Requirements Verification in CPSs

Luca Pulina[1] **Armando Tacchella**[2] **Simone Vuotto**[1,2]

[1]University of Sassari (Italy)
[2]University of Genoa (Italy)

CPS Summer School 2018
Alghero, September 18, 2018

Context

Requirements for Cyber-Physical Systems (CPSs)

- Understanding
- Writing
- Analyzing
- Refactoring
- Validating

Motivation

CPSs pose **unique** challenges in that they are

- Hierarchical
- Heterogeneous
- Distributed
- Connected
- Socio-technical
- Autonomous/adaptive

Challenges

CPSs **melt together** different communities:

- Hardware design
- Software engineering
- Control/automation
- Optimization
- Artificial Intelligence
- Psychology/sociology

"Tower of Babel" phenomenon: each community speaks its own language and sees the same problems from a different perspective.

Objectives

+

Transfer methodology/best practices from software to CPSs

What is in the tutorial

- A general introduction to requirement engineering
- Our research explained: from software requirements to CPSs requirements
- Introducing our prototype ReqV
- Hands-on session on ReqV

What is **not** in the tutorial

- A solution to requirement engineering in CPSs
- A common language that cuts across communities
- A focus on issues outside software and control (mostly)

Outline

1. Requirements Engineering – A quick overview
   - Functional and Non-functional Requirements
   - Requirements Engineering Processes
   - Requirements Specification
   - Requirements Validation
2. Formal consistency checking
3. Requirements Management with ReqV
4. Hands-on session on ReqV
5. Conclusions

Outline

1 Requirements Engineering – A quick overview
- Functional and Non-functional Requirements
- Requirements Engineering Processes
- Requirements Specification
- Requirements Validation

2 Formal consistency checking

3 Requirements Management with ReqV

4 Hands-on session on ReqV

5 Conclusions

Requirements Engineering

The **process** of establishing

- the **services** that a customer requires from a system; and
- the **constraints** under which it operates and is developed.

Requirements Engineering

The **process** of establishing

- the **services** that a customer requires from a system; and
- the **constraints** under which it operates and is developed.

Requirements

- Descriptions of the system services and constraints that are **generated during the requirements engineering process**.
- They may range from a **high-level abstract statement** of a service or of a system constraint to a **detailed mathematical** functional specification.

Types of requirement

- **User requirements**
  - Statements in **natural language plus diagrams** of the services the system provides and its operational constraints.
  - Written for customers.

Types of requirement

- **User requirements**
  - Statements in **natural language plus diagrams** of the services the system provides and its operational constraints.
  - Written for customers.
- **System requirements**
  - A structured document setting out **detailed descriptions** of the system's functions, services and operational constraints.
  - Defines what should be implemented so may be part of a contract between client and contractor.

Example – User and System Requirements

User Requirement

- **1.** The robotic arm should never reach its joint limits while moving.

Example – User and System Requirements

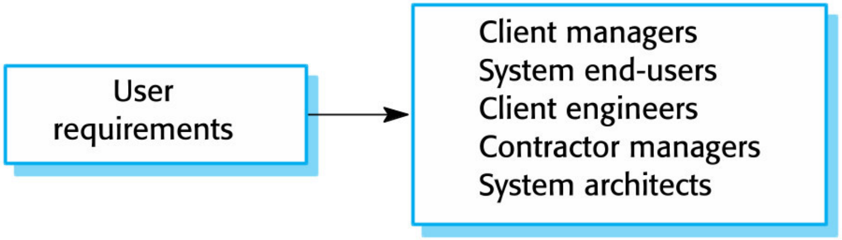
User Requirement

- **1.** The robotic arm should never reach its joint limits while moving.

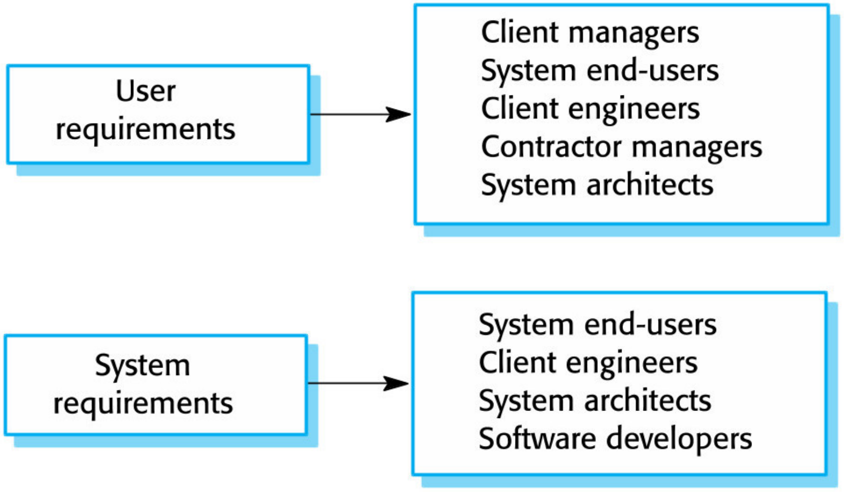
System Requirement

- **1.1** It is never the case that joint1_angle $>$ 45 deg holds.
- **1.2** It is never the case that joint2_angle $>$ 60 deg holds.
- . . .

Readers of different types of requirements specification



Readers of different types of requirements specification



Outline

1 Requirements Engineering – A quick overview
- Functional and Non-functional Requirements
- Requirements Engineering Processes
- Requirements Specification
- Requirements Validation

2 Formal consistency checking

3 Requirements Management with ReqV

4 Hands-on session on ReqV

5 Conclusions

Functional and Non-functional Requirements

Functional requirements

- **Statements of services the system should provide**
  - How the system should react to particular inputs;
  - How the system should behave in particular situations
- May state what the system should not do.

Functional and Non-functional Requirements

Functional requirements

- **Statements of services the system should provide**
  - How the system should react to particular inputs;
  - How the system should behave in particular situations
- May state what the system should not do.

Non-functional requirements

- Constraints on the services or functions offered by the system
  - e.g., timing constraints, constraints on the development process, standards, etc.
- Often apply to the **system as a whole** rather than individual features or services.

Functional requirements

- Describe functionality or system services.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Functional requirements

- Describe functionality or system services.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Problems arise when functional requirements **are not precisely stated**.

- Ambiguous requirements may be **interpreted in different ways** by developers and users.

Requirements Completeness and Consistency

In principle, requirements should be both **complete** and **consistent**.

