Real-Time Systems

Lecture 1: Introduction

2017-10-17

Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Content

- Introduction
  - a software engineering perspective
  - a theoretical computer science perspective

- Real-Time Systems
  - vs. reactive systems
  - vs. hybrid systems
  - safety-critical systems
  - examples

- Lecture Content Overview
  - and non-content

- Formalia
  - times/dates, procedures, exam

- A Formal Model of Real-Time Behaviour
  - state variables / observables
  - evolution / behaviour
Introduction: Software Engineering Perspective
**Recall: Software Engineering**

- **misunderstandings / errors** detected **late in development** can be **expensive:**
  - design and implementation may need to be **re-done.**

- **misunderstandings / errors** detected only **during use** can be **fatal:**
  - software malfunction may **harm business goals,** or even lead to **people being hurt.**
One approach to detect misunderstandings / errors early:

- describe **requirements** precisely / formally / **mathematically**
- try to **prove** requirements to be consistent, complete, etc.
- describe **design ideas** precisely / formally / **mathematically**
- try to **prove** that design satisfies requirements, i.e. that design is **correct**
To develop **software that is (provably) correct wrt. its requirements**, we need:

(i) a **formal model** of software **behaviour**

(ii) a **language*** to specify **requirements** on **behaviour**,  
(to distinguish desired from undesired behaviour),

(iii) a **language*** to specify **behaviour** of **design ideas**,  

(iv) a notion of **correctness**  
(a relation between requirements and design specifications),

(v) and a **method** to **verify (or prove) correctness**  
(that a given pair of requirements and design specifications are in correctness relation).

*: at best concisely and conveniently, with adequate expressive power.
Choose **observables**:
\( R: \) red light on, \( \bar{R}: \) red light off, \( Y: \) yellow light on, \( \bar{Y}: \) yellow light off, 
\( G: \) green light on, \( \bar{G}: \) green light off.

**Model of (finite) behaviour**: \( \Sigma^* \), where \( \Sigma = (\{R, \bar{R}\} \times \{Y, \bar{Y}\} \times \{G, \bar{G}\}) \). We write, e.g., \( RYG \) as shorthand for \((R, Y, G)\).

**Example behaviours**:
- \( RYG, RYG, RYG \)
- \( RYG, RYG, RYG \)

**Requirements**:

- Desired lights sequence: red, red-yellow, green, yellow, ...
  **Formalisation**: \( \text{Req}_1 := (RYG.RYG.RYG.RYG)^* \) regular expression
- Undesired configuration: red-green
  **Formalisation**: \( \text{Req}_2 := \Sigma^*.RYG.\Sigma^* \)

**Design**:

- Define **notion of correctness**:
  A design \( \text{Des} \) is correct wrt. requirement \( \text{Req} \) if and only if \( \mathcal{L}(\text{Des}) \subseteq \mathcal{L}(\text{Req}) \).
Example (Un-timed): Traffic Lights

- Choose **observables**:
  - $R$: red light on, $\bar{R}$: red light off,
  - $Y$: yellow light on, $\bar{Y}$: yellow light off,
  - $G$: green light on, $\bar{G}$: green light off.

- **Model of (finite) behaviour**: $\Sigma^*$, where $\Sigma = (\{R, \bar{R}\} \times \{Y, \bar{Y}\} \times \{G, \bar{G}\})$. We write, e.g., $RYG$ as shorthand for $(R, Y, G)$.

  **Example behaviours**:
  - $RYG$, $RYG$, $RYG$  
  - $RYG$, $RYG$, $RYG$

- **Requirements**:
  - Desired lights sequence: red, red-yellow, green, yellow, …

    **Formalisation**: $\text{Req}_1 := (RYG.RYG.RYG.RYG)^*$

  - Undesired configuration: red-green

    **Formalisation**: $\text{Req}_2 := \Sigma^*.RYG.\Sigma^*$

- **Design**:

  - $\text{Des}_0 := RYG \rightarrow RYG \rightarrow RYG \rightarrow RYG$

- **Define notion of correctness**:

  A design $\text{Des}$ is correct wrt. requirement $\text{Req}$ if and only if $\mathcal{L}(\text{Des}) \subseteq \mathcal{L}(\text{Req})$.

