Specifying and Reasoning about Generic Real-Time Requirements - *A Case Study*

R. Gotzhein, M. Kronenburg, C. Peper

SFB 501 Report 15/96

Specifying and Reasoning about Generic Real-Time Requirements - *A Case Study*

R. Gotzhein[‡], M. Kronenburg[†], C. Peper[‡]
{gotzhein, kronburg, peper} @ informatik.uni-kl.de

Report 15/96

Sonderforschungsbereich 501

[‡]Computer Networks Group
[†]Efficient Algorithms Group

Computer Science Department
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

Specifying and Reasoning about Generic Real-Time Requirements - *A Case Study*§

R. Gotzhein, M. Kronenburg, C. Peper

Computer Science Department, University of Kaiserslautern
Postfach 3049, 67653 Kaiserslautern, Germany
{gotzhein, kronburg, peper} @ informatik.uni-kl.de

Abstract

A non-trivial real-time requirement obeying a pattern that can be found in various instantiations in the application domain building automation*, and which is therefore called* generic*, is investigated in detail. Starting point is a description of a real-time problem in natural language augmented by a diagram, in a style often found in requirements documents. Step by step, this description is made more precise and finally transformed into a surprisingly concise formal specification, written in real-time temporal logic with customized operators. We reason why this formal specification precisely captures the original description - as far as this is feasible due to the lack of precision of natural language.*

Keywords: requirements, real-time, formal specification, reuse, application, formal reasoning, building automation, real-time temporal logic

1. Introduction

A central goal of the Sonderforschungsbereich (SFB) 501 is to devise methods and techniques for the generic development of large software systems [SFB94]. As an initial application domain, the area of building automation has been chosen as an experimental environment. Building automation systems usually exhibit reactivity, real-time behaviour, scalability, and distribution, and therefore qualify as instances of large software systems. Thus, problems typically occurring in such systems as well as methods and techniques to tackle them can be evaluated in this context.

Among the tasks of a building automation system are the control of temperature, humidity, light, air flow, security, and safety. Depending on the type of a building (e.g. house, research lab, hospital, airport terminal), hundreds or thousands of sensors and actuators can be involved in maintaining required conditions in a building. Our approach in the requirements phase in general was to use the initial informal problem description as delivered by the customer in order to work out a precise and formal specification in temporal logic. These

§ This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Sonderforschungsbereich (SFB) 501 "Development of Large Systems with Generic Methods"

formulas were then translated back to natural language, so that the resulting description could be compared with the initial one and be discussed with the customer. When, after necessary revisions, full agreement with the customer was reached, the temporal formulas formed the basis for subsequent design activities.

Depending on the size of the building and the functionality to be realized, the complexity of the automation system can be substantial. It is therefore vital to state system requirements on a high level of abstraction, to state them precisely, and to state them in such a way that conflicts between requirements can be detected and resolved as early as possible. In this report, we present a case study addressing the first and second aspect, and also address reusability. We start with an excerpt of a description of a real-time problem found in the requirements document of a customer, embodied by project D1 of the SFB 501, stated in natural language augmented by a diagram, as often found in this kind of documents (Section 2). Next, we present our final result, consisting of a surprisingly concise formal specification, written in real-time temporal logic with customized operators (Section 3). In Section 4, we briefly introduce the real-time temporal logic as far as we use it in this case study. Section 5 is then devoted to arguing and reasoning why this formal specification exactly and precisely captures the original description. In Section 6, we elaborate why we consider the real-time requirement as generic. We conclude with an outlook (Section 7).

2. The original problem description

Starting point is the original problem description, stated in natural language augmented by a diagram, which forms part of a larger document:

The comfort temperature shall be maintained during preset comfort times or reached as fast as possible, when the room is occupied for a longer period of time. This state is called "occupied" and is explained in the state chart below. This requirement shall be fulfilled during the heating period, and, if possible, also during the rest of the year. During the states "empty" and "entered" appropriate actions have to be taken.

