Integration between Requirements Modeling and Software Development in the Notification Oriented Paradigm: A Security System Case Study

Paulo J. D. Novaes[1], Jean M. Simão[1], Paulo C. Stadzisz[1]

[1]Graduation Program in Electrical Engineering and Computer Engineering (CPGEI)
Federal University of Technology - Paraná, (UTFPR), Curitiba - PR - Brasil

pnovaes@alunos.utfpr.edu.br, {jeansimao,stadzisz}@utfpr.edu.br

Abstract. *This paper presents the integration between the requirements modeling approach named Notification Oriented Requirements (NOR) and the Software Development method known as Notification Oriented Development (NOD). This integration is demonstrated by means of the case study of a simulated access control security system implemented in the Notification Oriented Paradigm (NOP). Results show that the integration between NOR model and NOD method is possible and facilitates the development of NOP software, since NOR clarifies the necessary elements to perform the software structural modeling (class model) and the behavioral modeling (high level states model and component model).*

1. Introduction

Requirements Engineering (RE) refers to the activity of formulating, documenting, and maintaining systems requirements in order to produce, from users' needs, a set of specification related to what the final system should be [YOUNG, 2004]. A requirement is a statement from the stakeholders' needs to define a product, a system or a process, and must be unambiguous, clear, unique, consistent, stand-alone, and verifiable [INCOSE, 2006]. Graphical approaches to enhance requirements specification (among others system´s characteristics) have gained prevalence, such as the SysML language, which is used in Model-Based Systems Engineering (MBSE) [FRIEDENTHAL et al., 2014].

In this context, Notification Oriented Requirements (NOR) emerges as a requirement modeling approach originated from concepts of the Notification Oriented Paradigm (NOP) and MBSE [SIMÃO et al., 2016]. In brief, NOP is an alternative paradigm using rules and notifications for composing software and hardware systems. Within this paradigm, NOR is a requirements specification approach applicable to both, software and system development processes. The practical integration between NOR and software development processes is an important experimentation for its validation. Currently, the Notification Oriented Development (NOD) method [MENDONÇA et al., 2015] is suited to develop NOP software. Thus, the following questions arise:

- Is it possible to integrate the NOR modeling approach into the NOD method?
- Are there advantages in integrating NOR and NOD into a NOP application project? Which are they?

To answer these questions, this study starts from a previous NOR model [SIMÃO et al., 2016], uses the NOD method to design and implement the corresponding NOP application, and concludes discussing the findings from the case study.

Therefore, the objectives of this study are to integrate NOR modeling approach into the NOD method and to identify the advantages of such integration during NOP application software development.

2. Notification Oriented Paradigm (NOP)

NOP has been improved in recent years by a group of researchers from the Federal University of Technology - Paraná (UTFPR). It is an alternative approach to develop software and hardware systems. NOP has several implementation versions in the form of frameworks and languages [FERREIRA, 2015]. It proposes to solve existing problems in usual programming paradigms [SIMÃO and STADZISZ, 2008] such as the Declarative Paradigm (PD) and the Imperative Paradigm (PI) [GABBRIELLI and MARTINI, 2010]. These problems are related to structural and temporal redundancies, and the strong coupling between computational entities [FERREIRA, 2015].

The fundamental proposal of NOP consists in the introduction of a notification-based inference mechanism, presenting a new way of structuring software in small and decoupled computational entities. These entities includes *Fact Base Elements* (*FBE*) and Rules [SIMÃO and STADZISZ, 2008]. An example of a NOP *Rule* is shown in figure 1 (a) and the NOP model is shown in figure 1 (b).

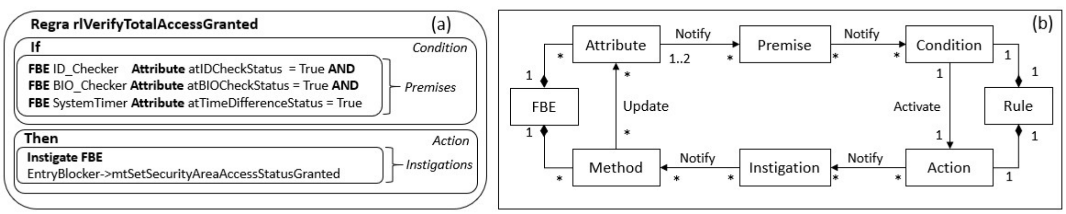

Figure 1. (a) NOP *Rule*. (b) NOP Model. Based on [SIMÃO et al., 2016].