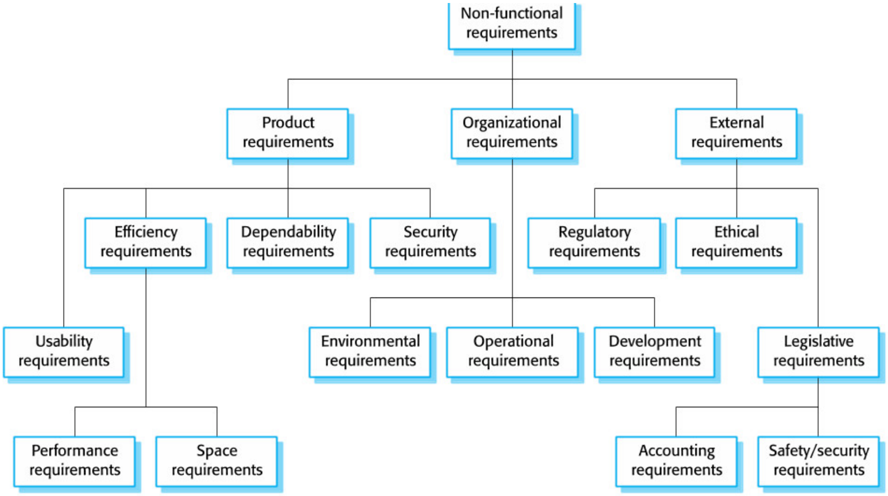
- **Complete**: They should include descriptions of all facilities required.
- **Consistent**: There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, because of system and environmental complexity, it is **impossible to produce a complete and consistent** requirements document.

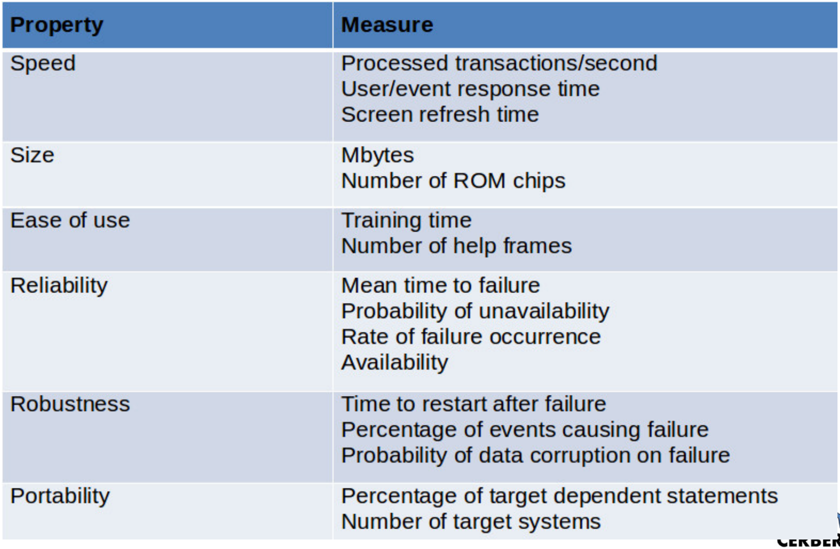
Non-functional Requirements

- These define system **properties** and **constraints**
  - Properties: reliability, response time, storage requirements...
  - Constraints: I/O device capability, system representations...
- Non-functional requirements may be more critical than functional requirements.
  - **If these are not met, the system may be useless.**

Types of Non-functional Requirement



Metrics for specifying non-functional requirements



Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Outline

1. Requirements Engineering – A quick overview
   - Functional and Non-functional Requirements
   - Requirements Engineering Processes
   - Requirements Specification
   - Requirements Validation
2. Formal consistency checking
3. Requirements Management with ReqV
4. Hands-on session on ReqV
5. Conclusions

Requirements Engineering (RE) processes

- The processes used for RE vary widely depending on
  - the application domain;
  - the people involved; and
  - the organization developing the requirements.

Requirements Engineering (RE) processes

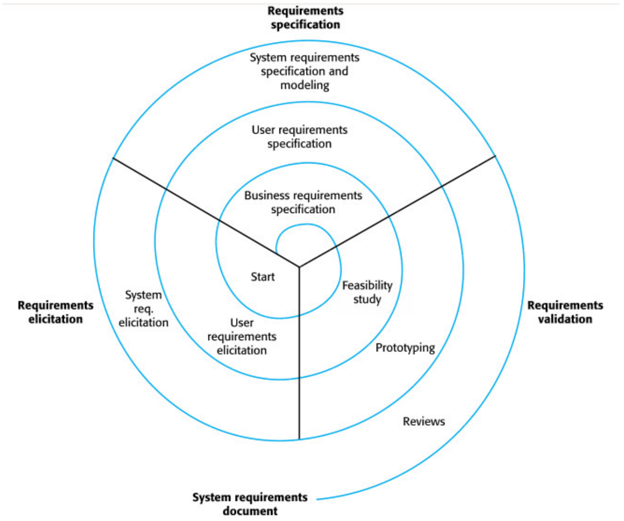
- The processes used for RE vary widely depending on
    - the application domain;
    - the people involved; and
    - the organization developing the requirements.
- However, there are a number of generic activities common to all processes
    - Requirements elicitation;
    - Requirements analysis;
    - Requirements validation;
    - Requirements management.

Requirements Engineering (RE) processes

- The processes used for RE vary widely depending on
  - the application domain;
  - the people involved; and
  - the organization developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.

RE is an iterative activity in which these processes are interleaved.

A spiral view


Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Prototyping
Reviews
Requirements elicitation
System req. elicitation
User requirements elicitation
Requirements validation
System requirements document

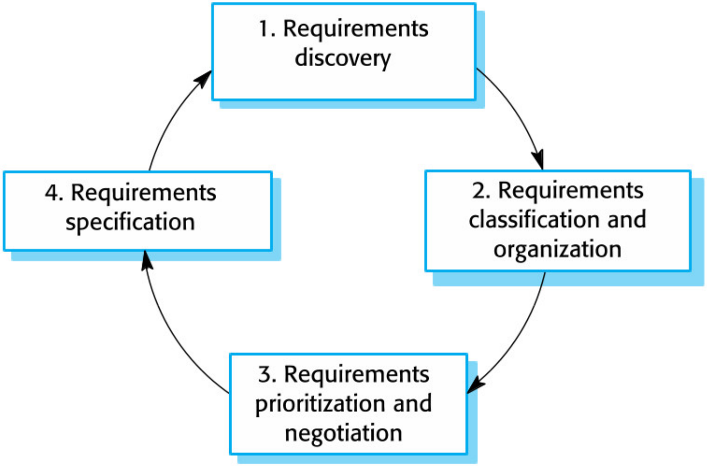
Requirements Elicitation

- Engineers work with a range of system stakeholders to find out about
  - the application domain;
  - the services that the system should provide;
  - the required system performance;
  - hardware constraints;
  - other system;
  - ...

Requirements Elicitation

- Engineers work with a range of system stakeholders to find out about
  - the application domain;
  - the services that the system should provide;
  - the required system performance;
  - hardware constraints;
  - other system;
  - ...
- Stages include:
  1. Requirements discovery,
  2. Requirements classification and organization,
  3. Requirements prioritization and negotiation,
  4. Requirements specification.