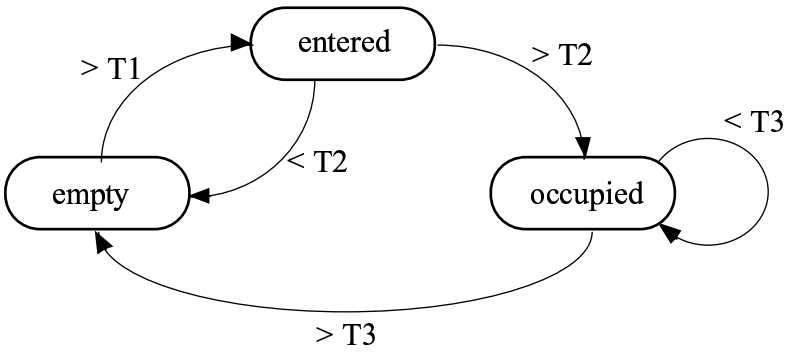

Figure 1: "state chart"

We distinguish three time spans for detecting occupancy. If a person is in a room shorter than T1 minutes, the control system does not try to reach the comfort temperature (state "empty", stand-by or off-time temperature, depending on the time period). If the person occupies the room for a time longer than T1, the system goes into the state "entered" and

starts to reach the comfort temperature. If the person leaves the room before T2 minutes, the system returns to the previous temperature. If the person occupies the room longer than T2, the system enters the state "occupied", where the comfort temperature should be held, until this person leaves the room for at least T3 minutes. Different settings for (T1, T2, T3) may exist for each room, depending on the expected usage (day, night, vacation, etc.).

3. The final formal specification of the real-time requirement

The final formal specification is written in real-time temporal logic with customized operators, and is listed below. It uses several predicates considered atomic in the sense that they are not refined at this stage.

roomEmpty	no person present (not to be confused with the state "empty"!)
comfortTime	time period where presence of persons is expected (e.g. during regular working hours)
comfortTemperature	valves of the heating system are controlled such that the comfort temperature is either reached within certain time bounds and maintained, or approximated, depending on outside temperatures
standByTemperature	as before, w.r.t. stand-by temperature
offTimeTemperature	as before, w.r.t. off-time temperature

Based on *roomEmpty*, the non-atomic (auxiliary) predicate *roomUsed* is defined:

D1. $roomUsed =_{Df} \blacksquare_{\leq T1} \neg\, roomEmpty \vee \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow \rangle \blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty$

Informally, a room is currently in use (written "*roomUsed*") iff one of the following statements holds

- during the past T1 time units, a person has been in the room continuously;
- there has already been a time span of length T1 + T2 such that a person has been in the room continuously, and since this has been the case for the last time, periods where the room was empty were not longer than or equal to T3.

The real-time requirement can then be formalized as follows:

R1. $\Box$ (*roomUsed* $\rightarrow$ *comfortTemperature*)

R2. $\Box$ ($\neg$ *roomUsed* $\wedge$ *comfortTime* $\rightarrow$ *standByTemperature*)

R3. $\Box$ ($\neg$ *roomUsed* $\wedge$ $\neg$ *comfortTime* $\rightarrow$ *offTimeTemperature*)

Informally, the comfort temperature has to be reached and/or maintained whenever the room is used as defined by *roomUsed* (R1). During the remaining time periods, the stand-by or off-time temperatures apply, depending on the setting of *comfortTime* (R2, R3).

4. A real-time temporal logic

The formation rules for the propositional real-time temporal logic we use in this paper are the following (for details, see [Got93] and [KrGoPe96]; for further explanations, see Section 5):

Let Φ be a set of atomic formulas.

i) Let $p \in \Phi$, then p is a formula.

ii) Let φ be a formula, then $\neg \varphi$ is a formula.

iii) Let φ_1, φ_2 be formulas, then $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \leftrightarrow \varphi_2)$ are formulas.

iv) Let φ be a formula, then $\Box\varphi$ is a formula.

v) Let φ be a formula, T be a time value, then $\blacksquare\varphi$, $\blacksquare_{\leq T}\varphi$, $\blacklozenge\varphi$, and $\blacklozenge_{\leq T}\varphi$ are formulas.

vi) Let φ be a formula, then $[\varphi]$ is a formula.

vii) Let φ_1, φ_2 be formulas, then $\langle [\varphi_1] \Rightarrow \rangle \varphi_2$ and $\langle *[\varphi_1] \Rightarrow \rangle \varphi_2$ are formulas.

