Politecnico
di Milano

Requirements Analysis and Specification

Structure of a RASD document and RASD assignment
Quick test on RE and Alloy
Exercises on RE with UML and Alloy

Politecnico
di Milano

Requirements Analysis and Specification Document (RASD)

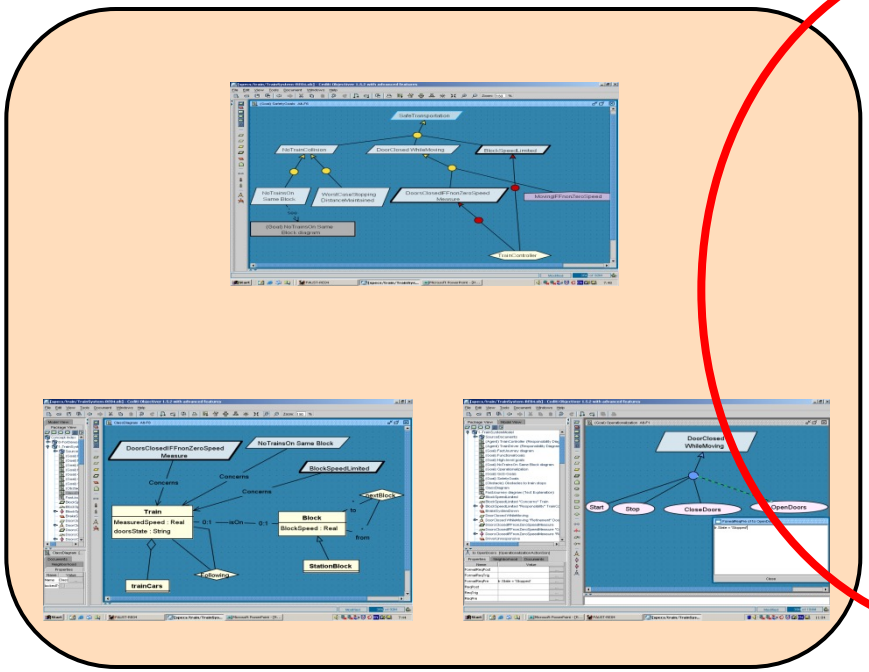
RASD

existing systems

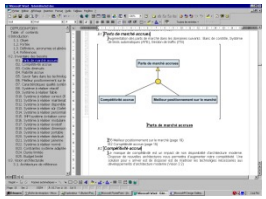
documents

elicitation & modelling

Requirements Models



generation of RE deliverables



requirements document

analysis & validation

Purposes of the RASD

- Communicates an understanding of the requirements
  - explains both the application domain and the system to be developed
- Contractual
  - may be legally binding!
- Baseline for project planning and estimation (size, cost, schedule)
- Baseline for software evaluation
  - supports system testing, verification and validation activities
  - should contain enough information to verify whether the delivered system meets requirements
- Baseline for change control
  - requirements change, software evolves

(slide: Steve Easterbrook)

