Real-Time Systems

Lecture 2: Timed Behaviour

2017-10-19

Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
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.
Content

- A formal model of real-time behaviour
  - state variables (or observables)
  - evolution over time (or behaviour)
  - discrete time vs. continuous (or dense) time

- Timing diagrams

- Formalising requirements
  - with available tools: logic and analysis
  - concise? convenient?

- Correctness of designs wrt. requirements

- Classes of timed properties
  - safety and liveness properties
  - bounded response and duration properties

- An outlook to Duration Calculus
A Formal Model of Real-Time Behaviour
We assume that the real-time systems we consider are characterised by a finite (!) set of state variables (or observables)

\[ \text{obs}_1, \ldots, \text{obs}_n \]

each associated with a set \( D(\text{obs}_i) \), the domain of \( \text{obs}_i \), \( 1 \leq i \leq n \).

Example: gas burner

\( G \)
State Variables (or Observables)

- We assume that the real-time systems we consider are characterised by a finite (!) set of state variables (or observables) $\text{obs}_1, \ldots, \text{obs}_n$

  each associated with a set $\mathcal{D}(\text{obs}_i)$, the domain of $\text{obs}_i$, $1 \leq i \leq n$.

- Example: gas burner

  - $G$, $\mathcal{D}(G) = \{0, 1\}$ – domain value 0 for $G$ models “valve closed” (value 1: “valve open”) (shorthand notation: $G : \{0, 1\}$)
  - $F : \{0, 1\}$ – domain value 0 models “no flame sensed” (value 1: “flame sensed”)
  - $I : \{0, 1\}$ – domain value 0 models “ignition device disabled” (value 1: “ignition enabled”)
  - $H : \{0, 1\}$ – domain value 0 models “no heating request sensed” (value 1: “heating request”)

[Diagram of a gas burner system with labels for gas valve, flame sensor, and ignition device]
Levels of Detail

We can describe a real-time system at various levels of detail by choosing an appropriate domain for each observable.

For example,

- if we need to model a gas valve with different positions (not only “open” and “closed”), we could use
  \[ G : \{0, 1, 2\} \quad (0: \text{“fully closed”, } 1: \text{“half-open”, } 2: \text{“fully open”}) \]
  (Note: domains are never continuous in the lecture, otherwise it’s a hybrid system!)

- if the thermostat (sending heating requests) and the gas burner controller are connected via a bus and exchange messages from \( \text{Msg} \), use
  \[ B : \text{Msg}^* \]
  to model gas burner controller’s receive buffer as a finite sequence of messages from \( \text{Msg} \).

- etc.

- Choice of observables and their domain is a creative (modelling) act.
  A choice is good if it conveniently serves the modelling purpose.
One possible evolution (over time), or: behaviour, of the considered real-time system is represented as a function

$$\pi : \text{Time} \rightarrow \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n).$$

where Time is the time domain (→ in a minute).

If (and only if) observable $obs_i$ has value $d_i \in \mathcal{D}(obs_i)$ at time $t \in \text{Time}$, $1 \leq i \leq n$, we set

$$\pi(t) = (d_1, \ldots, d_n).$$

For convenience, we use

$$obs_i : \text{Time} \rightarrow \mathcal{D}(obs_i)$$

to denote the projection of $\pi$ onto the $i$-th component.
What’s the time?

- There are two main choices for the time domain $\text{Time}$:
  - **discrete time**: $\text{Time} = \mathbb{N}_0$, the set of natural numbers.
  - **continuous or dense time**: $\text{Time} = \mathbb{R}_0^+$, the set of non-negative real numbers.

- Throughout the lecture we shall use the **continuous** time model and consider **discrete** time as a special case.