The *Fact Base Element* (*FBE*) stores system facts in *Attributes*. The *FBE* may have *Methods* that modify these *Attributes*. Each *Attribute* when it changes its status notifies only the related *Premises*. Similarly, each *Premise* when it changes its status notifies only the pertinent *Conditions*. Each *Condition* has one or more associated *Premises*, that becoming true, approve a *Rule*. The *Rule* has an *Action*, which notifies one or more *Instigations* to execute *Methods*, which in turn modify other *Attributes* [MENDONÇA et al., 2015]. This sequence characterizes the NOP inference mechanism, based on notifications. This mechanism avoids the need for matching and selection processes in order to execute rules, as usual in Rules Based Systems (RBS).

Many NOP applications have been developed [SANTOS, 2017]. However, NOP application development requires new specific software development methods. For this purpose, techniques such as NOD [WIECHETECK, 2011] and NOR modeling [SIMÃO et al., 2016] have been developed, which will be presented in the next sections.

3. Notification Oriented Requirements (NOR)

In NOR approach, the fundamental primitives of NOP models are used for graphical requirements modeling, such as *Rules*, *FBEs,* and *Notifications* (preconditions or postconditions). These primitives allow to describe the complete set of functional e non-functional software/system requirements [SIMÃO et al., 2016]. The NOR modeling notation is summarized in figure 2.

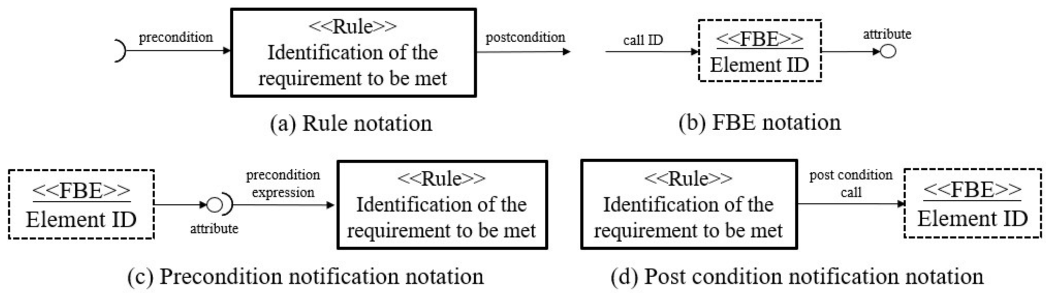

Figure 2. NOR requirements modeling notation [SIMÃO et al., 2016].

The NOR technique to construct the requirements model is [SIMÃO et al., 2016]:

For each requirement in the System Requirements Specification (SRS):

1. *To analyze the requirements sentence aiming at:*
   i) *Identifying the functional or non-functional request in the requirement.*
   ii) *Identifying the Conditions for the functional or non-functional request.*
   iii) *Identifying the attributes involved in the Conditions.*
   iv) *Identifying the Actions for the functional or non-functional request.*
   v) *Identifying the functions related to the Methods instigated in the Actions.*
   vi) *Identifying the FBEs related to the Attributes for the request.*
   vii) *Identifying the FBEs related to the Methods indicated by the request.*
2. *To create a Rule for every request identified in step 1.*
3. *To create a FBE for every entity identified in step 1.*
4. *To create links (i.e. notifications between Rules and FBEs according to conditions and Actions related to rules) identified in step 1.*
5. *To merge FBEs and Rules with analogous FBEs and Rules previously created.*

The use of NOR modeling was presented in a case study [SIMÃO et al., 2016] whose requirements were extracted from the INCOSE Systems Engineering Handbook [INCOSE, 2006]. Six requirements for the given security area access control system are presented below [SIMÃO et al., 2016]:

