A contract-based method to specify stimulus-response requirements

[1]*A. Naumchev <a.naumchev@innopolis.ru>*
[1]*M. Mazzara <m.mazzara@innopolis.ru>*
[1, 2, 3]*B. Meyer <Bertrand.Meyer@inf.ethz.ch>*
[3]*J.-M. Bruel <bruel@irit.fr>*
[3]*F. Galinier <galinier@irit.fr>*
[3]*S. Ebersold <ebersold@irit.fr>*
[1]*Innopolis University,*
1 Universitetskaya st., Innopolis, 420500, Russian Federation.
[2]*Politecnico di Milano,*
Piazza Leonardo da Vinci, 32, 20133 Milano MI, Italy.
[3]*Paul Sabatier University,*
118 Route de Narbonne, 31062 Toulouse, France.

Abstract. The verification of many practical systems – in particular, embedded systems – involves processes executing over time, for which it is common to use models based on temporal logic, in either its linear (LTL) or branching (CTL). Some of today's most advanced automatic program verifiers, however, rely on non-temporal theories, particularly Hoare-style logic. Can we still take advantage of this sophisticated verification technology for more challenging systems? As a step towards a positive answer, we have defined a translation scheme from temporal specifications to contract-equipped object-oriented programs, expressed in Eiffel and hence open for processing by the AutoProof program prover. We have applied this scheme to a published CTL model of a widely used realistic example, the "landing gear" system which has been the subject of numerous competing specifications. An attempt to verify the result in AutoProof failed to prove one temporal property, which on further inspection seemed to be wrong in the original published model, even though the published work claimed to have verified an Abstract State Machine implementation of that model. Correcting the CTL specification to reflect the apparent informal attempt, re-translating again to contracted Eiffel and re-running the verification leads to success. The LTL-to-contracted-Eiffel process is still ad hoc, and tailored to generate the kind of scheme that the target verification tool (AutoProof) can handle best, rather than the simplest or most elegant scheme. Even with this limitation, the results highlight the need for rigor in the verification process, and (on the positive side) demonstrate that the highly advanced mechanized proof technology developed over several decades for the verification of traditional programs also has the potential of handling the demanding needs of embedded systems and other demanding contemporary developments.

Keywords: seamless requirements; design by contract; autoproof; eiffel; landing gear system

DOI: 10.15514/ISPRAS-2017-29(4)-3

For citation: Naumchev A., Mazzara M., Meyer B., Bruel J.-M., Galinier F., Ebersold S. A contract-based method to specify stimulus-response requirements. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 39-54. DOI: 10.15514/ISPRAS-2017-29(4)-3

1. Overview and main results

The present article describes a technique for specification and verification of stimulus-response requirements using a general-purpose programming language (Eiffel) and a program prover (AutoProof [1]) based on the principles of Design by Contract [2].

Real-time, or reactive, systems are often run by a software controller that repeatedly executes one and the same routine and it is specified to take actions at specific time intervals or according to external stimuli [3]. This architecture is reasonable when the software has to react timely to non-deterministic changes in the environment. In this case the program should react to the external stimuli in small steps, so that in the event of a new change it responds timely.

Computation tree logics (CTL) [4] represent a frequent choice when it comes to capturing stimulus-response requirements. Although it may be easier to reason about requirements using declarative logic like CTL, the reasoning may be of little value for the software developer who will implement the requirements. Mainstream programming languages are all imperative, and the translation between declarative requirements and imperative programs is semi-formal.

Requirements have to be of imperative nature from the beginning. This would bridge the gap in how customers and developers understand them. For a software developer it is preferable to reason about the future program without switching to an additional formalism, notation and tools not connected to the original programming language and the IDE.

The present article describes a technique to achieve this goal, in particular:

- Introduces the Landing Gear System (LGS) case study and the LGS baseline requirements (Section 2).
- Generalizes the LGS baseline requirements, maps them to a well-established taxonomy, and complements the taxonomy (Section 3).
- Provides a general scheme for capturing semantics of the stimulus-response requirements in the form of imperative program routines with assertions (Section 4).
- Exercises utility of the approach by applying it to an Abstract State Machine (ASM) specification of the Landing Gear System case study (Section 5).
- Concludes the possibility of statically checking a sequential imperative program directly against a stimulus-response requirement whose semantics

is expressed in the same programming language through conditionals, loops, and assertions (Section 7).