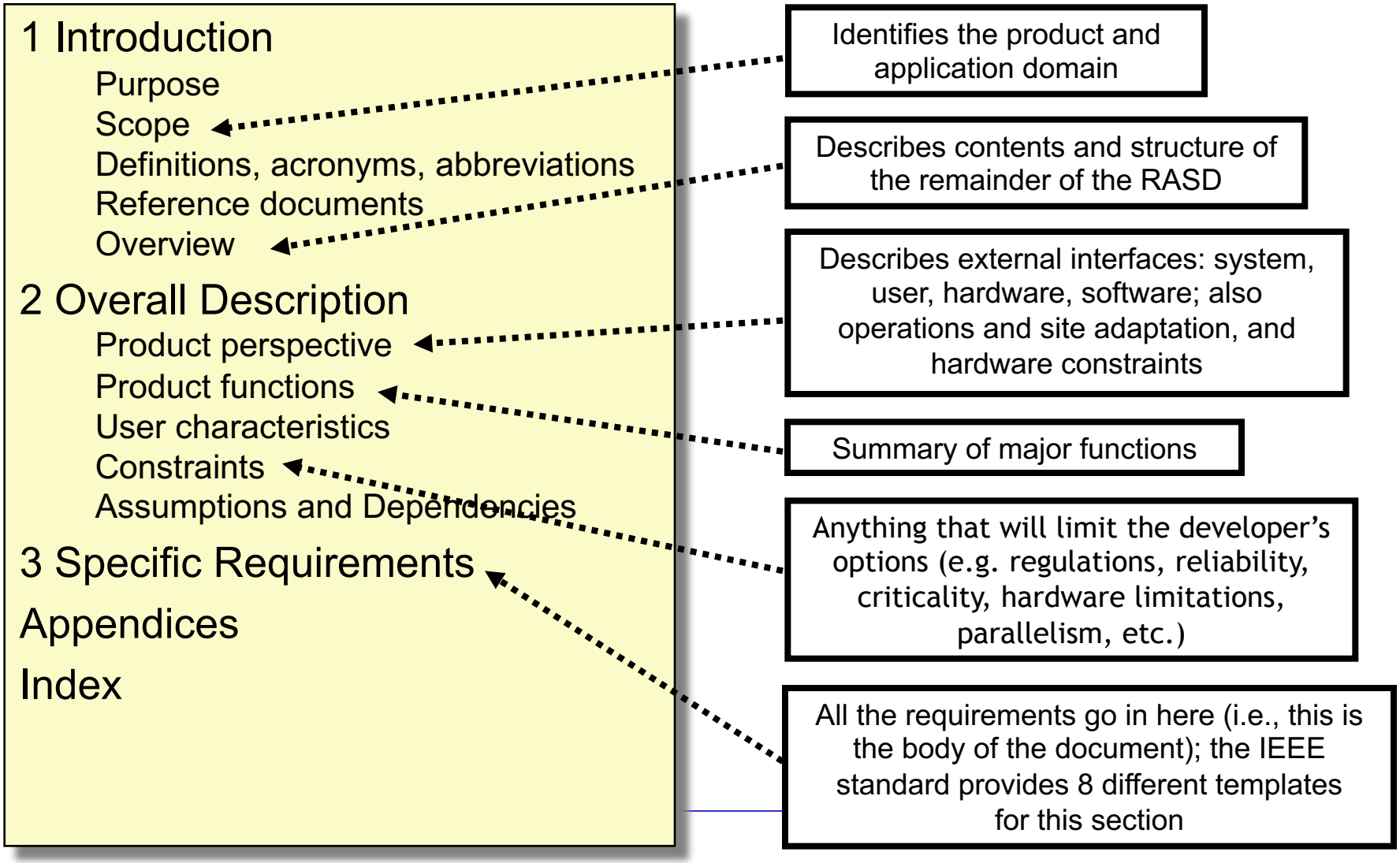
Audience of the RASD

- Customers & Users
  - most interested in validating system goals and high-level description of functionalities
  - not generally interested in detailed software requirements
- Systems Analysts, Requirements Analysts
  - write various specifications of other systems that inter-relate
- Developers, Programmers
  - have to implement the requirements
- Testers
  - determine that the requirements have been met
- Project Managers
  - measure and control the analysis and development processes

(slide: Steve Easterbrook)

IEEE Standard for RASD

Source: *Adapted from ISO/IEC/IEEE 29148 dated Dec 2011*



(slide: Steve Easterbrook)

IEEE STD Section 3 (example)

Source: *Adapted from ISO/IEC/IEEE 29148 dated Dec 2011*

3.1 External Interface Requirements
- 3.1.1 User Interfaces
- 3.1.2 Hardware Interfaces
- 3.1.3 Software Interfaces
- 3.1.4 Communication Interfaces

3.2 Functional Requirements

this section is organized by mode, user class, feature, etc. For example:

3.2.1 User Class 1

3.2.1.1 Functional Requirement 1.1

…

3.2.2 User Class 2

3.2.2.1 Functional Requirement 2.1

…

...