The Requirements Elicitation and Analysis Process


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

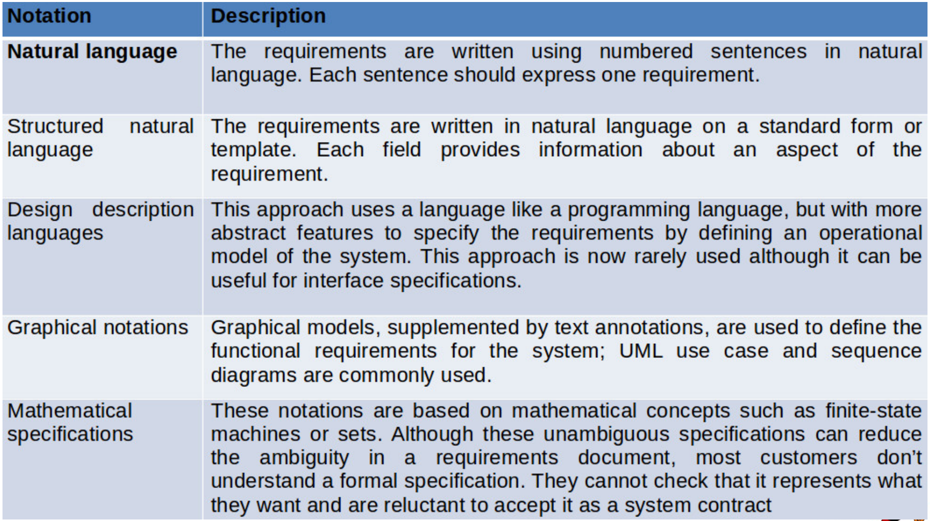
Outline

1. Requirements Engineering – A quick overview
   - Functional and Non-functional Requirements
   - Requirements Engineering Processes
   - Requirements Specification
   - Requirements Validation
2. Formal consistency checking
3. Requirements Management with ReqV
4. Hands-on session on ReqV
5. Conclusions

Requirements Specification

- The process of writing down the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

Ways of writing a system requirements specification



Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal.
  - This means that the requirements can be understood by users and customers.

Guidelines for Writing Requirements

- Invent a standard format and use it for all requirements.
- Use language in a consistent way.
  - Use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.
- Include an explanation (rationale) of why a requirement is necessary.

Problems with Natural Language

- **Lack of clarity**
  - Precision is difficult without making the document difficult to read.

Problems with Natural Language

- **Lack of clarity**
  - Precision is difficult without making the document difficult to read.
- **Requirements confusion**
  - Functional and non-functional requirements tend to be mixed-up.

Problems with Natural Language

- **Lack of clarity**
  - Precision is difficult without making the document difficult to read.
- **Requirements confusion**
  - Functional and non-functional requirements tend to be mixed-up.
- **Requirements amalgamation**
  - Several different requirements may be expressed together.

CERBERO

Structured (natural language) specifications (1/2)

- An approach to writing requirements where the **freedom of the requirements writer is limited** and requirements are written in a standard way.
- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Structured (natural language) specifications (2/2)

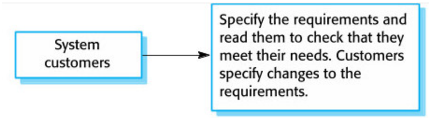
Form-based specifications

1. Definition of the function or entity.
2. Description of inputs and where they come from.
3. Description of outputs and where they go to.
4. Information about the information needed for the computation and other entities used.
5. Description of the action to be taken.
6. Pre and post conditions (if appropriate).
7. The side effects (if any) of the function.

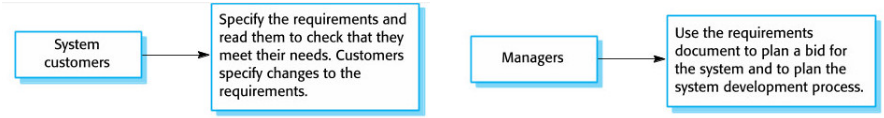
The Requirements Document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

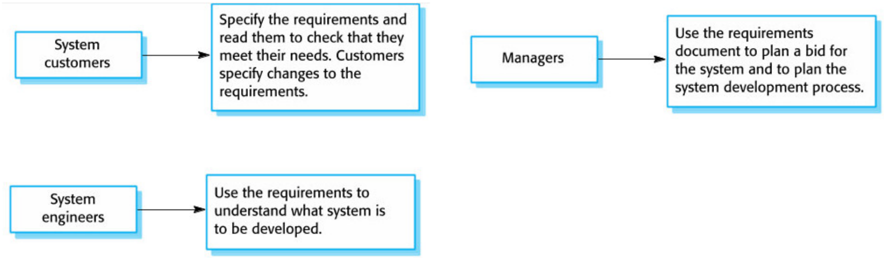
Users of a Requirements Document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.

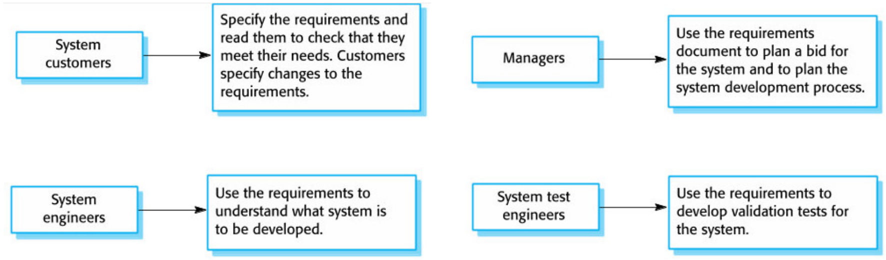
Users of a Requirements Document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.

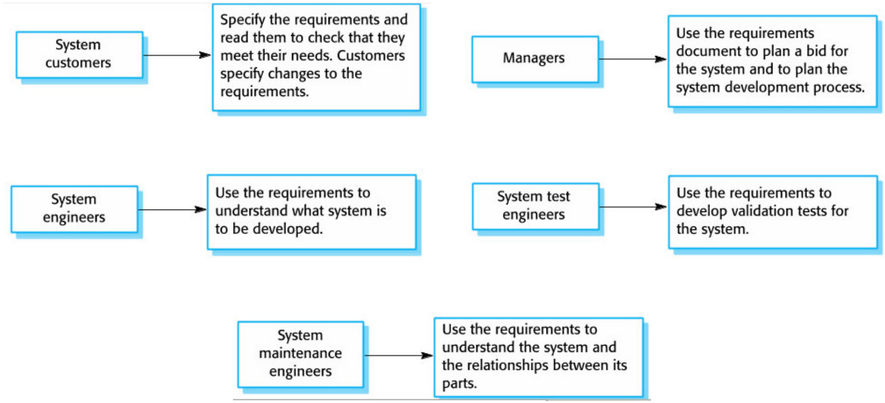
Users of a Requirements Document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.

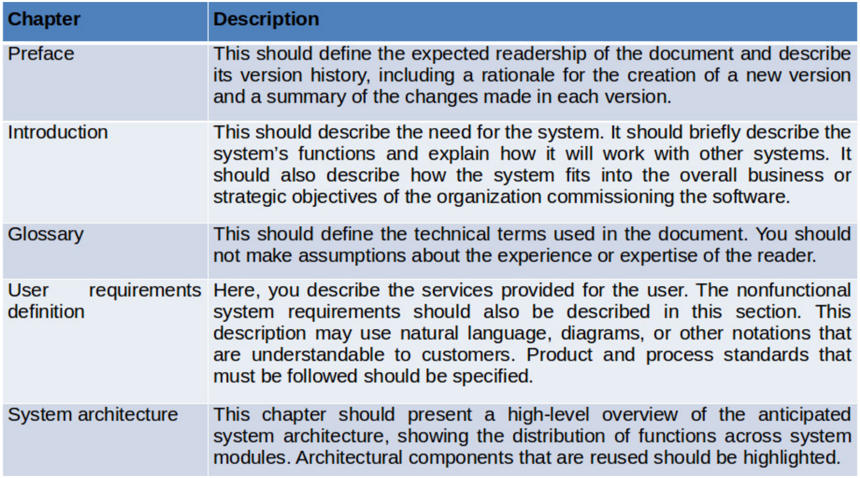
Users of a Requirements Document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.

