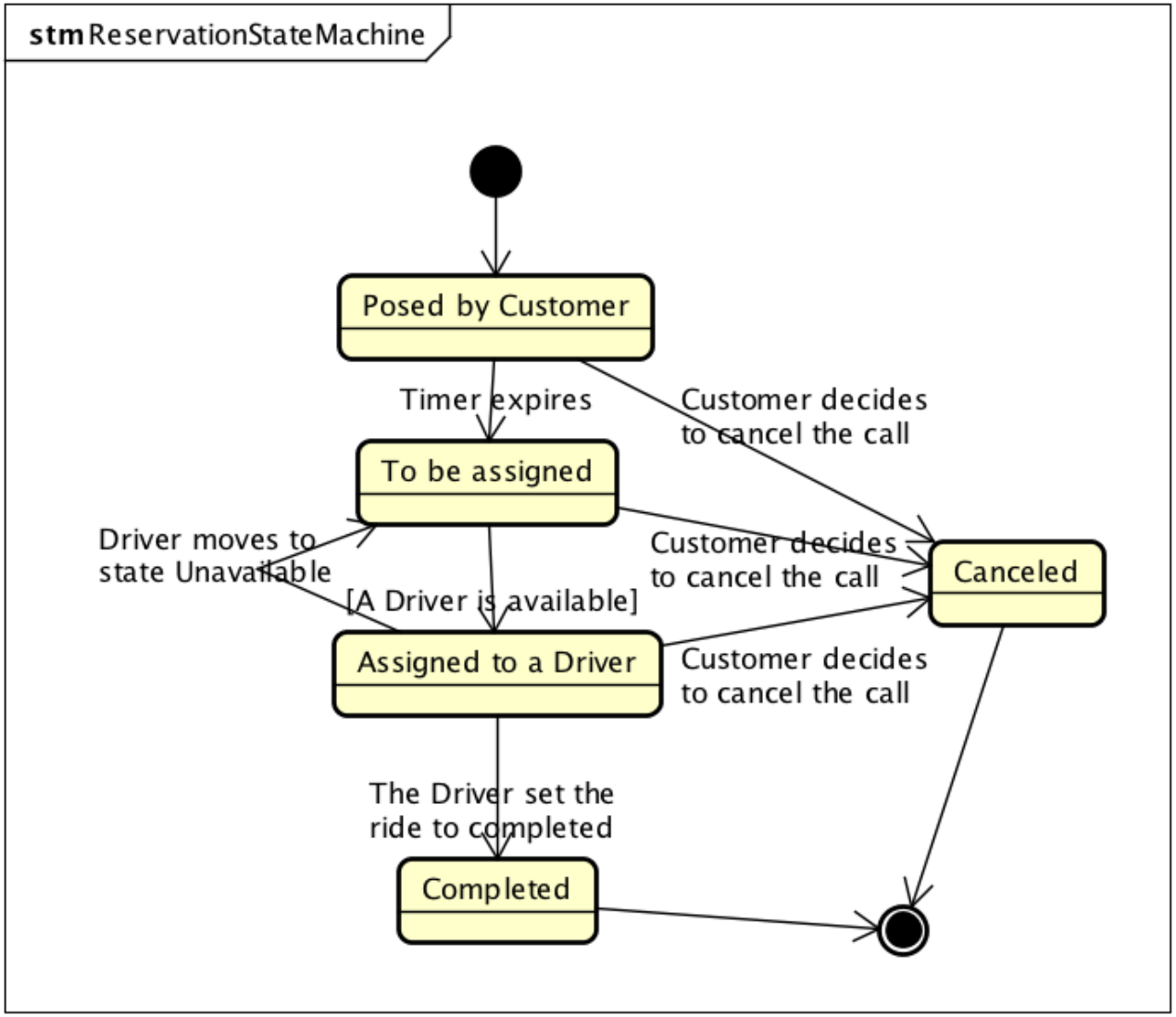
StateCharts for relevant data structures



StateCharts for relevant data structures



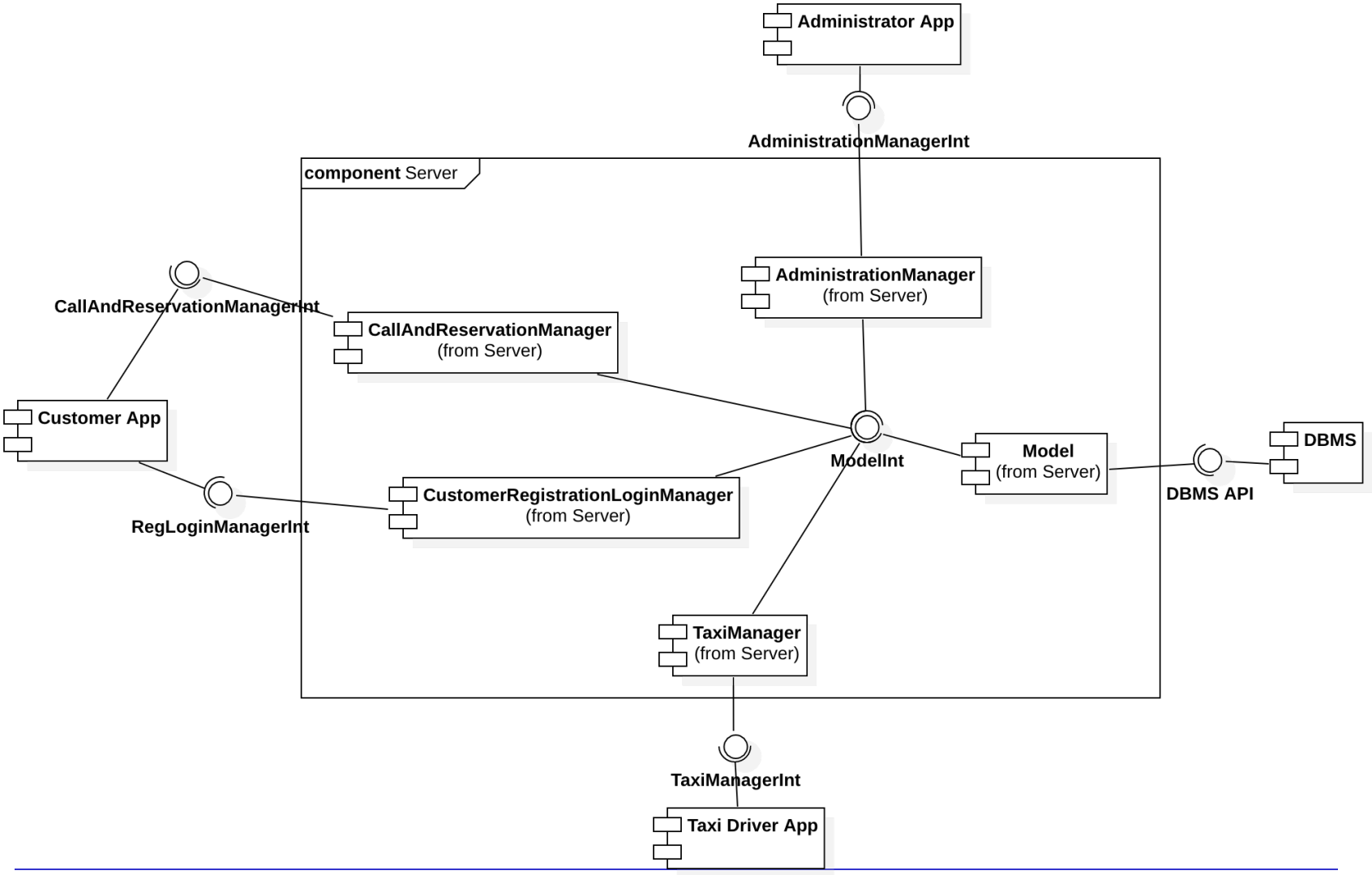
StateCharts for relevant data structures



