
Specification and Modelling of Embedded Systems

Prof. Dr. Holger Schlingloff
Fraunhofer FIRST & Humboldt Universität

holger.schlingloff@first.fraunhofer.de

Holger Schlingloff

Professor of Software Engineering at Humboldt University, Berlin
Research Director at the Fraunhofer Institute FIRST, Berlin

The image displays two overlapping web browser windows. The foreground window is titled "Humboldt Universität zu Berlin, Prof. Dr. Holger Schlingloff - Microsoft Internet Explorer" and shows a personal website. The background window is titled "Fraunhofer FIRST - Home" and shows the official website of the Fraunhofer Institute FIRST.

Prof. Dr. Holger Schlingloff
Humboldt-Universität zu Berlin, Institut für Informatik

Contact:

Prof. Dr. rer. nat. habil. Bernd-Holger Schlingloff
Professor of [Informatics](#) at the [Humboldt-Universität zu Berlin](#)
Group leader at the [Fraunhofer Institute FIRST](#)

Office: [Kekuléstr. 7, 12489 Berlin](#)
Tel.: ++49 30 6392 1907
Mobile: ++49 179 597 3372
Fax: ++49 30 6392 1805
Email: hs@informatik.hu-berlin.de

[Publications](#)

Fraunhofer FIRST
Fraunhofer FIRST / Institute

Areas:
Modeling
Systems Architecture
Quality Assurance

News from the Institute?
Please click on one of the pictures.

©2011 Fraunhofer FIRST | [Imprint](#) | [Sitemap](#) | [Top](#)

Structure of this Talk

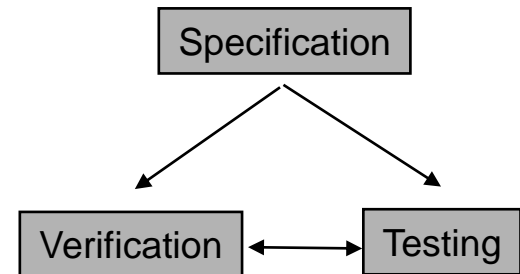
- My personal background
- Current projects
- Research issues in system specification
 - Formalization of use case descriptions
 - Modelling of informal requirements

My History: From Theory into Practice

- Ph.D. work: completeness and expressivity of temporal logics
e.g. models which are trees (not sequences or graphs):
needs not only “nexttime” and “until”, but also “sibling” operator
yields first-order expressive completeness via separation
- Habilitation work: model checking of real-time temporal logics
define temporal logic for timed Petri nets
partial-order reduction technique for state explosion problem
- Work as managing director of Bremen Institute for Safe Systems
model checking for industrial applications
avionics interface of ISS, satellite charge control, UMTS protocol stack
- Fraunhofer: Specification, verification and testing of embedded systems

Present

- Specification
 - Modeling of causality
- Verification
 - Static analysis and model checking
- Testing theory
 - Model-based testing of embedded systems



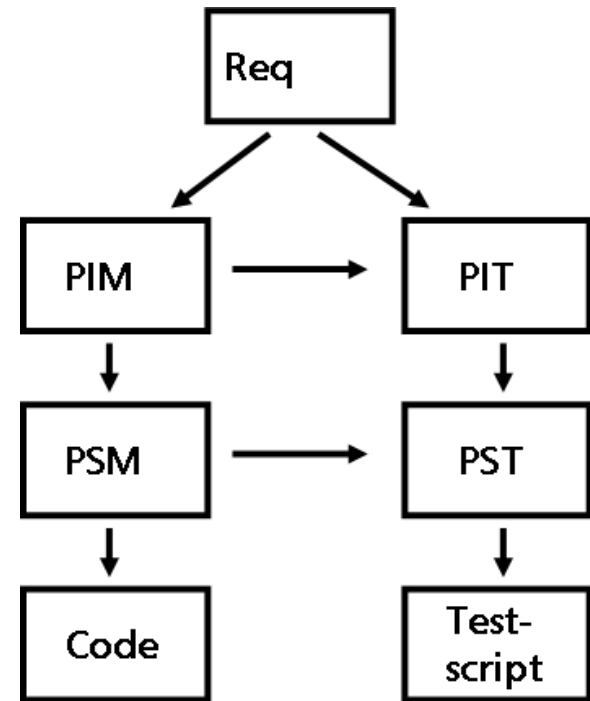
Current Projects

- ETCS radio block centre as software product line
- Verification of Paediatric Ventricular Assistant Device
- DSL metamodelling for dishwasher controls
- Model-based design of a gas burner
- Model-based testing of a crane master control