  Because
  - plant models usually live in **continuous** time,
  - we avoid too early introduction of hardware considerations,

- Interesting view: continuous-time is a well-suited **abstraction** from the discrete-time realms induced by clock-cycles etc.
Example: Gas Burner

An evolution over time of the considered real-time system is represented as function

$$\pi : \text{Time} \rightarrow \mathcal{D}(\text{obs}_1) \times \cdots \times \mathcal{D}(\text{obs}_n)$$

with $$\pi(t) = (d_1, \ldots, d_n)$$ if (and only if) observable $$\text{obs}_i$$ has value $$d_i \in \mathcal{D}(\text{obs}_i)$$ at time $$t \in \text{Time}$$, $$1 \leq i \leq n$$.

For convenience: use $$\text{obs}_i : \text{Time} \rightarrow \mathcal{D}(\text{obs}_i)$$. 
An evolution over time of the considered real-time system is represented as function

\[ \pi : \text{Time} \to \mathcal{D}(\text{obs}_1) \times \cdots \times \mathcal{D}(\text{obs}_n) \]

with \( \pi(t) = (d_1, \ldots, d_n) \) if (and only if) observable \( \text{obs}_i \) has value \( d_i \in \mathcal{D}(\text{obs}_i) \) at time \( t \in \text{Time} \), 1 \( \leq i \leq n \).

For convenience: use \( \text{obs}_i : \text{Time} \to \mathcal{D}(\text{obs}_i) \).
More Examples: Gas Burner Evolutions

- **One ignition failure, success, flame failure.**
- **No heating request, no heating.**
- **Reliable ignition, stable flame.**
- **Spontaneous flame, without request.**
An evolution (of a state variable) can be displayed in form of a timing diagram.

For instance, \( X : D(X) \) for \( X = \{d_1, d_2\} \).

Multiple observables can be combined into a single timing diagram:
Content

- A formal model of real-time behaviour
  - state variables (or observables)
  - evolution over time (or behaviour)
  - discrete time vs. continuous (or dense) time

- Timing diagrams

- Formalising requirements
  - with available tools:
    - logic and analysis
  - concise? convenient?

- Correctness of designs wrt. requirements

- Classes of timed properties
  - safety and liveness properties
  - bounded response and duration properties

- An outlook to Duration Calculus
**Necessary Ingredients**

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.
Formalising Requirements:
A First Approach with Available Tools
Requirements, More Formally

- A **requirement** ‘Req’ is a set of system behaviours (over observables) with the pragmatics that,
  - a **design** or **implementation** is correct wrt. ‘Req’
  - if and only if all observed behaviours
  - lie within the set ‘Req’.

- More formally,
  - \( \text{Req} \subseteq (\text{Time} \rightarrow \mathcal{D}(\text{obs}_1) \times \cdots \times \mathcal{D}(\text{obs}_n)) \)
    - (‘Req’ is the set of allowed evolutions),
  - let
    \[
    \text{Des} \subseteq (\text{Time} \rightarrow \mathcal{D}(\text{obs}_1) \times \cdots \times \mathcal{D}(\text{obs}_n))
    \]
    be the behaviours of a **design** or **implementation**;
  - ‘Des’ is **correct** wrt. ‘Req’ if and only if \( \text{Des} \subseteq \text{Req} \).

- **Inconvenient:**
  - ‘Req’ is usually an **infinite** set – we need ways to describe ‘Req’ conveniently.
A requirement on gas burner controller behaviours could be “do not ignite if the valve is closed”.

Thus, a design ‘Des’ is correct if

- for all evolutions $\pi \in \text{Des},$
  - for all points in time $t \in \text{Time},$
    - it is not the case that $I(t) = 1$ and $G(t) = 0.$
      (Recall: $I(t)$ is the projection of $\pi(t)$ on the $I$-component.)

We can already formalise the above requirement using a logical formula:

$$F := \forall t \in \text{Time} \neg (I(t) = 1 \land G(t) = 0).$$

Then $\text{Req} = \{\pi : \text{Time} \rightarrow D(H) \times D(G) \times D(I) \times D(F) \mid \pi \models F\}.$

In the following, we may identify a requirement and a logical formulae which defines the requirement. We say “requirement $F$”.