  Design $\text{Des}_0$ is correct wrt. requirements $\text{Req}_1$ and $\text{Req}_2$ (proof method: automata theory).
To develop **software that is (provably) correct wrt. its requirements**, we need:

(i) a **formal model** of software **behaviour**

(ii) a **language*** to specify **requirements** on **behaviour**,  
(to distinguish desired from undesired behaviour),

(iii) a **language*** to specify **behaviour** of **design ideas**,  

(iv) a notion of **correctness**  
(a relation between requirements and design specifications),

(v) and a **method** to **verify (or prove) correctness**  
(that a given pair of requirements and design specifications are in correctness relation).

*: at best concisely and conveniently, with adequate expressive power.
Example (Timed): Traffic Lights

- **Requirement**: yellow phases (RYG) should have a duration of 3 seconds on streets with speed limit 50 km/h.

- How do we **formally model** traffic lights behaviour with time?
  
  For example (informal):
  - red for 10 s
  - red-yellow for 2 s
  - green for 120 s
  - yellow for 3 s

- How do we **formalise** the timed requirement of 3 s?

- How do we **formally model** a controller design with time?

- What does it mean for a timed design to be correct wrt. a timed requirement?

- How do we prove timed designs correct wrt. timed requirements?

→ Lecture “Real-Time Systems”
Content

- Introduction
  - a software engineering perspective
  - a theoretical computer science perspective

- Real-Time Systems
  - vs. reactive systems
  - vs. hybrid systems
  - safety-critical systems
  - examples

- Lecture Content Overview
  - and non-content

- Formalia
  - times/dates, procedures, exam

- A Formal Model of Real-Time Behaviour
  - state variables / observables
  - evolution / behaviour
Introduction: Theoretical Computer Science Perspective
Lectures like *Introduction to Theoretical Computer Science* ("Informatik 3") cover content such as

- **propositional logic**
  - syntax, semantics, decision problems (e.g., satisfiability is decidable)

- **finite automata**
  - syntax, language of an automaton
  - decision problems (e.g., language emptiness is decidable)
  - properties, e.g., finite automata are closed under intersection

**Questions**: Are there logics whose models are timed behaviours?

- Is satisfiability still decidable?
- If not for the full logic, then for which fragment?

**Questions**: If we equip finite automata with real-time clocks,

- is language emptiness still decidable?
- are the set of such timed automata still closed under intersection?
- is it decidable whether a timed automaton satisfies a timed property?

→ Lecture “*Real-Time Systems*”
Content

- Introduction
  - a software engineering perspective
  - a theoretical computer science perspective

- Real-Time Systems
  - vs. reactive systems
  - vs. hybrid systems
  - safety-critical systems
  - examples

- Lecture Content Overview
  - and non-content

- Formalia
  - times/dates, procedures, exam

- A Formal Model of Real-Time Behaviour
  - state variables / observables
  - evolution / behaviour
Reactive Systems

- A **reactive system** interacts with its environment by reacting to inputs from the environment with certain outputs.

- Reactive systems usually **do not terminate**. For example, the traffic lights controller continues to run, unless there is a power outage or a scheduled maintenance.

- **Contrast**: terminating, transformational systems. For example: a sorting or searching function.

- **Reactive systems** can be partitioned into:

![Diagram of reactive system](image)

- “In constructing a **real-time system** the aim is to control a physically existing environment, the **plant**, in such a way that the controlled plant satisfies all desired (timing) requirements.”
A **Real-Time System** is a **reactive system** which, for certain inputs, has to compute the corresponding outputs **within given time bounds**.

A **Hybrid System** is a **real-time system** consisting of continuous and discrete components. The continuous components are time-dependent (!) physical variables ranging over a continuous value set.

A system is called **Safety Critical** if and only if a malfunction can cause loss of goods, money, or even life.
Another Definition Douglass (1999)

- “A real-time system is one that has performance deadlines on its computations and actions.”

- Sometimes distinguished:
  - “Hard deadlines: performance requirements that absolutely must be met each and every event or time mark.” (→ this lecture)
    “(Early / late data can be bad data.)”
  - “Soft deadlines: for instance about average response times.”
    “(Early / late data is still good data.)”

- Design Goal:
  A timely system, i.e. one which is meeting its performance requirements.

- Note: performance can in general be measured by any unit of quantities:
  - (discrete) number of steps or processor instructions,
  - (discrete or continuous) number of seconds, (→ this lecture)
  - etc.
**Example: Airbag Controller**

**Controller requirement:** “When a crash is detected, fire the airbag.”

- When firing **too early**: airbag ineffective.
- When firing **too late**: additional threat.

Say, 300ms (plus/minus small $\varepsilon$) after a crash is the right™ time to fire.

