Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 1377–1380

Towards deductive-based support for software development processes

Radosław Klimek
AGH University of Science and Technology
al. A. Mickiewicza 30, 30-059 Krakow, Poland
rklimek@agh.edu.pl

***Abstract*—The work relates two initial disciplines of the Rational Unified Process (RUP), i.e. Business Modeling and Requirements Engineering, to support them in an integrated way through deductive-based formal verification using temporal logic. On the other hand, Cyber-Physical Systems (CPS), which should be an effective orchestration of computations and physical processes, need careful development and formal verification to ensure they influence software reliability, trustworthiness and cost in a positive way. A method for building both business models and requirements models, including their logical specifications, is proposed and presented step by step. Applying the presented concepts bridges the gap between the benefits of deductive reasoning for correctness analysis and the difficulties in obtaining complete logical specifications.**

I. Introduction

CYBER-Physical Systems (CPS) are understood as integrations of computation with physical processes [1] and often refer to embedded systems. A CPS is designed as a network of interacting elements with physical input and output and focus on both technology and mathematical abstractions. CPS need to improve the development processes, in order to raise the level of abstraction, and to formally verify designs. On the other hand, the Rational Unified Process (RUP) provides a disciplined approach to assignment of tasks and responsibilities within software processes. Most iterations within RUP phases result in an executable deliverable. RUP consists of perspectives, disciplines, etc. [2]. The work focuses on the first two disciplines which are Business Modeling (BM) and Requirements Engineering (RE). BM involves higher level managing people. RE involves higher level software engineers. BM facilitates discovering RE. On the other hand, considering BM and RE together can result in a synergic effect. Formal methods enable the precise formulation of important artifacts, eliminating ambiguity during the software development process [3]. Deductive inference enables the analysis of infinite computation sequences and is an essential part of everyday life and scientific work. On the other hand, the important question for deductive approach is the lack of automatic methods for obtaining logical specifications understood as (large) sets of temporal logic formulas. Thus, the automation of this process seems justified and particularly important.

The motivation for the work is the lack of tools for deductive-based formal verification of RUP-like processes. Another motivation is the lack of tools for automatic extraction of logical specifications from software models. The contribution of the work is a method for automatic generation of logical specifications considered as sets of temporal logic formulas. Relatively simple yet illustrative examples of the approach are provided.

Work by Morimoto [4] contains a survey of formal verification methods for business processes. It discusses automata, model checking, communicating sequential processes, Petri nets, Markov networks, and all these issues are discussed in the context of business process management and web services. Work by Brambilla et al. [5] contains some aspects of workflows and temporal logic, but formulas are mostly created manually and formal verification is not discussed widely. In work by Kazhamiakin [6], a method based on formal verification of requirements using temporal logic and model checking approach is proposed, and a case study is discussed.

II. Basic assumptions

The idea of workflow patterns is crucial for this work. They constitute a kind of primitives to enable development of software models and modeling logical specifications. A set of temporal logic formulas is linked to every pattern. The basic issues related to temporal logics and their syntax and semantics are discussed in many works, e.g. [7]. The considerations in this work are limited to the *linear-time temporal logic* LTL, and attention is focused on the *propositional linear time logic* PLTL. The *elementary set* $pat()$ of formulas over atomic formulas a_i, where $i > 0$, which is also denoted $pat(a_i)$, is a set of temporal logic formulas $f_1, ..., f_m$ such that all formulas are syntactically correct. The example of an elementary set is $pat(a, b, c) = \{a \Rightarrow \Diamond b, b \Rightarrow \Diamond c, \Box\neg(a \wedge b \wedge c)\}$ which is a three-element set of formulas created over three atomic formulas. The *logical expression* W_L is a structure similar to the well-known regular expressions and allows to express the complex and nested workflow model in a literal notation. For example, $Seq(Split(a, b, c), Cond(d, e, f))$ shows the sequence of a parallel split followed by conditional execution of some tasks, and the meaning of all patterns is intuitive and not formally defined.

Every pattern has a predefined and countable set of linear temporal logic formulas. Thus, all acceptable patterns constitute a set of *predefined design patterns* Π. The example of such a set for business models is shown in Fig. 1. Software workflows should be modeled using predefined patterns only.

