Requirements Validation Through Scenario Generation and Comparison

Radek Kočí

Brno University of Technology, Faculty of Information Technology
Czech Republic
koci@fit.vut.cz
https://www.fit.vut.cz/person/koci

ICSEA 2020, 18.-22.10.2020

Presenter - dr. Radek Kočí

- assistant professor at Faculty of Information technology, Brno University of Technology
- teaching
  - software engineering and programming
  - artificial intelligence
  - operating systems
- research interest includes
  - modeling and simulation
  - formal models in software engineering
  - security-oriented research
  - genealogy databases
- projects
  - cooperation on modeling and simulation of technological processes, co-author of two software tools
  - he was/is a team member of several Czech Science Foundation projects and EU projects, he led the CSF project

Selected topics and activities

- Simulation Driven Development
  - formal models in the system design and requirements engineering
  - validation through simulation / scenarios
  - model continuity
  - supporting tools
- Agent Systems
  - improving the reasoning of agents described by the Agent-Speak language
  - competition of multiagent systems – 2nd place in the first round
- Genealogy databases and tools
  - systems for transcription of parish records
  - creation of genealogy models (family trees, . . .)
  - TACR (Technology Agency of the Czech Republic) project: Possibilities of creation of communite geneaogical databases with semantic information and uncertainity

Motivation

How to reduce the gap between real needs and specified needs to sofware system under development?

- prototypes, simulation/executable models, . . .
- prototypes/executable models may show a sketch of the system to help visualize what the system will do
- models can serve as a basis for validation of specification

How to validate the correctness of the behavior of the implemented models?

- model scenarios of expected behavior
- scenarios of real behavior under different conditions
- finding a match between the model scenario and real scenarios

Models for Requirements Modeling and Validation

- Domain model – the basis of each design, it captures system concepts
- Use case model – it captures user requirements to the system
- Behavioral model – specifies behavior the use case in the notions of the domain model
  - Workflow model – specific behavioral model
- Model scenario – models a concrete scenario of the use case requested behavior
- Real scenario – captures a real scenario of the model realization

When design the system

- different models are being created
- models for behavior specification are used too
- these models specifies scenarios (or a set of scenarios) of correct (requested) behavior under the given conditions

Possible uses of scenarios

- to verify whether the messaging sequence matches the expected behavior
- to find out when and under what conditions a specific method is called
- to verify whether a specific method is always called under certain conditions

How to achieve this?

Scenario validation

- is basically a simulation verification
- is based on scenarios prepared in advance during the creation of requirements and design
- if the models are modified during the development process, these scenarios are modified