Server: Manager components

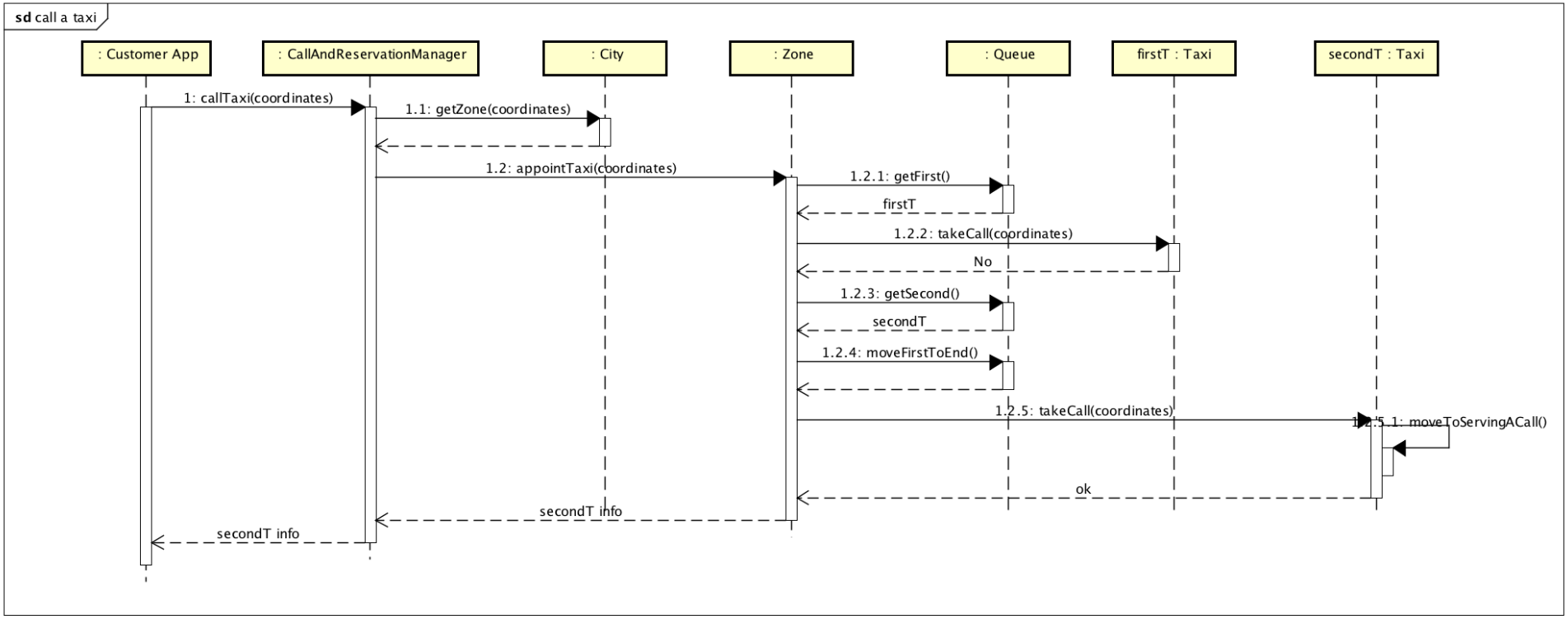
- CallAndReservationManager
  - Interacts with the customer app to acquire calls and reservations for taxis and to check status of calls
  - It waits for requests from the customer app
  - Pushes info back to the customer app when the call/reservation is assigned to a taxi driver
- CustomerRegistrationLoginManager
  - Interacts with the customer app to acquire data for customer registration and to support login
- AdministrationManager: supports all operations by admins
- TaxiManager: supports all operations by taxi drivers

Detailed component diagram



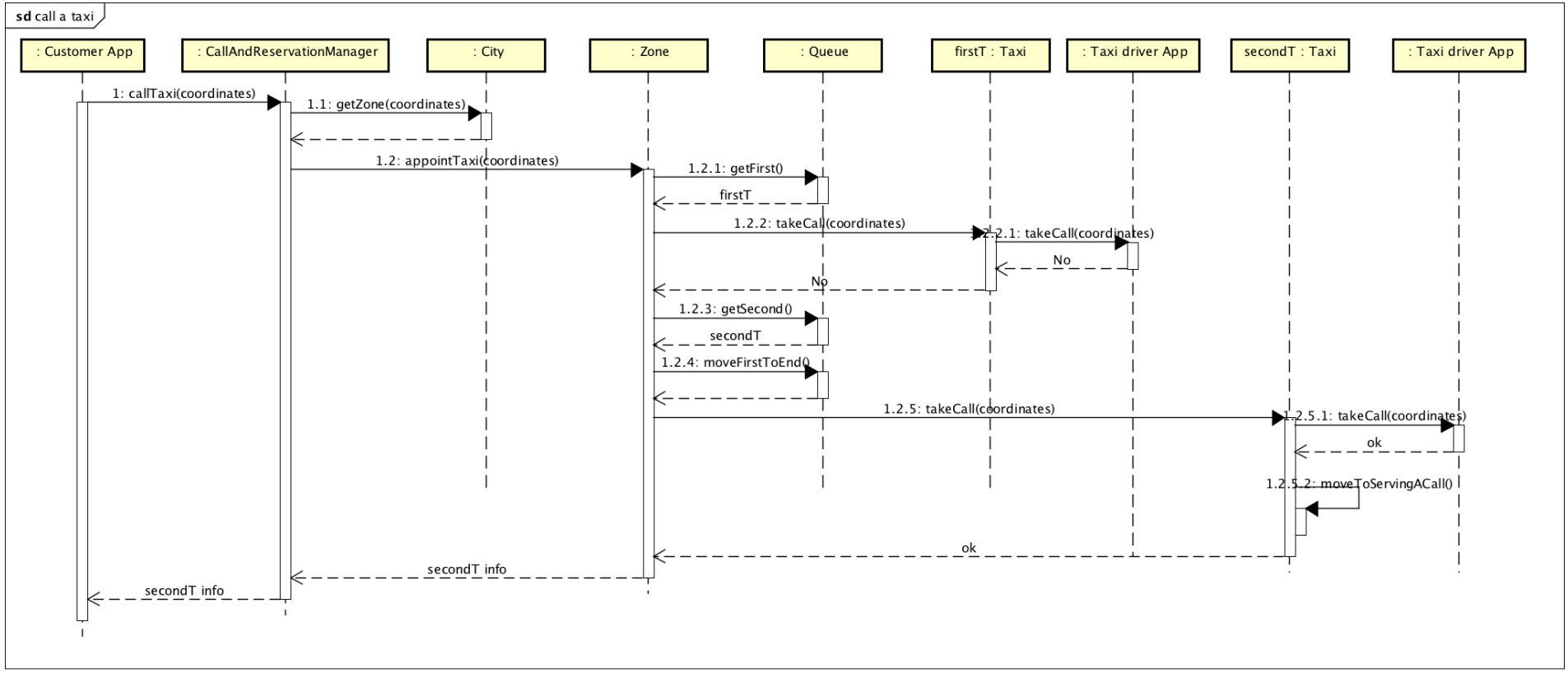
How do we understand how components are connected together?