Application of the technique leads to discovery of an error in the published model of the LGS ASM [5]. The error is not present in the specification the authors have actually used for proving the properties, but the error has found its way into the publication.

2. The landing gear system

Landing Gear System was proposed as a benchmark for techniques and tools dedicated to the verification of behavioral properties of systems [6]. It physically consists of the landing set, a gear box that stores the gear in the retracted position, and a door attached to the box (Figure 1). The door and the gear are actuated independently by a digital controller. The controller reacts to changes in position of a handle in the cockpit by initiating either gear extension or retraction process. The task is to program the controller so that it correctly aligns in time the events of changing the handle's position and sending commands to the door and the gear actuators.

3. Stimulus-response requirements

The LGS case study defines a number of requirements, including several for the normal mode of operation (Figure 2). The requirements communicate a common meaning of the form: *If* ***stimulus*** *holds, then* ***response*** *will eventually hold in the future*. For requirement $R_{11}bis$,
$stimulus \Leftrightarrow "The\ operation\ mode\ is\ normal\ and\ the\ handle\ is\ DOWN"$ and
$response \Leftrightarrow (stimulus \Rightarrow "The\ gear\ is\ down\ and\ the\ door\ is\ closed")$.
The implication in the definition of *response* reflects the "and stays DOWN" part of the original requirement. In addition to that, requirements R_{21} and R_{22} communicate something else:

- Once *response* holds in the presence of *stimulus*, and *stimulus* holds forever, *response* will hold forever.

3.1 Temporal interpretation of the requirements

The authors of the LGS ASM specification start with a ground model that satisfies a subset of requirements, and then refine the model to satisfy more requirements. The present article focuses on their ground model and the corresponding baseline requirements it covers (Figure 2). The work expresses the baseline requirements as CTL properties. The CTL interpretation assigns precise meanings to the requirements by assuming small-step execution semantics of ASM's. In particular, for requirements $R_{11}bis$ and $R_{12}bis$ "the future" means "after a finite number of execution steps", while for R_{21} and R_{22} "the future" means "after one execution step".

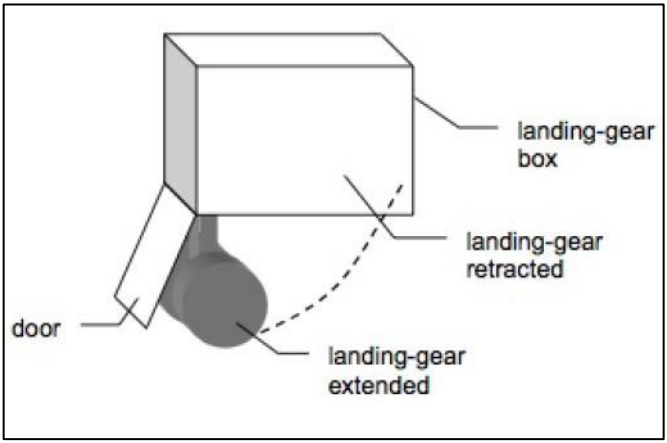

Fig. 1. Landing set (source: [6]).

($R_{11}bis$)	When the command line is working (normal mode), if the landing gear command handle has been pushed DOWN and stays DOWN, then eventually the gears will be locked down and the doors will be seen closed.
($R_{12}bis$)	When the command line is working (normal mode), if the landing gear command handle has been pushed UP and stays UP, then eventually the gears will be locked retracted and the doors will be seen closed.
(R_{21})	When the command line is working (normal mode), if the landing gear command handle remains in the DOWN position, then retraction sequence is not observed.
(R_{22})	When the command line is working (normal mode), if the landing gear command handle remains in the UP position, then outgoing sequence is not observed.

Fig. 2. Baseline LGS requirements.

The finite number of steps in $R_{11}bis$ and $R_{12}bis$ may be unacceptably large though for a system like an LGS of an aircraft. In particular, flights have some expected durations, and the gears have to react to commands in some limited time frame as well. The following two major categories of stimulus-response requirements stem from the speculations above:

- *If stimulus holds, then response will hold in not more than k execution steps.*

Requirements of this form are also called maximal distance requirements [7].

- *If stimulus holds, then response will hold in exactly k execution steps.* Requirements of this form are also called exact distance, or delay requirements.