- **the comparison of expected and real scenarios**

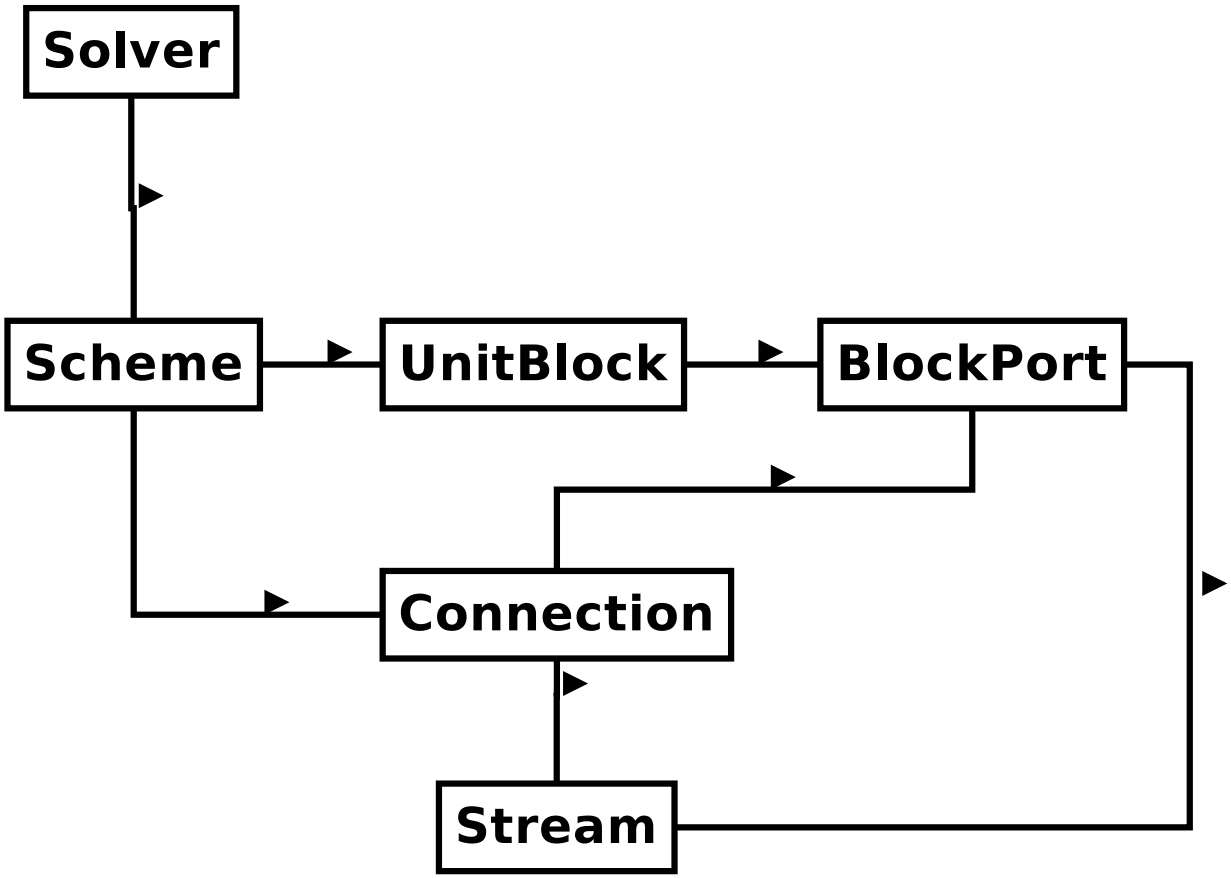
Demonstration method

We will use the following procedure to demonstrate the possibilities of working with scenarios

- We will present an example containing one simulation step in the balance calculation tool,
- we will design a domain model and a sequence diagram according to the standard procedure,
- we can create a workflow using, e.g., Petri nets that allow us to generate scenarios simply,
- based on these scenarios, we then create a so-called model scenario, which we compare with scenarios generated under different conditions.

Domain model

- the basis of each design
- captures the basic concepts of the proposed system
- these concepts appear in other models describing the behavior or interaction of objects



Behavioral model

- UML use case diagram is often use for capturing user requirements
- the behavior of use cases (functional requirements) is described in the text or modeled by other diagrams, such as the activity diagram (behavioral models)
- it is possible to use different formalisms, such as Petri nets
- the chosen concept then defines in what detail the use case's behavior can be specified and how difficult it is to simulate the models created in this way due to requirements verification or transform into the selected source code

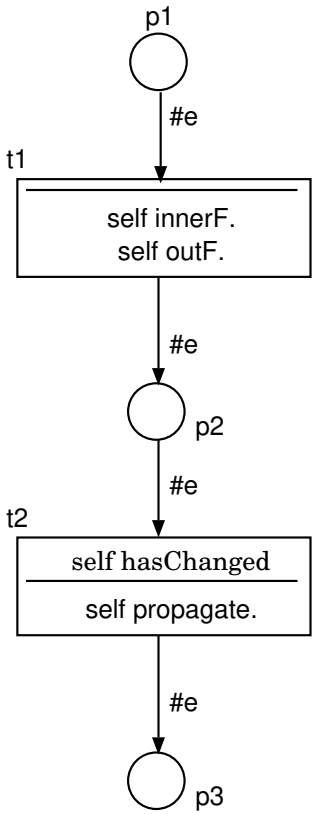
Behavioral model

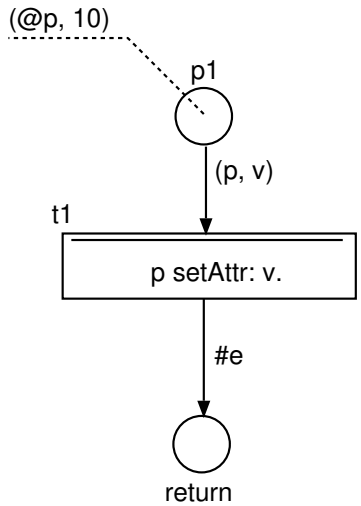
- example the use case balance calculation
- described by structured text

```
Data:
simList : a list of blocks
forall b ∈ simList do
    initialize b
end
forall b ∈ simList do
    if b.hasChanged() then
        b.innerFunction()
        b.outFunction()
        foreach p ∈ b.ports do
            if p.hasChanged() then
                recalculate a stream
                copy a stream
                send a stream copy to the connection
            end
        end
    end
end
```