The propositional operators ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) are interpreted as usual. Basic model is a set of timed state sequences, as defined in [AlHe91]. Note that time is dense. The semantics of the temporal operators is the following (see [KrGoPe96] for a formal semantics):

- $\Box\varphi$ (read "always φ") means that φ is true now and will be true at all points in time in the future.
- $\blacksquare\varphi$ (read "always in the past φ") means that φ is true now and has always been true at all points in time of the current context in the past.
- $\blacksquare_{\leq T}\varphi$ means that φ is true now and has always been true in the preceeding time span T, limited by the current context in the past. I.e., if the current context in the past is shorter than T, φ is only required to be true during that context.
- $\blacklozenge\varphi$ (read "sometimes in the past φ") means that φ is true now or has been true sometime in the current context in the past.
- $\blacklozenge_{\leq T}\varphi$ means that φ is true now or has been true sometime in the preceeding time span T, limited by the current context in the past.
- $[\varphi]$ (read "action of type φ") means that φ has just become true, i.e. φ is true now and has been false in the preceeding state (note that time is dense, however, the state sequence is discrete).
- $\langle [\varphi_1] \Rightarrow \rangle \varphi_2$ (read "φ_2 restricted on the context $[\varphi_1] \Rightarrow$") means that φ_2 holds in the context limited backwards by the last action of type φ_1; the formula is vacuously true if the context does not exist.
- $\langle *[\varphi_1] \Rightarrow \rangle \varphi_2$ is the same as $\langle [\varphi_1] \Rightarrow \rangle \varphi_2$, but is false if the context does not exist.

Note that $\blacksquare\varphi \leftrightarrow \neg \blacklozenge \neg\varphi$, $\blacksquare_{\leq T}\varphi \leftrightarrow \neg \blacklozenge_{\leq T} \neg\varphi$, and $\langle *[\varphi_1] \Rightarrow \rangle \varphi_2 \leftrightarrow \langle [\varphi_1] \Rightarrow \rangle \varphi_2 \wedge \blacklozenge[\varphi_1]$ are valid formulas.

5. Why the formal specification captures the original description

It is not obvious that the formal specification listed in Section 3 indeed captures the original problem description stated in Section 2. Actually, it doesn't, since natural language does not possess a formal semantics, and since the description in Section 2 exhibits some ambiguities. So the goal will be to argue why the formal specification captures the intentions of the informal description.

5.1. A first improvement of the original problem description

From Figure 1, it is not clear when the specified state transitions are actually intended to take place. For instance, the meaning of the transition from "empty" to "entered" seems to be: when in state "empty" for longer than T1, change to "entered". This is different from the intended meaning as explained in the text: when a person is present for longer than T1, and the current state is "empty", change to "entered". Also, it is unclear when the transition back to "empty" will occur. May or must this transition take place any time before expiration of T2, or will it be overruled by a change to "occupied" in certain cases? Again, the actual intention behind the "state chart", which is related to the real-time behaviour of a person entering and leaving the room, may be derived from the text, but is not expressed through the diagram.

