CPS Week 2018
Tutorial on Design of adaptive and secure CPS
April 10-13, 2018

Where All Projects Start: Requirements

Michael Masin, IBM Research - Haifa

Luca Pulina, Università degli Studi di Sassari

Horizon 2020
European Union funding
for Research & Innovation

CPS Week 2018
Tutorial on Design of adaptive and secure CPS
April 10-13, 2018

Where All Projects should Start: Requirements

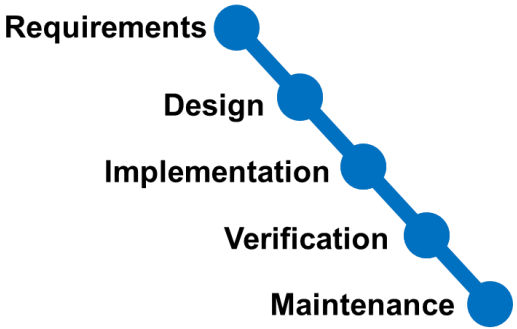
Michael Masin, IBM Research - Haifa
Luca Pulina, Università degli Studi di Sassari

Horizon 2020
European Union funding
for Research & Innovation

Outline

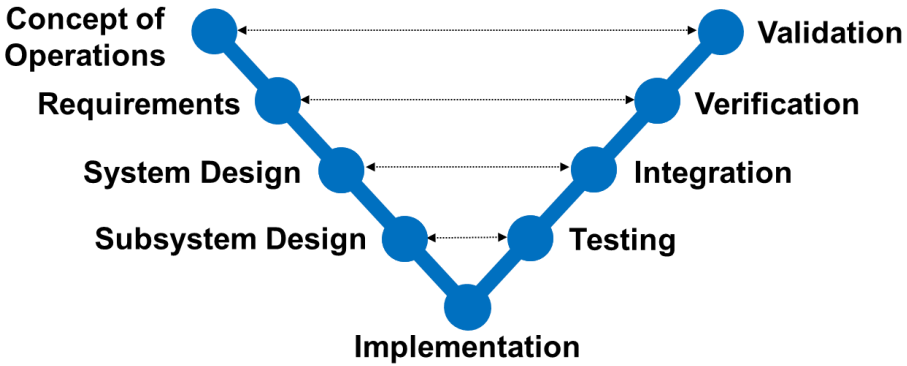
- Model Based Engineering Background
- PORTALS
  - Architecture
  - Interaction
  - Scenarios
  - Demo
- Requirements Formalization and Verification

Waterfall Model (from 1950s)



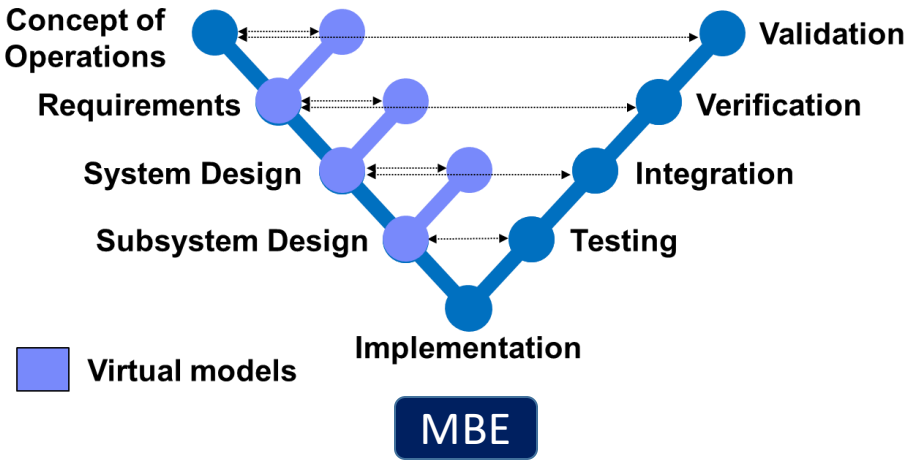
Start of Systems Engineering

V model (1980s)

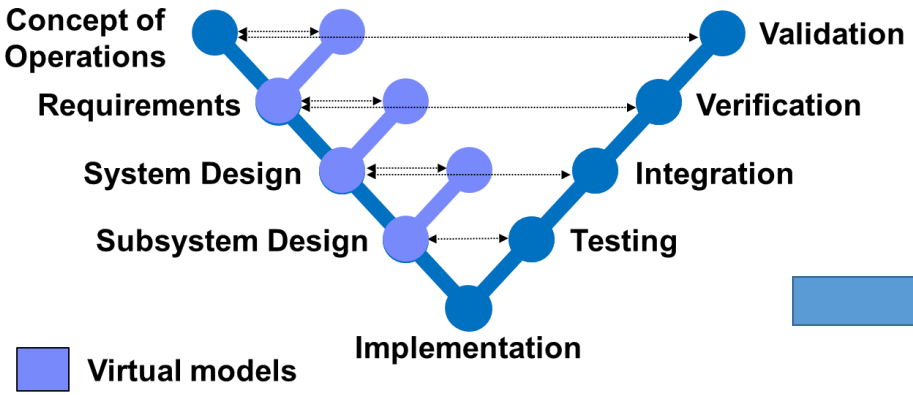


Systems Engineering today

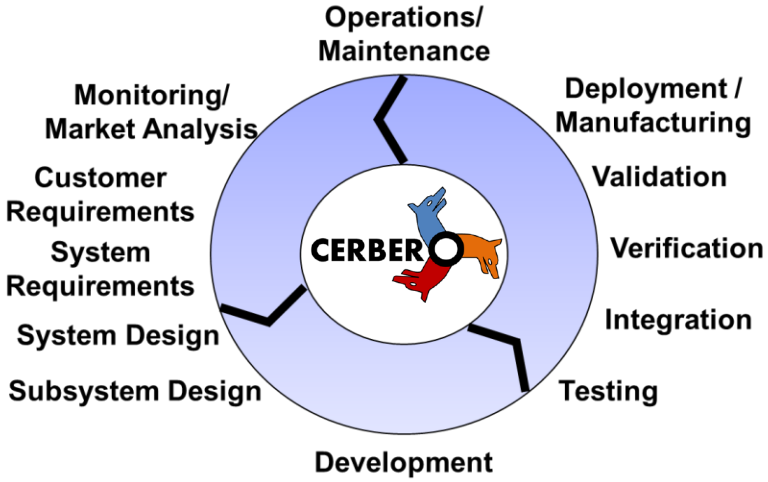
Model Based Engineering (< 10 years)