These two categories are not enough though for capturing stimulus-response requirements. For example, if according to $R_{11}bis$ the gears are locked down and the doors seen closed as the result of the handle staying down, we want this state to be stable if the handle stays down. This leads us to stimulusresponse requirements of the following form:

- *If response holds under stimulus, it will still hold after one execution step in the presence of that stimulus.*
  Let us call such requirements response stability requirements.

It makes sense to complement requirements ($R_{11}bis$) and ($R_{12}bis$) with the corresponding response stability requirements (Figure 3): not only do we want the LGS to respond to a change in the handle's position, but we also want it to maintain the response if the position does not change.

($R_{11}rs$)	If the gears are locked extended and the doors are closed when the landing gear command handle is DOWN, this state will still hold if the handle stays DOWN.
($R_{12}rs$)	If the gears are locked retracted and the doors are closed when the landing gear command handle is UP, this state will still hold if the handle stays UP.

Fig. 3. LGS response stability requirements.

4. Translation of stimulus-response requirements

Assuming the presence of an infinite loop **from until False loop** main **end** that runs a reactive system, a temporal stimulus-response requirement (Section 3.1) takes the form of a routine with an assertion (**check end** construct in Eiffel). The authors draw this idea from the notion of a specification driver [8] - a contracted routine that forms a proof obligation in Hoare logic. AutoProof is a prover of Eiffel programs that makes it possible to statically check the assertions.

```
response_holds_within_k_steps
-- If stimulus holds, response will hold within k steps.
  local
    steps: NATURAL
  do
```

```
    if (stimulus) then
      from
        steps := 0
      until
        response or (steps = k)
      loop
        main
        steps := steps + 1
      end
      check response end
    end
  end
```

Fig. 4. Representation of a maximal distance requirement. Regardless of the actual reason for the loop to terminate, the response has to hold if the stimulus held at the entry to the loop.

```
response_holds_in_k_steps
-- If stimulus holds, response will hold in k steps.
  local
    steps: NATURAL
  do
    if (stimulus) then
      from
        steps := 0
      until
        response or (steps=k)
      loop
        main
        steps := steps + 1
      end
      check (response and (steps = k)) end
    end
  end
```

Fig. 5. Representation of an exact distance requirement. Both of the loop exit conditions have to hold for the first time simultaneously if the stimulus held at the entry to the loop.

4.1 Maximal distance

In the representation of a maximal distance requirement (Figure 4) the "**if** stimulus **then**" clause captures the presence of the stimulus before the up-to-*k*-length execution fragment, and the "**check** response **end**" assertion expresses the need for the response upon completion of the subexecution. The sub-execution may complete for two possible reasons: either occurrence of the response or consumption of all of the available *k* steps. In the both cases the response has to hold.

4.2 Exact distance

Representation of an exact distance requirement (Figure 5) is very similar to that one of a maximal distance, with the "**check** (response **and** (steps = k)) **end**" assertion that makes the difference. Regardless of whether the loop terminates because of "response **or** steps = k", the both have to hold upon the termination.

4.3 Response stability

Representation of a response stability requirement (Figure 6) says: whenever response holds under stimulus in a state, it will still hold in the presence of the same stimulus in the next state.

```
response_is_stable_under_stimulus
-- response keeps holding under stimulus.
  do
    if (stimulus and response) then
      main
      check (stimulus implies response) end
    end
  end
```

Fig. 6. Representation of a response stability requirement. If response holds under stimulus in some state, the response should hold in the next state in the presence of the same stimulus.

5. Applying the translation scheme to the landing gear example

The article exercises the approach on the LGS ASM specification, which is operational by the definition and thus is a subject for translation into an imperative program. For this reason the present section starts with explanation of the rules according to which the authors converted the original specification into an Eiffel program.

5.1 Translation of ASM specifications

An ASM specification is a collection of rules taking one of the following three forms [9]: assignment (Section V-A1), do-in-parallel (Section V-A2), and

conditional (Section V-A3). If we have general rules for translating these operators into Eiffel then we will be able to translate an arbitrary ASM into an Eiffel program.

An ASM *assignment* looks as follows:

$$f\left(t_1, \dots, t_j\right) := t_0 \qquad (1)$$

The semantics is: update the current content of location $\lambda = (f,(a_1,..,a_j))$, where a_i are values referenced by t_i, with the value referenced by t_0.