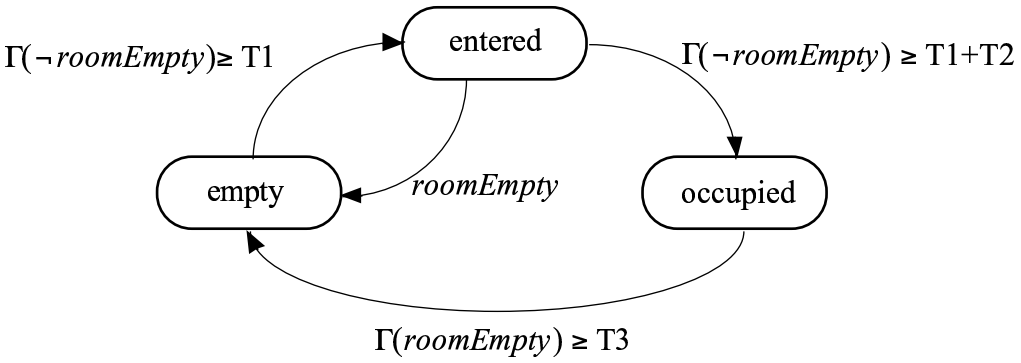

Figure 2: The improved diagram

As a first improvement, we modify the diagram by introducing a predicate *roomEmpty* (not to be confused with the state "empty" in the diagram) and a temporal function Γ denoting the "age" of a formula[1] (see Figure 2). State transitions are now clearly stated in the diagram without additional reference to the text. For instance, a transition from "empty" to "entered" occurs when the predicate *roomEmpty* has been false for at least T1 time units. The transition back to "empty" will then occur as soon as *roomEmpty* becomes true again. In a similar way, the remaining state transitions are to be interpreted.

We claim that the diagram shown in Figure 2 is "consistent" with the part of the original specification that is stated in natural language. This diagram forms the starting point for the derivation of the predicate *roomUsed* as defined in Section 3.

---

[1] As we use the temporal function Γ only in this intermediate stage of description, we will not give a formal definition.

5.2. A second improvement of the original problem description

The descriptions so far introduce and use states "empty", "entered", and "occupied" in order to express how the room temperature has to be controlled. These states can be understood as abstractions of the room history w.r.t. room occupancy. Therefore, they do not form part of the requirement, but are introduced for technical reasons. The major drawback of this description style is that the room history is only expressed *implicitly*, which makes it more difficult to understand the requirement. In this particular case, it would therefore be preferable to express the room history *explicitly*, i.e. directly in terms of room occupancy, without using auxiliary states and a state diagram.

Another drawback of the descriptions so far is that the notion of occupancy in the text significantly differs from the meaning of the states in both diagrams. In state "empty", a person may be present, while in state "occupied", the room may be empty. However, this is just a matter of suitable naming conventions.

As a second improvement of the original problem description, we now define an auxiliary predicate *roomUsed* that will finally allow us to get rid of the auxiliary states and the state diagram, and at the same time to make the description more formal, more intelligible, and more concise. The predicate is defined as follows:

D2. $roomUsed =_{Df}$ entered ∨ occupied

Note that this definition of *roomUsed* is given in terms of the state diagram, which will allow us a graceful transition to its final definition (see Section 3). Informally, when the room is in one of the states "entered" or "occupied", i.e. when *roomUsed* is true, then the valves of the heating system are controlled such that the comfort temperature is either reached within certain time bounds and maintained, or approximated, depending on outside temperatures.

D2 still uses the auxiliary states that we would like to get rid of. Therefore, as a next step, we will now list properties of *roomUsed* stated in terms of the predicate *roomEmpty* introduced in Section 3, without referring to the states in the diagram. We will justify each property by referring to D2 and Figure 2.

P1. $\blacksquare_{\leq T1} \neg\, roomEmpty \rightarrow roomUsed$

When the room has not been empty for at least T1, then *roomUsed* is true.

Reason: If the starting state, i.e. the state at the current time minus T1, was "empty", then there is a change to "entered" after T1. If the starting state was "entered", then the current state is "entered" or "occupied", depending on the room history and the choice of T2. If the starting state was "occupied", it has not changed during T1.

P2. $\langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow \rangle\ \blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty \rightarrow roomUsed$

If there has already been a time span of length T1 + T2 such that the room has continuously not been empty, and if since this has been the case for the last time,

periods where the room was empty were not longer than or equal to T3, then *roomUsed* is true.

Reason: At the beginning of the specified interval, the state is "occupied". Since in the interval, the room is never continuously empty for T3 or longer, the state "occupied" remains unchanged.

P3. $\blacksquare_{\leq T3}\, roomEmpty \rightarrow \neg\, roomUsed$

If the room has continuously been empty for T3, then *roomUsed* is false.

Reason: If the starting state was "empty", it has not changed during T3. If it was "entered", there has been a transition to "empty". Analogously for "occupied".

