Review

From: STATEMATE manual
Review

On

Show Clock

- Show 12hr clock
- Show 24hr clock

Show Timer

- but3[inState(TimerRunning)]: lapTime=timer
- but3: Show Timer
- but3[resetTimer(timer)]

Light

- but3/light

TimerNotRunning

- but3[inState(ShowTimer)]: beep

TimerRunning

- but2[inState(ShowTimer)]: beep

U Waterloo SE1 (Winter 2006) – p.4/44
Today’s Agenda

- Reference model for requirements engineering
  - Motivation
  - System, environment, interface
  - Requirements vs specifications (levels of requirements)
  - Domain knowledge

Required Reading: Jackson and Zave (course pack)
Review: Requirements Specifications

Requirements are the desired goals or behaviour of the system.

- What does the customer want?

A requirements specification is a description of the proposed behaviour of a computer-based system (CBS).

- What is assumed of the environment in which the CBS operates?

- Given these assumptions, what should the CBS do?

- What are the boundaries on this CBS?

We are concentrating on software requirements specifications (SRS), but we need to remember that often software is just one part of the system.
Main Ideas

Know the boundaries of your system (what is observable to the system?).
Main Ideas

Know the boundaries of your system (what is observable to the system?).

System requirements contribute towards business goals (i.e., there are different levels of requirements descriptions).
Motivation

You may have noticed a discrepancy between the types of requirements that seem to come out of the requirements elicitation process and the type of modelling you are doing for the course project — for the SRS that you are writing.

Requirements that are elicited tend to be:

- high-level descriptions of functions that the system should perform (goals),
- and some non-functional requirements that the system should exhibit.
Motivation

The questions that you are asking in your customer sessions are probably much more detailed, asking what the system’s response should be to a particular input event.

The process that is happening here is that you are going from high-level goals to descriptions in terms of only the INTERFACE between the system and the environment.
We view the hardware and software as building a **System** (machine) that operates in the **Environment** through an **Interface** that is shared between the System and the Environment (sensors, actuators), and both the System and the Environment sit in the World. Systems are built to improve the Environment.
A **System** can be any socio-technical artifact that is to be constructed and need not be physical; e.g., it could be a process.

Those aspects of the real world that are relevant to the improvement that we wish to achieve is called the **Environment**. We can ignore all but the relevant parts of the world.

This Environment is sometimes called the **application domain**.
System

Note that a system is not just hardware or software. It is a mixture of hardware and software. However, generally, only the software is modifiable. Hence, we tend to talk about Software Engineering and Software Requirements. Nevertheless, we really mean System Engineering and System Requirements.
The Interface between the Environment and the System are the phenomena that are shared between the Environment and the System (set of events and actions that are observable and/or controllable by both the Environment and the System).

These shared phenomena are visible to both the Environment and the System.

A given item in the Interface may be sensed or controlled by the Environment or the System but generally not by both. The usual situation is that a given item is controlled by one and sensed by the other and thus serves to communicate from the one to the other.
Requirements vs Specifications

When you are going from high-level goals to descriptions in terms of only the INTERFACE between the system and the environment, . . .

. . . you are experiencing the difference observed by Jackson and Zave between the system’s requirements and its specification (or you can think of these as different levels of requirements).

Often in the elicitation process, you hear requirements and have to produce a specification.
Requirements vs Specifications

Requirements, $R$, are a collection of statements about phenomena in the Environment, $Env$, that we want the System, $Sys$, to help make true.

A Specification, $S$, is a collection of statements that describe a System’s external behaviour, as observable through the Interface, $I$.

- A specification refers only to shared phenomena in the Interface.
- A specification can constrain only shared phenomena that the System itself can control.
Requirements vs Specifications

A Requirement statement may be also a Specification statement, but it does not have to be.

A Requirement statement is a Specification statement if it is expressed in terms of $I$.

A Requirement statement is not a Specification statement if it is not expressed in terms of $I$.

Requirements are often called user requirements or needs or business goals or domain requirements.

Specification is often called system requirements or product requirements.
Domain Knowledge

Requirements are concerned with describing things that we want the System to help make true. The System might not be able to accomplish these things by itself. Properties of the Environment might be necessary for the System to meet the Requirements.

These properties are called Domain Knowledge, $D$.

Domain Knowledge is properties that we know or assume to be true of the Environment.

Requirements are properties that we wish to make true of the Environment.
Reference Model

Thus, the following relationship must hold:

\[ D, S \models R \]

- \( D \) is domain knowledge
- \( S \) is the specification
- \( R \) is the requirements

The specification describes the behaviour of a System that realizes the requirements.