3.3 Performance Requirements

3.4 Design Constraints

3.4.1 Standards compliance

3.4.2 Hardware limitations

etc.

3.5 Software System Attributes
- 3.5.1 Reliability
- 3.5.2 Availability
- 3.5.3 Security
- 3.5.4 Maintainability
- 3.5.5 Portability

3.6 Other Requirements

(slide: Steve Easterbrook)

Target qualities for a RASD (1)

- Completeness
  - w.r.t. goals: the requirements are sufficient to satisfy the goals under given domain assumptions

    $$\text{Req and Dom} \models \text{Goals}$$

    - all Goals have been correctly identified, including all relevant quality goals
    - Dom represent valid assumptions; incidental and malicious behaviours have been anticipated
  - w.r.t. inputs: the required software behaviour is specified for all possible inputs
  - Structural completeness: no TBDs

(slide: adapted from A. van Lamsweerde)

Target qualities for a RASD (2)

- Pertinence
  - each requirement or domain assumption is needed for the satisfaction of some goal
  - each goal is truly needed by the stakeholders
  - the RASD does not contain items that are unrelated to the definition of requirements (e.g., design or implementation decisions)
- Consistency
  - no contradiction in formulation of goals, requirements, and assumptions

(slide: adapted from A. van Lamsweerde)

Target qualities for a RASD (3)

- Unambiguity
  - unambiguous vocabulary: every term is defined and used consistently
  - unambiguous assertions: goals, requirements and assumptions must be stated clearly in a way that precludes different interpretations
  - verifiability: a process exists to test satisfaction of each requirement
  - unambiguous responsibilities: the split of responsibilities between the software-to-be and its environment must be clearly indicated

(slide: adapted from A. van Lamsweerde)

Target qualities for a RASD (4)

- Feasibility
  - the goals and requirements must be realisable within the assigned budget and schedules
- Comprehensibility
  - must be comprehensible by all in the target audience
- Good Structuring
  - e.g., highlights links between goals, requirements and assumptions
  - every item must be defined before it is used
- Modifiability
  - must be easy to adapt, extend or contract through local modifications
  - impact of modifying an item should be easy to assess

(slide: adapted from A. van Lamsweerde)

Target qualities for a RASD (5)

- Traceability
  - must indicate sources of goals, requirements and assumptions
  - must link requirements and assumptions to underlying goals
  - facilitates referencing of requirements in future documentation (design, test cases, etc.)

(slide: adapted from A. van Lamsweerde)

RASD: In which sections do we include all we have learnt about requirements?

- The RASD does not necessarily follow the order of our mental process to the requirements
- Section 1
  - Purpose part → goals
  - Scope part → analysis of the world and of the shared phenomena
- Section 2
  - Product perspective → Scenarios, further details on the shared phenomena and a domain model (class diagrams and state diagrams)
  - Product functions → Requirements
  - User characteristics → Anything that is relevant to clarify their needs
  - Assumptions and dependencies → Domain assumptions

RASD: In which sections do we include all we have learnt about requirements?

- Section 3
  - More details on all aspects in Section 2 if they can be useful for the development team
  - Section 3.2 → Definition of use case diagrams, use cases and associated sequence/activity diagrams

A note on traceability

- Use cases are related to some requirements
- Keep track of this relationship through proper identifiers
  - E.g., RE.3 is associated with UC.3.1 and UC.3.2
- We may also have use cases that refer to multiple requirements
  - E.g., UC.3.1 may refer also to RE.2
    - …even thought the main relationship is with RE.3
  - Make this explicit in the presentation
    - E.g., you could build a traceability matrix