978-1-4673-4471-5/$25.00

Most elements of this predefined set, i.e. comments, two temporal logic operators, classical logic operators, are not in doubt. The slash allows to place more than one formula in a single line. The $entry$, or shortly en, and $exit$, or shortly ex, expressions are collections of atomic formulas which describe, informally speaking, the potential logical entry points end logical exit points for a particular pattern. The entry and exit formulas represent a pattern as a whole. f_1, f_2 etc. are atomic formulas for a pattern. They constitute a kind of formal arguments for a pattern.

The *logical specification* L consists of all formulas derived from the logical expression W_L using the algorithm $\mathcal{A}$, i.e.

$$L(W_L) = \{f_i : i \geq 0 \wedge f_i \in \mathcal{A}(W_L, P)\} \quad (1)$$

where f_i is a PLTL formula. The generation algorithm $\mathcal{A}$ has two inputs. The first one is a logical expression W_L which is a kind of a variable, i.e. it varies when a model (workflow) is subjected to any modification by software engineers. The second one is the predefined set P for business models or for activity models which is a kind of constant, i.e. it is once defined and then widely used. The output of the algorithm is a logical specification understood as a set given by a formula 1. However, generation of logical specifications is not a simple summation of formula collections from predefined design patterns. The algorithm ($\mathcal{A}$) is as follows:

1) at the beginning, the logical specification is empty, i.e. $L = \emptyset$;
2) the most nested pattern or patterns are processed first; then, less nested patterns are processed one by one, i.e. patterns that are located more towards the outside;
3) if the currently analyzed pattern consists of atomic formulas only, the logical specification is extended, by summing sets, by formulas linked to the currently being analyzed pattern $pat()$, i.e. $L = L \cup pat()$;
4) if any argument is a pattern itself, then the logical disjunction of its $entry$ and $exit$ formulas is substituted in the place of the pattern as an argument.

III. Business modeling

Business modeling BM allows to understand the structure and behavior of the organization in which a system is to be deployed. It also ensures stakeholders to have a common understanding of the target organization. *Business Process Modeling Notation* BPMN is a standard graphical notation provided by the Business Process Management Initiative (BPMI) for the modeling of business processes. BPMN is becoming the dominant modeling notation, bridging the gap between business process design and process implementation. The main goal of BPMN is to provide a notation that is understandable by all business users, from business analysts to technical developers, and finally, to business people who will manage and monitor those processes [8]. An important parts of BPMN are 21 patterns which are introduced by van der Aalst et al. [9]. Gradually building in complexity, process patterns were broken down into six categories: Basic Control Flow Patterns, Advanced Branching, Structural, Multiple Instances, State Based, and Cancellation. For example, the basic control flow patterns are divided into five particular patterns: Sequence, Parallel Split, Synchronization, Exclusive Choice, and Simple Merge, the meaning of which is defined in terms of temporal logic formulas in Fig. 1.

```
                                /* ver. 27.04.2013
/* Basic Control Patterns
Sequence(f1,f2):
entry=f1 / exit=f2
f1 => <>f2 / ~f1 => ~<>f2 / []~(f1 & f2)
ParallelSplit(f1,f2,f3):
en=f1 / ex=f2,f3
f1 => <>f2 & <>f3 / ~f1 => ~<>f2 & ~<>f3
[]~(f1&(f2|f3))
Synchronization(f1,f2,f3):
en=f1,f2 / ex=f3
f1 & f2 => <>f3 / ~f1 | ~f2 => ~<>f3
[]~((f1|f2)&f3)
ExclusiveChoice(f1,f2,f3):
en=f1 / ex=f2,f3
f1 => (<>f2 & ~<>f3)|(~<>f2 & <>f3)
~f1 => ~<>f2 & ~<>f3
[]~(f2 & f3) / []~(f1&(f2|f3))
SimpleMerge(f1,f2,f3):
en=f1 / ex=f2,f3
f1|f2 => <>f3 / ~f1 & ~f2 => ~<>f3
[]~(f1&f2) / []~((f1|f2)&f3)

/* ..... [other] Business Patterns
```

Fig. 1. A predefined set of patterns P for BPMN models

Let $\Pi_1 = \{Sequence, ParallelSplit, Synchronization, ExclusiveChoice, SimpleMerge, ...\}$ be a predefined set of patterns for business models, with aliases Seq, $Split$, $Synch$, $Choice$, and $Merge$, respectively. The meaning of patterns is formally defined in Fig. 1. Although the above set contains only some patterns to present the main ideas of the work, other workflow patterns together with their temporal logic formulas could be defined in future research.