Continuous Model Based Engineering



MBE



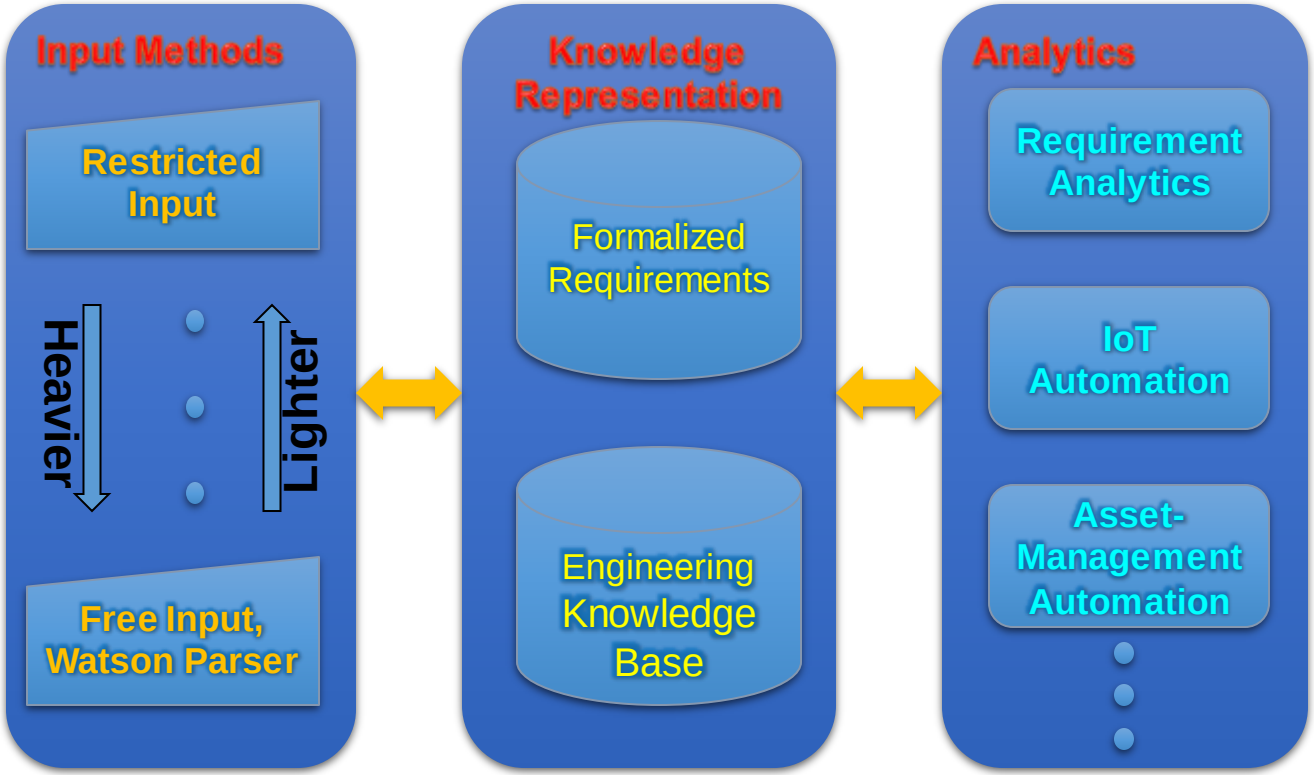
CMBE: Models create models

PORTALS – research project

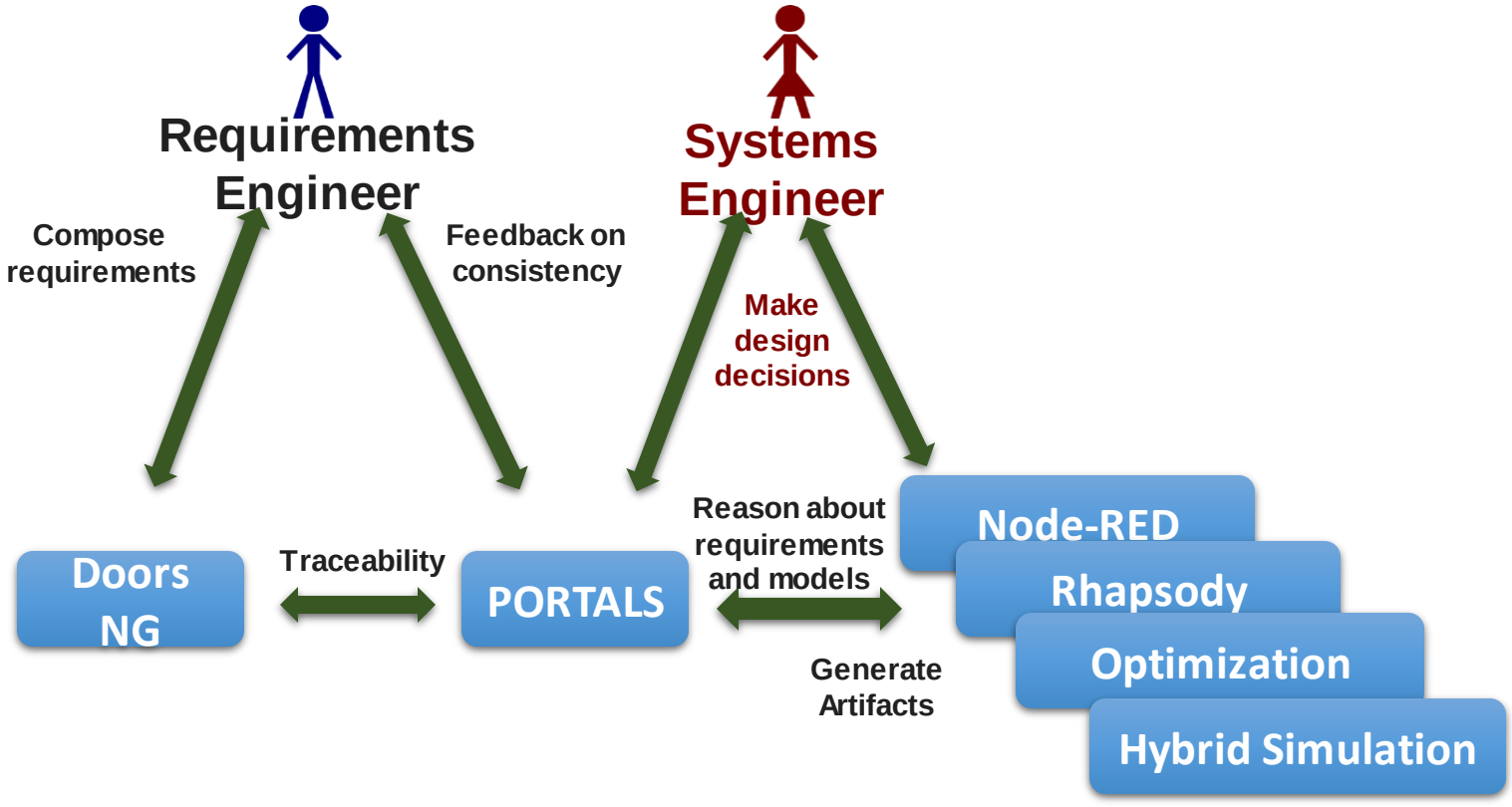
- to create tools to assist requirements engineers in incrementally raising the formalization level of system requirements, and
- to use formalized requirements to
  - provide feedback on the quality of the requirements (e.g., identifying omissions and contradictions), and
  - create downstream artifacts (e.g., models, monitors, tests, code)

- Yishai A. Feldman and Henry Broodney, 2016, “A Cognitive Journey for Requirements Engineering”, 26th Annual INCOSE International Symposium.

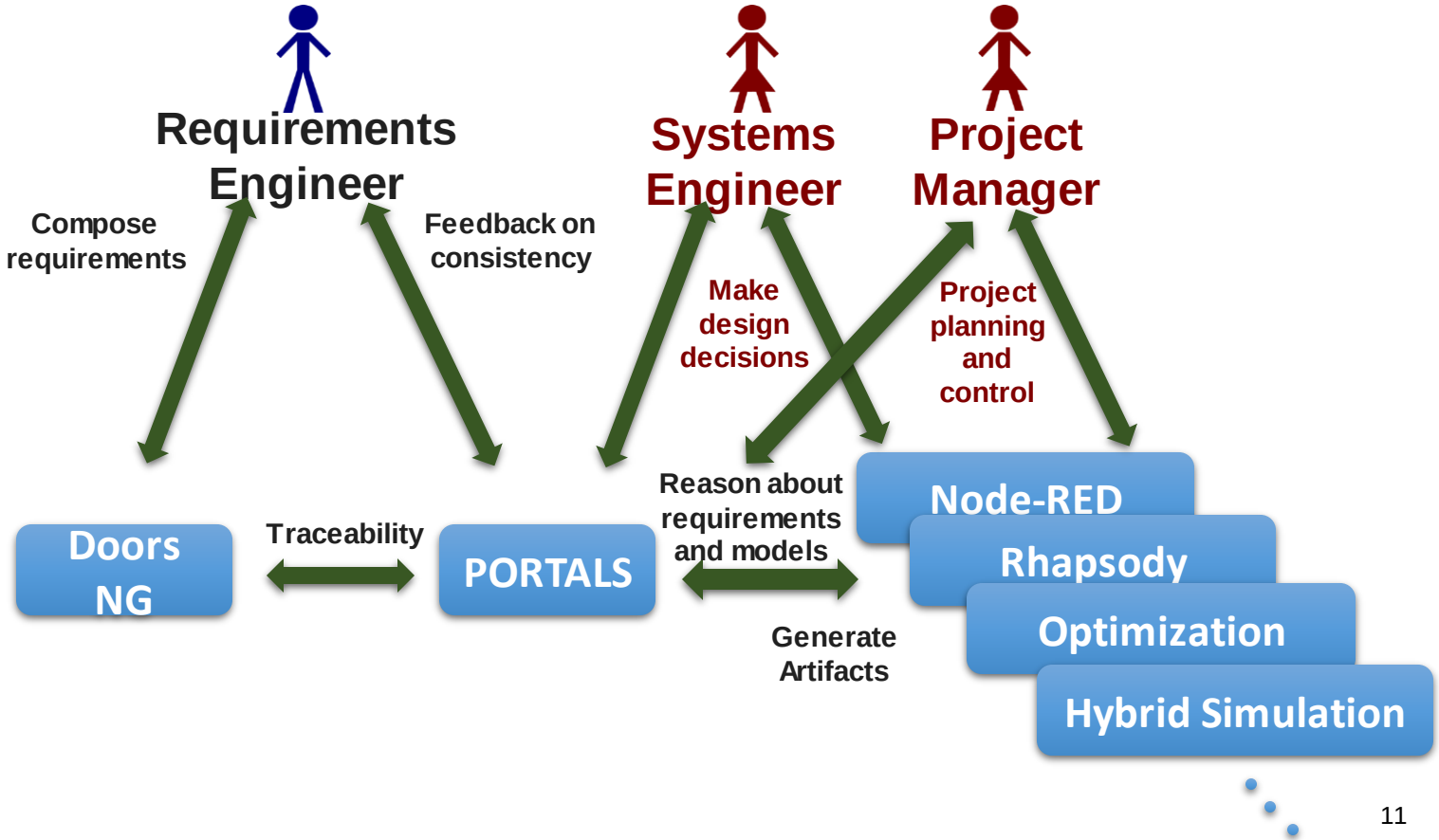
PORTALS Architecture



Personas and Interactions

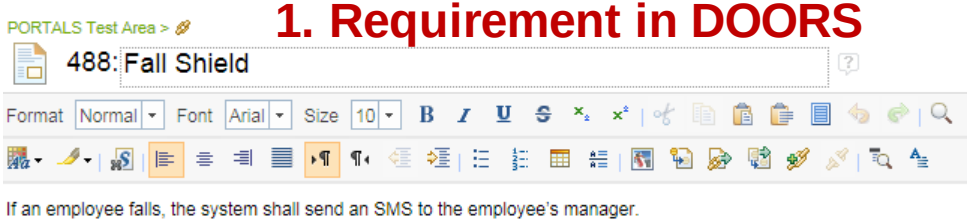


Personas and Interactions



Scenario 1: IBM IoT Safer Workplace

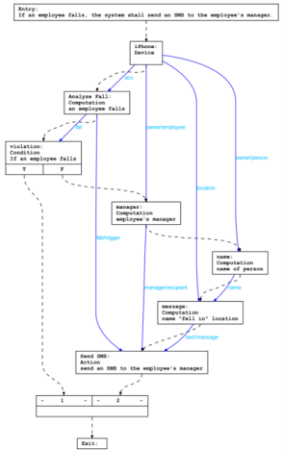
1. Requirement in DOORS



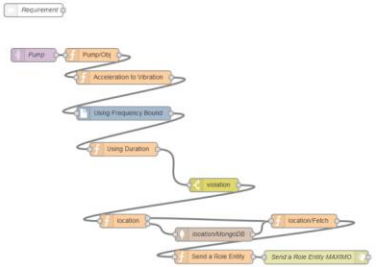
2. Paraphrase by PORTALS

if "an employee" falls then "the system" shall send [an abstract entity] "an SMS" (direction) "manager" of "the employee's"