- *SS11-a: Secure areas shall be protected by security check based upon employee ID.*
- *SS11-b: Secure areas shall be protected by a second independent security check based upon biometric data.*
- *SS11-c: The time between the two independent security checks shall not exceed a configurable period.*
- *SS11-d: The user shall be allowed three attempts at biometric identification.*
- *SS11-e: The user shall be allowed three attempts at card identification.*
- *SS11-f: Any denied access attempt shall be sent to the administrator.*

Based on the NOR technique above, a model for the given security system was created (figure 3) [SIMÃO et al., 2016] containing 7 requirements in the form of *Rules* and 7 entities in the form of *FBEs*.

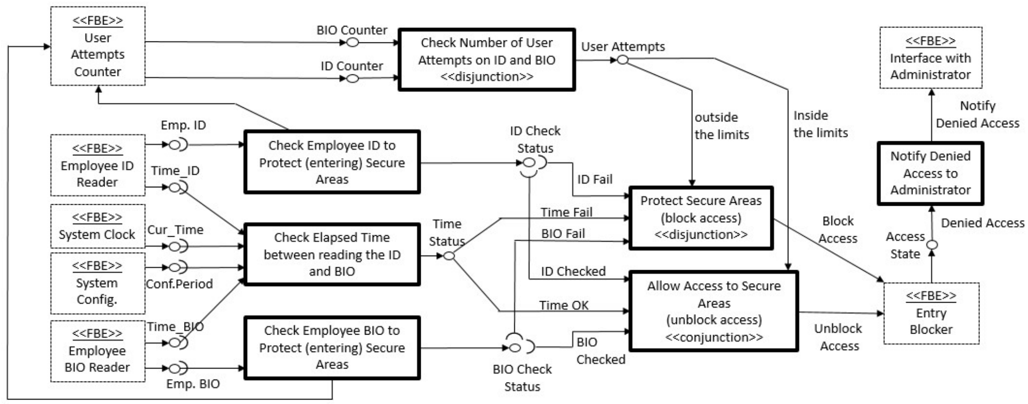

Figure 3. NOR requirements final modeling. Adjusted from [SIMÃO et al., 2016]

The model above facilitates requirements analysis and implicit knowledge identification, in addition to making explicit the dependencies between requirements [SIMÃO et al., 2016]. The next section presents the NOD method propped to conduct the development of NOP software applications.

4. Notification Oriented Development (NOD)

NOD is a software development method developed specifically for NOP applications [WIECHETECK, 2011], which consists of an extension of UML diagrams in the form of a UML profile, that properly represents NOP concepts (*NOP profile*). In addition, NOD establishes a sequence of steps to guide NOP software development.

The *NOP profile* enables to characterize NOP elements more precisely during design phase, allowing to particularize UML for a specific domain of applications. This is done by determining a new syntax and semantics for UML elements using stereotypes, tagged values, and constraints [WIECHETECK et al., 2011].

NOD method contains 8 steps (figure 4). The first two steps are: 1. *Capture Requirements* and 2. *Create Use Case Model*. The next six steps focus on software design through diagrams creation: 3. *Class Model*; 4. *High Level States Model*; 5. *Component Model*; 6. *Sequence Model* (optional, not created in this study[1]); 7. *Communication Model* (optional, not created in this study[1]); 8. *Petri Net Model* [WIECHETECK, 2011].