- Try with sequence diagrams
- Use them to identify all main operations to be offered by components
- … and the way components interact



Question 1

- Can firstT and secondT make decisions on behalf of taxi drivers?
- Maybe not! Updated diagram:

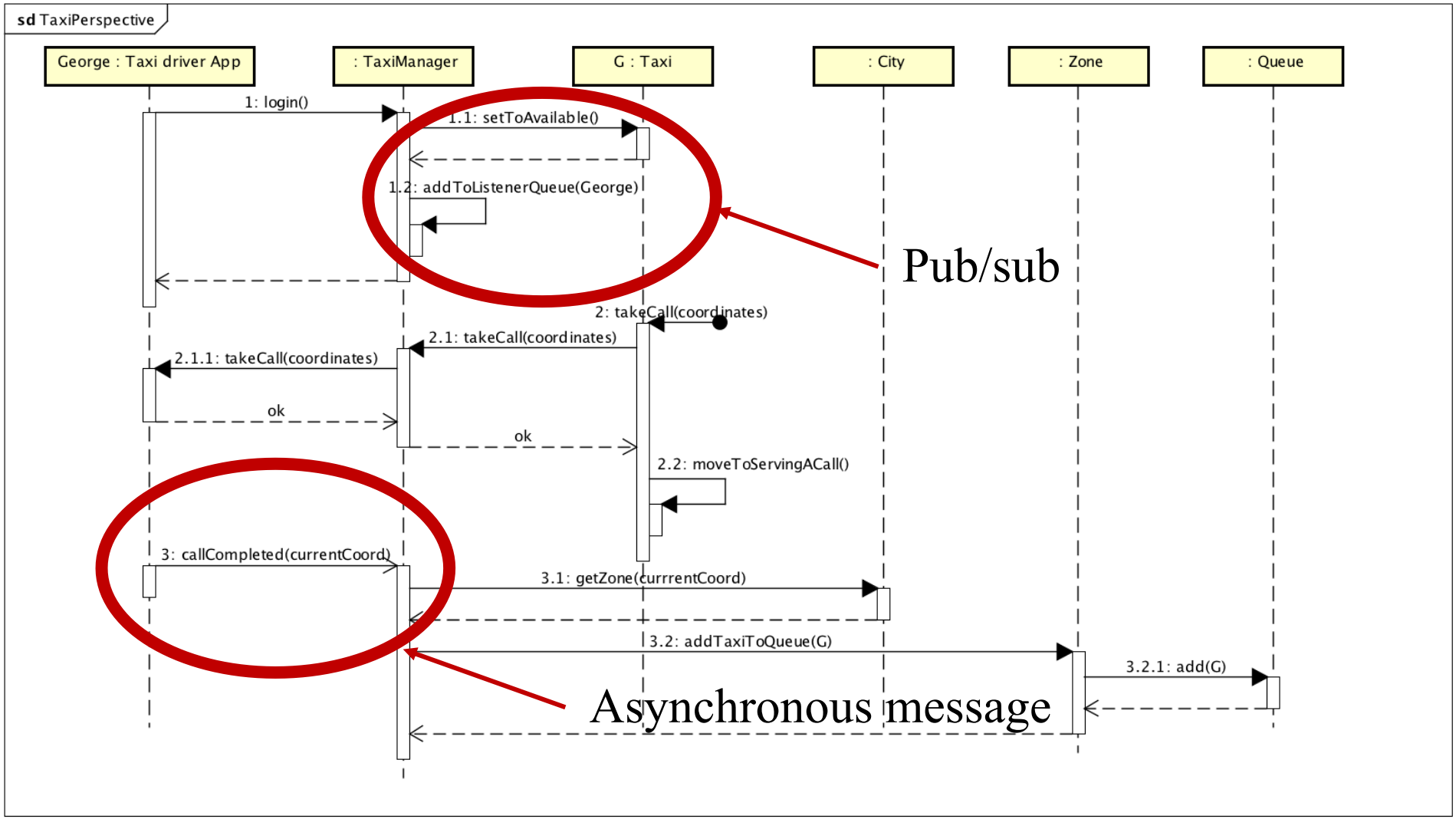


Question 2

- Can firstT and secondT interact directly with the Taxi driver App?
- Shouldn’t we delegate this interaction to the TaxiManager component?
- You do the change (but see next slide …)

We need to continue exploring behaviors…

- How can server-side components send messages to the Taxi driver App?



Other issues

- How many instances per component?
  - All managers could be available in multiple instances, one per request
  - They could be stateless
  - What about components encapsulating data structures?
- How to manage communication between distributed components?
- Which database?
- How to interface with database?
- Studying all this and assessing our technological ability, schedule, budget, should drive us to the selection of some underlying technology
  - Or, alternatively, to rely only on the standard OS and communication levels (see the NginX case)

So, how do we continue?