„Real“ Specifications I

Requirements can be given as

- contract specification
- use case descriptions
- algebraic or logical formula
- class descriptions with pre- and postconditions
- UML state diagrams
- timed or hybrid automata
- Matlab/Simulink files,
- Code / Pseudo-Code,
- ...



Use Case Descriptions

Requirements often are “use case descriptions” (standardized natural language)

Natural language is informal, imprecise, allows over/under-specification, ...

- inconsistencies
- ambiguities
- incompleteness

→ Methodology of transforming textual use case descriptions into models to verify them

2. Typical and Critical Scenarios

2.1 Brake failure while brake is closed

This scenario is part of the realization of DriveWrp.FMon.Brake operation failure [1].

2.1.1 Preconditions

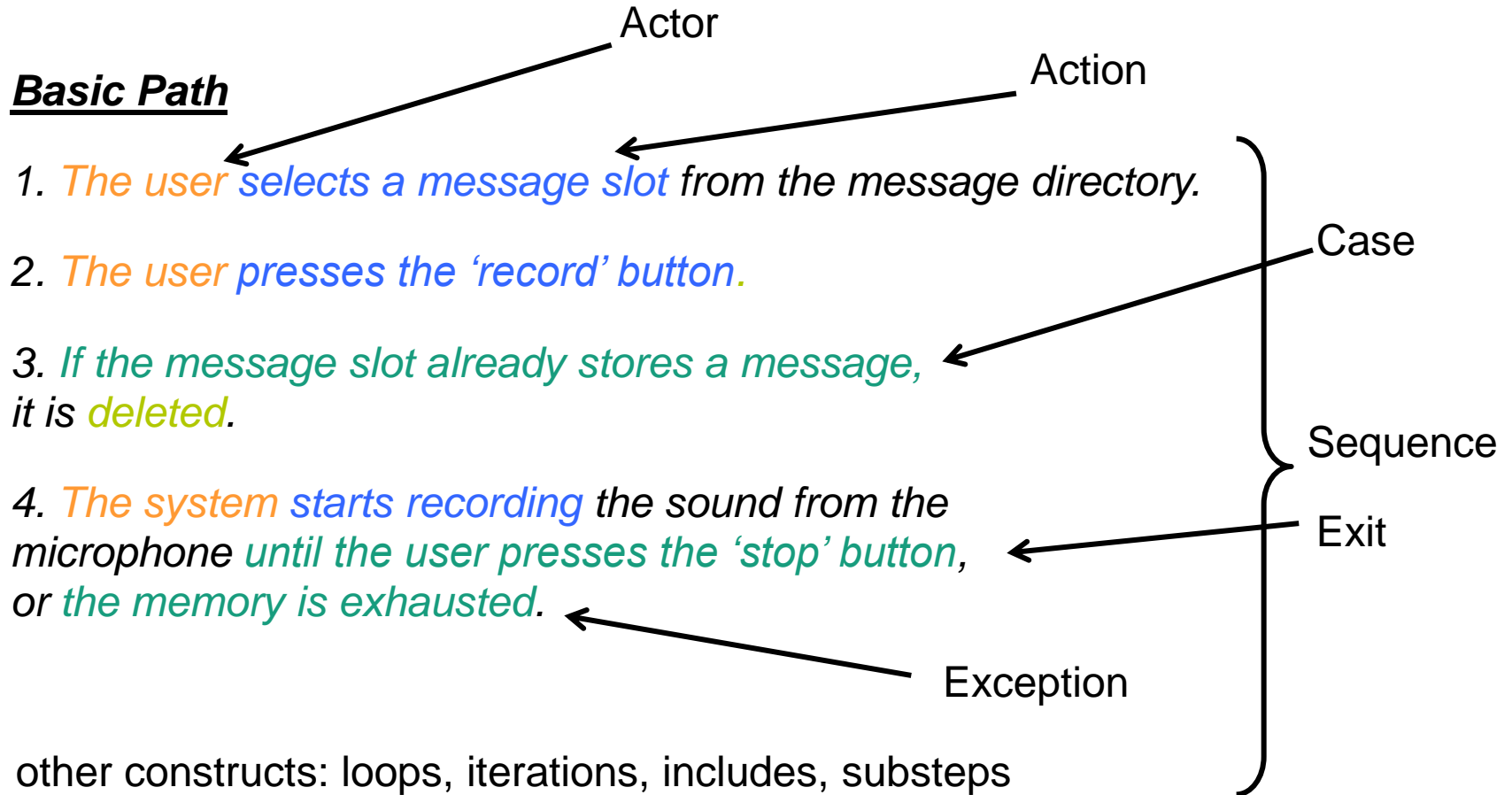
Brake is in state closing or closed, no Alarm E_FC_ELEVATOR_MECHANICAL_BRAKE_KB or _FC_ELEVATOR_MECHANICAL_BRAKE_KB1 during previous trip occurred. (**Note:** The case of an Alarm E_FC_ELEVATOR_MECHANICAL_BRAKE_KB or _FC_ELEVATOR_MECHANICAL_BRAKE_KB1 during a trip is handled in ch. 2.2)