In Eiffel locations are represented with class attributes, so an ASM's location update corresponds in Eiffel to an attribute assignment.

An ASM *do-in-parallel* operation can apply several rules simultaneously in one step:

$$R_1 || \dots || R_k \qquad (2)$$

In order to emulate a parallel assignment in a synchronous setting, one needs to assign first to fresh variables and then assign their values to the original ones. For example, an ASM do-in-parallel statement

$$a, b := \max(a - b, b), \min(a - b, b) \qquad (3)$$

in Eiffel would look like:

```
local
  a_intermediate, b_intermediate: INTEGER
do
  a_intermediate := max (a−b, b)
  b_intermediate := min (a−b, b)
  a := a_intermediate
  b := b_intermediate
end
```

An attempt to update in parallel identical locations in an ASM corresponds semantically to a crash. The translation scheme not only preserves but strengthens this semantics: an Eiffel program with two local variables declared with identical names will not compile.

Conditional: An ASM conditional **if** t **then** R1 **else** R2 carries the same meaning as in Eiffel, so the translation is straightforward.

5.2 An error in the ground model

Translation of the original LGS ASM specification into Eiffel is publicly available in a GitHub repository [10] The error is not handling the situation when the door is closing and the handle is pushed down, in which case the ground model will not meet requirement ($R_{11}bis$). To catch this error with the SVR method one needs first to introduce it back by commenting out two lines in the "open_door" routine of the Eiffel translation:

```
open_door
 do
  inspect door_status
  when closed_position then
    door_status := opening_state
  -- when closing_state then
  --   door_status := opening_state
```

and then submit routine r11_bis to verification with AutoProof; the verification will fail. We have contacted an author of the article that contains the erroneous ASM specification, and he admitted the presence of the error.

5.3 Requirements

The two classes include the translations of the baseline requirements plus the response stability requirements introduced in the present article. We do not discuss all of them here: requirements ($R_{11}bis$) and ($R_{12}bis$), (R_{21}) and (R_{22}), ($R_{11}rs$) and ($R_{12}rs$) are pairwise similar, which is why we prefer to pick one from each pair.

Translation of requirement r11_bis (Figure 7) is an application of the response_holds_within_k_steps pattern (Figure 4), where:

- stimulus equates to:
  is_normal_mode **and** (handle_status = is_handle_down)
- response equates to:
  (**not** (is_normal_mode **and** (handle_status = is_handle_down))) **or**
  ((gear_status = is_gear_extended) **and** (door_status = is_door_closed))

The idea behind the response is that there may be two reasons for the gear not to extend and the door not to close:

- An abnormal situation that leads to quitting the normal mode.
- The crew changes their mind and pushes the handle up.

```
r11_bis
-- If (is_normal_mode and (handle_status = is_handle_down)) hold and remain,
-- ((gear_status = is_gear_extended) and (door_status = is_door_closed)) will hold within 10 steps.
```

```
 local
  steps: NATURAL
 do
  if (is_normal_mode and (handle_status = is_handle_down)) then
   from
    steps := 0
   until
    (not (is_normal_mode and (handle_status = is_handle_down)))
or ((gear_status = is_gear_extended) and
    (door_status = is_door_closed)) or (steps = 10)
   loop
    main
    steps := steps + 1
   end
   check (not (is_normal_mode and (handle_status =
is_handle_down))) or
    ((gear_status = is_gear_extended) and (door_status =
is_door_closed)) end
  end
 end
```

Fig. 7. Translation of the "r11 bis" requirement.

r21

```
-- If (is_normal_mode and (handle_status = is_handle_up)) holds and
remains,
-- (gear_status= is_gear_extending) will hold within 1 step.
 local
  steps: NATURAL
 do
  if (is_normal_mode and (handle_status = is_handle_up)) then
   from
    steps := 0
   until
    (not (is_normal_mode and (handle_status = is_handle_up))) or
    (gear_status = is_gear_extending) or
    (steps = 1)
   loop
    main
    steps := steps + 1
```

```
    end
    check (not (is_normal_mode and (handle_status = is_handle_up))) or
      (gear_status = is_gear_extending) end
  end
end
```

Fig. 8. Translation of the "r21" requirement.

```
r11_rs
-- ((gear_status = is_gear_extended) and (door_status = is_door_closed)) keeps holding under
-- (is_normal_mode and (handle_status = is_handle_down))
 do
  if ((is_normal_mode and (handle_status = is_handle_down)) and
     ((gear_status = is_gear_extended) and (door_status = is_door_closed))) then
    main
    check ((is_normal_mode and (handle_status = is_handle_down)) implies
     ((gear_status = is_gear_extended) and (door_status = is_door_closed))) end
  end
 end
```

Fig. 9. Translation of the "r11 rs" requirement.