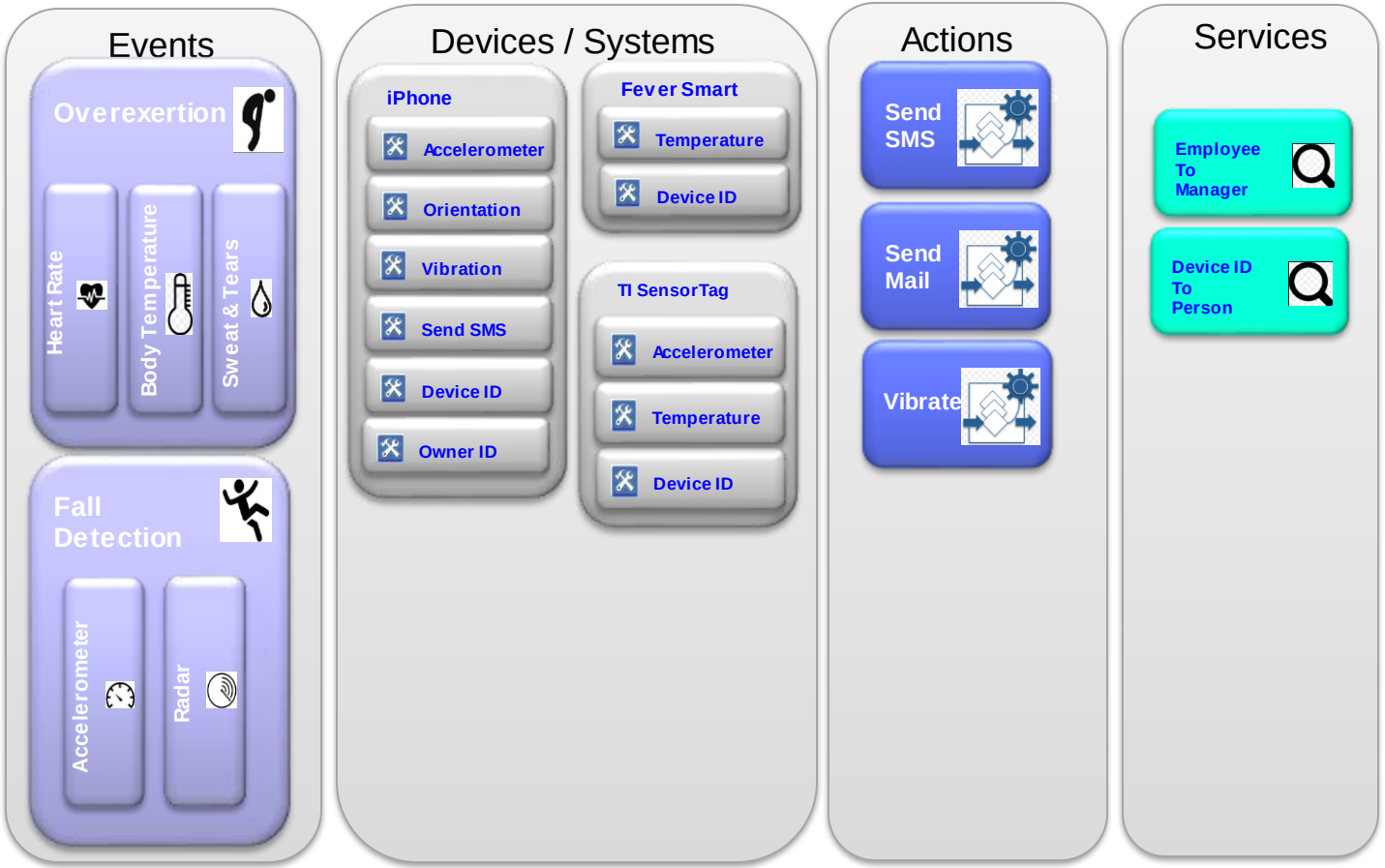
3. Process Model



4. Implementat in Node-RED



Engineering Knowledge Base:



Scenario 2: IoT Pump

1. Requirement in DOORS

PORTALS Test Area >

487: Too much vibration

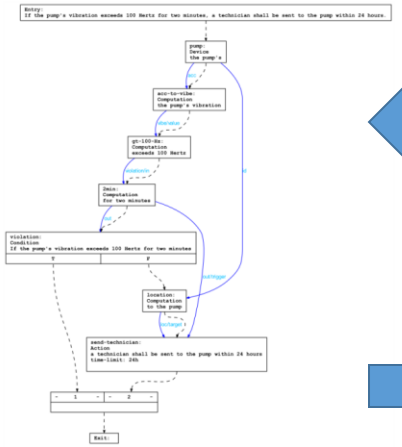
Format Normal Font Arial Size 10

If the pump's vibration exceeds 100 Hertz for two minutes, a technician shall be sent to the pump within 24 hours.

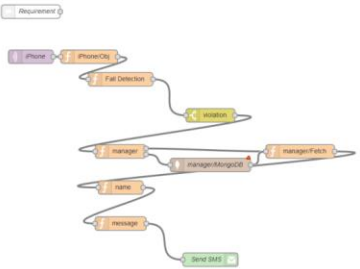
2. Paraphrase by PORTALS

if "vibration" of "the pump's" is greater than 100 Hz (duration) 2 min then "?" shall send [a role entity] "a technician" (duration) 24 hr; (direction) "the pump"