The business model for flight dispatch is considered below. The BPMN model is shown in Fig. 2. When the decision to realize the flight is taken, then some processes are carried out concurrently. Some of them relate to the preparation of the aircraft and the crew, while others – to passenger handling and loading of baggage. Taking off must be preceded by a permission to start with all of its internal actions. Every business process should be decomposed into a business use case and a series of activities. The logical expression W_L is

$$Seq(Split(Initiate, Split(PreFlightPreparation, AircraftPreparation, PassengerHandling), FlightCoordination), Synch(Synch(CrewPreparation, BaggageLoading, AircraftTaxiing), PermissionStart, TakingOff))$$

or after the substitution of propositions (business processes) as capital letters of the Latin alphabet: A – Initiate, B – PreFlightPreparation, C – AircraftPreparation, D – Passenger-Handling, E – FlightCoordination, F – CrewPreparation, G – BaggageLoading, H – AircraftTaxiing, I – PermissionStart, and J – TakingOff. A logical specification L for the above logical expression is built in such a way that patterns are processes in the following order: most nested $Split$, most

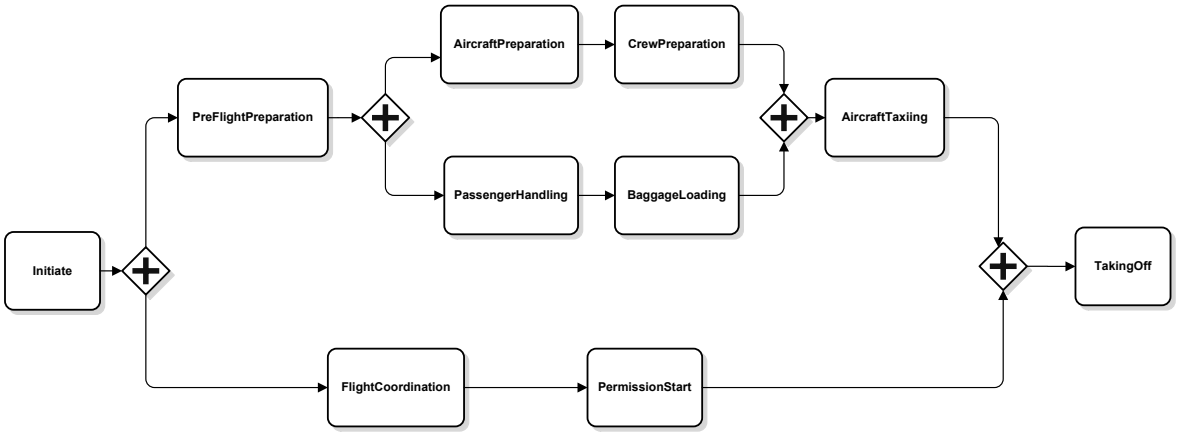


Fig. 2. A business model for a flight dispatch

nested $Synch$, outer $Split$, outer $Synch$, and Seq. The resulting logical specification contains formulas

$$\begin{aligned} L = \{&B \Rightarrow \Diamond C \wedge \Diamond D, \neg B \Rightarrow \neg\Diamond C \wedge \neg\Diamond D,\\ &\Box\neg(B \wedge (C \vee D)), F \wedge G \Rightarrow \Diamond H, \neg F \vee \neg G \Rightarrow \neg\Diamond H,\\ &\Box\neg((F \vee G) \wedge H), A \Rightarrow \Diamond(B \vee C \vee D) \wedge \Diamond E,\\ &\neg A \Rightarrow \neg\Diamond(B \vee C \vee D) \wedge \neg\Diamond E,\\ &\Box\neg(A \wedge ((B \vee C \vee D) \vee E)), (F \vee G \vee H) \wedge I \Rightarrow \Diamond J,\\ &\neg(F \vee G \vee H) \vee \neg I \Rightarrow \neg\Diamond J,\\ &\Box\neg(((F \vee G \vee H) \vee I) \wedge J),\\ &(A \vee C \vee D \vee E) \Rightarrow \Diamond(F \vee G \vee I \vee J),\\ &\neg(A \vee C \vee D \vee E) \Rightarrow\\ &\neg\Diamond(F \vee G \vee I \vee J),\\ &\Box\neg((A \vee C \vee D \vee E) \wedge (F \vee G \vee I \vee J))\} \end{aligned} \quad (2)$$