6. Related work

Modeling of real-time computation and related requirements is a well-investigated matter [12]. Representation of real-time requirements, expressed in general or specific form, is a challenging task that has been attacked by the use of several formalisms both in sequential and concurrent settings, and in a broad set of application domains. The difficulty (or impossibility) to fully represents general real-time requirements other than in natural language, or making use of excessively complicated formalisms (unsuitable for software developers), has been recognized.

In [13] the domain of real-time reconfiguration of system is discussed, emphasizing the necessity of adequate formalisms. The problem of modeling real time in the context of services orchestration in Business Process, and in presence of abnormal behavior has been examined in [14] and [15] by means, respectively, of process algebra and temporal logic. Modeling of protocols also requires real-time aspects to be represented [16]. Event-B has also been used as a vector for real-time extension [17] in order to handle embedded systems requirements.

In all these studies, the necessity emerged of focusing on specific typology of requirements using ad-hoc formalisms and techniques, and making use of abstractions. The notion of "real-time" is often abstracted as *number of steps*, a metric commonly used. In this paper we follow the same approach, inheriting both strength (simplicity of the model and effectiveness for applicative purposes) and limitations (temporal logic and time automata themselves miss to capture a precise notion of *real-time*).

7. Conclusions and future work

Software developers reason in an *imperative/operational* manner. This claim is supported both by anecdotal experience and by empirical evidence [18]. Requirements expressed in imperative/operational fashion would therefore results of easier comprehensions for developers and would simplify the process of negotiation behind requirements elicitation. In the method described in this paper, requirements are expressed in a formalism (or language) that seamlessly stay the same along the whole process, without the need of switching between different instruments or mental paradigms. At the same time, the linguistic tool used to define them also allows for automatic verification of correctness.

The meaning of correctness here remains subject to the assumption that requirements engineers and stakeholders agree on a list of desiderata that is indeed the intended one. Assuming a non-faulty process of intention transferring (and this assumption is common to any other approach too), requirements are now more easily manageable by software engineerings all the way from elicitation to verification.

The result of elicitation process is a set of requirements in natural language. The full realization of the presented method would imply an automatic (or semi-automatic) translation from natural language into a structured representation that, although completely intuitive for software developers, it is possibly not easy to manage for average stakeholders. The first part of this process, i.e., the translation from natural language into the current representation (and back) is under development. A tool automatically translates semi-structured natural language into the Hoare-triple-based representation [19], allowing also the opposite direction, i.e. back to natural language [20], so that software engineers would be able to negotiate back requirements with stakeholders using a format they would comprehend. The role of the requirement engineers would then consist in concluding the elicitation phase with a set of requirements in semi-structured natural language, which the tool would be able to process in an entirely automatic manner.

This paper supports the idea of seamless development describing a method supported by a formalism that stay the same along the whole process, from requirements to deployment. Alternative approaches have also been experimented which make use of formalism-based toolkits, where ad hoc notations are adopted for each development phase [21].

References

[1]. J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, "Autoproof: Auto-active functional verification of object-oriented programs," *arXiv preprint arXiv:1501.03063*, 2015.

[2]. B. Meyer, *Touch of Class: learning to program well with objects and contracts*. Springer, 2009.

[3]. I. J. Hayes, M. A. Jackson, and C. B. Jones, *Determining the Specification of a Control System from That of Its Environment*, pp. 154–169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[4]. E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," *Logics of programs*, pp. 52–71, 1982.

[5]. P. Arcaini, A. Gargantini, and E. Riccobene, "Modeling and analyzing using asms: the landing gear system case study," in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 36–51, Springer, 2014.

[6]. F. Boniol and V. Wiels, "The landing gear system case study," in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 1–18, Springer, 2014.

[7]. R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255– 299, 1990.

[8]. A. Naumchev and B. Meyer, "Complete contracts through specification drivers," in *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 160–167, July 2016.