2.1.2 Basic Flow

Seq-Nr	Description	Junction
1.	FC sends error E_FC_ELEVATOR_MECHANICAL_BRAKE_KB or E_FC_ELEVATOR_MECHANICAL_BRAKE_KB1 to the Drive component	
2.	The FC blocks itself	
3.	The Drive component logs the Error "Error Brake Partial Failure"	
4.	The Drive components sets the state of the monitoring function "MOB" in the MonitoringSupervisor to "Error"	
5.	The MonitoringSupervisor logs the Status "Breakdown"	
6.	The MonitoringSupervisor activates the service ESLN in the service handler	
7.	The ServiceHandler blocks the Drive interface and opens the door.	
8.	The Drive component sets the state of the monitoring function "MOB" in the MonitoringSupervisor to "ReadyForReset"	
9.	END SysUC	

Formalization of Use Case Descriptions

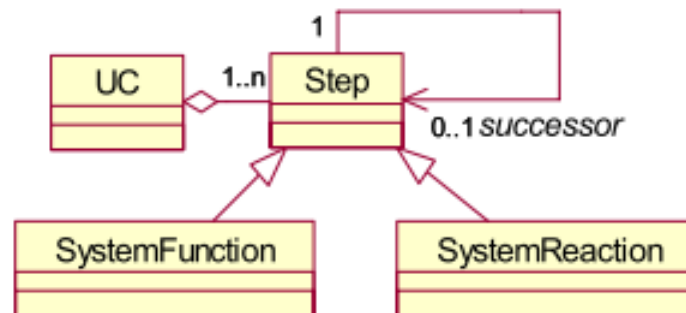
Example Use Case “Start Record”



Task: transform this into a (semi-) formal representation

- Statecharts or state transition diagrams ?
- MSCs or sequence diagrams ?
- Petri nets or activity diagrams ?
- live sequence charts ?

- ➔ use special metamodel of necessary constructs for intermediate format
- ➔ transform into appropriate representation with automated model transformation



Translation into Intermediate Format

Procedure:

1. Identification of system interfaces
2. Identification of system functions and reactions
3. Formalization of control flow
4. Formalization of individual steps
5. Translation of pre- and postconditions

```
User.selectMessageSlot(Slot)
User.startRecording()
if (Slot.full())
    System.deleteMessageSlot(Slot)
exception (System.exhaustMemory() || User.stopRecording())
    System.recordMessage(Slot)
```

Use Case Validator

The screenshot displays the Eclipse IDE in the Use Case Validator Perspective for the project 'SoundRecorder.ucf'. The interface is divided into several panes:

- Navigator:** Shows the project structure with 'SoundRecorder2' selected.
- SoundRecorder.ucf:** The main editor showing the use case model. It contains a table with the following data:

Step	Assigned Interaction
Basic Path	Basic Path
1. The user selects a ...	User.selectMessageSlot(Slot)
2. The user presses t...	User.startRecording()
unnamed Switch	Slot.full()
If	If
3. If the messa...	System.deleteteMessageSlot(Slot)
4. The system starts r...	System.recordMessage(Slot)

- Action View:** Shows the system and user actions. The system actions are recordMessage(SlotNo: Integer), deleteteMessageSlot(SlotNo: Integer), and exhaustMemory(). The user actions are startRecording(), selectMessageSlot(SlotNo: Integer), and stopRecording().
- Outline:** Shows the structure of the use case 'Record a Message', including the Basic Path and the four steps.
- Console:** Displays the output of the validation process, including the message 'Starte Generierung des .xmi ...' and the file paths for the generated models.
- Binding View:** Shows a table with two columns: Parameter and Type. Both columns contain the text 'no parameters'.
- Variable View:** Shows the variable 'Slot: Integer'.

„Real“ Specifications II: A Fuel Cell Controller

Task: Protect a valve to freeze, by killing the engine. (The valve controls the gas flow from the tank to the engine)

Informal specification:

- If the temperature sensor is more than 3s (short delay) "too cold" a quick stop occurs and the engine is shut off.
- If the temperature sensor was invalid and switches to valid again and during the following 3s the temperature is not warm a long delay of 15s is activated. In this state a "too cold" triggers the quick-stop after 15s (long delay). (Long delay replaces the initial short delay).
- If the temperature is "warm" then the 3s (short delay) is valid again.
- If the valid temperature switches to invalid the 3s (short delay) is valid again.
- If during the delay the valid temperature is not "too cold" for more than 0.2s the delay timer is reset to start a new delay period.

Analysis

Definitions

- Temperature Sensor reads: warm, cold, tooCold, invalid
- Time Window: shortDelay (3s), longDelay (15s)
- Actions: quickStop

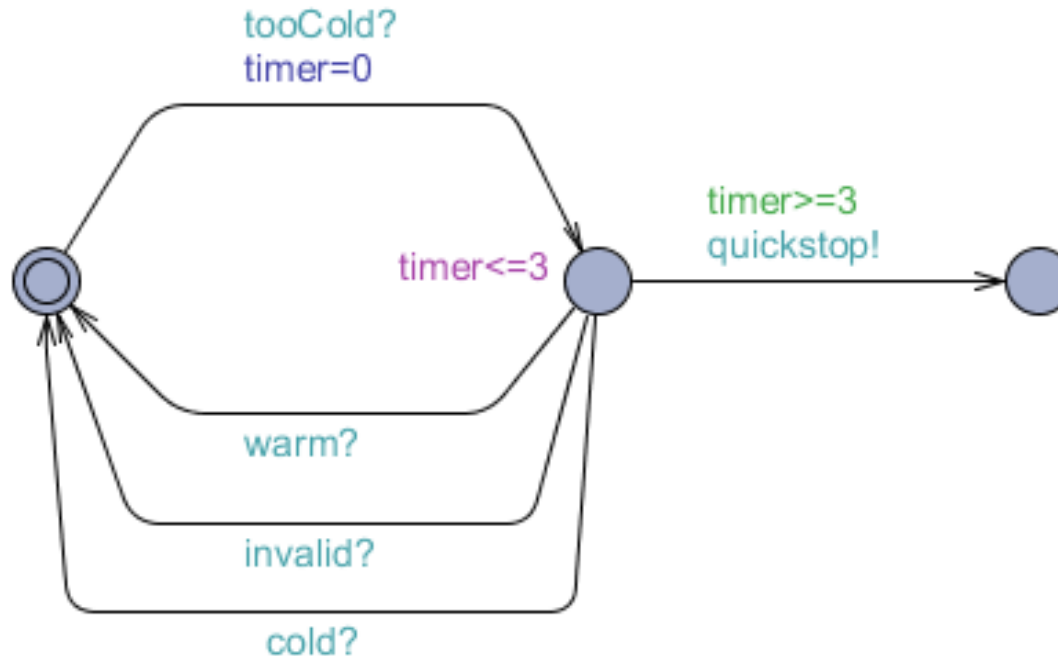
Design decisions

- State- or event-based modelling, e.g. of temperature sensor
- Modeling of timing and timers