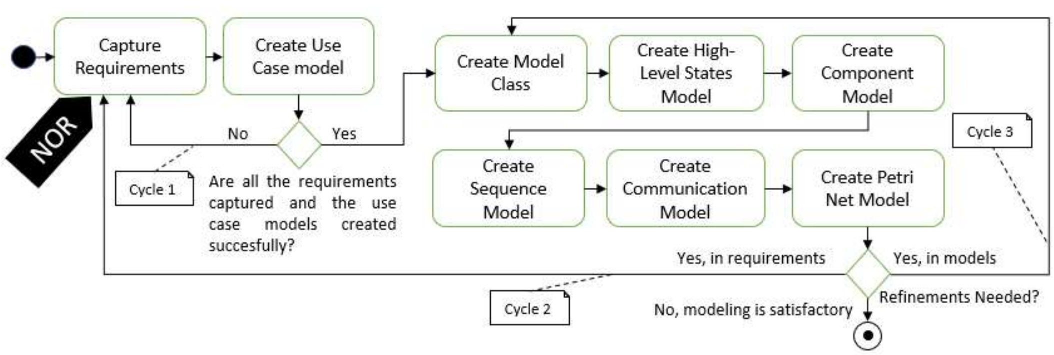

Figure 4. Integrating NOR into NOD. Based on [MENDONÇA et al., 2015].

---

[1] The creation of the *Sequence Model* and *Communication model* is an optional part of DON method and was not performed in the current study, without compromising the article results.

The method is divided in three cycles (figure 4). In **first cycle**, the requirements are documented and the *Use Case Model* is created. In **second cycle**, diagrams are created and requirements are refined as necessary. In **third cycle**, the models are refined to ensure compliance with the requirements [MENDONÇA et al., 2015] [WIECHETECK, 2011].

Software coding can either occur at the end of the method (cascade software process) or during cycles (incremental software process). The incremental software process was used in this case study.

5. Case Study: NOD Modeling of a Simulated Security System

The requirements and NOR modeling described previously were used as the basic scope for this case study. This is the main point of integration between the methods, in which the **NOD 1st Cycle => 1st step** *Capture Requirements* of the NOD method is now performed by the NOR technique, as pointed out in figure 4.

The models for this study were created using tools of the *Enterprise Architect® v.13.5* (*Sparx Systems*) suite by applying the *NOP Profile* [WIECHETECK et al., 2011], except for the Petri Net model, created using *CPN Tools® v.4.0.1* (*Eindhoven University of Technology*).

Given this, based on NOR modeling, the **NOD 1st Cycle => 2nd step** *Create Use Case Model* was performed and resulted in the model shown in figure 5.

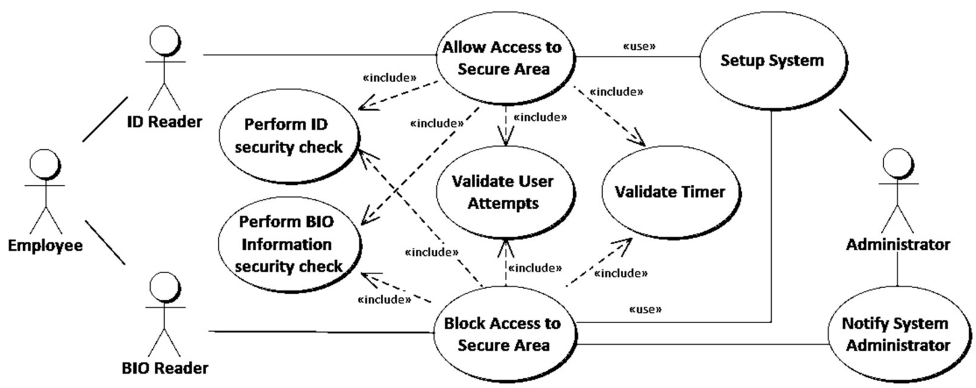

Figure 5. Use Case Model (NOD)

In **NOD 2nd cycle** modeling, the following diagrams were created, later refined in the **NOD 3rd cycle** (figure 4): 1. *Class Model* (*FBEs* definition); 2. *High Level States Model* (*Rules modeling*); 3. *Component Model*; 4. *Petri Net Model*.