[9]. Y. Gurevich, "Sequential abstract-state machines capture sequential algorithms," *ACM Transactions on Computational Logic (TOCL)*, vol. 1, no. 1, pp. 77–111, 2000.

[10]. A. Naumchev, "Lgs asm ground model in eiffel" https://github.com/anaumchev/lgs_ground_model, 2017.

[11]. N. Polikarpova, J. Tschannen, C. A. Furia, and B. Meyer, "Flexible invariants through semantic collaboration," in *FM 2014: Formal Methods*, pp. 514–530, Springer, 2014.

[12]. H. Yamada, "Real-time computation and recursive functions not real-time computable," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 753–760, Dec 1962.

[13]. M. Mazzara and A. Bhattacharyya, "On modelling and analysis of dynamic reconfiguration of dependable real time systems," in *Proceedings of the 2010 Third International Conference on Dependability*, DEPEND '10, (Washington, DC, USA), pp. 173–181, IEEE Computer Society, 2010.

[14]. M. Mazzara, "Timing issues in web services composition," in *Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings*, pp. 287–302, 2005.

[15]. L. Ferrucci, M. M. Bersani, and M. Mazzara, "An LTL semantics of business workflows with recovery," in ICSOFTPT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 2931 August, 2014, pp. 29–40, 2014.

[16]. M. Berger and K. Honda, "The two-phase commitment protocol in an extended pi-calculus," Electr. Notes Theor. Comput. Sci., vol. 39, no. 1, pp. 21–46, 2000.

[17]. A. Iliasov, A. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala, "Augmenting event-b modelling with real time verification," in Proceedings of the First International

Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, FormSERA ’12, 2012.

[18]. D. Fahland, D. Lubke, J. Mendling, H. Reijers, B. Weber,¨ M. Weidlich, and S. Zugal, Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. Springer Berlin Heidelberg, 2009.

[19]. A. Bormotova, “Translation of natural language into hoare triples.” https://github.com/An-Dole/ Semantic-mapping.

[20]. V. Skukov, “Translation of hoare triples into natural language.” https://github.com/flosca/hybrid.

[21]. R. Gmehlich, K. Grau, F. Loesch, A. Iliasov, M. Jackson, and M. Mazzara, “Towards a formalism-based toolkit for automotive applications,” in 1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013, San Francisco, CA, USA, May 25, 2013, pp. 36–42, 2013.

Контрактный метод спецификации реактивных требований

[1]*А. Наумчев <a.naumchev@innopolis.ru>*
[1]*М. Маццара <m.mazzara@innopolis.ru>*
[1, 2, 3]*Б. Мейер <Bertrand.Meyer@inf.ethz.ch>*
[3]*Ж.-М. Брюэль <bruel@irit.fr>*
[3]*Ф. Галинье <galinier@irit.fr>*
[3]*С. Эберсоль <ebersold@irit.fr>*
[1]*Университет Иннополис,*
420500, Российская Федерация, г. Иннополис, ул. Университетская, д. 1.
[2]*Миланский технический университет,*
20133, Италия, г. Милан, Piazza Leonardo da Vinci, 32.
[3]*Университет Тулузы,*
31062, Франция, г. Тулуза, Route de Narbonne, 118.

Аннотация. Верификация многих прикладных систем – в частности, встроенных, - включает в себя процессы, исполняющиеся во времени, для моделирования которых обычно используется временная логика, линейная (LTL) или ветвящаяся (CTL). Наиболее развитые автоматические доказатели программ, однако, основаны на невременных теориях: например, на логике Хоара. Возможно ли все же применение этой развитой технологии верификации к более сложным системам? В качестве шага на пути к положительному ответу, мы разработали схему перевода подмножества LTL спецификаций в объектно-ориентированные программы с контрактами на языке Eiffel, которые являются естественными целями для доказателя программ AutoProof. Мы применили эту схему к опубликованной временной модели широко используемого реалистичного примера, авиационной системы контроля шасси, являющейся своего рода эталонной задачей для сравнения применимости различных методов спецификации. Верификация переведенной спецификации с помощью AutoProof обнаружила ошибку в одном из временных свойств. Углубленное изучение данной ошибки привело к обнаружению ошибки в опубликованной абстрактной машине состояний (ASM), которая реализует переведенную модель; авторы публикации, в свою очередь, заявили об успешной верификации. Корректировка исходной

спецификации и перевод результата в Eiffel с контрактами с последующей верификацией привели к успешному результату. Процесс перевода из LTL в Eiffel все еще находится в зачаточном состоянии и оптимизирован для используемого инструмента верификации (AutoProof), поэтому схема перевода не выглядит простой и элегантной. Даже с учетом указанных ограничений полученные результаты демонстрируют потенциал технологии автоматического доказательства традиционных программ в части ее применимости к специфичным проблемам встроенных систем.