Modelling with MTL and Timed Automata

Rule 1: If the temperature sensor is more than 3s "too cold" a quick stop occurs.

$$\square(\text{tooCold} \wedge \square_{\leq 3}(\neg \text{warm} \wedge \neg \text{cold} \wedge \neg \text{invalid}) \rightarrow \square_{=3} \text{quickStop})$$



Alternative

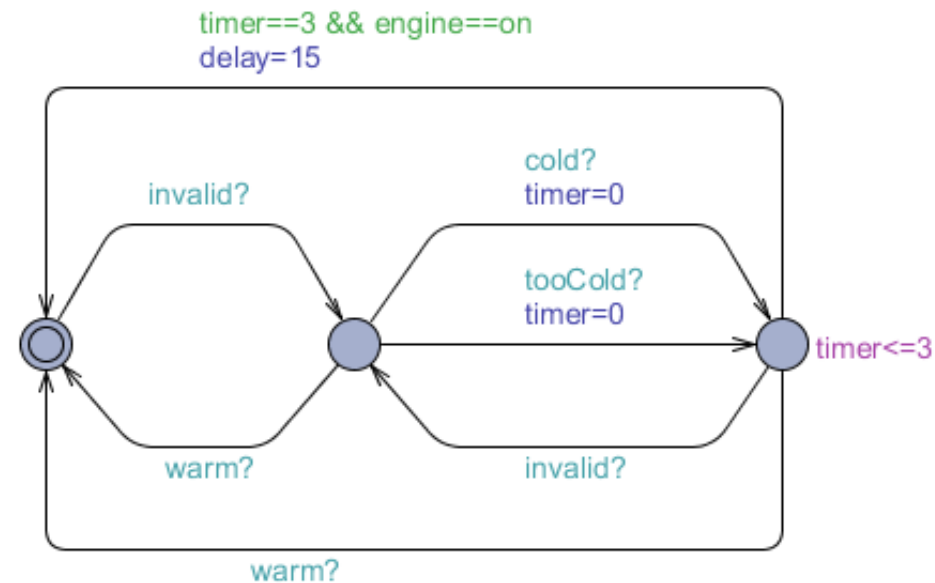
Rule 2: If the temperature sensor was invalid and switches to valid again and during the following 3s the temperature is not warm a long delay of 15s is activated. In this state a "too cold" triggers the quick-stop after 15s (long delay).

$$(\neg \text{valid} \mathcal{S} \text{ invalid}) \wedge \text{valid} \wedge \square_{\leq 3} \neg \text{warm} \rightarrow \text{longDelay}$$

$$(\text{longDelay} \wedge \text{tooCold} \wedge \square_{\leq 15} (\neg \text{warm} \wedge \neg \text{cold} \wedge \neg \text{invalid}) \rightarrow \square_{=15} \text{quickStop})$$

$$(\neg \text{longDelay} \wedge \text{tooCold} \wedge \square_{\leq 3} (\neg \text{warm} \wedge \neg \text{cold} \wedge \neg \text{invalid}) \rightarrow \square_{=3} \text{quickStop})$$

where $\text{valid} \triangleq (\text{warm} \vee \text{cold} \vee \text{tooCold})$