Traceability matrix

Raw ID	Goal ID	Req ID	Use Case ID	Comments
r1	G.1	RE.3	UC.3.1	
r2	G.1	RE.2	UC.3.1	

- This may grow during the development process, example:

Raw ID	Goal ID	Req ID	Use Case ID	Test case ID	Comments
r1	G.1	RE.3	UC.3.1	TC.3.1.1	
r2	G.1	RE.2	UC.3.1		

Homework

- Review the RASD available on Webeep, direct link
  - https://webeep.polimi.it/pluginfile.php/515844/mod_folder/content/0/ProjectToBeReviewed/RASD.pdf?forcedownload=1
    - It refers to the assignment described in this document: https://webeep.polimi.it/pluginfile.php/515844/mod_folder/content/0/ProjectToBeReviewed/Assignment.pdf?forcedownload=1
- Answer to the questionnaire here (one set of answers per group)
  - https://forms.office.com/r/mcxmaSTcdN
  - We will assign up to 1 point to clear and convincing answers.
- Deadline: November 1st at 23.59
- Your answers will be used as the basis for discussion during the labs of Nov 2 (Camilli's and Di Nitto's classes) and Nov 3 (Rossi's class)

Quick test on RE and Alloy

Politecnico
di Milano

The LuggageKeeper case – identification of phenomena, use of UML and Alloy

See also exam of February 16th, 2018

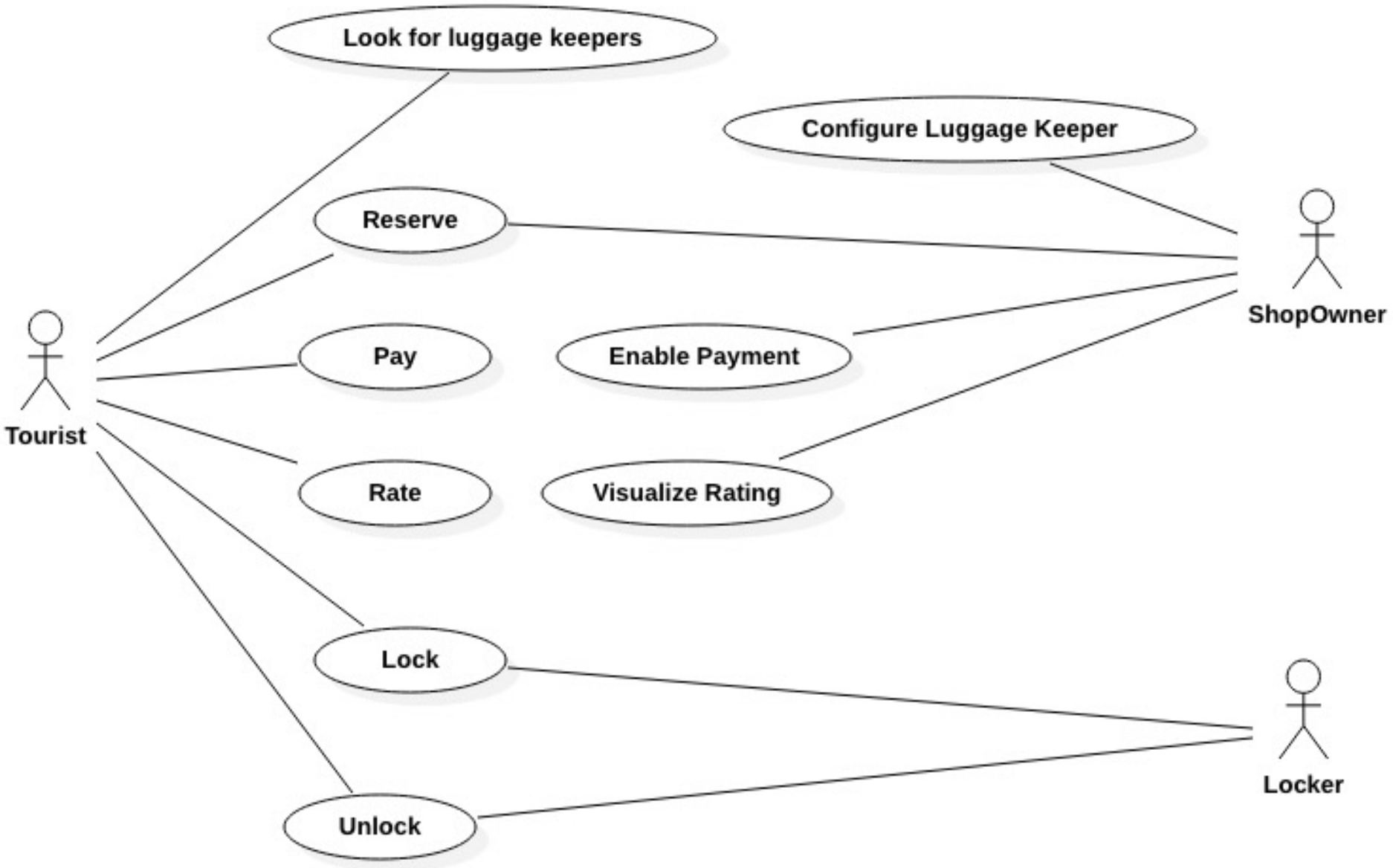
Informal description

- The company *TravelSpaces* decides to help tourists visiting a city in finding places that can keep their luggage for some time. The company establishes agreements with small shops in various areas of the city and acts as a mediator between these shops and the tourists that need to leave their luggage in a safe place.
- To this end, the company wants to build a system, called *LuggageKeeper*, that offers tourists the possibility to: look for luggage keepers in a certain area; reserve a place for the luggage in the selected place; pay for the service when they are at the luggage keeper; and, optionally, rate the luggage keeper at the end of the service.

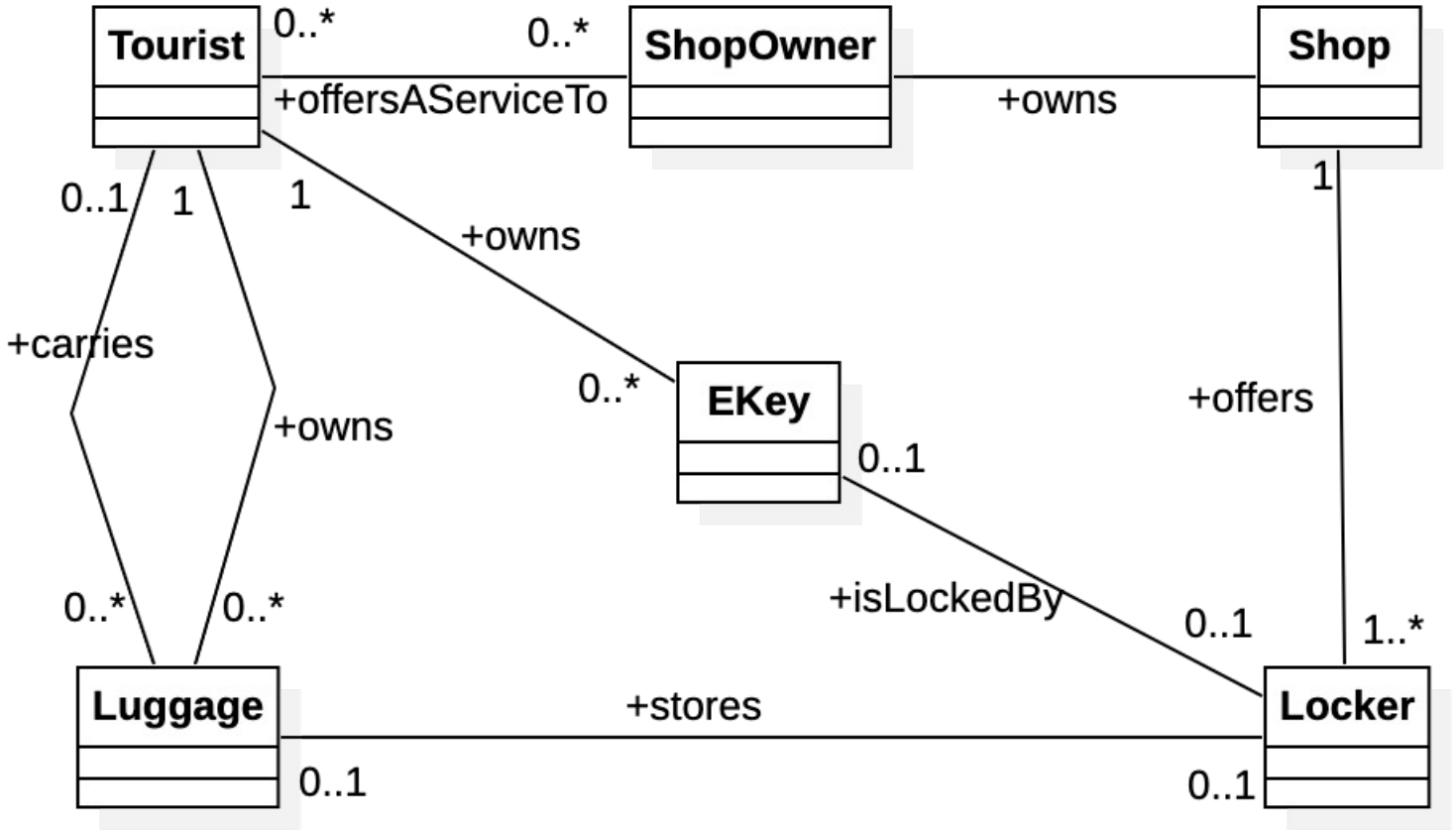
(some) possible world and machine phenomena