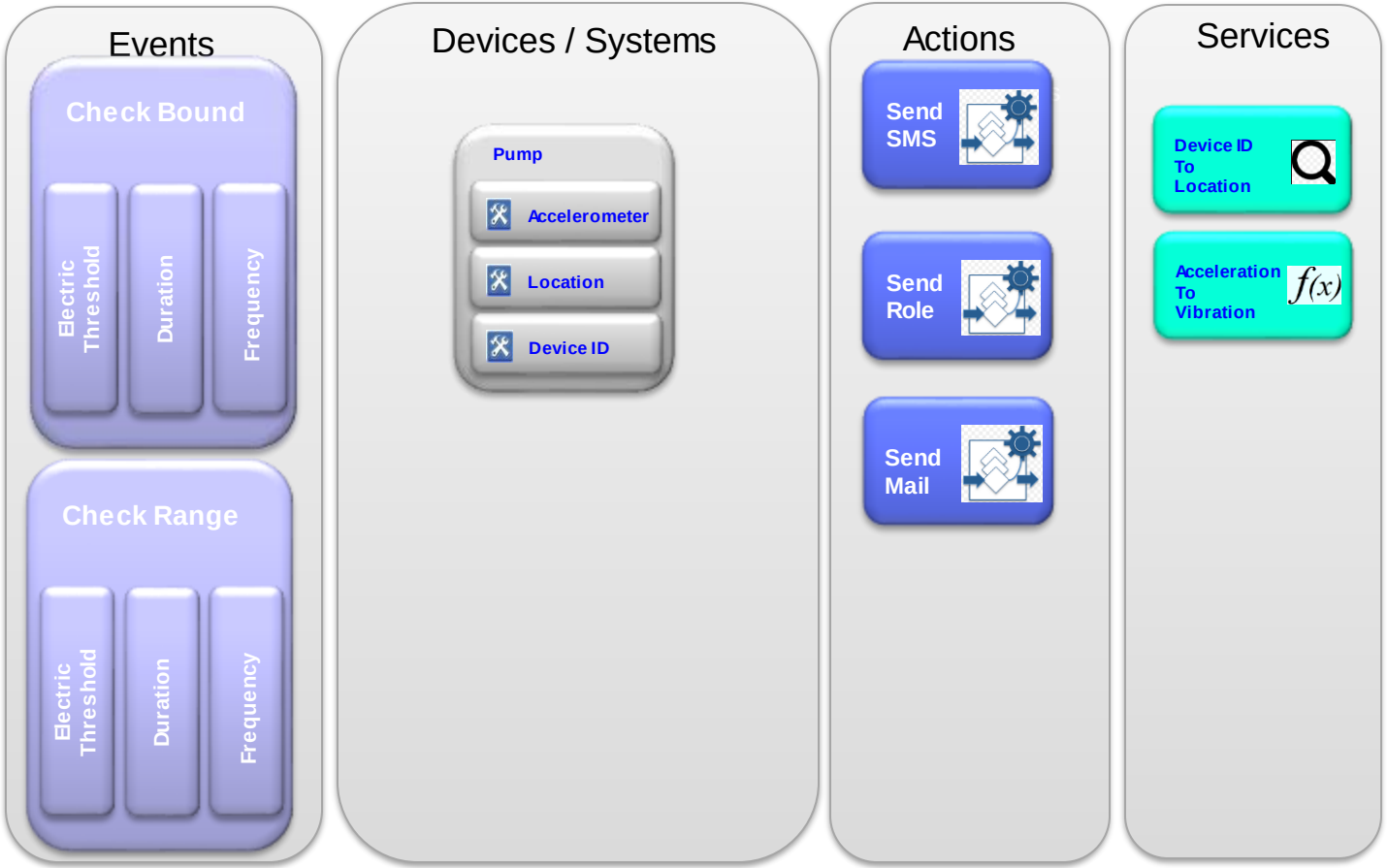
3. Process Model



4. Implement in Node-RED



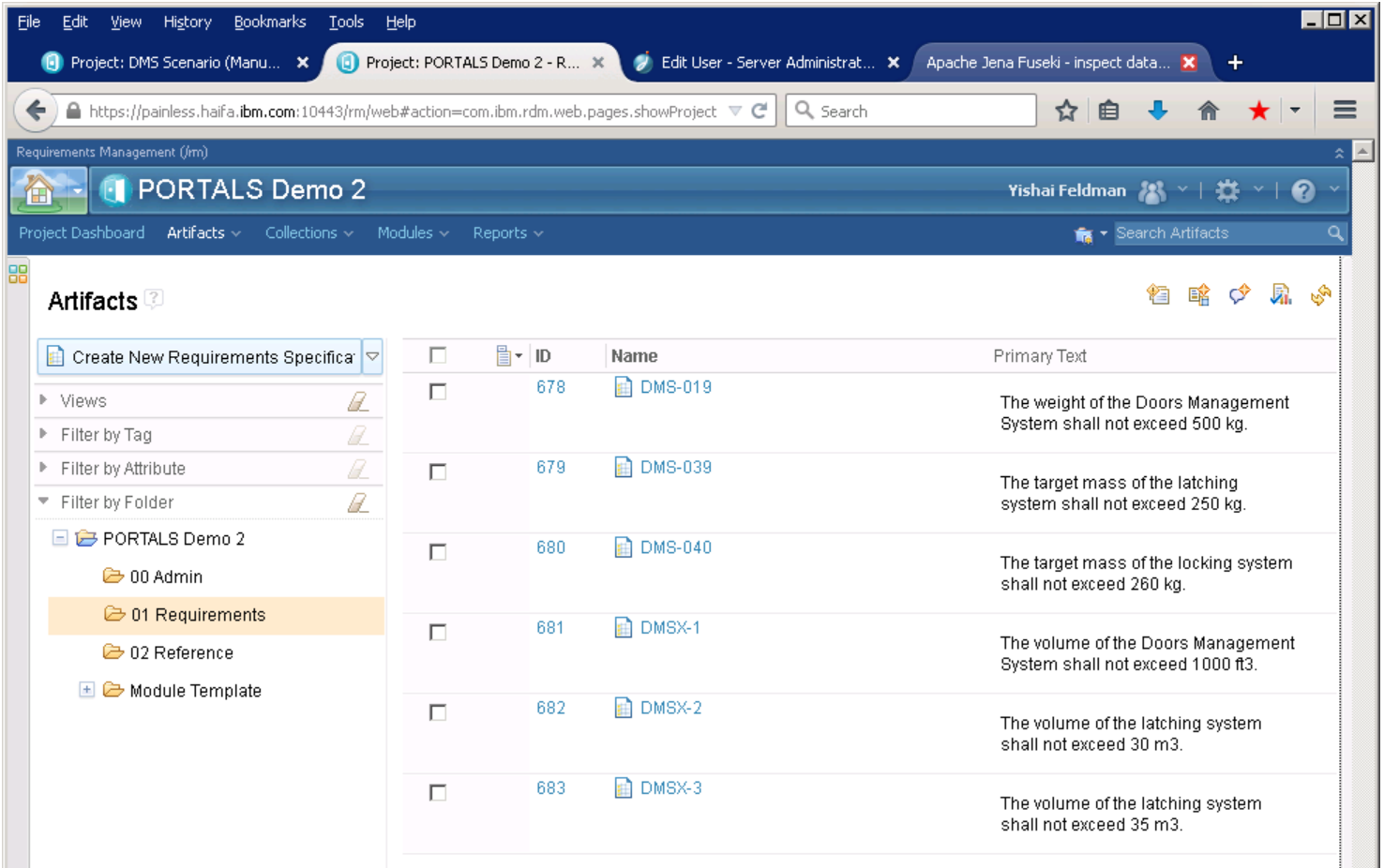
Engineering Knowledge Base:



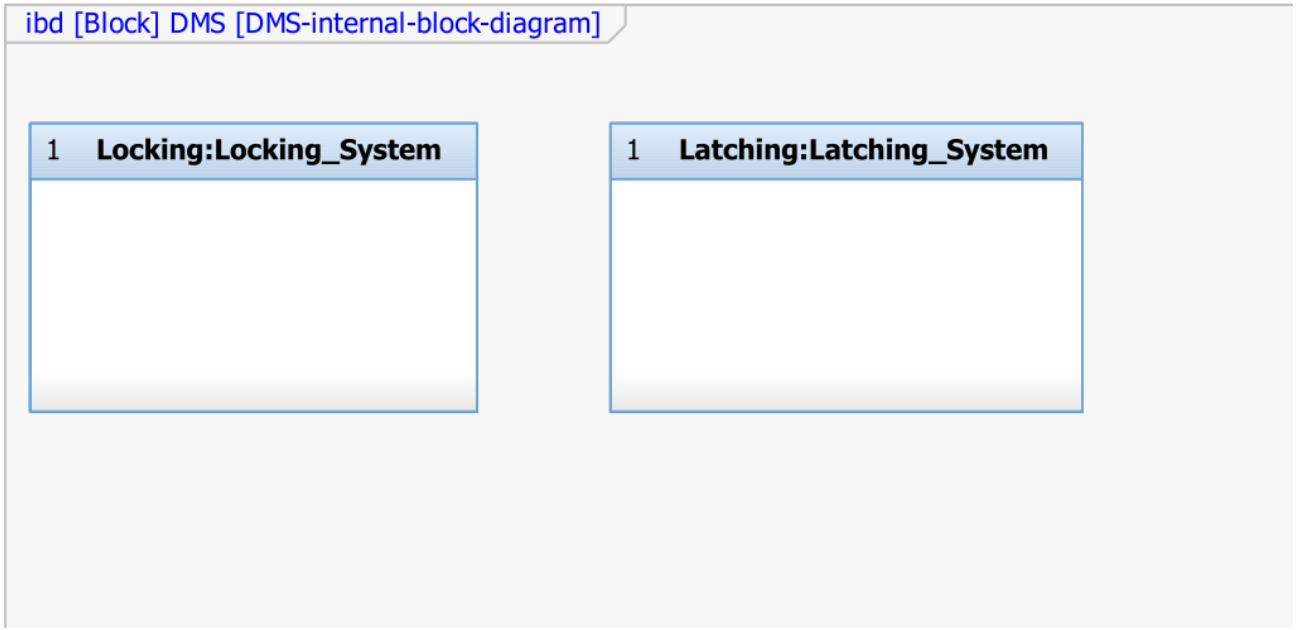
Demo for Scenarios 1 and 2

Scenario 3: Door Management System

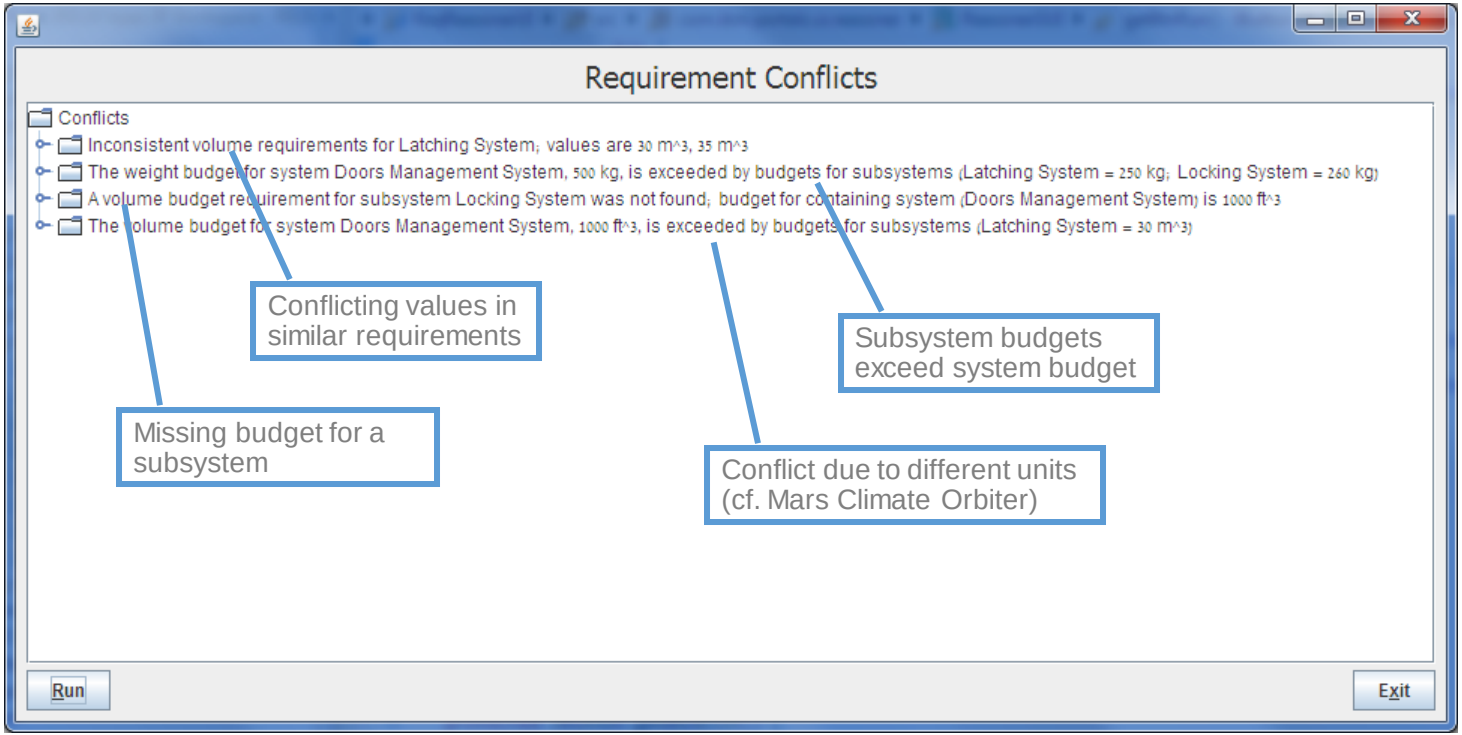
- The weight of the Doors Management System shall not exceed 500 kg.
- The target mass of the locking system shall not exceed 260 kg.
- The target mass of the latching system shall not exceed 250 kg.
- The volume of the Doors Management System shall not exceed 1000 ft^3.
- The volume of the latching system shall not exceed 30 m^3.
- The volume of the latching system shall not exceed 35 m^3.