Workflow model

- captures the functional requirements for each case as a sequence of events
- the formalism of Petri Nets can be used
- example of the case calculation (on the left) and the function outF (on the right)





Sequence diagrams

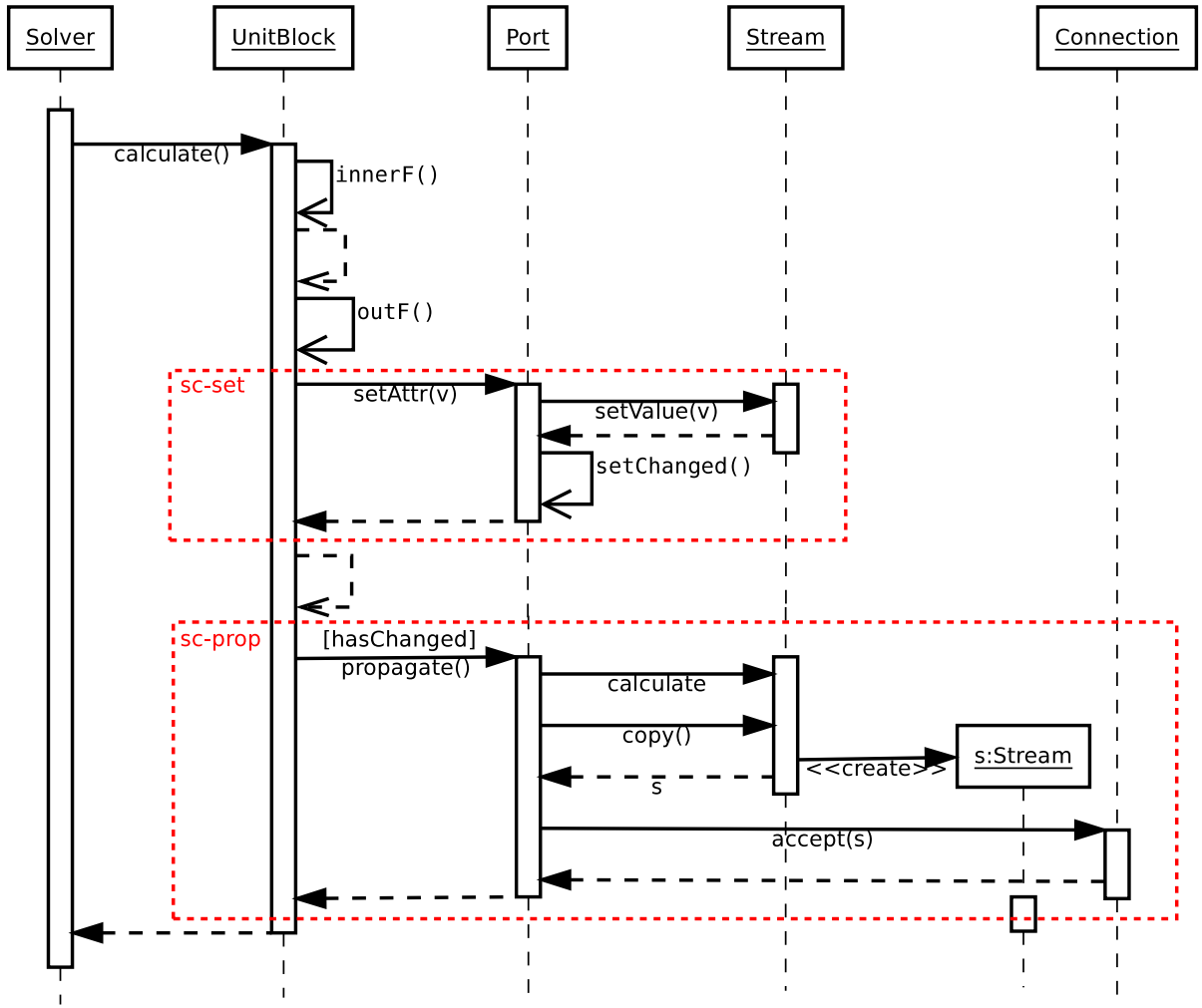
- scenarios help to specify the correct, expected system behavior for a particular task
- scenarios are often modeled using sequence diagrams
- the disadvantage is that the designer often creates these diagrams manually and must follow the rules for their creation, such as following the names defined by the domain model