- World phenomena
  - Some users own various pieces of luggage.
  - Some users carry around various pieces of luggage.
  - Some pieces of luggage are safe
  - Some pieces of luggage are unsafe.
  - Small shops store the luggage in lockers.
- Shared phenomena
  - Some lockers are opened with an electronic key.
  - Some users hold various electronic keys.

Use cases


Look for luggage keepers
Configure Luggage Keeper
Reserve
Pay
Enable Payment
Rate
Visualize Rating
Lock
Unlock
Tourist
ShopOwner
Locker

Problem domain model (class diagram)



Alloy signatures


```
abstract sig Status{}
one sig Safe extends
Status{}
one sig Unsafe
        extends Status{}

sig Luggage{
   luggageStatus :
                one Status
}

sig EKey{}
```

```
sig User{
   owns : set Luggage,
   carries : set Luggage,
   hasKeys : set EKey
}

sig Locker{
   hasKey : lone EKey,
   storesLuggage : lone Luggage
}

sig Shop{
   lockers : some Locker
}
```

Various constraints could be added
(e.g., the owner of a luggage is unique)

A domain assumption

- any piece of luggage is safe if, and only if, it is with its owner, or it is stored in a locker that has an associated key, and the owner of the piece of luggage holds the key of the locker

```
fact DAsafeLuggages {
  all lg : Luggage |
    lg.luggageStatus in Safe
    iff
    all u : User |
        lg in u.owns
        implies
        ( lg in u.carries
          or
          some lk : Locker | lg in lk.storesLuggage and
                             lk.hasKey != none and
                             lk.hasKey in u.hasKeys )
}
```

A requirement

- a key opens only one locker

```
fact requirement {
  all ek : EKey |
    no disj lk1, lk2: Locker |
       ek in lk1.hasKey and ek in lk2.hasKey
}
```

A goal

- for each user all his/her luggage is safe

```
pred goal {
  all u : User |
    all lg : Luggage |
        lg in u.owns
        implies
        lg.luggageStatus in Safe
}
```

Operation GenKey

- Given a locker that is free, GenKey associates with it a new electronic key

```
pred GenKey[lk, lk' : Locker] {
  //precondition
  lk.hasKey = none
  //postcondition
  lk'.storesLuggage = lk.storesLuggage
  one ek : EKey | lk'.hasKey = ek
}
```

Politecnico
di Milano

Modeling the Airbus braking logic with Alloy

Is the UML spec complete?