File Edit View History Bookmarks Tools Help
Project: DMS Scenario (Manu...
Project: PORTALS Demo 2 - R...
Edit User - Server Administrat...
Apache Jena Fuseki - inspect data...
https://painless.haifa.ibm.com:10443/rm/web#action=com.ibm.rdm.web.pages.showProject
Search
Requirements Management (/rm)
PORTALS Demo 2
Yishai Feldman
Project Dashboard
Artifacts
Collections
Modules
Reports
Search Artifacts
Artifacts
Create New Requirements Specifica
Views
Filter by Tag
Filter by Attribute
Filter by Folder
PORTALS Demo 2
00 Admin
01 Requirements
02 Reference
Module Template
ID
Name
Primary Text
678
DMS-019
The weight of the Doors Management System shall not exceed 500 kg.
679
DMS-039
The target mass of the latching system shall not exceed 250 kg.
680
DMS-040
The target mass of the locking system shall not exceed 260 kg.
681
DMSX-1
The volume of the Doors Management System shall not exceed 1000 ft3.
682
DMSX-2
The volume of the latching system shall not exceed 30 m3.
683
DMSX-3
The volume of the latching system shall not exceed 35 m3.

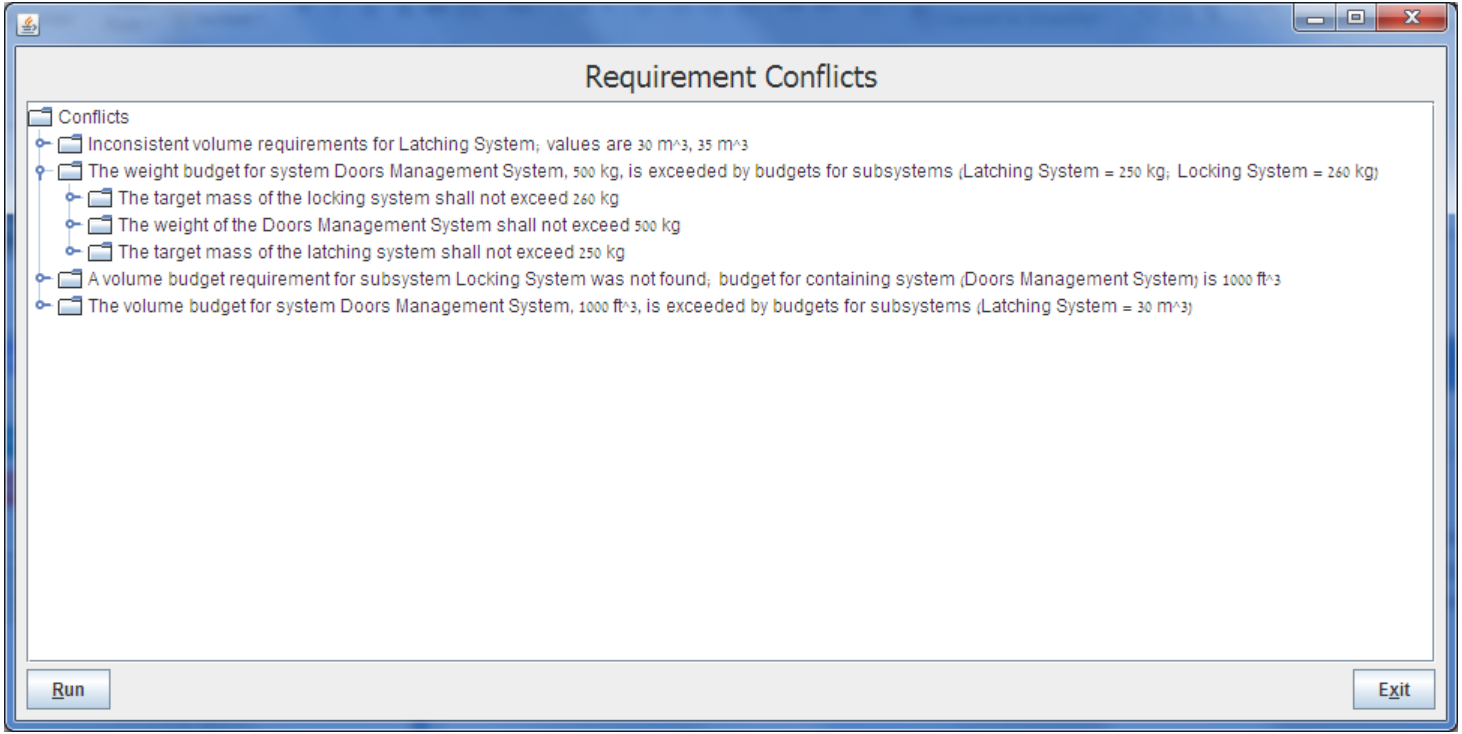
Engineering Knowledge Base:



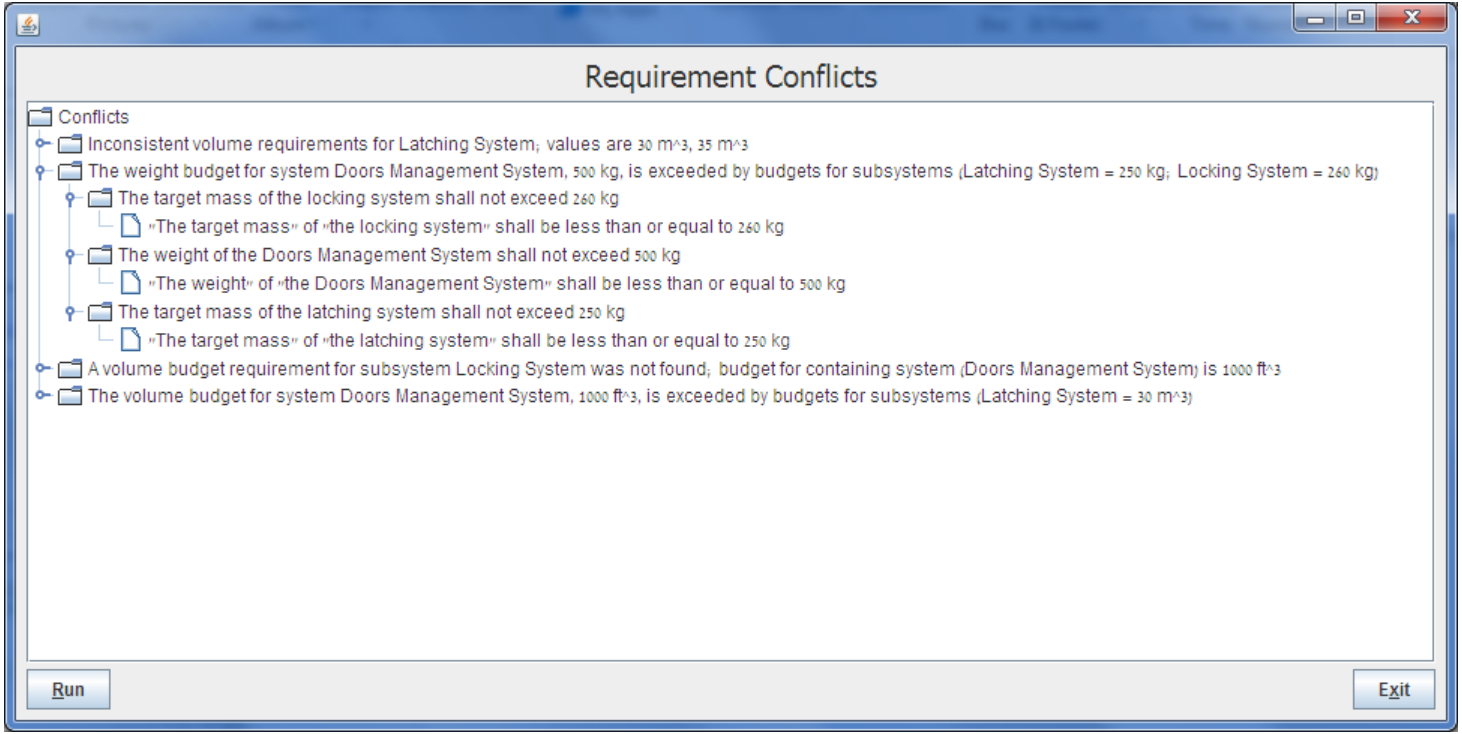
Requirements Analysis



Requirements Analysis



Requirements Analysis



Requirements Formalization and Verification

In the context of adaptive CPSs, checking the consistency of requirements is an indisputable, yet challenging task.