Users of a Requirements Document

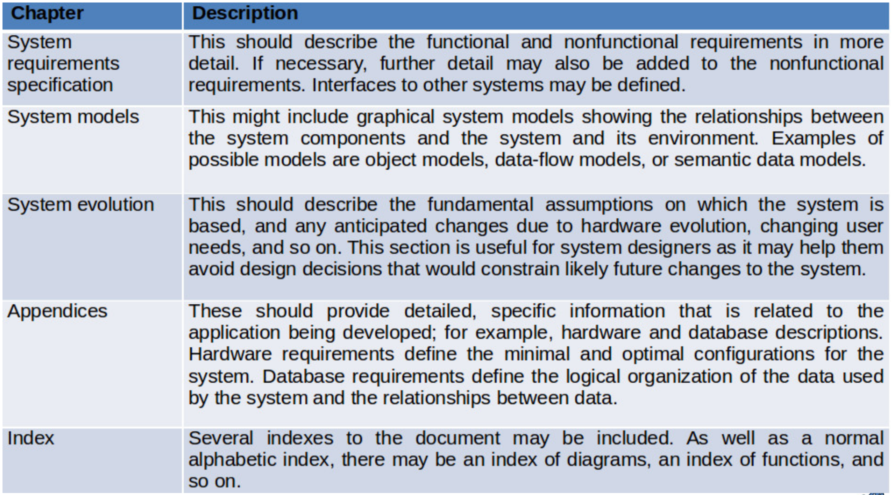

System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

The Structure of a Requirements Document (1/2)



Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The Structure of a Requirements Document (2/2)



Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Outline

1. Requirements Engineering – A quick overview
   - Functional and Non-functional Requirements
   - Requirements Engineering Processes
   - Requirements Specification
   - Requirements Validation
2. Formal consistency checking
3. Requirements Management with ReqV
4. Hands-on session on ReqV
5. Conclusions

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements Checking

- **Validity.** Does the system provide the functions which best support the customer's needs?

Requirements Checking

- **Validity.** Does the system provide the functions which best support the customer's needs?
- **Consistency.** Are there any requirements conflicts?

Requirements Checking

- **Validity.** Does the system provide the functions which best support the customer's needs?
- **Consistency.** Are there any requirements conflicts?
- **Completeness.** Are all functions required by the customer included?

Requirements Checking

- **Validity.** Does the system provide the functions which best support the customer’s needs?
- **Consistency.** Are there any requirements conflicts?
- **Completeness.** Are all functions required by the customer included?
- **Realism.** Can the requirements be implemented given available budget and technology

Requirements Checking

- **Validity.** Does the system provide the functions which best support the customer's needs?
- **Consistency.** Are there any requirements conflicts?
- **Completeness.** Are all functions required by the customer included?
- **Realism.** Can the requirements be implemented given available budget and technology
- **Verifiability.** Can the requirements be checked?

Requirements Checking

- **Validity.** Does the system provide the functions which best support the customer's needs?
- **Consistency.** Are there any requirements conflicts?
- **Completeness.** Are all functions required by the customer included?
- **Realism.** Can the requirements be implemented given available budget and technology
- **Verifiability.** Can the requirements be checked?

Consistency checking

The issue: How to do consistency checking

Consistency checking

The issue: How to do consistency checking

- of requirements expressed in natural language;

Consistency checking

The issue: How to do consistency checking

- of requirements expressed in natural language; and
- in a completed automated way;

Consistency checking

The issue: How to do consistency checking

- of requirements expressed in natural language; and
- in a completed automated way; and
- exahustively.

Consistency checking

The issue: How to do consistency checking

- of requirements expressed in natural language; and
- in a completed automated way; and
- exahustively.

Let us see in the second part of the tutorial!

Outline

1 Requirements Engineering – A quick overview
- Functional and Non-functional Requirements
- Requirements Engineering Processes
- Requirements Specification
- Requirements Validation

2 Formal consistency checking

3 Requirements Management with ReqV

4 Hands-on session on ReqV

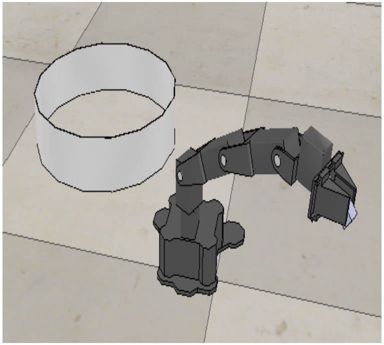
5 Conclusions

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)



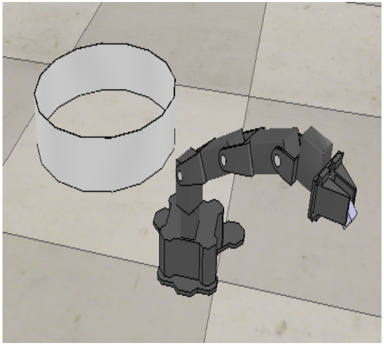
Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Tasks

- Find the object



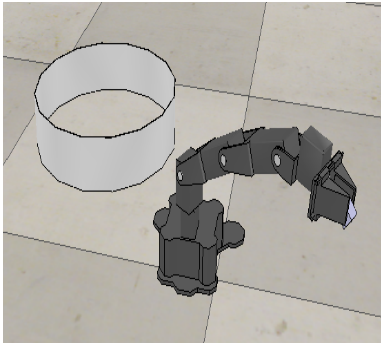
Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Tasks

- Find the object
- Move to the target position


Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

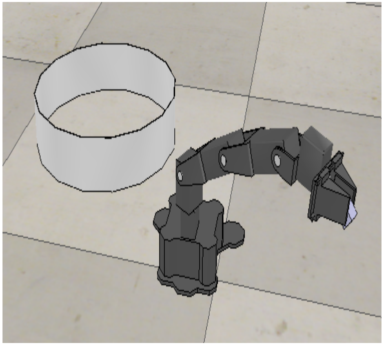
CERBERO

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Tasks

- Find the object
- Move to the target position
- Grab



Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

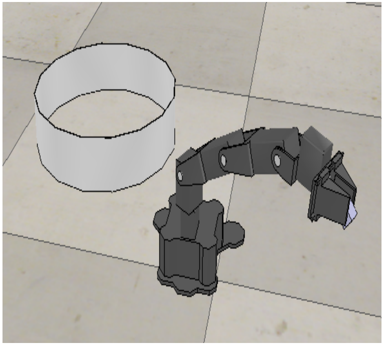
CERBERO

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Tasks

- Find the object
- Move to the target position
- Grab
- Move to the bucket



Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

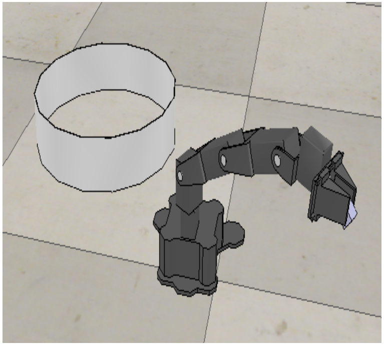
CERBERO

Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Tasks

- Find the object
- Move to the target position
- Grab
- Move to the bucket
- Release the object


Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

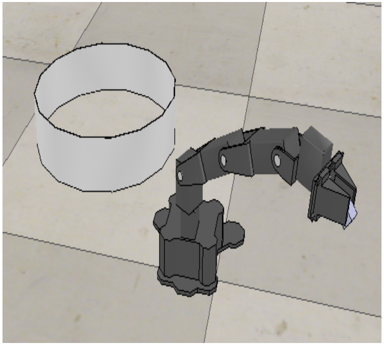
Context - Robotic Manipulator

H2020 EU project *CERBERO* (Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments)

Tasks

- Find the object
- Move to the target position
- Grab
- Move to the bucket
- Release the object

The robot should operate without damaging itself or the surrounding environment.



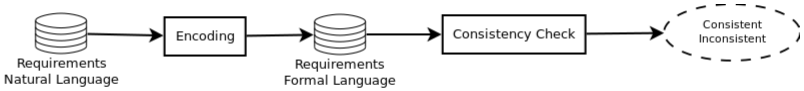
Trossen Robotics WidowX arm with 4 degrees-of-freedom and a gripper

CERBERO

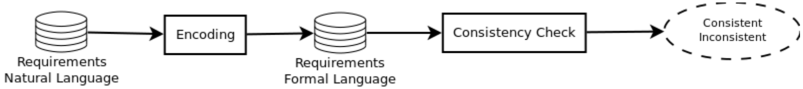
Robotic Manipulator Requirements

- **Safety**
  The joint1 speed should never exceed 90 degrees/s.
- **State Evolution**
  The successors of grabbing state are move-to-target or alarm states.
- **Conditional Events**
  If the proximity sensor detects an obstacle, the robot arm should stop.
- **Mutual Exclusion**
  The controller can be in only one state at a time.

Requirements Consistency Check

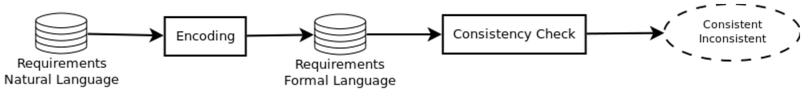


Requirements Consistency Check



- Encode each requirement into Linear Temporal Logic

Requirements Consistency Check



- Encode each requirement into Linear Temporal Logic
- Reduce the consistency check to the LTL satisfiability check.

Robotic Manipulator Requirements - LTL

- **Safety**
  The joint1 speed should never exceed 90 degrees/s.
- **State Evolution**
  The successors of grabbing state are move-to-target or alarm
- **Conditional Events**
  If the proximity sensor detects an obstacle, the robot arm should stop.
- **Mutual Exclusion**
  The controller can be in only one state at a time.

Robotic Manipulator Requirements - LTL

- **Safety**
  The joint1 speed should never exceed 90 degrees/s.
  $\Box(\neg(joint1gt90))$
- **State Evolution**
  The successors of grabbing state are move-to-target or alarm
  $\Box\neg s\_grabbing \lor \Diamond(s\_grabbing \land \Diamond(s\_movetotarget \lor s\_alarm))$
- **Conditional Events**
  If the proximity sensor detects an obstacle, the robot arm should stop.
  $\Box(object\_detected \rightarrow \Diamond arm\_idle)$
- **Mutual Exclusion**
  The controller can be in only one state at a time.
  $\Box(s\_init \rightarrow \neg(s\_grabbing \lor s\_state\_alarm \lor ...)) \land \Box(state\_grabbing \rightarrow ...) \land ...$

Issues

High Level Skills

LTL is far from the natural language.

Lack of expressiveness

R_1 “The angle of joint1 shall never be greater than 170 degrees”.

R_2 “After the MOTOR_SPEED signal arrived, the angle of joint1 shall never be lower than 90 degrees”.

Property Specification Patterns

Issue 1 calls for a language that is “more” Natural and “less” Formal.

Property Specification Patterns

Issue 1 calls for a language that is “more” Natural and “less” Formal.

Property Specification Patterns

PSPs are a collection of parameterizable, high-level, formalism-independent specification abstractions, originally developed to capture recurring solutions to the needs of requirement engineering.

Property Specification Patterns

Issue 1 calls for a language that is “more” Natural and “less” Formal.

Property Specification Patterns

PSPs are a collection of parameterizable, high-level, formalism-independent specification abstractions, originally developed to capture recurring solutions to the needs of requirement engineering.

Each pattern is

- parameterizable
- written in natural language
- directly encoded in a formal language(LTL/CTL/...)

PSP

Body

The *body* of a pattern, describes the behavior that we want to specify.

Scope

The *scope* is the extent of the program execution over which the pattern must hold.

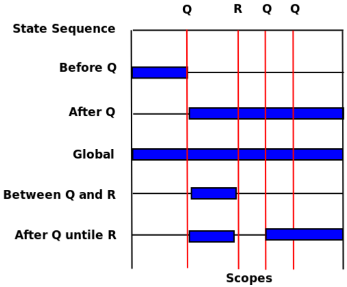
PSP

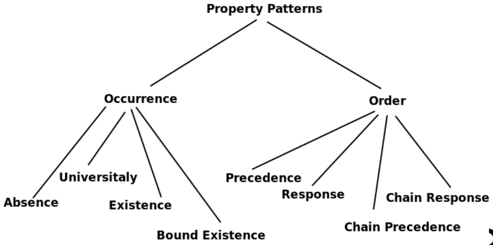
Body

The *body* of a pattern, describes the behavior that we want to specify.

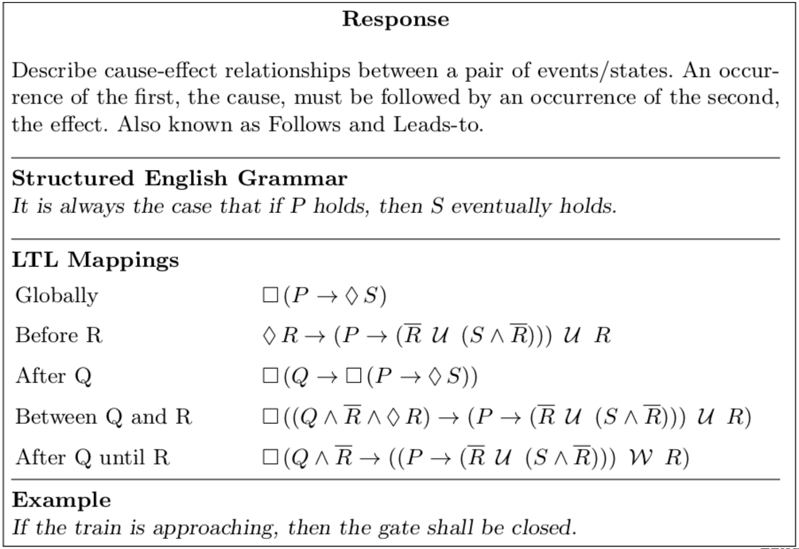
Scope

The *scope* is the extent of the program execution over which the pattern must hold.