Then the **precise requirement** is

“When a crash is detected at time $t$, fire the airbag at $t + 300ms \pm \varepsilon$.”
**Example: Airbag Controller**

**Controller requirement:** “When a crash is detected, fire the airbag.”

- When firing **too early**: airbag ineffective.
- When firing **too late**: additional threat.

Say, 300ms (plus/minus small $\varepsilon$) after a crash is the right™ time to fire.

Then the **precise requirement** is

“When a crash is detected at time $t$, fire the airbag at $t + 300\text{ms} \pm \varepsilon$.”

What is the plant, what is the controller?
Example: Gas Burner

Where is the plant, where is the controller?
A situation where the gas valve is open but there is no flame is called leakage.

Leakage is practically unavoidable:

- for ignition, first open valve,
- then ignite the available gas;
- ignition may fail...

Leakage is safety critical:

Igniting large amounts of leaked gas may lead to a dangerous explosion.

Requirement: Leakage phases should have a limited duration.
Sketch of the Methodology: Gas Burner Example

- **Requirements**
  - At most 5% of any at least 60s long interval amounts to leakage.

- **Reflective Design**
  - Time intervals with leakage last at most 1s.
  - After each leak, wait 30s before opening valve again.

- **Constructive Design**
  - PLC Automaton:
    - (open valve for 0.5s; ignite; if no flame after 0.1s close valve)

- **Implementation**
  - IEC 61131-3 program
Wireless fire alarm systems are regulated by European Norm EN-54, Part 25.

EN 54-25 states the following requirements:

(i) The loss of the ability of the system to transmit a signal from a component to the central unit is detected in less than 300 seconds and displayed at the central unit within 100 seconds thereafter.

(ii) Out of exactly ten alarms occurring simultaneously, the first should be displayed at the central unit within 10 seconds and all others within 100 seconds.

(iii) There must be no spurious displays of events at the central unit.

(iv) The above requirements must hold as well in the presence of radio interference by other users of the frequency band. Radio interference by other users of the frequency band is simulated by a jamming device specified in the standard.

(Arenis et al., 2016)
Content

- Introduction
  - a software engineering perspective
  - a theoretical computer science perspective

- Real-Time Systems
  - vs. reactive systems
  - vs. hybrid systems
  - safety-critical systems
  - examples

- Lecture Content Overview
  - and non-content

- Formalia
  - times/dates, procedures, exam

- A Formal Model of Real-Time Behaviour
  - state variables / observables
  - evolution / behaviour
Content Overview
Introduction

- Observables and Evolutions
- Duration Calculus (DC)
- Semantical Correctness Proofs
- DC Decidability
- DC Implementables
- PLC-Automata

\[ \text{obs} : \text{Time} \rightarrow \mathcal{D}(\text{obs}) \]

\[ \langle \text{obs}_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_0} \langle \text{obs}_1, \nu_1 \rangle, t_1 \ldots \]

- Timed Automata (TA), Uppaal
- Networks of Timed Automata
- Region/Zone-Abstraction
- TA model-checking
- Extended Timed Automata
- Undecidability Results

- Automatic Verification...
  ...whether a TA satisfies a DC formula, observer-based

- Recent Results:
  - Timed Sequence Diagrams, or Quasi-equal Clocks,
  or Automatic Code Generation, or ...
Tying It All Together

- **abstraction level**
- **formal description language I**
- **semantic integration**
- **automatic verification**
- **formal descr. language II**

**Requirements**
- **Duration Calculus**
- **Constraint Diagrams**

**Designs**
- **PLC-Automata**
- **C code**

**Programs**
- **PLC code**

- **DC**
  - logical semantics
  - operational semantics
- **timed automata**
  - equiv.
- **Live Seq. Charts**

- **satisfied by**
- **equiv.**
**Worst Case Execution Time**

- Over-simplified airbag controller program:
  ```c
  while (true) do
    poll_sensors();
    if (crash) tmr.start(300ms);
    if (tmr.elapsed()) fire := 1;
    update_actuators();
  od
  ```

- The execution of `poll_sensors()` and `update_actuators()` also **takes time**! (And we have to consider it!)

- **Not in lecture:**
  How to determine the WCET of, for instance, C code.
  (A science of its own.)
Scheduling

• A bit less over-simplified airbag controller:

  \[
  \text{Sens} \quad \text{Controller} \quad \text{Act}
  \]

  \[m/s\]

  Bus

• **Not in lecture**: Specialised methods to determine...
  
  • ...whether the bus provides sufficient bandwidth.
  