- Requirements written in natural language call for time-consuming and error-prone manual reviews, BUT
- enabling automated consistency verification often requires overburdening formalizations.

Given the increasing pervasiveness of CPSs, their stringent time-to-market and product budget constraints, practical solutions to enable automated verification of requirements are in order.

Formal Specification

Goal: (Semi) Automatic Translation from Natural Language Specification to Formal Specification.

Desiderata: Unambiguous language with high expressiveness, that can be automatically translated in some logic and then used for verification/validation.

Expressiveness vs Unambiguity!

Actually...

Property Specification Patterns (PSPs) offer a viable path towards this goal.

- PSP: collection of parameterizable, high-level, formalism-independent specification abstractions, originally developed to capture recurring solutions to the needs of requirement engineering.
- Each pattern can be directly encoded in a formal specification language, such as linear time temporal logic (LTL), computational tree logic (CTL), or graphical interval logic (GIL).
- Because of their features, PSPs may ease the burden of formalizing requirements, yet enable their verification using current state-of-the-art automated reasoning tools (e.g., for LTL).

Linear Temporal Logic (LTL)

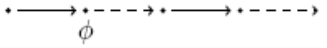
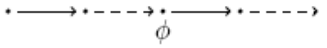
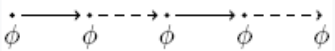
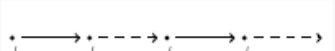
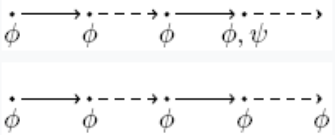
Modal temporal logic with *modalities* referring to time

- 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 $p \in AP$ then p is an LTL formula;
  - if ψ and ϕ are LTL formulas then $\neg\psi$, $\phi \vee \psi$, **X** ψ, and ϕ **U** ψ are LTL formulas.
  - Additional temporal operators: **G** (**g**lobally), **F** (eventually), **R** (**r**elease)

Linear Temporal Logic (LTL) - Semantics

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.	

Property Specification Patterns (PSPs)

- PSPs are meant to describe the essential structure of system's behaviours and provide expressions of such behaviors in a range of common formalisms.
- A pattern is comprised of a
  - name;
  - an informal statement describing the behaviour captured by the pattern;
  - a (structured English) statement that should be used to express requirements.

Property Specification Patterns (PSPs)

The LTL mappings corresponding to different declinations of the pattern are also given, where capital letters (P, Q, R, ...) stands for Boolean states/events.

A complete list of patterns is available at http://patterns.projects.cs.ksu.edu

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.

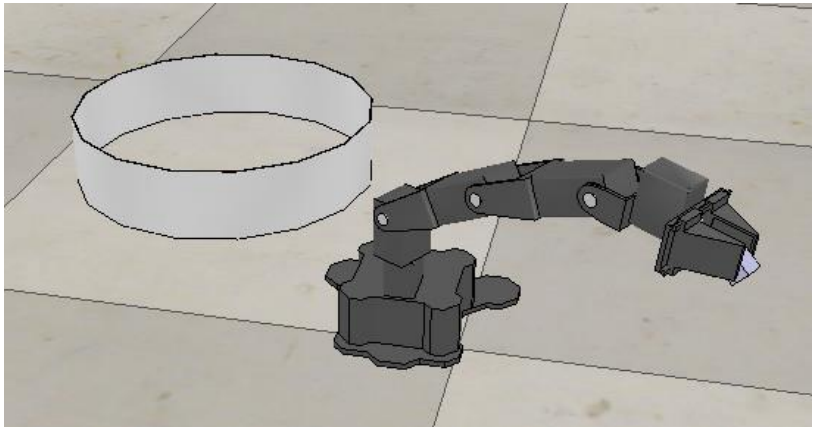
Extending PSPs

- The original formulation of PSPs caters for temporal structure over Boolean variables: **for most practical applications, such expressiveness is too restricted.**
- Example: embedded controller for robotic manipulators (from CERBERO use case)
  - With original PSPs, requirements such as "*The angle of joint1 shall never be greater than 170 degrees*" cannot be expressed.
- Solution proposed in CERBERO: PSPs with Boolean and Constrained Numerical Signals (with sound translation to LTL).

Controller for a Robotic Manipulator

Let consider a set of requirements from the design of an embedded controller for a robotic manipulator:

- the controller should direct a properly initialized robotic arm to look for an object placed in a given position and move to such position in order to grab the object;
- once grabbed, the object is to be moved into a bucket placed in a given position and released without touching the bucket.
- The robot must stop also in the case of an unintended collision with other objects or with the robot itself.
- collisions can be detected using torque estimation from sensors placed in the joints.



The manipulator is a 4 degrees-of-freedom Trossen Robotics WidowX Arm equipped with a gripper

Requirements

- Constrained numerical signals are used to represent requirements related to various parameters
  - angle, speed, acceleration, and torque of the 4 joints, size of the object picked, and force exerted by the end-effector.
- 75 requirements in total.

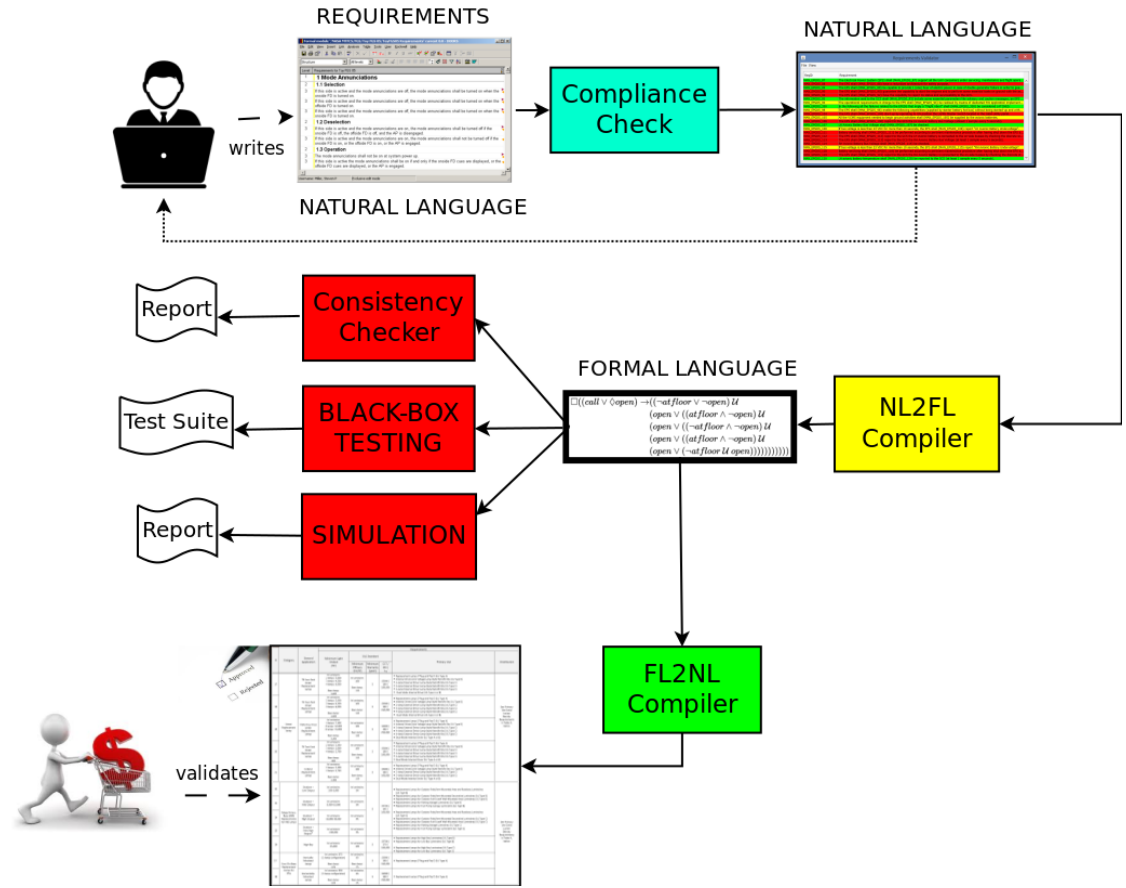
```
Globally, it is never the case that joint1_angle < -170 or joint1_angle > 170 holds.
                                                    …
Globally, it is always the case that if ef_idle holds, then ef_speed = 0 and ef_acc = 0 holds
as well.
                                                    …
After state_init until state_scanning, it is never the case that state_moving_to_target
holds.
```

The complete list is available at https://github.com/SAGE-Lab/robot-arm-usecase

Consistency checking

- The formal representation of all requirements is "glued" together.
- The resulting formula is checked with a Model Checker or Theorem Prover.
- If the formula is satisfiable, then the system can be realized.
- Otherwise, **inconsistency => Impossible to build a system that satisfy all the requirements!**