PSP example: Response Pattern



Response

Describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect. Also known as Follows and Leads-to.

Structured English Grammar
It is always the case that if P holds, then S eventually holds.

LTL Mappings

Globally	$\Box (P \rightarrow \Diamond S)$
Before R	$\Diamond R \rightarrow (P \rightarrow (\overline{R}\ \mathcal{U}\ (S \wedge \overline{R})))\ \mathcal{U}\ R$
After Q	$\Box (Q \rightarrow \Box (P \rightarrow \Diamond S))$
Between Q and R	$\Box ((Q \wedge \overline{R} \wedge \Diamond R) \rightarrow (P \rightarrow (\overline{R}\ \mathcal{U}\ (S \wedge \overline{R})))\ \mathcal{U}\ R)$
After Q until R	$\Box (Q \wedge \overline{R} \rightarrow ((P \rightarrow (\overline{R}\ \mathcal{U}\ (S \wedge \overline{R})))\ \mathcal{W}\ R)$

Example
If the train is approaching, then the gate shall be closed.

Constraint System

Issue 2 calls for atomic constraint.

Constraint System

Issue 2 calls for atomic constraint.

Constraint System $\mathcal{D}$

$\mathcal{D} = (D, R_1, \ldots, R_n, \mathcal{I})$,

- D is a non-empty set called *domain*,
- R_i is a predicate symbol of arity a_i,
- $\mathcal{I}(R_i) \subseteq D^{a_i}$ it is the interpretation of R_i over the domain D^{a_i}

Given a set of variables X and a set of constants C such that
$C \cap X = \emptyset$,
A *term* is a member of the set $T = C \cup X$;

$R_i(t_1, \ldots, t_{a_i})$ is an (atomic) $\mathcal{D}$-*constraint* over a set of terms T
where $1 \leq i \leq n$ and $t_j \in T$ for all $1 \leq j \leq a_i$

Constraint Property Specification Pattern

PSP($\mathcal{D}$)

A Constraint Property Specification Pattern is a PSP where specification contains both boolean and atoms from a constraint system $\mathcal{D}$.

Constraint Property Specification Pattern

PSP($\mathcal{D}$)

A Constraint Property Specification Pattern is a PSP where specification contains both boolean and atoms from a constraint system $\mathcal{D}$.

restriction

D_C = ($\mathbb{R}$,$<$,$=$)
with atomic constraints of the form $x < c$ and $x = c$, where c is a constant in $\mathbb{R}$

Constraint Property Specification Pattern

PSP($\mathcal{D}$)

A Constraint Property Specification Pattern is a PSP where specification contains both boolean and atoms from a constraint system $\mathcal{D}$.

restriction

D_C = ($\mathbb{R}$,$<$,$=$)
with atomic constraints of the form $x < c$ and $x = c$, where c is a constant in $\mathbb{R}$

PSP($\mathcal{D}_c$)

is a PSP($\mathcal{D}$) system where atomic constraint are in the form of $x < c$ or $x = c$ ($c \in \mathbb{R}$).

Robotic Manipulator Requirements - PSP($\mathcal{D}_C$)

- **Safety**
  The joint1 speed should never exceed 90 degrees/s.
- **State Evolution**
  The successors of grabbing state are move-to-target or alarm
- **Conditional Events**
  If the proximity sensor detects an obstacle, the robot arm should stop.
- **Mutual Exclusion**
  The controller can be in only one state at a time.

Robotic Manipulator Requirements - PSP($\mathcal{D}_C$)

- **Safety**
  The joint1 speed should never exceed 90 degrees/s.
  `Globally, it is never the case that joint1_speed > 90 holds.`
- **State Evolution**
  The successors of grabbing state are move-to-target or alarm
  `After state_grabbing, state_moving_to_bucket or state_alarm eventually holds.`
- **Conditional Events**
  If the proximity sensor detects an obstacle, the robot arm should stop.
  `Globally, it is always the case that if proximity_sensor < 20 holds, then arm_idle eventually holds.`
- **Mutual Exclusion**
  The controller can be in only one state at a time.
  `Globally, it is always the case that if state_init holds, then not (state_scanning or state_moving_to_target or ...) holds as well.`

Linear Temporal Logic (LTL) - 1/2

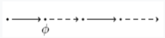
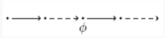
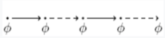
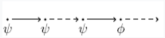
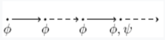
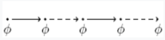
Modal temporal logic with *modalities* referring to time

- One can encode formulae about the future of **paths**, e.g., a condition will eventually be true, a condition will be true until another fact becomes true, etc.

Syntax:

- ❑ LTL is built up from a finite set of propositional variables *AP*, the logical operators ¬ and ∨, and the temporal modal operators **X** (next) and **U** (until).
- ❑ the set of LTL formulas over *AP* is inductively defined as follows:
  - if p ∈ *AP* then p is an LTL formula;
  - if ψ and ϕ are LTL formulas then ¬ψ, ϕ ∨ ψ, **X** ψ, and ϕ **U** ψ are LTL formulas.
  - ❑ Additional temporal operators: **G** (**g**lobally), **F** (eventually), **R** (**r**elease)

Linear Temporal Logic (LTL) - 2/2

Textual	Symbolic	Explanation	Diagram
Unary operators:			
X ϕ	$\bigcirc\phi$	ne**X**t: ϕ has to hold at the next state.	
F ϕ	$\Diamond\phi$	**F**inally: ϕ eventually has to hold (somewhere on the subsequent path).	
G ϕ	$\Box\phi$	**G**lobally: ϕ has to hold on the entire subsequent path.	
Binary operators:			
ψ **U** ϕ	$\psi\,\mathcal{U}\,\phi$	**U**ntil: ψ has to hold *at least* until ϕ becomes true, which must hold at the current or a future position.	
ψ **R** ϕ	$\psi\,\mathcal{R}\,\phi$	**R**elease: ϕ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, ϕ must remain true forever.	 

Linear Temporal Logic modulo Constraint

PSP system encode requirements directly in LTL. What about PSP($\mathcal{D}_C$)?

Linear Temporal Logic modulo Constraint

PSP system encode requirements directly in LTL. What about PSP($\mathcal{D}_C$)?

The Linear Temporal Logic modulo Constraint (LTL($\mathcal{D}$)) is an extension of LTL with atoms in a constraint system $\mathcal{D}$.

$$\phi = p \mid R_i(t_1, \ldots, t_{a_i}) \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \mathcal{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2$$

where

- *Prop* is a set of Boolean Propositions
- $\mathcal{D} = (D, R_1, \ldots, R_n, \mathcal{I})$ is a constraint system
- $T = C \cup X$ is a set of terms

abbreviations

we consider $p \vee \neg p$ as $\top$, $p \wedge \neg p$ as $\bot$, “$\wedge$” and “$\rightarrow$”, $\Diamond\,\phi$ (“eventually”) denote $\top\,\mathcal{U}\,\phi$ and $\Box\,\phi$ (“always”) denote $\neg\Diamond\,\neg\phi$.