IAW: predicate logic formula $F$ serves as concise description of requirement ‘Req’.
Example: Gas Burner

\[ \text{Req} : \iff \forall t \in \text{Time} \bullet \neg (I(t) \land \neg G(t)) \]

\( \pi \in \text{Req?} \)

\[ \forall t \in \{t_1, t_2\} \neg (I(t) \land \neg G(t)) \]

\[ \neg \exists t \in \{t_1, t_2\} (I(t) \land \neg G(t)) \]

\( \pi \in \text{Req?} \)

\( \pi : \)

\( H \):

\( G \):

\( I \):

\( F \):

\( t_1 \)

\( t_2 \)

\( \checkmark \)

\( \checkmark \)
Correctness

- Let ‘Req’ be a requirement,
- ‘Des’ be a design, and
- ‘Impl’ be an implementation.

Recall: each is a set of evolutions, i.e. a subset of \((\text{Time} \rightarrow \times_{i=1}^{n} D(\text{obs}_i))\).

We say

- ‘Des’ is a correct design (wrt. ‘Req’) if and only if

  \[
  \text{Des} \subseteq \text{Req}.
  \]

- ‘Impl’ is a correct implementation (wrt. ‘Des’ (or ‘Req’)) if and only if

  \[
  \text{Impl} \subseteq \text{Des} \quad \text{(or} \quad \text{Impl} \subseteq \text{Req})
  \]

If ‘Req’ and ‘Des’ are described by formulae of first-order predicate logic, proving the design correct amounts to proving validity of

\[\vdash \text{Des} \implies \text{Req}.\]
Content

- A formal model of real-time behaviour
  - state variables (or observables)
  - evolution over time (or behaviour)
  - discrete time vs. continuous (or dense) time

- Timing diagrams

- Formalising requirements
  - with available tools: logic and analysis
  - concise? convenient?

- Correctness of designs wrt. requirements

- Classes of timed properties
  - safety and liveness properties
  - bounded response and duration properties

- An outlook to Duration Calculus
Classes of Timed Properties
Safety Properties

- A safety property states that something bad must never happen [Lamport].

- Example: “do not ignite if the valve is closed”

\[ \text{Req} := \forall t \in \text{Time} \bullet \neg(I(t) \land \neg G(t)). \]

is a safety property.

- In general, a safety property is characterised as a property that can be falsified in bounded time:

  - If a gas burner controller does not satisfy ‘Req’, there is an evolution \( \pi \) and a time \( t \in \text{Time} \) such that

  \[ \neg(I(t) \land \neg G(t)) \]

  does not hold. All later times \( t' > t \) do not make it better.

- But safety is not everything...
Liveness Properties

- The simplest form of a **liveness property** states that **something good eventually does happen.**

- **Example:** “heating requests are finally served”

  \[
  \forall t \in \text{Time} \bullet (H(t) \land \neg F(t)) \implies (\exists t' \geq t \bullet G(t') \land I(t'))
  \]

  is a **liveness property.**

  **Note:** a gas burner controller can guarantee that finally the valve is opened and ignition is enabled – but a **flame cannot be guaranteed.**

- **Note:** liveness properties not **falsified** in finite time.

  - if there is a heating request at time \( t \), and at time \( t' > t \), the controller did not enforce \( G(t) \land I(t) \), there may be a later time \( t'' > t' \) where the formula holds.

- With real-time systems, liveness is too weak...
A bounded response property states that the desired reaction on an input occurs in time interval \([b, e]\).

**Example**: heating requests are served within 3 seconds ±\(\varepsilon\)

\[
\forall t \in \text{Time} \bullet (H(t) \land \neg F(t)) \implies (\exists t' \in [t + 3 s - \varepsilon, t + 3 s + \varepsilon] \bullet G(t') \land I(t'))
\]

is a bounded liveness property.

Here, the interval is \([b, e] = [t + 3 s - \varepsilon, t + 3 s + \varepsilon]\); it depends on the time \(t\) of the heating request.

This property can again be falsified in finite time.