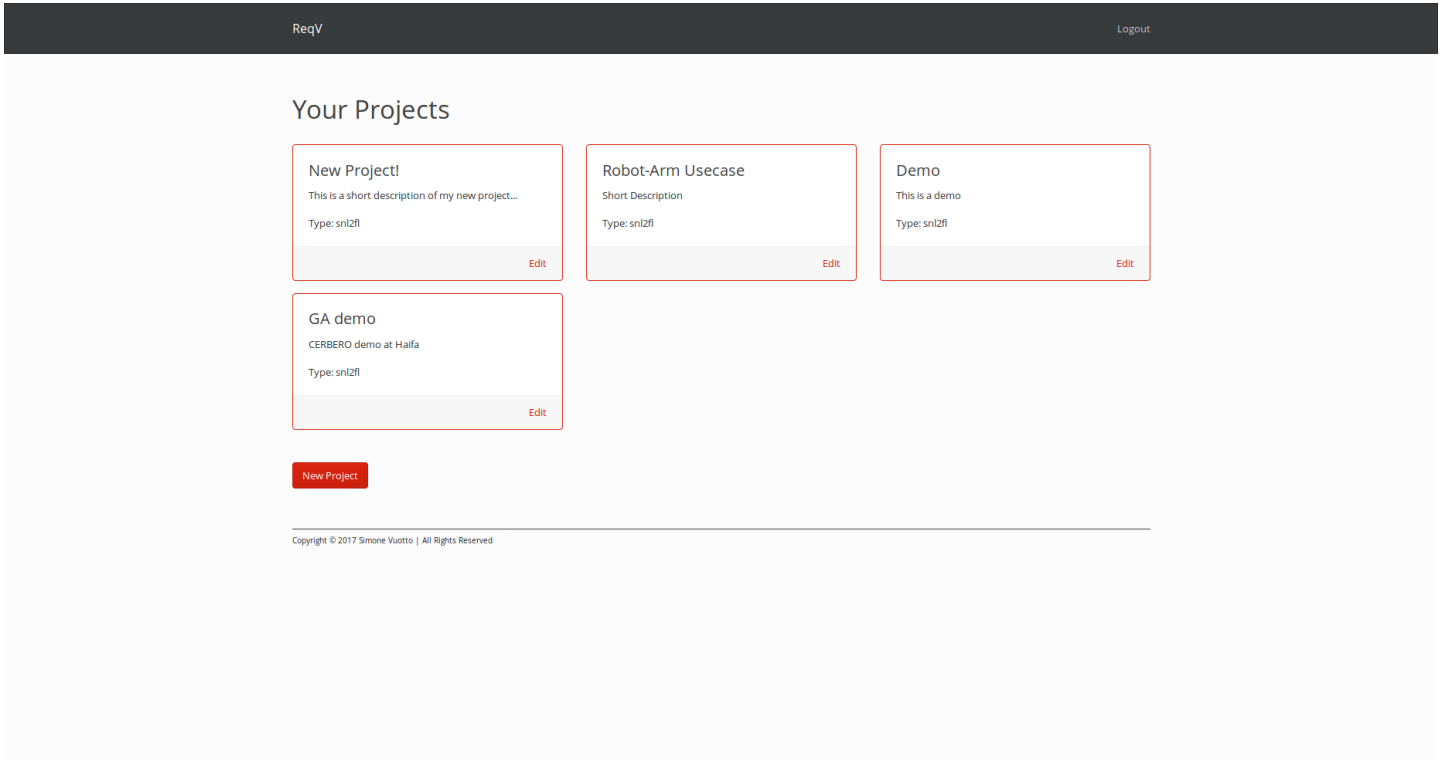
The ReqV tool



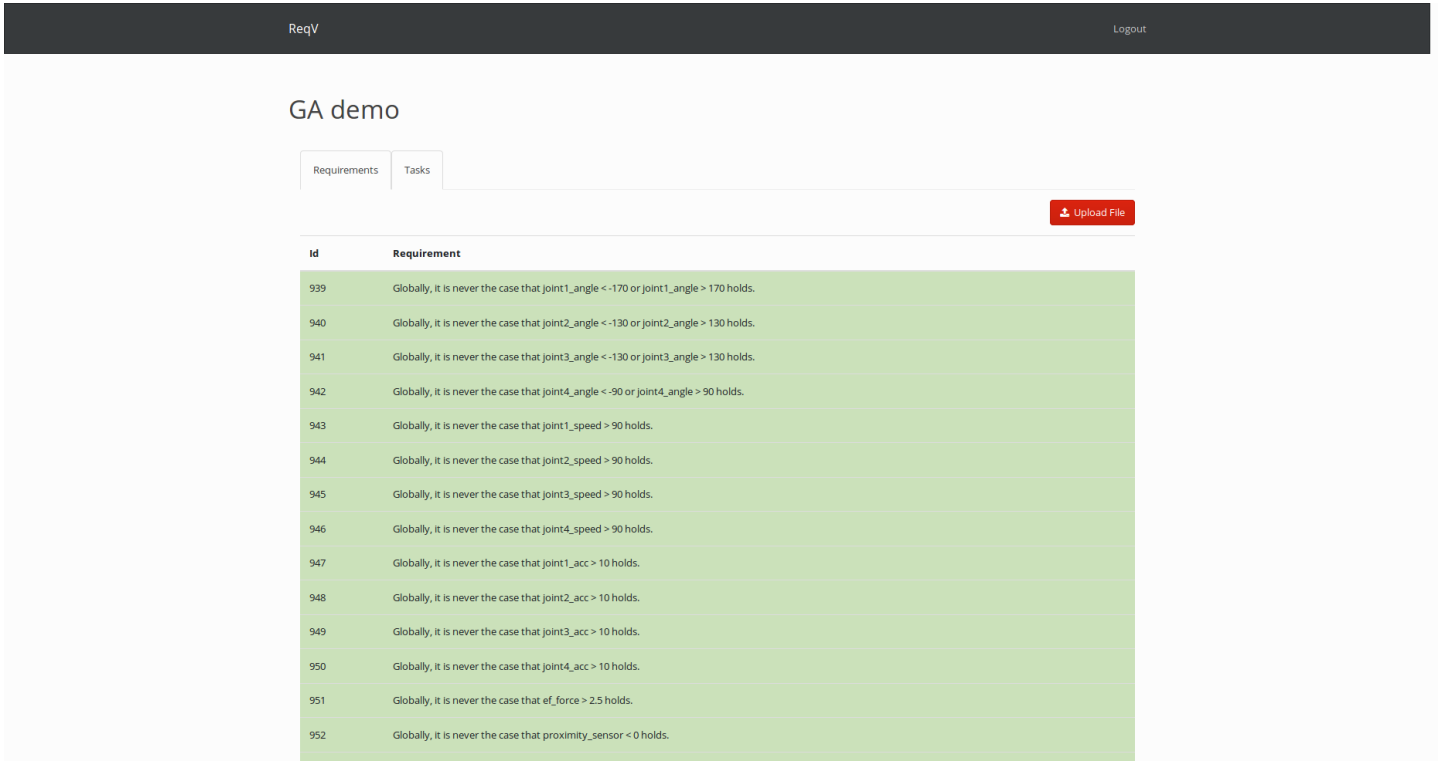
- ReqV (with NuSMV as back engine) checked automatically the requirements in about 37 seconds.

Avaliable at
https://github.com/SimoV8/ReqV-webapp
https://github.com/SimoV8/ReqV-backend