Model scenario

- if the behavior is described by workflow models, it is possible to generate these scenarios and make our work easier
- generated scenarions serve as model scenarios for further validation of model realization
- our approach works with Petri net models but can be easily adapted to messaging-defined scenarios

Model scenario of the case study

- sequence diagram of the behavior (workflow) balance calculation



Scenario definition

- the scenario model is described as a messaging sequence, where messages can be grouped into blocks
- these blocks (red highlighted sequences on the previous slide) represent one sub-scenario
- messages and sub-scenarios may be repeated – we need a tool for specification of scenario templates
- it is possible to define their repetitions using regular expressions
- a message is denoted as a pair of $\mathbf{msg} = (\mathbf{msg^s}, \mathbf{msg^r})$ representing the sending of the message and its return
- between $\mathbf{msg^r}$ and $\mathbf{msg^r}$, there may be a sequence of additional messages that express the calculation to achieve the desired goal of the $\mathbf{msg}$ message

Message definitions

- $\mathbf{msg^s = (C_1\{o_1\}, C_2\{o2\}, msg_n\{a_1, ..., a_n\})}$, where $\mathbf{C_1}$ is the classifier of the class whose instance sends the message $\mathbf{msg_n}$ of the object of class $\mathbf{C_2}$
- $\mathbf{msg^s = (C_1\{o_1\}, C_2\{o2\}, msg_n\{a_1, ..., a_n\}, ret)}$, where the first three elements semantically correspond to the message $\mathbf{msg^s}$ and $\mathbf{ret}$ is the return value (object) of the message

- each of the listed elements can be parameterized; the parameters are given in curly braces
- for the class classifier it is possible to mark (name) their instances ($\mathbf{o_1}$, $\mathbf{o_2}$), for the sent message its attributes can be defined ($\mathbf{a_1}, ..., \mathbf{a_n}$)
- attributes can have a form of specific values or just formal parameters.

Scenario specification

- the scenario (and sub-scenario) model is denoted by the symbol δ
- the model consists of a sequence of symbols $\mathbf{msg^s}$, $\mathbf{msg^r}$, and δ, which can be repeated
- the rules are simple, it is necessary to follow the pairing of $\mathbf{msg^s}$ and $\mathbf{msg^r}$, and can be described by a context-free grammar $G_M = (\Sigma, N, P, \{S\})$, where $\Sigma = \{msg^s, msg^r, \delta, *\}$ ($*$ represents the iteration symbol, i.e., the possibility of repetition), $\mathbf{N} = \{\mathbf{S}\}$ and $\mathbf{P}$ is a set of rewriting rules in the following form:

$$
\begin{array}{lcl}
\mathbf{S} & \Rightarrow & \delta\ \mathbf{S} \\
\mathbf{S} & \Rightarrow & \delta *\ \mathbf{S} \\
\mathbf{S} & \Rightarrow & \mathbf{msg^s}\ \mathbf{S}\ \mathbf{msg^r} \\
\mathbf{S} & \Rightarrow & \mathbf{msg^s}\ \mathbf{msg^r}\ \mathbf{S} \\
\mathbf{S} & \Rightarrow & \varepsilon
\end{array}
$$

Repeatable sub-scenarios

- correspond to the red highlighted sequences in the sequence diagram

$$
\begin{aligned}
\delta_{\text{sc\_set}} = \quad & (\text{UnitBlock}, \text{Port}, \text{setAttr}\{\text{v}\}), \\
& (\text{Port}, \text{Stream}, \text{setValue}\{\text{v}\}), \\
& (\text{Port}, \text{Stream}, \text{setValue}\{\text{v}\}, \varepsilon), \\
& (\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged}), \\
& (\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged}, \varepsilon), \\
& (\text{UnitBlock}, \text{Port}, \text{setAttr}\{\text{v}\}, \varepsilon)
\end{aligned}
$$

$$
\begin{aligned}
\delta_{\text{sc\_prop}} = \quad & (\text{UnitBlock}, \text{Port}, \text{propagate}), \\
& \ldots \\
& (\text{Port}, \text{Stream}, \text{copy}), \\
& (\text{Port}, \text{Stream}, \text{copy}, \text{s}), \\
& \ldots \\
& (\text{Port}, \text{Connection}, \text{accept}\{\text{s}\}), \\
& \ldots \\
& (\text{UnitBlock}, \text{Port}, \text{propagate}, \varepsilon)
\end{aligned}
$$