P4. $(\langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow \rangle \blacklozenge\blacksquare_{\leq T3}\, roomEmpty) \wedge \blacklozenge_{\leq T1}\, roomEmpty \rightarrow \neg\, roomUsed$

If there has already been a time span of length T1 + T2 such that the room has continuously not been empty, and if since this has been the case for the last time, the room has sometime been continuously empty for T3 or longer, and if in addition, the room was empty at least once during T1, then *roomUsed* is false.

Reason: At the beginning of the specified interval, the state is "occupied". Since in the interval, the room has sometime been continuously empty for T3 or longer, the state is changed from "occupied" to "empty". If changes to "entered" have occurred since then, then the additional condition ensures that at the current point in time, "empty" has been reached again.

P5. $\neg\blacklozenge[\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \wedge \blacklozenge_{\leq T1}\, roomEmpty \rightarrow \neg\, roomUsed$

If so far, there has been no time span of length T1 + T2 such that a person has been in the room continuously, and if in addition, the room was empty sometime during the previous time span of length T1, then *roomUsed* is false.

Reason: The first part of the antecedents ensures that the state "occupied" has not been reached so far. If there exists a transition to "entered", then the second part of the antecedents ensures that at the current point in time, "empty" has been reached again.

We have argued that *roomUsed* satisfies properties P1 through P5. Each property has been justified by referring to D2 and the state diagram in Figure 2. Therefore, we define *roomUsed* by the logical conjunction of P1 through P5. This definition certainly is in line with our reasoning so far, but it is at the moment not clear whether it is sound and complete. Also, there may be some redundancy contained in this definition. We will examine these questions in the next section.

5.3. Proving and reducing the second improvement of the original description

We will now formally prove that the definition of *roomUsed* by the conjunction of properties P1 through P5 is sound and complete. *Sound* means that at each point in time, not more than one value of *roomUsed* is specified. *Complete* means that at each point in time, at least one value of *roomUsed* is specified (a definition allowing for more than one value at some point in time is called *partial*). By showing that the definition of *roomUsed* is sound and complete, we therefore prove that at each point in time, exactly one value is specified.

As a by-product of the proof, it will turn out that the definition of *roomUsed* contains substantial redundancy. By removing this redundancy, we obtain an equivalent, concise definition of *roomUsed* (see D1 in Section 3).

Following is a sketch of the proof. The detailed proof is listed in the Appendix.

Theorem 1: The definition of *roomUsed* by the conjunction of P1 through P5 is sound and complete.

Proof sketch: Let AP1 through AP5 be the antecedents of properties P1 through P5, respectively.

a) In order to show that the definition of *roomUsed* is sound, we prove:
(AP1 ∨ AP2) impl ¬ (AP3 ∨ AP4 ∨ AP5).

This means that if *roomUsed* is required to be true, it is not required to be false (and vice versa, by contraposition).

b) In order to show that the definition of *roomUsed* is complete, we prove:
¬ (AP1 ∨ AP2) impl (AP3 ∨ AP4 ∨ AP5).

This means that if *roomUsed* is not required to be true, it is required to be false (and vice versa, by contraposition).

From the proof, it follows immediately:

(AP1 ∨ AP2) iff ¬ (AP3 ∨ AP4 ∨ AP5)

Together with (AP1 ∨ AP2) → *roomUsed* and (AP3 ∨ AP4 ∨ AP5) → ¬ *roomUsed* (due to P1 to P5), we get (AP1 ∨ AP2) ↔ *roomUsed*, which is an equivalent, concise definition of *roomUsed*.

5.4. Final improvements of the original problem description

Based on the results of Section 5.3, we can replace D2, the state diagram in Figure 2, and the corresponding text in natural language (see Section 5.1) by the following improved description:

roomEmpty no person present

$$roomUsed \;=_{Df}\; \blacksquare_{\leq T1} \neg\, roomEmpty \vee \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow \rangle\; \blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty$$

The comfort temperature shall be reached and/or maintained whenever the room is used as defined by *roomUsed*. When *roomUsed* is false, the stand-by or off-time temperature shall be reached, depending on the time period. Different settings for T1, T2, T3 may exist for each room, depending on the expected usage (day, night, vacation, etc.).