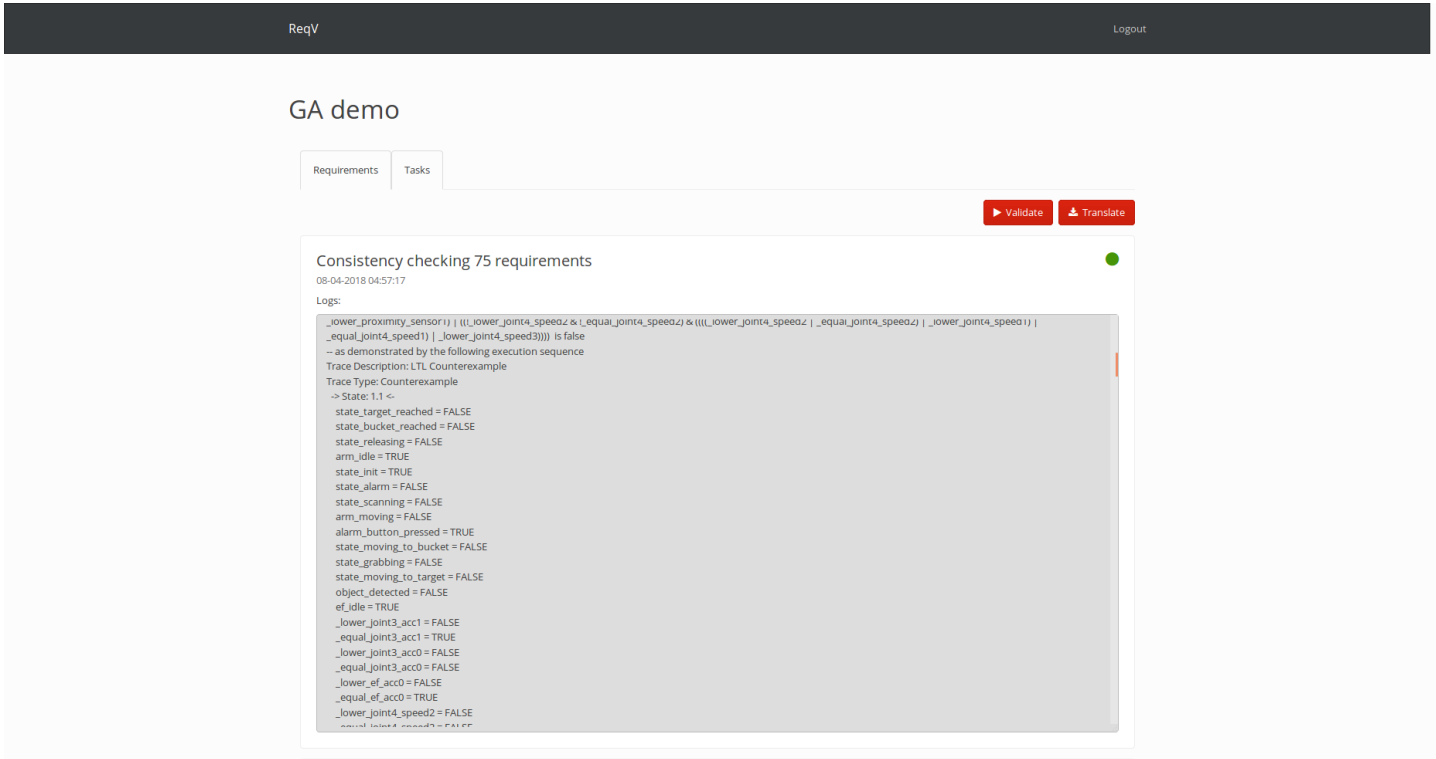
The ReqV tool



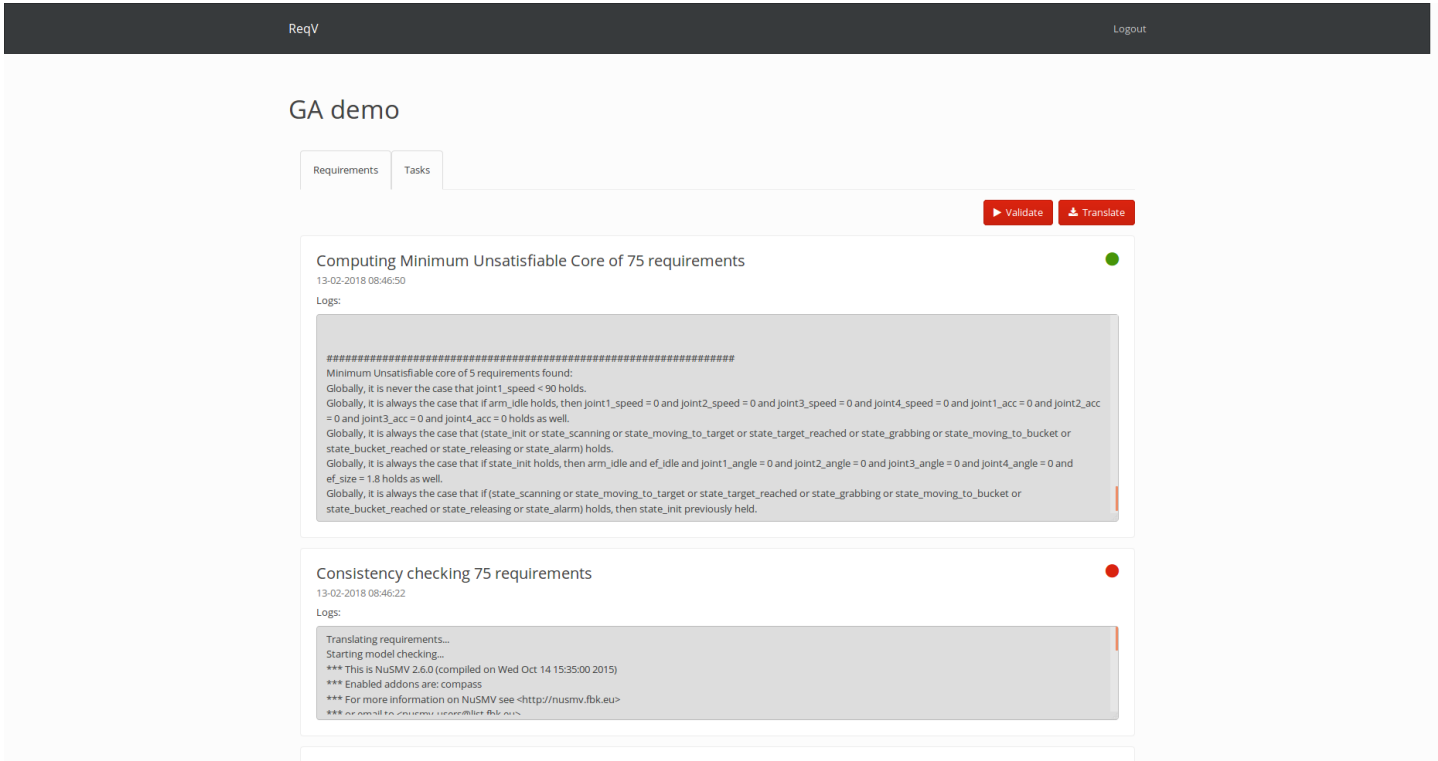
The ReqV tool



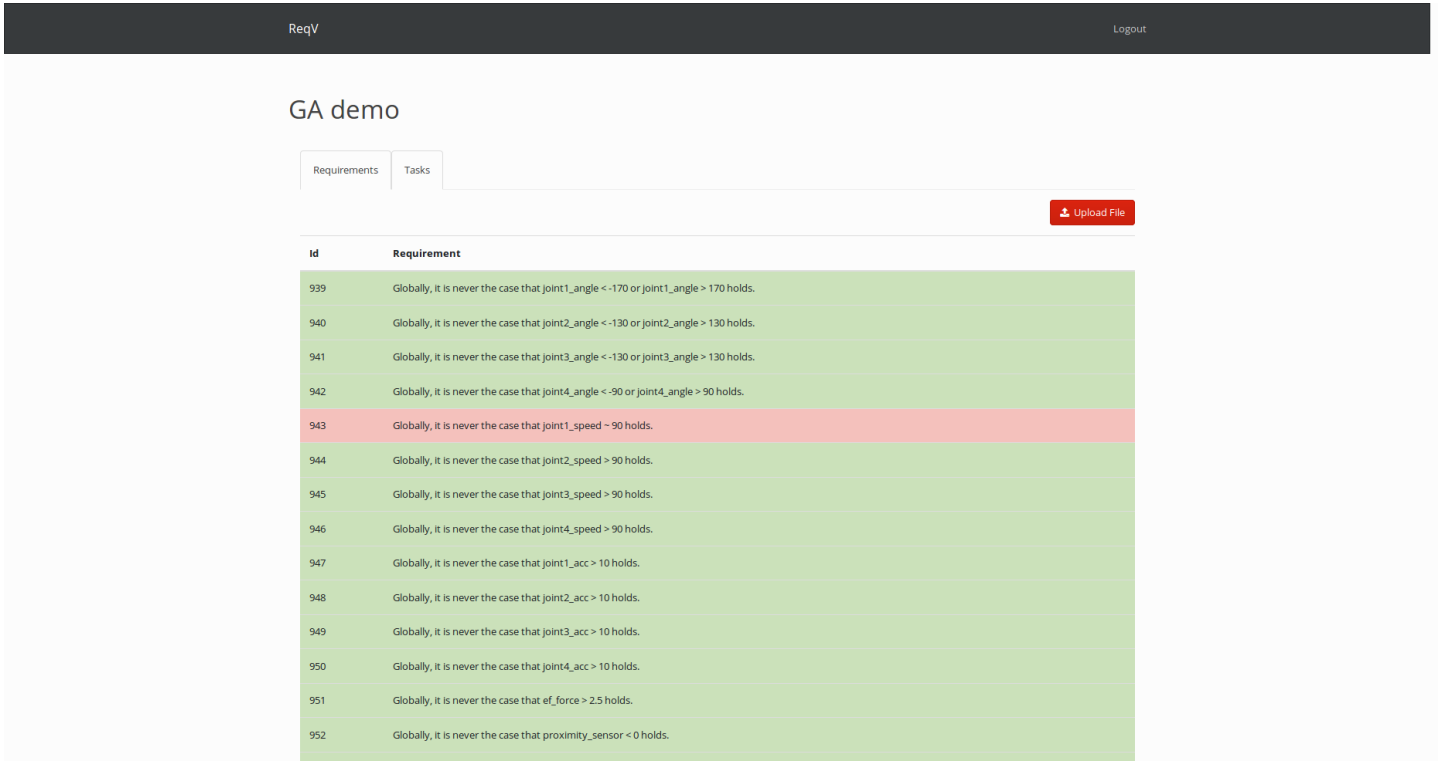
The ReqV tool



The ReqV tool



The ReqV tool



Conclusions

- Enabling the automated (formal) verification of requirements is one of the key aspects towards the development of safety- and security-critical CPSs.
- The expressiveness of original PSPs is often too restricted for practical applications.
  - Hybrid systems? Probabilistic models? Real-time constraints?
- Main issue: scalability!

Some references

- PSPs: Dwyer, Matthew B., George S. Avrunin, and James C. Corbett. "Patterns in property specifications for finite-state verification." *Proceedings of the 21st international conference on Software engineering*. ACM, 1999.
- LTL: Pnueli, Amir, and Zohar Manna. "The temporal logic of reactive and concurrent systems." *Springer* 16 (1992): 12.
- NuSMV model checker: Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., ... & Tacchella, A. (2002, July). Nusmv 2: An opensource tool for symbolic model checking. In *International Conference on Computer Aided Verification* (pp. 359-364). Springer.
- Model checking: Baier, Christel, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.
- PSPs with boolean and constrained numerical signals: Narizzano, M., Pulina, L., Tacchella, A., & Vuotto, S. (2018, April). Consistency of Property Specification Patterns with Boolean and Constrained Numerical Signals. In *NASA Formal Methods Symposium* (pp. 383-398). Springer, Cham.