The resulting model scenario

- corresponds to the sequence diagram of balance calculation

$$
\begin{aligned}
\delta_{\mathrm{m}} = \quad & (\mathrm{Solver}, \mathrm{UnitBlock}, \mathrm{calculate}), \\
& (\mathrm{UnitBlock\{b\}}, \mathrm{UnitBlock\{b\}}, \mathrm{innerF}), \\
& (\mathrm{UnitBlock\{b\}}, \mathrm{UnitBlock\{b\}}, \mathrm{innerF}, \varepsilon), \\
& (\mathrm{UnitBlock\{b\}}, \mathrm{UnitBlock\{b\}}, \mathrm{outF}), \\
& \delta_{\mathrm{sc\_set}}*, \\
& (\mathrm{UnitBlock\{b\}}, \mathrm{UnitBlock\{b\}}, \mathrm{outF}, \varepsilon), \\
& \delta_{\mathrm{sc\_prop}}*, \\
& (\mathrm{Solver}, \mathrm{UnitBlock}, \mathrm{calculate}, \varepsilon)
\end{aligned}
$$

Validation process

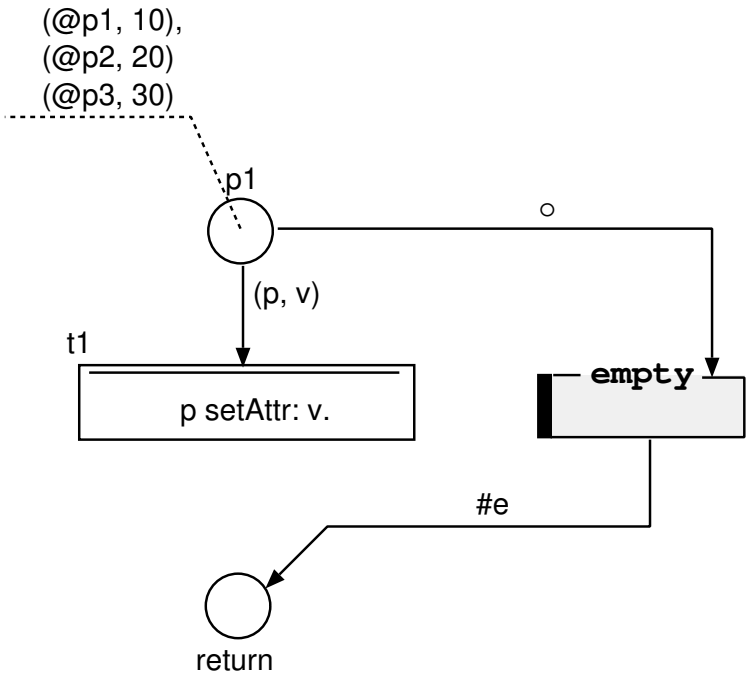
- a comparison of model scenarios and actual scenarios
- a comparison has to respect control characters of the regular expressions
- a tool based on finite state machines can be used

Modes of evaluation

- Entire scenario validation – validation of the whole sequence of the scenario. E.g., it is possible to verify that a method is called that should not be called, the method is not called in the correct place, or the method with the wrong parameters is called.
- Pass validation. The model scenario defines only the key aspects and their order that must be followed. If there are other calls outside these defined points, they are ignored for evaluation.