- Try to answer to all questions that come to your mind
- Continue exploring all possible behaviors
- Highlight all possible kinds of communication between distributed components
- You will be soon ready to define your architectural views and the interfaces of your components
- When do you stop?
  - Ideally, when you know you are at a point where you are able to start splitting the work and coding

Question

- Aren't we missing some elements/components from our design level diagrams?
  - Yes, we do
  - The analysis is incomplete
  - If you check the class diagram against the sequence you will see that we are missing some interactions…
  - We encourage you to further work on this

A note on non functional requirements

- We know they have a strong impact on architecture
  - Some can be addressed with stateless components, replication, load balancing
- We have seen some architectural cloud patterns to address them
- Others you will see in other courses
- Keep both functional and non-functional requirements into account while reasoning on your architecture…

Exercises from some written exams

Availability assessment

- Assume you have available three components offering the same functionality with different availability:
  - LOO shows 95% availability
  - BOO shows 98% availability
  - MOO shows 70% availability
- What component configuration (combination of the three components) would you use to obtain an overall availability equal to or above 99%?
  - Note that you can use only one copy of each component, and you could use fewer than all available components.

Availability assessment - solution

- the most natural way to achieve >=99% availability is to place components in parallel.
- It turns out it is sufficient to place BOO in parallel to LOO to achieve: 1-(0.02*0.05)=0.999 availability.
- If LOO could be available in multiple instances, even two instances of LOO would be sufficient: 1-(0.05*0.05)=0.9975

Exercise 1 on architecture definition

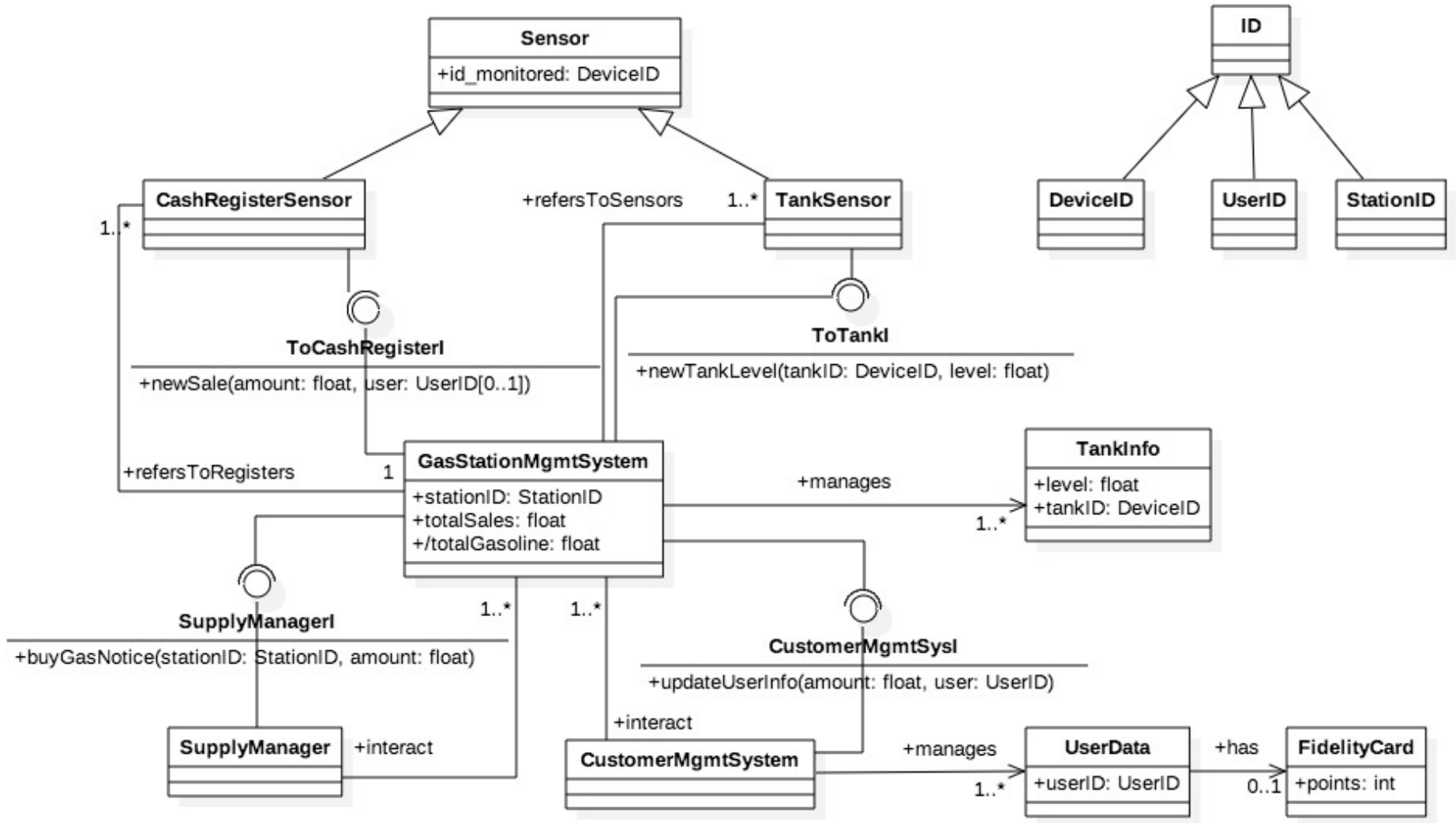
- A company managing gas stations aims at developing an information system offering the following functions:
  1. Visualize the status of all gas stations in terms of quantity of gasoline available and sales. The main system acquires data from sensors located within the gasoline tanks and from the cash registers.
  2. Manage the process through which a customer refuels his/her tank. If the customer has a fidelity card, the points corresponding to the gasoline purchase have to be added to this card.
  3. Alert the supplying department about the need to buy new gasoline when it goes below a certain threshold.
  4. Manage the fidelity cards assigned to customers, e.g., to assign points, apply discounts, …

Exercise 1 on architecture definition (cont.)