- We have described all phenomena
  - Aircraft, wheels, sensor, reverse thrust system
- … and a use case EnablingReverseThrust
- Are we missing something?
- Are we representing goals, domain properties and requirements?
  - Goal
    - Reverse_enabled ⇔ Moving_on_runway
  - Domain properties
    - Wheel_pulses_on ⇔ Wheels_turning
    - Wheels_turning ⇔ Moving_on_runway
  - Requirements
    - Reverse_enabled ⇔ Wheels_pulses_on

Is the UML spec complete?

- Pure UML does not help us in expressing assertions
- We can complement its usage with
  - Some formal or informal description of these assertions

Modeling the Airbus braking logic with Alloy


```
abstract sig Bool {}
one sig True extends Bool {}
one sig False extends Bool {}

abstract sig AirCraftState {}
one sig Flying extends AirCraftState {}
one sig TakingOff extends AirCraftState {}
one sig Landing extends AirCraftState {}
one sig MovingOnRunaway extends AirCraftState {}
```

… for landing, we are not considering the movement due to takeoff as it is not relevant to our analysis

Modeling the Airbus braking logic with Alloy


```
sig Wheels {
        retracted: Bool,
        turning: Bool
}{turning = True implies retracted = False}

sig Aircraft {
        status: one AirCraftState,
        wheels: one Wheels,
        wheelsPulsesOn: one Bool,
        reverseThrustEnabled: one Bool
}{status = Flying implies wheels.retracted = True}
```

Modeling the Airbus braking logic with Alloy


```
fact domainAssumptions {
all a: Aircraft | a.wheelsPulsesOn = True
                  <=> a.wheels.turning = True
all a: Aircraft | a.wheels.turning = True
                  <=> a.status = MovingOnRunaway}

fact requirement {
all a: Aircraft | a.reverseThrustEnabled = True
                  <=> a.wheelsPulsesOn = True}

assert goal {
all a: Aircraft | a.reverseThrustEnabled = True
                  <=> a.status = MovingOnRunaway}
check goal
```

No counterexamples are found!

But note that, still, this is the wrong model of our world: the spec is internally coherent, but it does not correctly represent the world

References and interesting sources about Requirement Engineering

- M. Jackson, P. Zave, "Deriving Specifications from Requirements: An Example", Proceedings of ICSE 95, 1995
- M. Jackson, P. Zave, "Four Dark Corners of Requirements Engineering",TOSEM, 1997
- B. Nuseibeh, S. Easterbrook, "Requirements Engineering: A Roadmap", Proceedings ICSE 2000
- A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley and Sons, 2009
- M. Jackson, Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices, ACM Press Books, 1995
- S. Robertson and J. Robertson, Mastering the Requirements Process, Addison Wesley, 1999
- Requirements Engineering Specialist Group of the British Computer Society http://www.resg.org.uk/
- B. Bruegge & A.H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, September 25, 2003
- T. E. Bell and T. A. Thayer. 1976. Software requirements: Are they really a problem?. In Proceedings of the 2nd international conference on Software engineering (ICSE '76). IEEE Computer Society Press, Los Alamitos, CA, USA, 61-68.
- F. P. J. Brooks, "No Silver Bullet Essence and Accidents of Software Engineering," in Computer, vol. 20, no. 4, pp. 10-19, April 1987. doi: 10.1109/MC.1987.1663532
- Slides by Emmanuel Letier UCL