Note that the above definition of *roomUsed* coincides with D1 in Section 3. Formalization of the paragraph that is still stated in natural language leads to the final specification listed in Section 3.

6. On the genericity of the real-time requirement

Genericity is an important general (software) engineering concept, applying both to products (requirements, design, implementation) and development processes. Genericity of products is supported by concepts such as compositionality, adaptation, parameterization, and reusability[2]. Genericity of the development process is additionally supported by the concepts of synthesis and generation.

While working on application requirements in the area of building automation, we have found that most of them obey a small number of syntactic patterns ([PeGoKr96]). One of the patterns we found is D3, where p, q, T1, T2, and T3 are parameters to be instantiated in order to obtain a particular requirement. By instantiating p and q, a more specific pattern is obtained. The time parameters T1, T2, and T3 can be replaced by specific values, and/or restricted by adding constraints (note that a replacement by a value is a special case of an added constraint).

D3. $p \;=_{Df}\; \blacksquare_{\leq T1}\, q \vee \langle * [\blacksquare_{\leq T1+T2}\, q] \Rightarrow \rangle\; \blacksquare\blacklozenge_{\leq T3}\, q$

Below, we list some requirements occurring in heating control, air flow control, motion detection, and safety control that are instantiations of D3.

Heating control

By instantiating p and q with *roomUsed* and ¬ *roomEmpty*, respectively, we may obtain D1. The time parameters can either be replaced by specific values, or restricted by additional properties. Once a set of time values is determined, it may be possible to transform the resulting formula into an even more concise specification. We illustrate this for the requirement investigated in this paper:

a) T1 = T2 = T3 = 0: $roomUsed =_{Df} \neg\, roomEmpty$
The room is used iff it is not empty.

[2] These concepts are not orthogonal.

b) T2 = T3 = 0: $roomUsed =_{Df} \blacksquare_{\leq T1} \neg\ roomEmpty$

The room is used iff during the past T1 time units, it has continuously not been empty.

c) T2 = 0: $roomUsed =_{Df} \langle * [\blacksquare_{\leq T1} \neg\ roomEmpty] \Rightarrow \rangle\ \blacksquare\blacklozenge_{\leq T3} \neg\ roomEmpty$

The room is used iff there has already been a time span of length T1 such that the room has continuously not been empty, and since this has been the case for the last time, periods where the room was empty were not longer than or equal to T3.

d) T1 = T2 = 0: $roomUsed =_{Df} \blacklozenge_{\leq T3} \neg\ roomEmpty$

The room is used iff it has not been continuously empty during the past T3 time units.

Note that we have used a second, very simple pattern underlying the definition of R1, R2, and R3, namely $\Box\ (r \rightarrow s)$. The instantiations are obvious here.

Air flow control

By instantiating *p*, *q*, T1, T2, and T3 with *roomUsed*, *lightOn*, 70, 0, and 155 (seconds), respectively, we obtain part of the specification of a ventilation system that is installed in the bathroom of one of the authors (see D4). To detect the presence of a person, a very simple sensor implementation is used: the position of the light switch. Since the bathroom has no windows, a person using it may be expected to switch on and off the light on entering and leaving, respectively. Also, more sophisticated sensors could be used, which would not change the pattern applied in D4.

lightOn	the light in the bathroom is switched on
ventilationOn	the ventilator in the bathroom is switched on

D4. $roomUsed =_{Df} \langle * [\blacksquare_{\leq 70}\ lightOn] \Rightarrow \rangle\ \blacksquare\blacklozenge_{\leq 155}\ lightOn$

R4. $\Box\ (roomUsed \leftrightarrow ventilationOn)$

Motion detection

The specification of a motion detector found in a hardware store was obtained by instantiating *p*, *q*, T1, T2 with *personPresent*, *motion*, 0, and 0, respectively, by restricting T3 as shown in D6, and by adding R5:

motion	motion detected (implemented by an infrared sensor)
darkness	light intensity below a threshold value, adjustable from 2 to 2000 lux
lightOn	a light is switched on to illuminate the area where motion has been detected