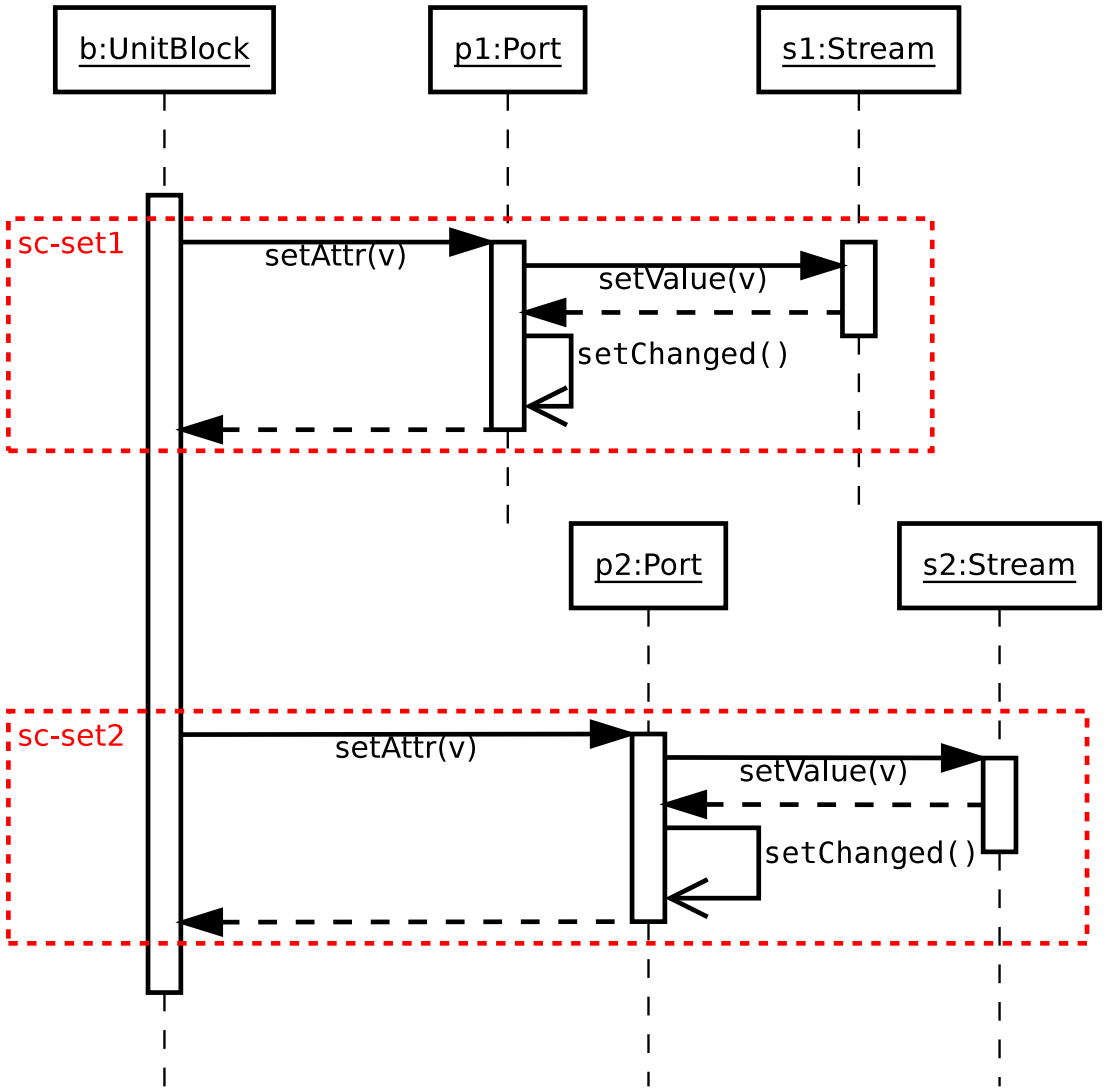
Process

- we must obtain a scenario of the real models or real implementations
- this real scenario must, however, structurally correspond to the model scenario



Generated real scenario

- sequence diagram of the resulted scenario δ_1



Specification of a part of generated scenario

- sequence of calls $\delta_{\mathrm{sc-set1}}$ and $\delta_{\mathrm{sc-set2}}$ corresponding to marked blocks sc-set1 and sc-set2 (only an example for sc-set1 is presented)

$$\begin{aligned}\delta_{\mathrm{sc\_set1}} = \quad & (\mathbf{UnitBlock}, \mathbf{Port}, \mathbf{setAttr}\{10\}),\\ & (\mathbf{Port}, \mathbf{Stream}, \mathbf{setValue}\{10\}),\\ & (\mathbf{Port}, \mathbf{Stream}, \mathbf{setValue}\{10\}, \varepsilon),\\ & (\mathbf{Port}\{\mathbf{p}\}, \mathbf{Port}\{\mathbf{p}\}, \mathbf{setChanged}),\\ & (\mathbf{Port}\{\mathbf{p}\}, \mathbf{Port}\{\mathbf{p}\}, \mathbf{setChanged}, \varepsilon),\\ & (\mathbf{UnitBlock}, \mathbf{Port}, \mathbf{setAttr}\{10\}, \varepsilon)\end{aligned}$$