Formal *verification* is the act of proving the correctness of a system. Liveness and safety are a taxonomy of system properties. *Liveness* means that the computational process achieves its goals (something good eventually happens). *Safety* means that the computational process avoids undesirable situations (nothing bad ever happens). The liveness property for the model can be

$$A \Rightarrow \Diamond J \quad (3)$$

which means that **if the initiation is executed then sometime in the future the aircraft take off**, or formally $Initiate \Rightarrow \Diamond TakeOff$. When considering the property, the entire formula to be analyzed using, for example, the semantic tableaux method is

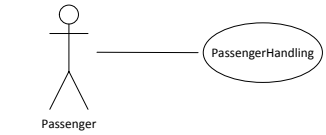
$$C(L) \Rightarrow (A \Rightarrow \Diamond J) \quad (4)$$

where $C(L)$ is a logical conjunction of all formulas that belong to the logical specification L, c.f. formula 2. In a similar way, safety formulas are considered, e.g. $\Box\neg(\neg PermissionStart \wedge TakingOff)$ or even formula $\Box\neg(PermissionStart \wedge \neg CrewPreaparation.c \wedge \neg BaggageLoading.c)$ the meaning of which seems understandable, and the $.c$ suffix means the logical condition associated with an activity. However, the presentation of a full inference tree for these cases exceeds the size of the work. In the case of the semantic tableaux method for logical reasoning, when the falsification of the semantic tree is received, the open branches are obtained and provide information about the source of the error. This is another advantage of the method.

IV. Requirements modeling

Once the business model is built, a requirements model is developed. The transition between the models is done in such a way that a single business process is mapped to a single business use case [2]. A business use case is always actor-centric. A business use case *scenario* is a description that illustrates the behavior from the actors's point of view. Every scenario allows to identify and extract atomic activities. Afterwards, the *activity diagram* enables the modeling of workflows for atomic activities using predefined workflow patterns. It supports choice, concurrency and iteration. The activity diagram shows how an activity depends on others. Nesting of activities is permitted. The following design patterns for activities are introduced [10], [11]: *sequence*, *concurrent fork/join*, *branching* and *loop-while* for iteration. Thus, the predefined set of patterns for activity workflows is $\Pi_2 = \{Sequence, Concurrency, Branching, LoopWhile\}$, and aliases are: Seq, $Concur$, $Branch$, and $Loop$, respectively.

The business use case "PassengerHandling" is discussed below. This is one of the processes received from a business model shown in Fig. 2. The use case scenario is in Fig. 3. The use case scenario is in Fig. 4. When the passenger holding a valid ticket arrives at the airport, the airport check-in, i.e. counter or self, must be made to confirm the passenger's presence, and then a boarding pass is issued. When the passenger has hold baggage, then it must be registered. (In Europe) when the passenger is outside the non-Schengen countries, then border and custom controls need to be performed. The last step before boarding is the security control. Not to introduce the simple branching (if-then) as another pattern for the activity



Use Case: PassengerHandling
Preconditions: Passenger has a valid ticket without check-in
Scenario: 1. Passenger's "Check-in"or"SelfCheck-in" 2. If necessary then "HoldBaggage" 3. If non-Schengen then "BorderControl" and "CustomControl" 4. Passenger's "SecurityControl" 5. Passenger's "Boarding"
Postconditions: Passenger completed all formalities related to the flight

Fig. 3. A business use case (top) and its scenario (bottom)

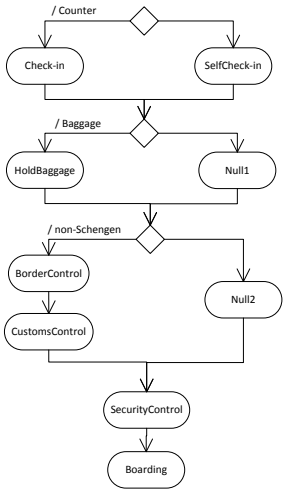


Fig. 4. An activity diagram for the scenario from the Fig. 3

diagram, the empty activity ("Null") is introduced instead a full branching (if-then-else). The Null activity does not consume time, i.e. after reaching the activity it is always immediately completed.