Consistency Check

PSP($\mathcal{D}_c$) encoding into LTL($\mathcal{D}_c$) follows the PSP to LTL encoding

Consistency Check

PSP($\mathcal{D}_c$) encoding into LTL($\mathcal{D}_c$) follows the PSP to LTL encoding

The consistency check of a set of PSP($\mathcal{D}_C$) requirement is encoded into a satisfiability of a LTL($\mathcal{D}_c$).

Consistency Check

PSP($\mathcal{D}_c$) encoding into LTL($\mathcal{D}_c$) follows the PSP to LTL encoding

The consistency check of a set of PSP($\mathcal{D}_C$) requirement is encoded into a satisfiability of a LTL($\mathcal{D}_c$).

Unluckily, to the best of our knowledge there is still not a tool computing the satisfiability of a LTL($\mathcal{D}_c$) formula.

Consistency Check

PSP($\mathcal{D}_c$) encoding into LTL($\mathcal{D}_c$) follows the PSP to LTL encoding

The consistency check of a set of PSP($\mathcal{D}_C$) requirement is encoded into a satisfiability of a LTL($\mathcal{D}_c$).

Unluckily, to the best of our knowledge there is still not a tool computing the satisfiability of a LTL($\mathcal{D}_c$) formula.

Luckily a LTL($\mathcal{D}_c$) formula can be reduced to a LTL formula.

LTL($\mathcal{D}_c$) reduction to LTL

Given a LTL($\mathcal{D}_c$) formula, ϕ, let

- $A(\phi)$ be the set of atomic constraint in ϕ ($x < c$ and $x = c$)
- $X(\phi)$ be the set of variables occurring in $A(\phi)$
- $C(\phi)$ be the set of constants that occur in $A(\phi)$.
- $S_x(\phi) \subseteq C(\phi)$ be the set of constants occurring in each atomic constraint where the variable x occurs (ordered ascending)

LTL($\mathcal{D}_c$) reduction to LTL

Example

- R_1 Globally, it is always the case that **v** $\leq$ **5.0** holds.
- R_2 After **a**, **v** $\leq$ **8.5** eventually holds.
- R_3 After **a**, it is always the case that if **v** $\geq$ **3.2** holds, then **z** eventually holds.

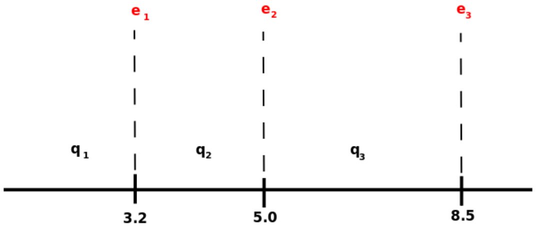
- $A(\phi) = \{v < 5.0, v = 5.0, v < 8.5, v = 8.5, v < 3.2\}$
- $X(\phi) = \{v\}$
- $C(\phi) = \{3.2, 5.0, 8.5\}$.
- $S_v(\phi) = \{3.2, 5.0, 8.5\}$.

LTL($\mathcal{D}_c$) reduction to LTL

Given $S_x(\phi)$ we construct two sets of boolean proposition, Q_x and E_x

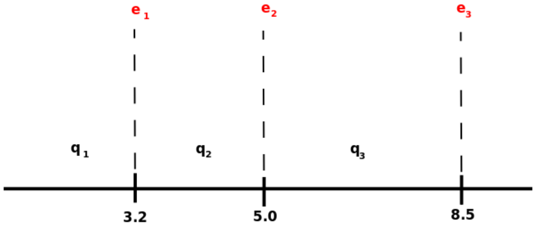
LTL($\mathcal{D}_c$) reduction to LTL

Given $S_x(\phi)$ we construct two sets of boolean proposition, Q_x and E_x



LTL($\mathcal{D}_c$) reduction to LTL

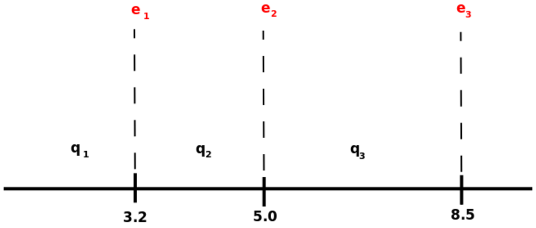
Given $S_x(\phi)$ we construct two sets of boolean proposition, Q_x and E_x



$Q_v = \{q_1, q_2, q_3\}$ and $E_v = \{e_1, e_2, e_3\}$.

LTL($\mathcal{D}_c$) reduction to LTL

Given $S_x(\phi)$ we construct two sets of boolean proposition, Q_x and E_x

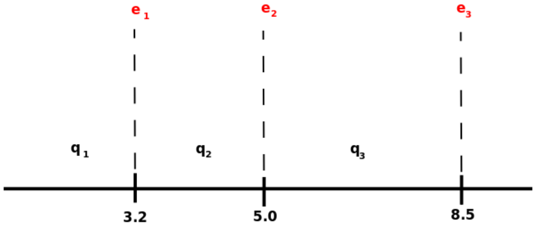


$Q_v = \{q_1, q_2, q_3\}$ and $E_v = \{e_1, e_2, e_3\}$.
We substitute in ϕ each atomic proposition with x, as follows

$$x < t_k \rightsquigarrow \bigvee_{j=1}^{k} q_j \vee \bigvee_{j=1}^{k-1} e_j \qquad \text{and} \qquad x = t_k \rightsquigarrow e_k.$$

LTL($\mathcal{D}_c$) reduction to LTL

Given $S_x(\phi)$ we construct two sets of boolean proposition, Q_x and E_x



$Q_v = \{q_1, q_2, q_3\}$ and $E_v = \{e_1, e_2, e_3\}$.
We substitute in ϕ each atomic proposition with x, as follows

$$x < t_k \rightsquigarrow \bigvee_{j=1}^{k} q_j \vee \bigvee_{j=1}^{k-1} e_j \qquad \text{and} \qquad x = t_k \rightsquigarrow e_k.$$

For example $x < 8.5 \rightsquigarrow \{q_1 \vee q_2 \vee q_3 \vee e_1 \vee e_2\}$

LTL($\mathcal{D}_c$) reduction to LTL

$$\begin{array}{lcl}
\Box(v < 5.0 \vee v = 5.0) & \rightsquigarrow & \Box(q_1 \vee q_2 \vee e_1 \vee e_2) \\
\Box(a \rightarrow \Diamond(v < 8.5) \vee (v = 8.5)) & \rightsquigarrow & \Box(a \rightarrow \Diamond(q_1 \vee q_2 \vee q_3 \vee e_1 \vee e_2 \vee e_3)) \\
\Box(a \rightarrow \Box(v \geq 3.2 \rightarrow \Diamond z)) & \rightsquigarrow & \Box(a \rightarrow \Box((q_2 \vee q_3 \vee e_1 \vee e_2 \vee e_3) \rightarrow \Diamond z))
\end{array}$$

LTL($\mathcal{D}_c$) reduction to LTL

$$\begin{array}{lcl}
\square(v < 5.0 \vee v = 5.0) & \rightsquigarrow & \square(q_1 \vee q_2 \vee e_1 \vee e_2) \\
\square(a \rightarrow \lozenge(v < 8.5) \vee (v = 8.5)) & \rightsquigarrow & \square(a \rightarrow \lozenge(q_1 \vee q_2 \vee q_3 \vee e_1 \vee e_2 \vee e_3)) \\
\square(a \rightarrow \square(v \geq 3.2 \rightarrow \lozenge z)) & \rightsquigarrow & \square(a \rightarrow \square((q_2 \vee q_3 \vee e_1 \vee e_2 \vee e_3) \rightarrow \lozenge z))
\end{array}$$

Consistency may report wrong results

A satisfiable formula may be satisfiable only by models containing conflicting boolean atoms such as q_1 (encoding of $x \leq 3.2$) and q_2 (encoding of $3.2 \leq x \leq 5.0$).