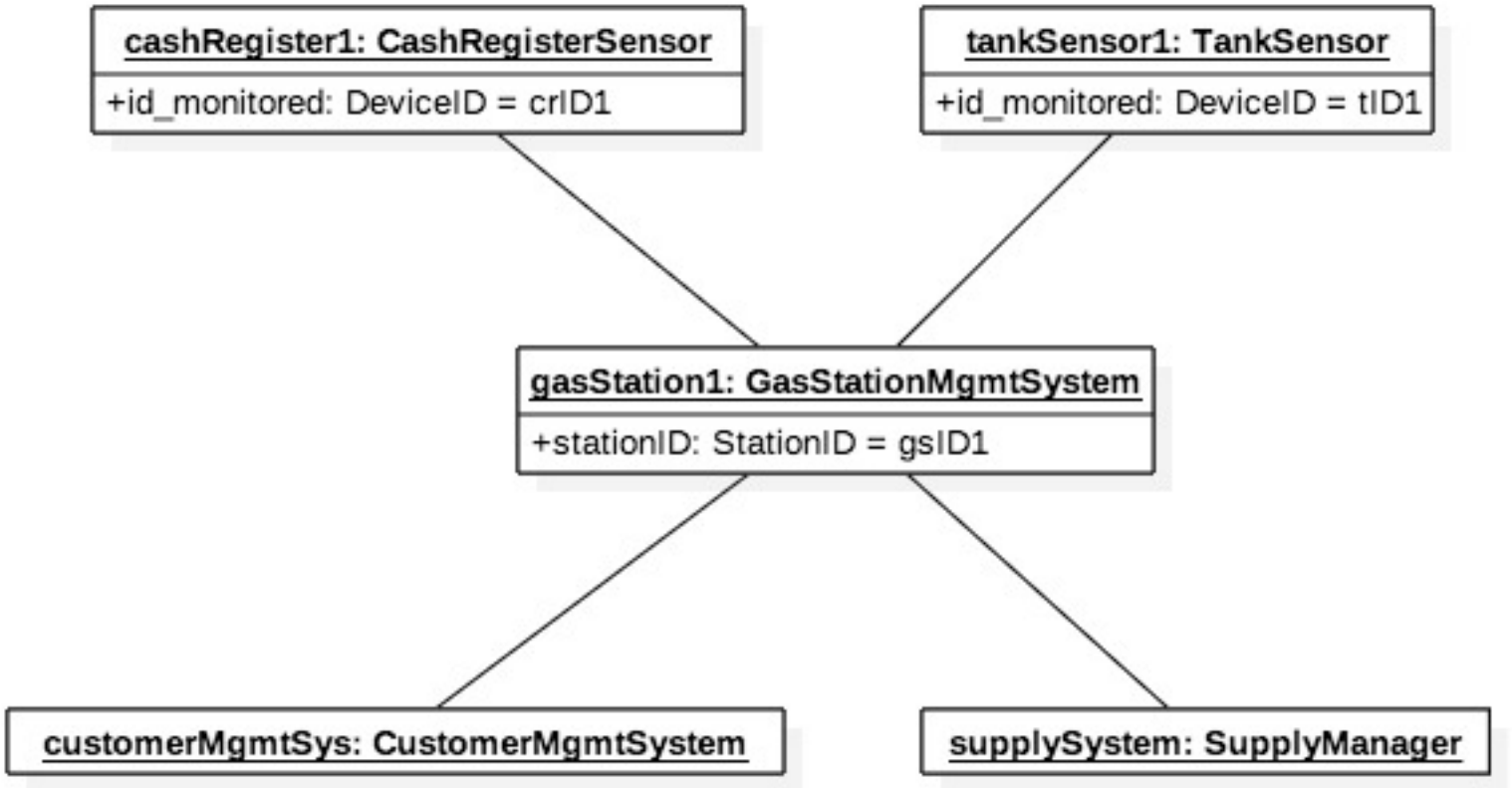
- Define the system architecture.
  - List all components belonging to the architecture, describe their functions and the way they interact with each other.
  - Draw a UML class or component diagram to provide a visual representation of the architecture.
  - Define a UML sequence diagram that describes the interaction between components needed to accomplish function 3 (alert the supplying department about the need to buy new gasoline).

Solution: class diagram

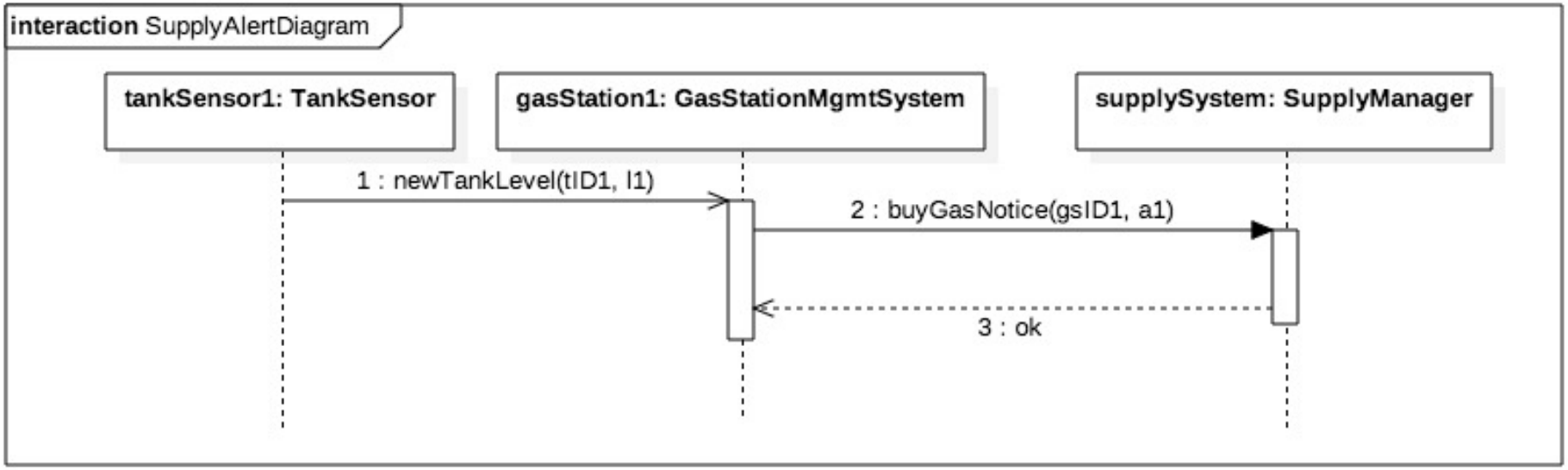
(without description; note that the exercise requires you to include also a description of elements)



Solution: object diagram
(not required, but useful in this case)



Solution: sequence diagram



Exercise 2 on architecture definition

- A public transportation company of a large urban area aims at opening a service for enabling sharing of electric vehicles. The vehicles can be cars, scooters and electric bicycles. The company offers to its users a number of vehicles and the charging stations plugs. Each user can either rent a vehicle (only one at a time), or use the charging stations provided by the company to recharge his own vehicle.
- You have been contacted by the company to develop the software system supporting the operation of the service.