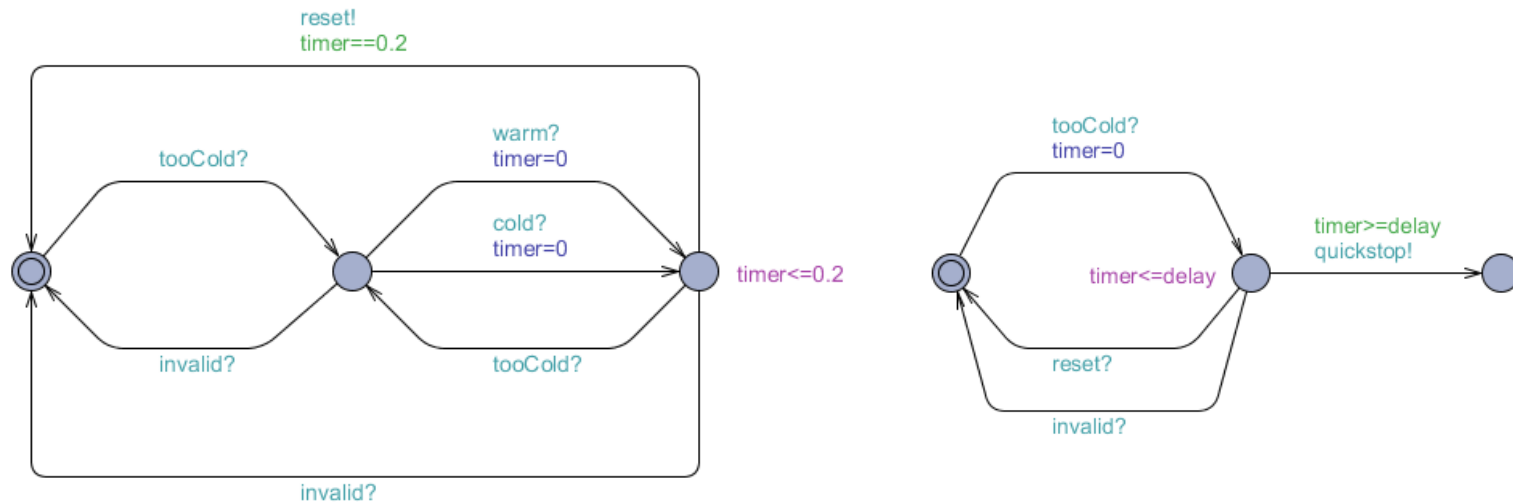
Revision

Rule 5: If during the delay the valid temperature is not "too cold" for more than 0.2s the delay timer is reset to start a new delay period.

If during the delay the valid temperature is not "too cold" for *less* than 0.2s the delay timer is *not* reset and the current delay period is continued.

$$(\neg \text{longDelay} \wedge \text{tooCold} \wedge \square_{\leq 3}(\text{warm} \vee \text{cold} \vee \text{invalid} \rightarrow \diamond_{< 0.2} \text{tooCold}) \rightarrow \square_{=3} \text{quickStop})$$

$$(\text{longDelay} \wedge \text{tooCold} \wedge \square_{\leq 15}(\text{warm} \vee \text{cold} \vee \text{invalid} \rightarrow \diamond_{< 0.2} \text{tooCold}) \rightarrow \square_{=15} \text{quickStop})$$