The *Class Model* is presented in figure 6. In addition to the stereotyped class *<<NOP_Application>>*, that is a default in NOP applications, stereotyped classes *<<NOP_FBE>>* were created for each *FBE*. In this step, it is possible to notice the easiness achieved by the previous existence of the NOR model, **since it becomes possible to correlate the *FBEs*** (this does not imply necessarily a 1 to 1 relationship) modeled in NOR to those included in the *Class Model*:

- *Employee ID Reader* (NOR) ⇔ *ID_Checker* (NOD)
- *Employee BIO Reader* (NOR) ⇔ *BIO_Checker* (NOD)
- *System Clock* (NOR) ⇔ *SystemTimer* (NOD)
- *User Attempts Counter* (NOR) ⇔ *UserAttemptsCounter* (NOD)

- *Entry Blocker* (NOR) ⇔ *EntryBlocker* (NOD)
- *System Config.* (NOR) ⇔ *SystemConfigurator* and *EmployeeController* (NOD)
- *Interface with Administrator* (NOR) ⇔ *SysAdminNotificationController* (NOD)

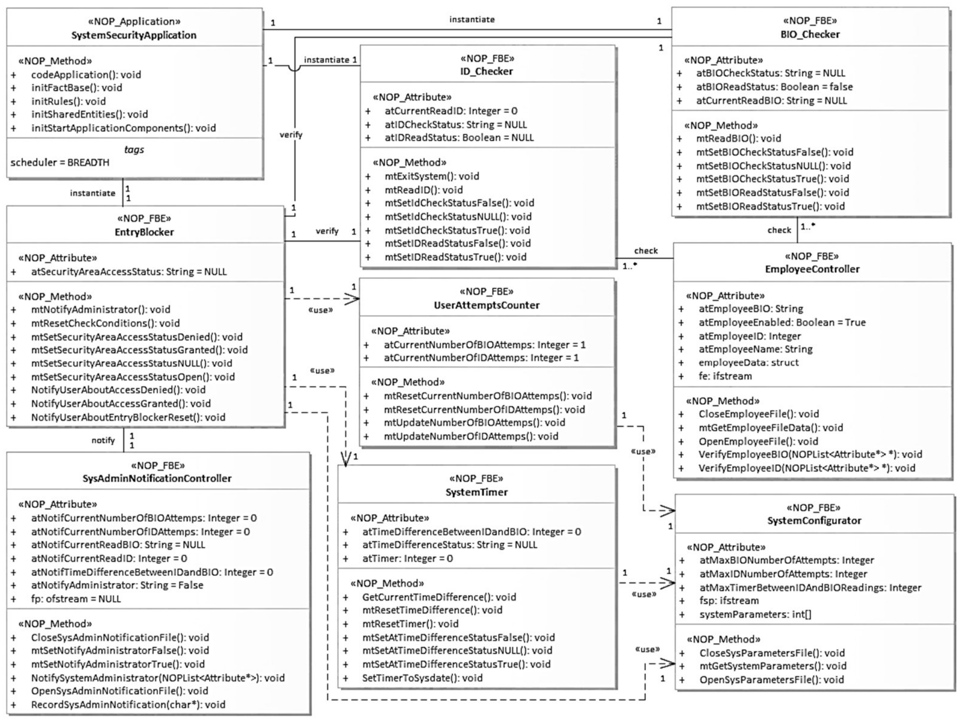

Figure 6. Class Model (NOD)

The *High-Level States Model* establishes the basic logic of system operation and bases the identification of NOP Application *Rules* (figure 7).