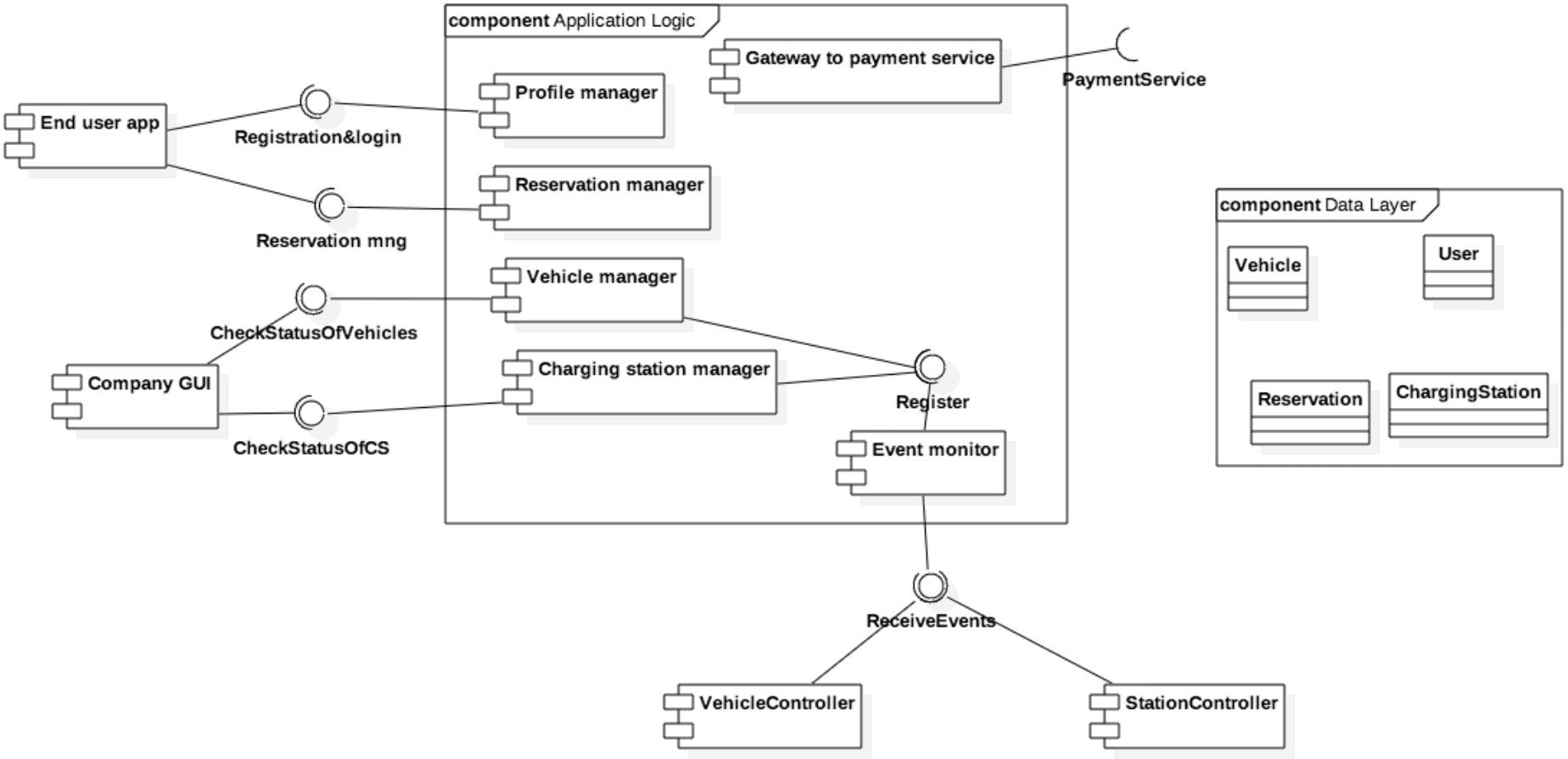
Exercise 2 on architecture definition (cont.)

- This system should allow:
  - Customers to place/modify/remove their reservations. When a customer places a reservation he/she should specify the time slot in which he/she will use the vehicle.
  - Customers to return a vehicle or a charging station plug. Vehicles have to be returned before the time slot associated to the corresponding reservation expires.
  - The company to enter information about vehicles and charging stations, to monitor the actual situation of its vehicles and charging stations, to charge users for the usage of their service.
- From the side of customers, the service should be accessible through a mobile device.

Exercise 2 on architecture definition (cont.)

- Define the high level architecture of the software system. Describe each component and the way it is connected to the others.
- Define the UML sequence diagrams to describe the behaviour of the system in the following case:
  - A customer is using one of the vehicles of the company and the time slot he has declared at the reservation is expiring. Thus, the system sends to the customer information about the closest charging stations where he can leave the vehicle. The customer replies asking for charging stations available in a different area. The system provides them together with the information on the extra-time needed to reach these other charging stations (this extra-time is calculated by relying on an external map service).

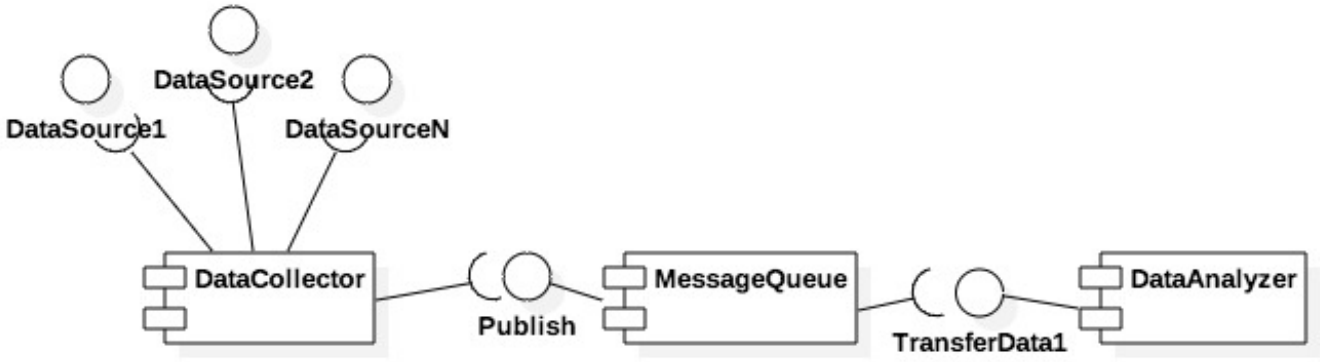
Incomplete high level architecture



- You are asked to complete it as needed

Exercise on Alloy and architectures
(exam of July 13th, 2018)

- Consider the following UML component diagram.



This diagram describes a software system that acquires and elaborates information from a number of different sources by polling them periodically. The `DataCollector` component is exploiting the interfaces offered by some data sources to acquire data and the interface of the `MessageQueue` to pass collected data to the other components. The `MessageQueue` exploits the interface offered by the `DataAnalyzer` to pass the data to this component.

Exercise (cont.)