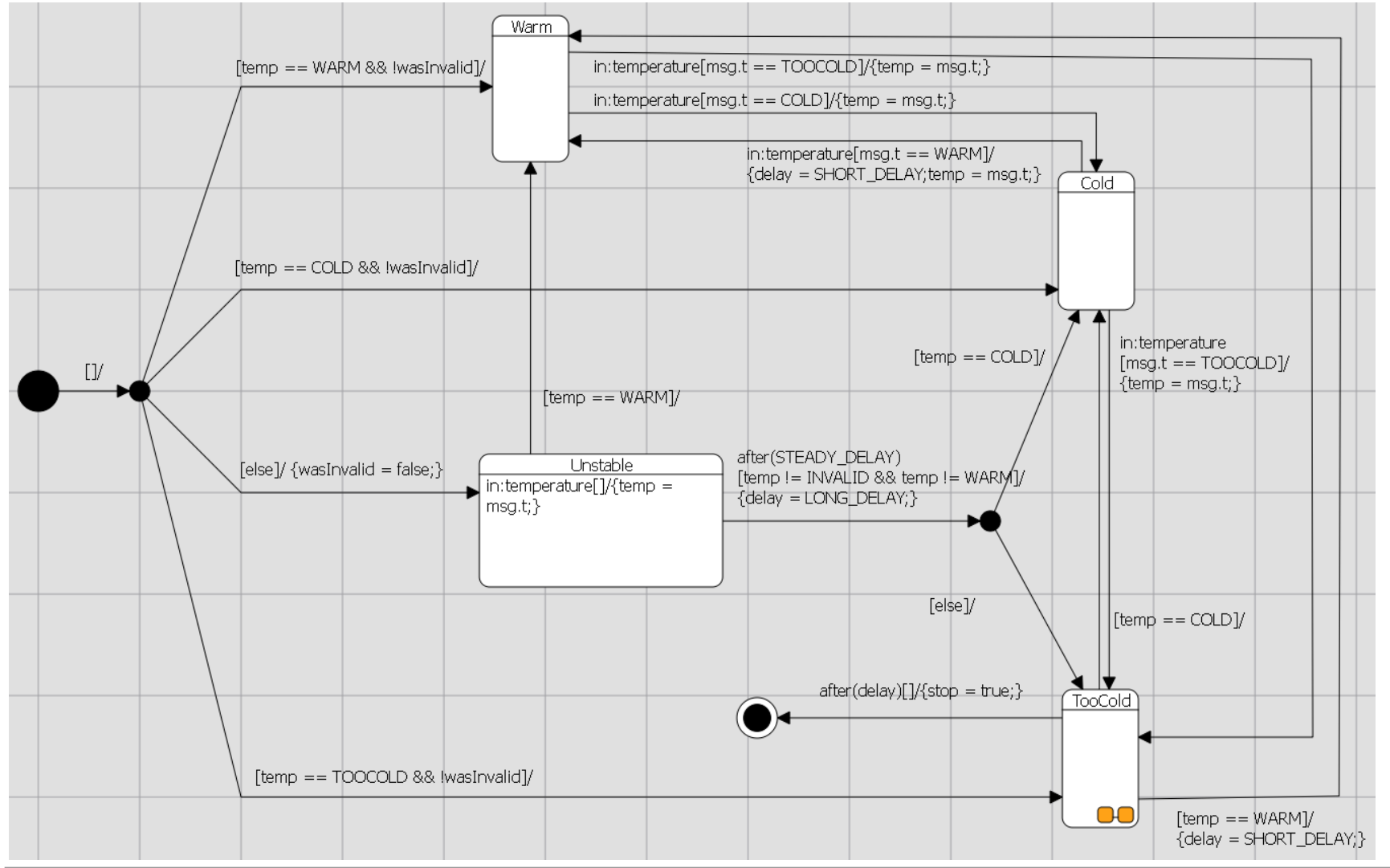
Observations

- Mixture of declarative and state-based description
- New rules modify / alternate previous ones
 - compositional in TA
 - noncompositional in TL
- Formulation with a particular implementation in mind

- Formalization is used for
 - systems development, or
 - test generation

- How to evaluate the “quality” of a formalization? (validation, not verification)
 - simulation
 - test generation
 - test execution