LTL($\mathcal{D}_c$) reduction to LTL

$$\begin{array}{lcl}
\Box(v < 5.0 \vee v = 5.0) & \rightsquigarrow & \Box(q_1 \vee q_2 \vee e_1 \vee e_2) \\
\Box(a \rightarrow \Diamond(v < 8.5) \vee (v = 8.5)) & \rightsquigarrow & \Box(a \rightarrow \Diamond(q_1 \vee q_2 \vee q_3 \vee e_1 \vee e_2 \vee e_3)) \\
\Box(a \rightarrow \Box(v \geq 3.2 \rightarrow \Diamond z)) & \rightsquigarrow & \Box(a \rightarrow \Box((q_2 \vee q_3 \vee e_1 \vee e_2 \vee e_3) \rightarrow \Diamond z))
\end{array}$$

Consistency may report wrong results

A satisfiable formula may be satisfiable only by models containing conflicting boolean atoms such as q_1 (encoding of $x \leq 3.2$) and q_2 (encoding of $3.2 \leq x \leq 5.0$).

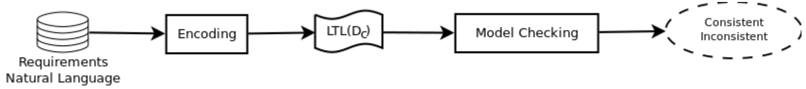
Mutual Exclusion

Given any two boolean variables in $Q_\xi(\phi) \cup E_\xi(\phi)$, they can not be true at the same time.

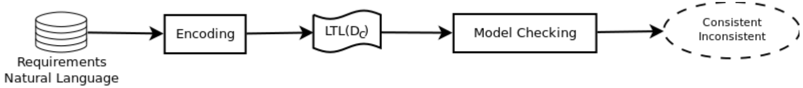
$$\phi_M = \bigwedge_{\xi \in X(\phi)} \left(\bigwedge_{a,b \in M_\xi(\phi), a \neq b} \Box\neg(a \wedge b) \right) \quad (1)$$

LTL($\mathcal{D}_c$) satisfiability via Model Checking

LTL($\mathcal{D}_c$) satisfiability via Model Checking



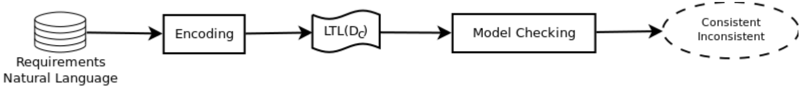
LTL($\mathcal{D}_c$) satisfiability via Model Checking



Given a set of requirements in PSP(D_c)

- we encode them in a formula LTL($\mathcal{D}_c$), ϕ

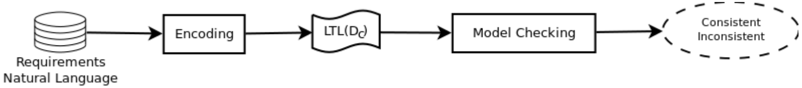
LTL($\mathcal{D}_c$) satisfiability via Model Checking



Given a set of requirements in PSP(D_c)

- we encode them in a formula LTL($\mathcal{D}_c$), ϕ
- we construct the the formula $\phi_M \rightarrow \phi'$,

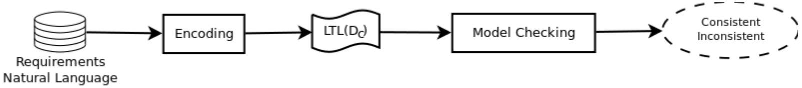
LTL($\mathcal{D}_c$) satisfiability via Model Checking



Given a set of requirements in PSP(D_c)

- we encode them in a formula LTL($\mathcal{D}_c$), ϕ
- we construct the the formula $\phi_M \rightarrow \phi'$,
- we construct the universal model M, i.e. a model that encodes all the possible computation over a set of *Prop*.

LTL($\mathcal{D}_c$) satisfiability via Model Checking



Given a set of requirements in PSP(D_c)

- we encode them in a formula LTL($\mathcal{D}_c$), ϕ
- we construct the the formula $\phi_M \rightarrow \phi'$,
- we construct the universal model M, i.e. a model that encodes all the possible computation over a set of *Prop*.
- $\phi_M \rightarrow \phi'$ is satisfiable precisely when M does not satisfy its negation.

Summing up

- We extended basic PSPs over the constraint system $\mathcal{D}_c$
- We provided an encoding from any PSP($\mathcal{D}_c$) into a LTL formula
- We showed how to check the consistency of a set of requirements written a PSP($\mathcal{D}_c$) by reduction to model checking problem
- Our approach scales on realistically sized sets of requirements
- Our approach is feasible to check specifications and uncover injected faults

Outline

1 Requirements Engineering – A quick overview
- Functional and Non-functional Requirements
- Requirements Engineering Processes
- Requirements Specification
- Requirements Validation

2 Formal consistency checking

3 Requirements Management with ReqV

4 Hands-on session on ReqV

5 Conclusions

Requirements Management with ReqV

Outline

1 Requirements Engineering – A quick overview
- Functional and Non-functional Requirements
- Requirements Engineering Processes
- Requirements Specification
- Requirements Validation

2 Formal consistency checking

3 Requirements Management with ReqV

4 Hands-on session on ReqV

5 Conclusions

Hands-on session on ReqV

`https://reqv.sagelab.it/`

Outline

1 Requirements Engineering – A quick overview
- Functional and Non-functional Requirements
- Requirements Engineering Processes
- Requirements Specification
- Requirements Validation

2 Formal consistency checking

3 Requirements Management with ReqV

4 Hands-on session on ReqV

5 Conclusions

Conclusion (1/2)

- Requirements for a system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.

Conclusion (2/2)

- The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.
- Requirements specification is the process of formally documenting the user and system requirements and creating a requirements document.
- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.

Acknowledgments

- ReqV and related research is a joint work with our colleague Massimo Narizzano (University of Genoa).
- This work is partially funded by the EU Commission H2020 Programme under grant agreement N.732105 (CERBERO project).

Acknowledgments

- ReqV and related research is a joint work with our colleague Massimo Narizzano (University of Genoa).
- This work is partially funded by the EU Commission H2020 Programme under grant agreement N.732105 (CERBERO project).

Please, fill the questionnaire!