  • ...whether the Real-Time OS controlling CPU ‘Controller’ schedules the airbag control code in time.
  
  • ...how to distribute tasks over multiple CPUs.
  
  • etc.

(Also a science of its own.)
Content

• Introduction
  • a software engineering perspective
  • a theoretical computer science perspective

• Real-Time Systems
  • vs. reactive systems
  • vs. hybrid systems
  • safety-critical systems
  • examples

• Lecture Content Overview
  • and non-content

• Formalia
  • times/dates, procedures, exam

• A Formal Model of Real-Time Behaviour
  • state variables / observables
  • evolution / behaviour
Formalia
Formalia: Event

- **Lecturer:** Dr. Bernd Westphal
- **Support:** Liridon Musliu

- **Homepage:**
  
  http://swt.informatik.uni-freiburg.de/teaching/WS2017-18/rtsys

- **ILIAS course:** see homepage.

- **Location:**
  - Tuesday, Thursday: here
Formalia: Dates/Times, Break

- **Schedule:**
  - Thursday, week $N$: 14:00–16:00 lecture (exercises $M$ online)
  - Tuesday, week $N+1$: 14:00–16:00 lecture
  - Thursday, week $N+1$: 14:00–16:00 lecture
  - Monday, week $N+2$: 14:00 (exercises $M$ early turn-in)
  - Tuesday, week $N+2$: 14:00 (exercises $M$ late turn-in)
  - Tuesday, week $N+2$: 14:00–16:00 tutorial
  - Thursday, week $N+2$: 14:00–16:00 lecture (exercises $M+1$ online)

  With a prefix of lectures, with public holidays; see homepage for details.

- **Break:**
  - Unless a majority objects now, we’ll have a **10 min. break** in the middle of each event from now on.
Formalia: Lectures

- **Course language:** English
  (slides/writing, presentation, questions/discussions)

- **Presentation:**
  half slides/half on-screen **hand-writing** – for reasons

- **Script/Media:**
  - **slides** without annotations on homepage,
    trying to put them there **before** the lecture
  - **slides** with annotations on homepage, 2-up for printing,
    typically **soon after** the lecture
  - **recordings** in ILIAS course with max. 1 week delay.

- **Interaction:**
  absence often moaned but **it takes two**, so please ask/comment immediately
Schedule/Submission:

- **Recall**: exercises **online** on Thursday before (or soon after) lecture, regular **turn in** on corresponding tutorial day until **14:00 local time**
- should work in groups of **max. 3**, clearly give **names** on submission
- please submit **electronically** by Mail to me (cf. homepage), some \LaTeX\ styles on homepage; paper submissions are tolerated

Didactical aim:

- deal more extensively with notions from lecture (easy)
- explore corner cases or alternatives (medium)
- evaluate/appreciate approaches (difficult)
- additional **difficulty**: imprecise/unclear tasks – by intention

True aim: most complicated rating system ever, namely two ratings

- Good-will (“reasonable solution with knowledge before tutorial”)
- Evil/Exam (“reasonable solution with knowledge after tutorial”)

10% **bonus** for **early** submission.
Formalia: Exam

- Exam Admission:
  50% of the maximum possible non-bonus **good-will points** in total are **sufficient** for admission to exam

- Exam Form: (oral or written) not yet decided
Speaking of grading and examination...

- **Mid-term Evaluation:**
  We will have a *mid-term evaluation*, but we’re *always* interested in comments/hints/proposals concerning form or content.

---

1 that is, students are asked to evaluate lecture, lecturer, and tutor...
Questions:

“online”:
(i) ask immediately or in the break

“offline”:
(i) try to solve yourself
(ii) discuss with colleagues
(iii)

Exercises: contact tutor via ILIAS forum or by mail

Rest: contact lecturer by mail (cf. homepage)
or just drop by: Building 52, Room 00-020
Speaking of questions:

Any questions so far…?
Tell Them What You’ve Told Them...

- **Real-Time Systems**...
  - ... have to compute outputs for certain inputs within *(quantitative!)* time bounds,
  - ... are often *safety critical*,
    then construction requires a high degree of precision.

- (discrete) **reactive system**: without time (other lecture),
- **hybrid system**:
  other continuous components than clocks (other lecture).

- The lecture presents approaches for the **precise development** of real-time systems,
  - logic-based: *Duration Calculus*
  - automata-based: *Timed Automata*

- **Non-content**: (other lectures)
  - Real-time operating systems,
  - Scheduling,
  - Worst-case execution time, etc..
References
References