- Write in Alloy the signatures that model a `DataSource`, a `DataCollector`, a `MessageQueue` and a `DataAnalyzer`. Make sure that you represent in the model the connections between components that are highlighted in the UML component diagram.
- Assume that we decide to replicate the `DataCollector` component. Model in Alloy the following possible configurations of the system:
  - **Configuration 1**: Each `DataCollector` replica is connected to a disjoint subset of `DataSource` components.
  - **Configuration 2**: All `DataCollector` replicas are connected to all `DataSource` components.
  - **Configuration 3**: `DataCollector` components are classified in master and slaves. There is always one `DataCollector` that acts as *master*.

A possible solution


```
sig DataSource {}
sig DataCollector {
  sources: set DataSource,
  queue : MessageQueue
}

sig MessageQueue {
  analyzer: DataAnalyzer
}

sig DataAnalyzer {}
```

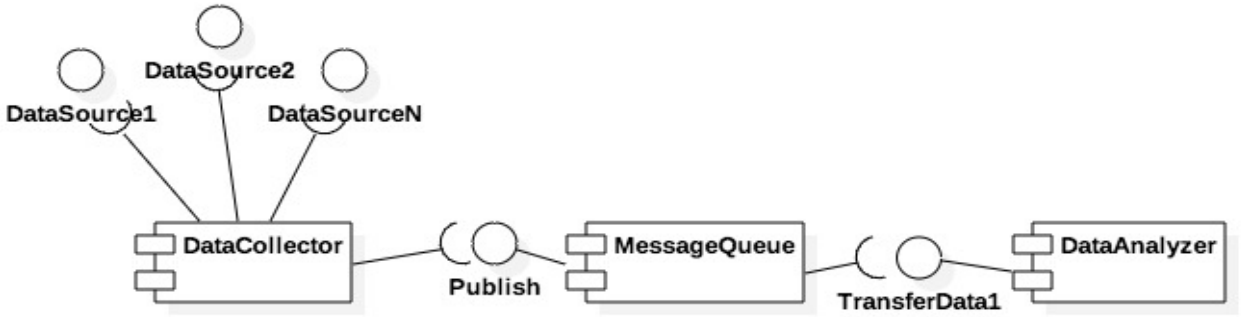
A possible solution (cont.)


```
sig Configuration {
  sources: set DataSource,
  collectors: set DataCollector,
  queue: MessageQueue,
  analyzer: DataAnalyzer
}
{ // all DataCollector components are connected to the
  // same MessageQueue, which is connected to the
  // DataAnalyzer of the configuration
  all coll : collectors | coll.queue = queue
  queue.analyzer = analyzer
  // also, the DataSource components used by the
  // DataCollector ones are exactly
  // those of the configuration
  collectors.sources = sources
}
```

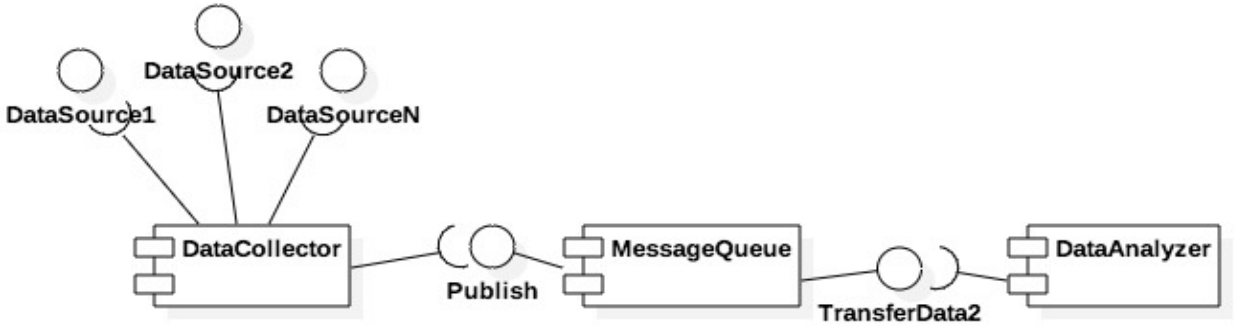
A possible solution (cont.)


```
// We capture the different configurations through
// extensions of the Configuration
// signature above; they add the necessary constraints
sig Configuration1 extends Configuration{}
{ all disj coll1, coll2 : collectors |
            coll1.sources & coll2.sources = none }

sig Configuration2 extends Configuration{}
{ all coll : collectors | coll.sources = sources }

sig MasterDataCollector extends DataCollector {}
sig SlaveDataCollector extends DataCollector {}

sig Configuration3 extends Configuration{}
{ all coll : collectors |
    coll in (MasterDataCollector | SlaveDataCollector)
  one coll : collectors | coll in MasterDataCollector
}
```

More on the data analysis example
(from the exam of June, 27th 2018)



- Consider this second version of the same system



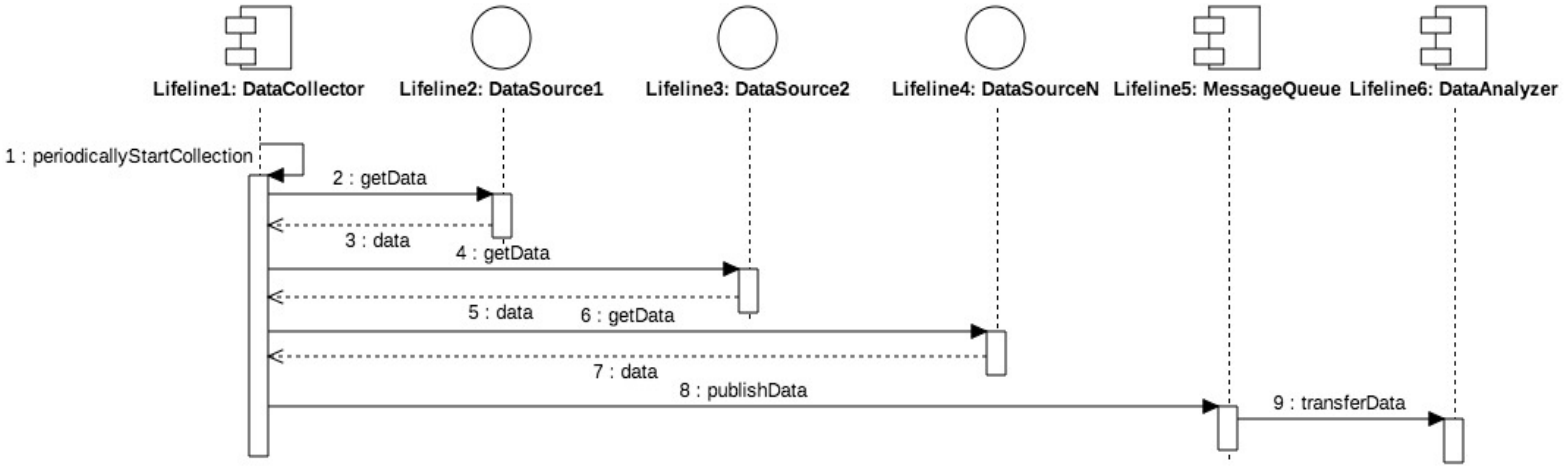
- Q1: What is the difference between the two?