D5. $personPresent =_{Df} \blacklozenge_{\leq T3}\ motion$

D6. $T3 \in [10\ \text{seconds}, 15\ \text{minutes}]$

R5. $\Box\ (darkness \wedge personPresent \leftrightarrow lightOn)$

Safety control

By instantiating *p* and *q* with *hazardousCondition* and *hazardousSituation*, respectively, and by adding R6, we obtain a safety control requirement (see also [Got+96]). As already discussed, the time parameters can either be replaced by specific values, or restricted by additional properties.

hazardousSituation	holds in case of heavy winds, heavy rainfall, or danger of burglary (e.g., during darkness)
upperSashClosed	upper sash closed

D7. $hazardousCondition =_{Df}$
$\blacksquare_{\leq T1}\ hazardousSituation \vee$
$\langle * [\blacksquare_{\leq T1+T2}\ hazardousSituation] \Rightarrow \rangle\ \blacksquare\blacklozenge_{\leq T3}\ hazardousSituation$

R6. $\Box\ (hazardousCondition \Rightarrow_{\leq T}\ upperSashClosed)$

R6 uses a tailored operator of the real-time temporal logic introduced in [KrGoPe96]. Informally, R6 states that each time *hazardousCondition* holds continuously for at least T time units, the upper sash must be closed within this time span and must then remain closed as long as *hazardousCondition* holds. R6 is an instantiation of another pattern occurring quite frequently in requirements on building automation systems, namely $\Box\ (r \Rightarrow_{\leq T}\ s)$.

Screen saver

Requirements matching the same pattern can also be found outside the area of building automation. For instance, by instantiating *p*, *q*, T1, T2, and T3 with *terminalUsed*, *terminalTouched*, 0, 0, and 300 (seconds), respectively, and by adding R7, we obtain part of the specification of the authors' screen saver as it is currently parameterized. The predicate *terminalTouched* is evaluated by observing key and mouse activity. For security reasons, time periods with expected usage of the terminal – similar to of heating control – may be distinguished in addition.

terminalTouched	key or mouse activity
screenOn	terminal screen active

D8. $terminalUsed =_{Df}\ \blacklozenge_{\leq T3}\ terminalTouched$

R7. $\Box\ (terminalUsed \leftrightarrow screenOn)$

7. Outlook

Starting with a description of a real-time problem in natural language augmented by a diagram, we have developed a precise and concise formal specification in a step-by-step fashion. We

have reasoned why this formal specification precisely captures the original description - as far as this has been feasible due to the lack of precision of natural language, and due to the ambiguities of the diagram.

The definition of *roomUsed* follows a pattern that can be found in various instantiations in the area of building automation. For instance, it can be used to control air flow in a bathroom that has no windows. In the home of one of the authors, the ventilation system is active iff *roomUsed* is true, with settings T1 = 70, T2 =0, and T3 = 155 (seconds). Further requirements making use of the above pattern exist in all areas of building automation including the control of light and security, and also outside this domain. Thus, the original investment in developing a formal and rigorous specification pays off.

Another benefit of the reuse of the above pattern is that generic solutions that can be adapted to particular machine environments can be developed. In project B4 of the SFB 501, we are currently developing a distributed solution for another real-time pattern, based on message passing. We expect that by a generic layout of systems and the resulting reuse, customized solutions can be developed with significantly less effort.

Acknowledgements

We are grateful for the stimulating environment provided by the SFB 501 and the discussions with other project members, in particular with the project leader of D1, Prof. Gerhard Zimmermann.

References

[AlHe91] R. Alur, and T. A. Henzinger: Logics and Models of Real Time: A Survey, in: J. W. de Bakker, C. Huizing, W. P. de Roever, G. Rozenberg (eds.), Real-Time: Theory and Practice, LNCS 600

[Got93] R. Gotzhein: Open Distributed Systems - On Concepts, Methods and Design from a Logical Point of View, Vieweg Wiesbaden, 1993

[Got+96] R. Gotzhein, B. Geppert, C. Peper, and F. Rößler: Generic Layout of Communication Subsystems - A Case Study, SFB 501 Report 14/96, University of Kaiserslautern, Germany, 1996