Validation / Evaluation

- comparing the sub-scenario $\delta_{\text{sc\_set}}$ with the sequence of scenarios $\delta_{\text{sc\_set1}}$ and $\delta_{\text{sc\_set2}}$

$\delta_{\text{sc\_set}}$	$\delta_{\text{sc\_set1}}$
$(\text{UnitBlock}, \text{Port}, \text{setAttr}\{\text{v}\})$,	$(\text{UnitBlock}, \text{Port}, \text{setAttr}\{10\})$,
$(\text{Port}, \text{Stream}, \text{setValue}\{\text{v}\})$,	$(\text{Port}, \text{Stream}, \text{setValue}\{10\})$,
$(\text{Port}, \text{Stream}, \text{setValue}\{\text{v}\}, \varepsilon)$,	$(\text{Port}, \text{Stream}, \text{setValue}\{10\}, \varepsilon)$,
$(\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged})$,	$(\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged})$,
$(\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged}, \varepsilon)$,	$(\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged}, \varepsilon)$,
$(\text{UnitBlock}, \text{Port}, \text{setAttr}\{\text{v}\}, \varepsilon)$	$(\text{UnitBlock}, \text{Port}, \text{setAttr}\{10\}, \varepsilon)$

- we find that they are structurally identical, only substitutions $\{\text{v}/10\}$ and $\{\text{v}/20\}$ occur
- it can be concluded that a sequence $\delta_{\text{sc\_set1}}, \delta_{\text{sc\_set2}}$ equals $\delta_{\text{sc\_set}}$ $*$, formally $\delta_{\text{sc\_set}} \; * \; \tilde{=} \; \delta_{\text{sc\_set1}}, \delta_{\text{sc\_set2}}$
- similarly it can be concluded that $\delta_{\text{m}} \tilde{=} \delta_1$

$\Rightarrow$ the newly generated scenario δ_1 corresponds to the scenario model δ_{m}

Pass Validation

- is not interested in the whole scenario, but only the fulfillment of some condition
- a model scenario containing those call only those calls that we consider crucial for validation is generated

Pass Validation Example

- condition: the setChanged method must always be called after any call of the setAttr method
- model scenario:

$$\delta_{\text{pass}} = (\text{UnitBlock}, \text{Port}, \text{setAttr}\{\text{v}\}), (\text{UnitBlock}, \text{Port}, \text{setChanged})$$

Pass Validation Example

- when comparing the model scenario with the generated one, we will only be interested in whether the above sequence is followed and other parts of the scenario will be uninteresting for us
- comparing the sub-scenario δ_{pass} with the sequence of scenarios $\delta_{\text{sc\_set1}}$

δ_{pass}

(UnitBlock, Port, setAttr{v}),

(Port{p}, Port{p}, setChanged),

$\delta_{\text{sc\_set1}}$

(UnitBlock, Port, setAttr{10}),
$(\text{Port}, \text{Stream}, \text{setValue}\{10\}),$
$(\text{Port}, \text{Stream}, \text{setValue}\{10\}, \varepsilon),$
(Port{p}, Port{p}, setChanged),
$(\text{Port}\{\text{p}\}, \text{Port}\{\text{p}\}, \text{setChanged}, \varepsilon),$
$(\text{UnitBlock}, \text{Port}, \text{setAttr}\{10\}, \varepsilon)$

- we find that
  - $\delta_{\text{pass}} \cong \delta_{\text{sc\_set1}} \{\text{v}/10\}$
  - $\delta_{\text{pass}} \cong \delta_{\text{sc\_set2}} \{\text{v}/20\}$

Present state

- the concept of requirements validation through scenarios has been introduced
- scenarios can be described, e.g., by sequence diagrams
- workflow models offer a more general description ability and allow the generation of specific scenarios or models, i.e., some patterns that can then be used for comparison
- we currently have a tool for generating sequence diagrams from models described by Petri nets; the presented concept works only with a structural comparison

Future work

- to parameterize the sequences themselves – this feature would make it possible, for verification purposes, to specify precisely which specific objects are involved in the communication, in what state, etc.
- extension of the tool to other models

The end.