After the substitution of propositions (atomic activities) as small letters of the Latin alphabet: a – Counter, b – CheckIn, c – SelfCheckIn, d – Baggage, e – HoldBaggage, $n1$ – Null1, f – nonSchengen, g – BorderControl, h – CustomControl, $n2$ – Null2, i – SecurityControl, and j – Boarding, then the expression W_L is

$$Seq(Seq(Branch(a,b,c),Branch(d,e,n1)),\\ Seq(Branch(f,Seq(g,h),n2),Seq(i,j)))$$

A logical specification L for the above logical expression is build in the same way as it is shown in the previous section. An example of the liveness property for the model can be

$$e \Rightarrow \Diamond j \tag{5}$$

which means that **if the hold baggage for a passenger is registered then sometime in the future the passenger is boarding**, or more formally $HoldBaggage \Rightarrow \Diamond Boarding$.

V. Conclusion

A method for a deductive-based support developing software models for the first two RUP disciplines is proposed. Also, a method for automatic generation of logical specifications is defined.

Future works might result in development of CASE software which supports deduction-based formal verification of software development processes for CPS systems. Considering Concurrent Communicating Lists [12] is encouraging for strengthen and join these directions of research.

Acknowledgment

This work was supported by the AGH UST internal grant no. 11.11.120.859.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, ser. ISORC 2008. IEEE Computer Society, 2008, pp. 363–369. [Online]. Available: http://dx.doi.org/10.1109/ISORC.2008.25

[2] P. Kruchten, *The Rational Unified Process: An Introduction*, ser. The Addison-Wesley Object Technology Series. Addison Wesley, 2004.

[3] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM Computing Survey*, vol. 41, no. 4, pp. 19:1–19:36, 2009.

[4] S. Morimoto, "A survey of formal verification for business process modeling," in *Proceedings of the 8th International Conference Computational Science (ICCS 2008), June 23–25, 2008, Kraków, Poland, Part II*, ser. Lecture Notes in Computer Science, M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5102. Springer–Verlag, 2008, pp. 514–522.

[5] M. Brambilla, A. Deutsch, L. Sui, and V. Vianu, "The role of visual tools in a web application design and verification framework: A visual notation for ltl formulae," in *Proceeding of the 5th International Conference on Web Engineering (ICWE 2005), July 27-29, 2005, Sydney, Australia*, ser. Lecture Notes in Computer Science, D. Lowe and M. Gaedke, Eds., vol. 3579. Springer–Verlag, 2005, pp. 557–568.

[6] R. Kazhamiakin, M. Pistore, and M. Roveri, "Formal verification of requirements using spin: A case study on web services," in *Proceedings of 2nd International Conference on Software Engineering and Formal Methods (SEFM 2004), 28–30 September 2004, Beijing, China*, 2004, pp. 406–415.

[7] E. Emerson, *Handbook of Theoretical Computer Science*. Elsevier, MIT Press, 1990, vol. B, ch. Temporal and Modal Logic, pp. 995–1072.

[8] OMG, "Business process modeling notation specification, version 1.2," January 2009, OMG Document dtc/2009-01-03, Tech. Rep., 2009.

[9] W. M. van der Aalst, A. ter Hofstede, B. Kiepusewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 4(1), pp. 5–51, 2003.

[10] R. Klimek, "Proposal to improve the requirements process through formal verification using deductive approach," in *Proceedings of 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012), 29–30 June, 2012, Wrocław, Poland*, J. Filipe and L. Maciaszek, Eds. SciTePress, 2012, pp. 105–114.

[11] ——, "From extraction of logical specifications to deduction-based formal verification of requirements models," in *Proceedings of 11th International Conference on Software Engineering and Formal Methods (SEFM 2013), 25–27 September 2013, Madrid, Spain*, ser. Lecture Notes in Computer Science, R. Hierons, M. Merayo, and M. Bravetti, Eds., vol. 8137. Springer Verlag, 2013, pp. 61–75.

[12] K. Kułakowski and T. Szmuc, "Modeling robot behavior with ccl," in *Simulation, Modeling, and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science, I. Noda, N. Ando, D. Brugali, and J. Kuffner, Eds. Springer, 2012, vol. 7628, pp. 40–51. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34327-8_7