[KrGoPe96] M. Kronenburg, R. Gotzhein, and C. Peper: A Tailored Real-Time Temporal Logic for Specifying Requirements of Building Automation Systems, SFB 501 Report 16/96, University of Kaiserslautern, Germany, 1996

[PeGoKr96] C. Peper, R. Gotzhein, and M. Kronenburg: A Generic Approach to the Formal Specification of Real-Time Requirements of Building Automation Systems, SFB 501 Report 1/97, University of Kaiserslautern, Germany, 1997

[SFB94] Development of Large Systems with Generic Methods, Project Proposal, SFB 501, Computer Science Department, University of Kaiserslautern, 1994

Appendix

Lemma 1: Let AP1 through AP5 be the antecedents of properties P1 through P5, respectively. Then (AP1 ∨ AP2) iff ¬ (AP4 ∨ AP5) holds.

Proof:

(AP1 ∨ AP2)

iff $(\blacksquare_{\leq T1} \neg\, roomEmpty \vee \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle \blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty)$

iff $\neg (\neg\, \blacksquare_{\leq T1} \neg\, roomEmpty \wedge \neg \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle \blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty)$

iff $\neg (\blacklozenge_{\leq T1} roomEmpty \wedge (\neg\blacklozenge[\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \vee \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle \neg\blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty))$

iff $\neg ((\blacklozenge_{\leq T1} roomEmpty \wedge \neg\blacklozenge[\blacksquare_{\leq T1+T2} \neg\, roomEmpty]) \vee (\blacklozenge_{\leq T1} roomEmpty \wedge \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle \neg\blacksquare\blacklozenge_{\leq T3} \neg\, roomEmpty))$

iff $\neg ((\blacklozenge_{\leq T1} roomEmpty \wedge \neg\blacklozenge[\blacksquare_{\leq T1+T2} \neg\, roomEmpty]) \vee (\blacklozenge_{\leq T1} roomEmpty \wedge \langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle \blacklozenge\blacksquare_{\leq T3} roomEmpty))$

iff ¬ (AP4 ∨ AP5)

q.e.d.

Lemma 2: Let AP3 through AP5 be the antecedents of properties P3 through P5, respectively. Then (AP3 → (AP4 ∨ AP5)) holds.

Proof:

If AP3 ($\blacksquare_{\leq T3} roomEmpty$) does not hold, the proposition is trivially satisfied. Therefore, it remains to show that the proposition also holds in case AP3 is true. To prove this, we distinguish two cases:

Case a) $\blacklozenge[\blacksquare_{\leq T1+T2} \neg\, roomEmpty]$ is valid

It follows that the interval specified by $\langle[\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle$ exists. Because of AP3, it has a length of at least T3. This means that the first part of AP4, i.e. $\langle * [\blacksquare_{\leq T1+T2} \neg\, roomEmpty] \Rightarrow\rangle \blacklozenge\blacksquare_{\leq T3} roomEmpty$, holds. Furthermore, the second part of AP4, i.e. $\blacklozenge_{\leq T1} roomEmpty$, follows directly from AP3.

Case b) $\neg \blacklozenge [\blacksquare_{\leq T1+T2} \neg$ *roomEmpty*$]$ is valid

This is equivalent to the first part of AP5. In addition, the second part of AP5, i.e. $\blacklozenge_{\leq T1}$ *roomEmpty*, is immediately implied by AP3.

q.e.d.

Proof of Theorem 1: Let AP1 through AP5 be the antecedents of properties P1 through P5, respectively. We prove a) and b) by showing (AP1 ∨ AP2) iff ¬ (AP3 ∨ AP4 ∨ AP5):

(AP1 ∨ AP2)

iff ¬ (AP4 ∨ AP5) Lemma 1

iff ¬ AP4 ∧ ¬ AP5 ∧ (AP3 → (AP4 ∨ AP5)) Lemma 2

iff ¬ AP4 ∧ ¬ AP5 ∧ (¬ AP3 ∨ AP4 ∨ AP5)

iff ¬ AP4 ∧ ¬ AP5 ∧ ¬ AP3

iff ¬ (AP3 ∨ AP4 ∨ AP5)

q.e.d.