With gas burners, this is still not everything...
It is not so easy to read out

“Heating requests are served within 3 seconds \( \pm \varepsilon \).”

from (lengthy) formula

\[
\forall t \in \text{Time} \bullet (H(t) \land \neg F(t)) \implies (\exists t' \in [t + 3 s - \varepsilon, t + 3 s + \varepsilon] \bullet G(t') \land I(t')).
\]

The Duration Calculus formula

\[
((\lceil H \land \neg F \rceil; \text{true}) \land \lceil \neg (G \land I) \rceil) \implies 3 - \varepsilon \leq \ell \leq 3 + \varepsilon
\]

is more concise (fewer symbols),
and considered easier to read out by some.

\( \rightarrow \) in a week.
A **duration property** states that

- for observation interval $[b, e]$ characterised by a condition $A(b, e)$,
- the **accumulated time**
- in which the system is in a certain critical state characterised by condition $C(t)$
- has an upper bound $u(b, e)$.

$$
\forall b, e \in \text{Time} \Rightarrow (\int_b^e C(t) \, dt) \leq u(b, e)
$$

**Example:** leakage in gas burner,

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

$$
\forall b, e \in \text{Time} \Rightarrow (b \leq e \land (e - b) \geq 60) \Rightarrow (\int_b^e G(t) \land \neg F(t) \, dt) \leq (0.05 \cdot (e - b))
$$

is a **duration property**.
## Duration Properties

- A **duration property** states that
  - for observation interval \([b, e]\) characterised by a condition \(A(b, e)\),
  - the **accumulated time**
  - in which the system is in a certain critical state characterised by condition \(C(t)\)
  - has an upper bound \(u(b, e)\).

\[
\forall b, e \in \text{Time} \bullet A(b, e) \Rightarrow \int_b^e C(t) \, dt \leq u(b, e)
\]

- **Example**: leakage in gas burner,

  “At most 5% of any at least 60s long interval amounts to leakage.”

\[
\forall b, e \in \text{Time} \bullet (b \leq e \land (e - b) \geq 60) \Rightarrow \int_b^e G(t) \wedge \neg F(t) \, dt \leq 0.05 \cdot (e - b)
\]

is a **duration property**.

- This property can again be falsified in finite time.
An Outlook to Duration Calculus (DC)
Duration Calculus is an interval logic.

Formulae are evaluated in an (implicitly given) interval.

Strangest operators:

- almost everywhere – Example: $\lceil G \rceil$
  (Holds in a given interval $[b, e]$ iff the gas valve is open almost everywhere.)

- chop – Example: $(\lceil \neg I \rceil ; \lceil I \rceil ; \lceil \neg I \rceil) \implies \ell \geq 1$
  (Ignition phases last at least one time unit.)

- integral – Example: $\ell \geq 60 \implies \int L \leq \frac{\ell}{20}$
  (At most 5% leakage time within intervals of at least 60 time units.)
Content

- A formal model of real-time behaviour
  - state variables (or observables)
  - evolution over time (or behaviour)
  - discrete time vs. continuous (or dense) time

- Timing diagrams

- Formalising requirements
  - with available tools: logic and analysis
  - concise? convenient?

- Correctness of designs wrt. requirements

- Classes of timed properties
  - safety and liveness properties
  - bounded response and duration properties

- An outlook to Duration Calculus
Tell Them What You’ve Told Them…

- **Evolutions** over state variables
  - are a (simple but powerful) **formal model** of timed behaviour, and
  - can be represented by **timing diagrams**.

- A **requirements specification** denotes a set of **desired** behaviours.

- Example **classes** of properties are
  - **safety**: something bad never happens,
  - **liveness**: something good finally happens,
  - **bounded response**: good things happen with deadlines,
  - **duration**: critical conditions have limited duration.

- Real-time requirements **can be formalised** using just **logic and analysis**.

  But: these specifications easily become **hard to read**.

- Something **more concise** and **more readable** (?): **Duration Calculus** (→ next week)
References
References