UML-Modelling

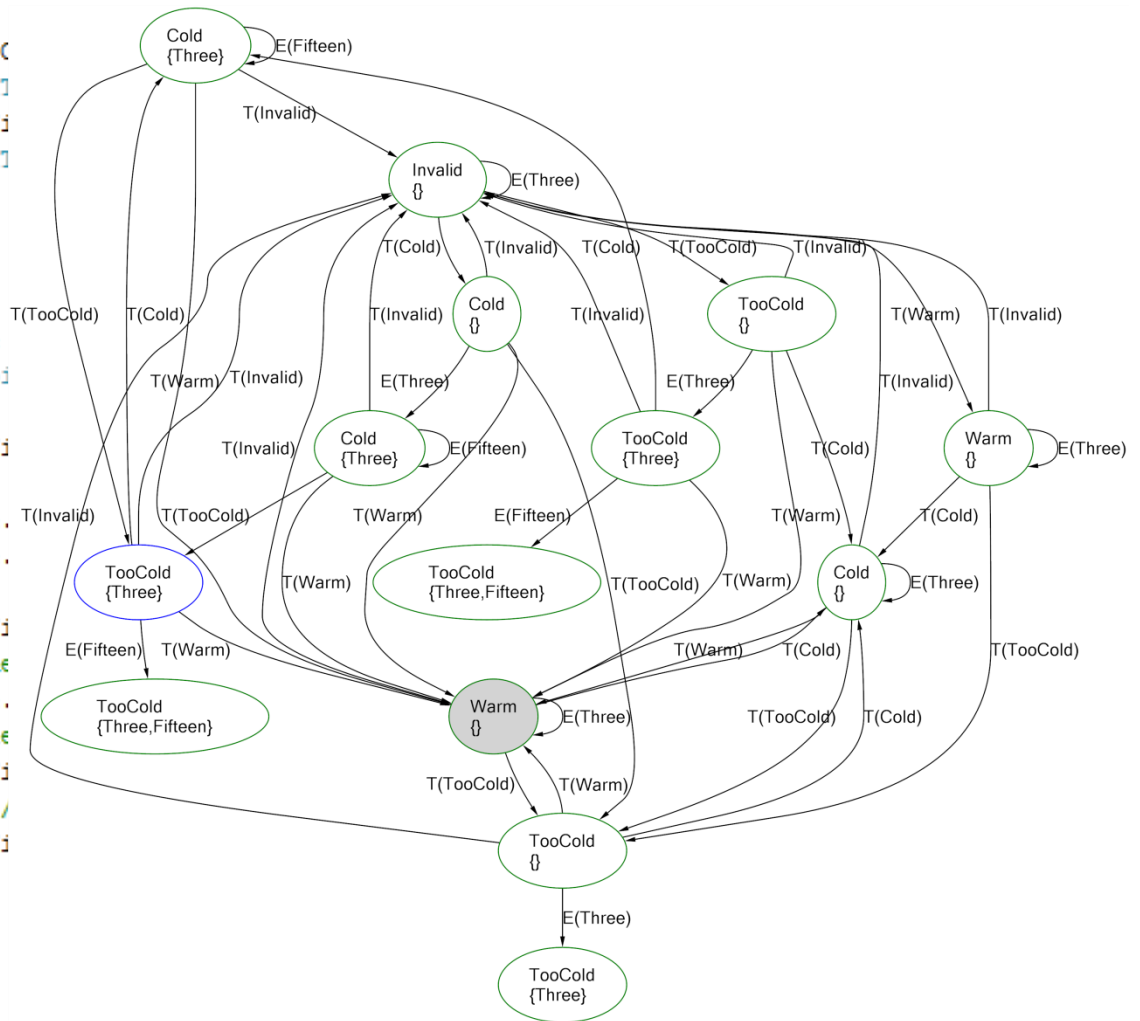


C#-Modelling

```

// Rule 1 and 2nd Half of Rule 5
if (temperature == Temperature.TooCold
    (expiredTimers.Contains(ExpiredTimer d == Delay.Short) || (expiredTimers.Contains(ExpiredTimer d == Delay.Long)
    isStopped = true;
    engineRunning = false;}
else {
    // Rule 2
    if (wasInvalid && (temperature == Temperature.TooCold
        expiredTimers.Contains(ExpiredTimer d = Delay.Long;
        expiredTimers = expiredTimers.Expire(d);
    // Rule 3 & Rule 4
    if (temperature == Temperature.TooCold
        (temperature == Temperature.TooCold
            d = Delay.Short;
            expiredTimers = expiredTimers.Expire(d);
    // Rule 5 (Based on a 0.2s Time)
    if (temperature == Temperature.TooCold
        if (d == Delay.Short) // Re-expire
            expiredTimers = expiredTimers.Expire(d);
        else if (d == Delay.Long) // Re-expire
            expiredTimers = expiredTimers.Expire(d);
    }
}

```



Summary

- My personal background – see <http://www2.informatik.hu-berlin.de/~hs/>
- Current projects – maybe more next time
- Research issues in system specification
 - formalization of use case descriptions
 - UCV for interactively generating various UML-models
 - connection to consistency checker pending
 - Modelling of informal requirements
 - “revision” operation for formulas and models
 - metrics for quality of formalization

Thank you for your attention!