Ключевые слова: бесшовные требования; проектирование по контракту; autoproof; эйфель; система контроля шасси

DOI: 10.15514/ISPRAS-2017-29(4)-3

Для цитирования: Наумчев А., Маццара М., Мейер Б., Брюэль Ж.-М., Галинье Ф., Эберсоль С. Контрактный метод спецификации реактивных требований. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 39-54 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-3

Список литературы

[1]. J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, “Autoproof: Auto-active functional verification of object-oriented programs,” *arXiv preprint arXiv:1501.03063*, 2015.

[2]. B. Meyer, *Touch of Class: learning to program well with objects and contracts*. Springer, 2009.

[3]. I. J. Hayes, M. A. Jackson, and C. B. Jones, *Determining the Specification of a Control System from That of Its Environment*, pp. 154–169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[4]. E. Clarke and E. Emerson, “Design and synthesis of synchronization skeletons using branching time temporal logic,” *Logics of programs*, pp. 52–71, 1982.

[5]. P. Arcaini, A. Gargantini, and E. Riccobene, “Modeling and analyzing using asms: the landing gear system case study,” in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 36–51, Springer, 2014.

[6]. F. Boniol and V. Wiels, “The landing gear system case study,” in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 1–18, Springer, 2014.

[7]. R. Koymans, “Specifying real-time properties with metric temporal logic,” *Real-time systems*, vol. 2, no. 4, pp. 255– 299, 1990.

[8]. A. Naumchev and B. Meyer, “Complete contracts through specification drivers,” in *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 160–167, July 2016.

[9]. Y. Gurevich, “Sequential abstract-state machines capture sequential algorithms,” *ACM Transactions on Computational Logic (TOCL)*, vol. 1, no. 1, pp. 77–111, 2000.

[10]. A. Naumchev, “Lgs asm ground model in eiffel..” https://github.com/anaumchev/lgs_ground_model, 2017.

[11]. N. Polikarpova, J. Tschannen, C. A. Furia, and B. Meyer, "Flexible invariants through semantic collaboration," in *FM 2014: Formal Methods*, pp. 514–530, Springer, 2014.

[12]. H. Yamada, "Real-time computation and recursive functions not real-time computable," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 753–760, Dec 1962.

[13]. M. Mazzara and A. Bhattacharyya, "On modelling and analysis of dynamic reconfiguration of dependable real time systems," in *Proceedings of the 2010 Third International Conference on Dependability*, DEPEND '10, (Washington, DC, USA), pp. 173–181, IEEE Computer Society, 2010.

[14]. M. Mazzara, "Timing issues in web services composition," in *Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings*, pp. 287–302, 2005.

[15]. L. Ferrucci, M. M. Bersani, and M. Mazzara, "An LTL semantics of business workflows with recovery," in ICSOFTPT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 2931 August, 2014, pp. 29–40, 2014.

[16]. M. Berger and K. Honda, "The two-phase commitment protocol in an extended pi-calculus," Electr. Notes Theor. Comput. Sci., vol. 39, no. 1, pp. 21–46, 2000.

[17]. A. Iliasov, A. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala, "Augmenting event-b modelling with real time verification," in Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, FormSERA '12, 2012.

[18]. D. Fahland, D. Lubke, J. Mendling, H. Reijers, B. Weber,¨ M. Weidlich, and S. Zugal, Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. Springer Berlin Heidelberg, 2009.

[19]. A. Bormotova, "Translation of natural language into hoare triples." https://github.com/An-Dole/ Semantic-mapping.

[20]. V. Skukov, "Translation of hoare triples into natural language." https://github.com/flosca/hybrid.

[21]. R. Gmehlich, K. Grau, F. Loesch, A. Iliasov, M. Jackson, and M. Mazzara, "Towards a formalism-based toolkit for automotive applications," in 1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013, San Francisco, CA, USA, May 25, 2013, pp. 36–42, 2013.