The requirements are a goal-oriented description of the desired behaviour.

When we write the specification, we have to start asking how the System will realize the requirements.
Consistency

\[ D, S \models R \]

- \( D \) is domain knowledge
- \( S \) is the specification
- \( R \) is the requirements

Additionally:

There must exist an environment that satisfies \( D \) (consistency of domain knowledge). The specification can be satisfied in a choice of environment that satisfies \( D \).

Consistency

\( D, S \vdash R \)

- \( D \) is domain knowledge
- \( S \) is the specification
- \( R \) is the requirements

Additionally:

- There must exist an environment that satisfies \( D \) (consistency of domain knowledge).
- The specification can be satisfied in a choice of environment that satisfies \( D \).

Example: Airplane

**Requirements:** An airplane may engage reverse thrust only when on the runway.

**Domain:** The wheels are turning if and only if they are on the runway.

**Domain:** The wheels pulse if and only if the wheels are turning.

**Specification:** The airplane can reverse if and only if the wheels pulse.
Example: Park User Fees

Suppose that the city of Waterloo decides to raise funds by instituting users fees for public parks. It needs to implement a complete system of money collection, security, etc. The idea is to collect a fee from each human park user on entry to park (no fee to leave).

Requirements:

1. Ensure that no person may enter park without paying.

2. Ensure that anyone who has paid may enter the park.
Example: Park User Fees

Notice that there is no mention of possible solutions. They may decide to have

1. a manned kiosk, in which a person collects money and lifts and lowers the gate, or

2. an automated system.

Only the fundamental goal of the client has been stated — to collect money from visitors to the park.
**Example: Park User Fees**

**Solution #1:** Employ human fee collectors. Enforce security by instituting the Waterloo Park Militia, armed guards who make certain no one uses a park without paying a user fee.
Example: Park User Fees

Solution #1: Employ human fee collectors. Enforce security by instituting the Waterloo Park Militia, armed guards who make certain no one uses a park without paying a user fee.

Solution #2: Use chain link fences for security, use turnstiles with automated token collection. After some research, we find appropriate turnstile hardware, but it’s brand new technology so we must create the embedded software system.

There is a barrier through which to enter a park. A person inserts a token, the barrier unlocks, allowing the person to push the barrier and enter the park.
## Example: Park User Fees

### Inputs:

<table>
<thead>
<tr>
<th>Environment (Non-interface)</th>
<th>Interface</th>
</tr>
</thead>
</table>

### Outputs:

<table>
<thead>
<tr>
<th>Environment (Non-interface)</th>
<th>Interface</th>
</tr>
</thead>
</table>
Designations

A designation is the term by which a phenomena will be known in requirements and specifications.

Interface Designations:

- $Push(e)$
- $Token(e)$
- $Lock(e)$
- $Unlock(e)$

The above is in the style of Jackson and Zave, where each designation is a predicate, e.g., $Push(e)$ means event $e$ is a push phenomena.
Interface

The interface for the park user fees system consists of:

- **token slot**
  - system *observes* a token entering the token slot
  - environment *controls* the insertion of tokens \( Token(e) \)

- **barrier**
  - locking and unlocking
    - system *controls* whether the barrier is locked or unlocked \( Lock(e), Unlock(e) \)
    - environment *observes* whether the barrier is locked or unlocked
  - motion
    - environment *controls* the pushing of the barrier \( Push(e) \)
Specification

- The specification is a description of the proposed behaviour of the system.
- Describes what needs to occur in terms of the behaviour of the shared phenomena, $S$.

\[ S \subseteq Env \cap Sys (= I) \]

Example Specification: If a token has been inserted into the token slot, then the barrier can be pushed one rotation.

Note that this doesn’t say: If a visitor puts a token into the token slot, then he or she can push the barrier one rotation. That references parts of $Env$ that aren’t in $I$ and is a requirement, rather than a specification!
Specification

A specification maps inputs to outputs at the interface.

Even though you’re not enumerating a function from inputs to outputs, you are in effect describing the System’s response to all possible inputs:

In a UML system state diagram, you divide the System’s behaviour into states, in which every state defines

- the input events to which the System is ready to react and
- what the System’s resulting actions will be.
We would like it that

\[ S \vdash R \quad \text{(Specification implies Requirements)} \]

If we want to demonstrate that the specifications meet the requirements, normally we are not able to do so without adding some additional information.
Domain Knowledge

We would like it that

\[ S \vdash R \quad (\text{Specification implies Requirements}) \]

If we want to demonstrate that the specifications meet the requirements, normally we are not able to do so without adding some additional information.

\[ D, S \vdash R \quad (D = \text{Domain Knowledge}) \]

We need to document our assumptions about the connection between the interface events and other environmental events.