More on the data analysis example
(from the exam of June, 27th 2018)

- Solution
  - In the second case, the MessageQueue does not actively push the data to the DataAnalyzer, but it offers interface TransferData2 so that the DataAnalyzer can pull data as soon as it is ready to process them. Also in this case, both a batch or a per data approach is possible. The rest of the system behaves as first one.
- Q2: Define two sequence diagrams that describe how data flow through the system in the two versions of the architecture

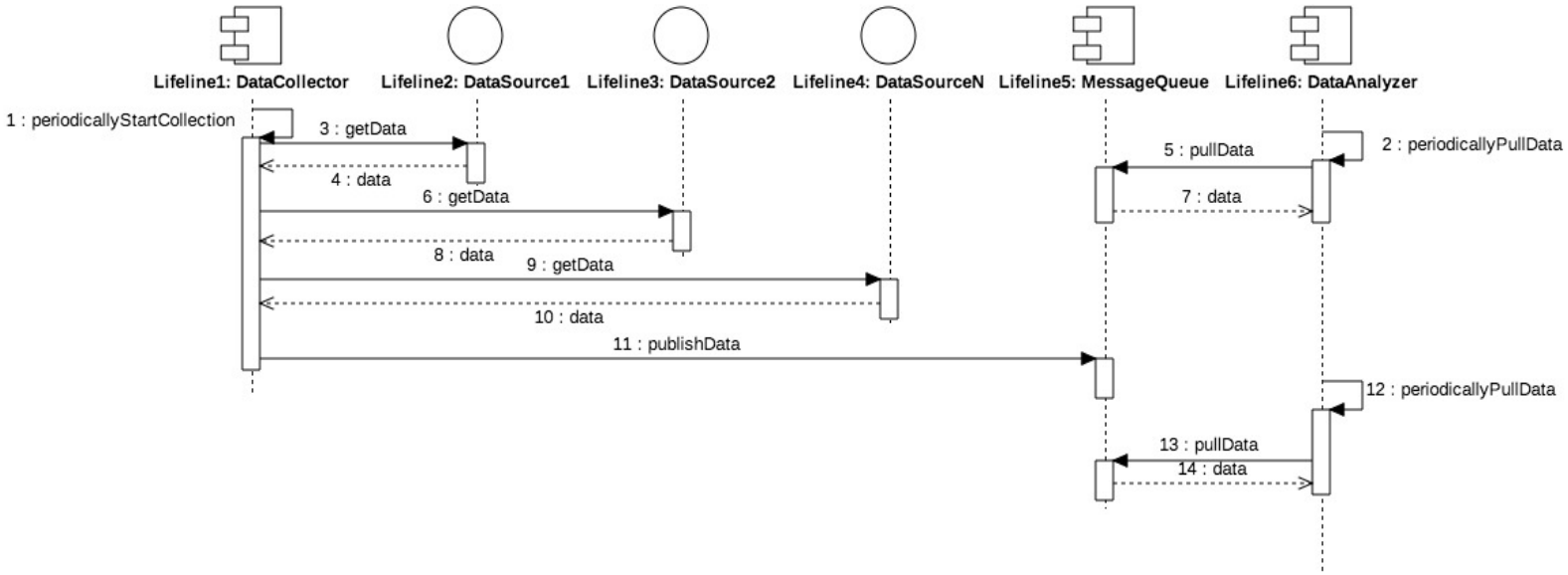
More on the data analysis example
(from the exam of June, 27th 2018)

- Solution to Q2
  - Sequence diagram compatible with the first component diagram



More on the data analysis example
(from the exam of June, 27th 2018)

- Solution to Q2
  - Sequence diagram compatible with the second component diagram



More on the data analysis example
(from the exam of June, 27th 2018)

- Assume that the components of your system offer the following availability:
  - DataCollector: 99%
  - MessageQueue: 99.99%
  - DataAnalyzer: 99.5%
- Provide an estimation of the total availability of your system (you can provide a raw estimation of the availability without computing it completely).

More on the data analysis example
(from the exam of June, 27th 2018)

- Data flow through the whole chain of components to be processed => series of component.
- The total availability of the system is determined by the weakest element, that is, the DataCollector.
  - A_{Total} = 0.99*0.9999*0.995 = 0.985
- Assuming that you wanted to improve this total availability by exploiting replication, which component(s) would you replicate? Please provide an argument for your answer.

More on the data analysis example
(from the exam of June, 27th 2018)

- If we parallelize the data collector adding a new replica, we can achieve the following availability:
  - $(1-(1-0.99)^2) * 0.9999 * 0.995 = 0.995$
- if we increase the number of DataCollector replica, we do not achieve an improvement as the weakest component becomes the DataAnalyzer.
- We can parallelize this component as well to further improve the availability of our system.

- How would such replication impact on the way the system works and is designed?

References (Architecture)

- Len Bass, Paul Clements & Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
- Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl, *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley and Sons, 1996.
- Christine Hofmeister, Robert Nord, Dilip Soni, *Applied Software Architecture*, Addison-Wesley 1999.
- Eric Gamma, John Vlissides, Richard Helm, Ralph Johnson, *Design Patterns*, Addison-Wesley 1995.
- Philippe Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, 12 (6), November 1995, IEEE.
  - http://www.rational.com/support/techpapers/ieee/
- Eberhardt Rechtin, *Systems Architecting: Creating and Building Complex Systems*, Englewood Cliffs NJ, Prentice-Hall, 1991.

References (Architecture)

- Eberhardt Rechtin & Mark Maier, *The Art of System Architecting*, CRC Press, 1997.
- *Recommended Practice for Architectural Description*, Draft 2.0 of IEEE P1471, May 1998
  - http://www.pithecanthropus.com/~awg/
- Mary Shaw, and David Garlan, *Software Architecture—Perspectives on an Emerging Discipline*, Upper Saddle River, NJ, Prentice-Hall, 1996.
- Bernard I. Witt, F. Terry Baker, and Everett W. Merritt, *Software Architecture and Design—Principles, Models, and Methods*, New York NY, Van Nostrand Reinhold, 1995.
- The World-wide Institute of Software Architects
  - http://www.wwisa.org

References (UML4Arch)

- Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- Kruchten, P.; Selic, B.; Kozaczynski, W.; Larsen, G. & Brown, A. W. (2001), Describing Software Architecture with UML., *in* Hausi A. Müller; Mary Jean Harrold & Wilhelm Schäfer, ed., 'ICSE' , IEEE Computer Society, pp. 777.