We may need to document other assumptions about how the Environment behaves.
Domain Knowledge

- Domain Knowledge ($D$) is a description of the problem domain.
  i.e., facts & assumptions about the Environment

\[ D \subseteq Env \]

- Usually, system will behave as desired only if these assumptions, $D$, hold.

- It is worthwhile to make explicit all of those facts that seem “obvious”.
  The shared phenomena and the interactions through them are the visible glue in the system; often, the domain is the invisible piece that causes problems if missing.
Example: Park User Fees

Requirements:

1. Ensure that no person may enter park without paying.
2. Ensure that anyone who has paid may enter the park.

Specifications:
Example: Park User Fees

The domain assumptions include the following facts about the environment:

Thus, we realize requirements in two ways

1. building a system that performs the specified actions and

2. making assumptions about how the environment will behave.
Observable Behaviour

For every event or action, we need to decide

1. Can the system observe the event or action? (i.e., Is this event/action part of the interface?)

2. If it is part of the interface, is the event or action controlled by
   - the system? or
   - the environment?,

3. If it is part of the environment, are there any domain assumptions about the event/actions?
Validation

Validation is the evaluation of a specification:

\[ D, S \vdash R \]

- Must be able to argue that the specification of the system plus the assumptions about the domain are enough to satisfy the requirements.

- If you can’t make this argument successfully, then you need to:
  1. strengthen the specification, or
  2. strengthen the domain knowledge, or
  3. weaken the requirements.

It must also be possible to verify that a program meets $S$. 
Ex: Danish Shipyard

- Our precalculations shall be accurate to within 5%.
- The product shall have recording and retrieval functions for experience data.
- The system shall have screen pictures as shown in Appendix xx.
Ex: Danish Shipyard

Our precalculations shall be accurate to within 5%.

The product shall have recording and retrieval functions for experience data.

The system shall have screen pictures as shown in Appendix xx.

WHY
Ex: Danish Shipyard

- Our precalculations shall be accurate to within 5%.
- The product shall have recording and retrieval functions for experience data.
- The system shall have screen pictures as shown in Appendix xx.

Lauesen considers the third one a design-level requirement. But we’ll discuss in a later lecture that we want to have this in the SRS because it’s important to the user.
Example: Traffic Light

Requirement: Allow road traffic to cross an intersection safely, without colliding with traffic travelling in a different direction.

A System that can satisfy this requirement is a traffic light.

The specification of such a System includes statements about switching on coloured lights at particular times and in defined orders.

How can lights stop one direction of traffic and allow the other traffic to continue?
Example: Traffic Light

The answer lies in the domain knowledge ($D$), which includes such statements as:

From these $D$ statements combined with $S$, one can prove $R$. 
Example: Train Crossing

Req: train is in crossing $\implies$ gate must be down

S1: if approaching train is 200m away, lower gate
Example: Train Crossing

Req: \[ \text{train is in crossing} \implies \text{gate must be down} \]

S1: \[ \text{if approaching train is 200m away, lower gate} \]
Summary

Requirements (business goals) are useful to include in an SRS because:

- Help all stakeholders understand the purpose of the system
- Help in considering new solutions
- Make it possible to check that the specification contributes towards the requirements

The specification:

- Describes the input/output relationship for the product
- Is used by the developers
- Is validated
- Should be as complete as possible
- Is verified
SRS

There are really two documents that are produced during the requirements phase:

1. one that documents the requirements, and

2. another that documents the specifications.
SRS: Requirements

- Requirements Document
  - requirements (goals)
  - domain model
    - entities, relations, attributes (some may not be observable by the system)
    - event sequences (include some domain knowledge)
    - actors
  - purpose
  - preferences and priorities
  - design constraints
SRS: Specification

- Specification Document
  - interfaces (just what is observable by the system)
  - specifications
  - screens
  - protocols
  - training
The SRS you are writing includes requirements, specifications, and domain knowledge.

The requirements and domain knowledge are documented in Section 1 and 2, as part of the documentation of the purpose of the system.

The specifications are documented in Section 3–7, which describes the system’s interfaces and its behaviour in terms of inputs and responses.
Summary

- Reference model for requirements engineering
  - Motivation
  - System, environment, interface
  - Requirements vs specifications (levels of requirements)
  - Domain knowledge

Next Lecture: UML Concept (Class) Diagrams

Reading: Arlow and Neustadt Ch. 6,7,8,9 (skip 9.5), 10, 11, 18.1-18.10