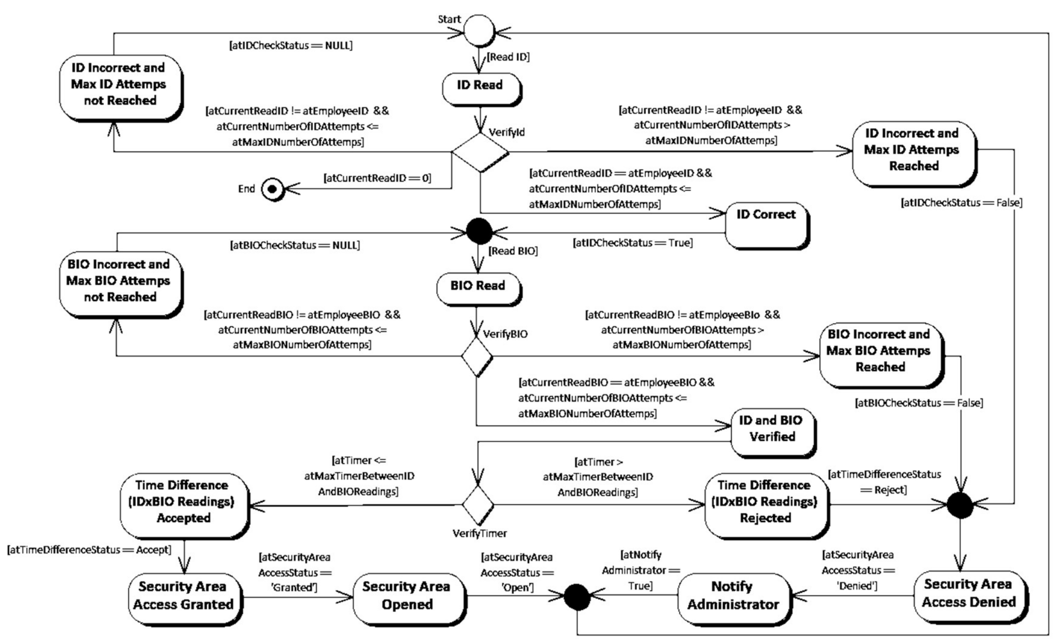

Figure 7. High-Level States Model (NOD)

While creating the *High-Level States Model*, **the NOR model was used as an aid**, **allowing to visually identify states** or **facts** (e.g.: *ID Checked, ID Fail, Time OK, Time Fail, BIO Checked, BIO Fail,* etc.) and **activities** (e.g.: *Check employee ID, Check Elapsed Time, Check Employee BIO, Block Access, Unblock Access,* etc.).

Based on *Class* and *High-Level States* models, a ***Rules*** **Table** containing the *Rules, Premises, and Instigations* was created (table 1). NOP elements were named using a standard that facilitates identification during both system design and programming. Prefixes were used as follows: *rl* for *Rules*; *pr* for *Premises*; *in* for *Instigations*; *at* for *Attributes* and *mt* for *Methods*, following definitions given by [RONSZCKA et al., 2017].

Table 1. NOP *Rules* identified for the System (NOD)

N	Use Cases	Rules	Premises	Instigations
1	Perform ID Security Check	rlReadID	prAtIDReadStatusFalse	inReadID && inVerifyEmployeeID && inSetAtIDReadStatusTrue
2	Perform ID Security Check	rlVerifyIDIncorrectRetry	prAtIDCheckStatusNULL && prAtIDReadStatusTrue && prIDIncorrect && prAtCurrentNumIDAttemptsSmallerThanMax	inIncrementAtCurrentNumberOfIDAttempts && inSetAtIDCheckStatusNULL && inResetTimer &&
3	Perform ID Security Check	rlVerifyIDIncorrectDenyAcc	prAtIDCheckStatusNULL && prAtIDReadStatusTrue && prIDIncorrect &&	inSetAtIDCheckStatusFalse && inResetTimer
4	Perform ID Security Check	rlVerifyIDCorrectProceed	prAtIDCheckStatusNULL && prAtIDReadStatusTrue && prIDCorrect &&	inSetAtIDCheckStatusTrue && inSetAtTimerToSysdate
5	Perform BIO Security Check	rlReadBIO	prAtIDCheckStatusTrue && prAtBIOReadStatusFalse	inReadBIO && inVerifyEmployeeBIO && inSetAtBIOReadStatusTrue
6	Perform BIO Security Check	rlVerifyBIOIncorrectRetry	prAtBIOCheckStatusNULL && prAtBIOReadStatusTrue && prAtIDCheckStatusTrue && prBIOIncorrect && prAtCurrentNumBIOAttemptsSmallerThanMax	inIncrementAtCurrentNumberOfBIOAttempts && inSetAtBIOCheckStatusNULL &&
7	Perform BIO Security Check	rlVerifyBIOIncorrectDenyAc	prAtBIOCheckStatusNULL && prAtBIOReadStatusTrue && prAtIDCheckStatusTrue && prBIOIncorrect && prAtCurrentNumBIOAttemptsGreaterOrEqualThanMax	inSetAtBIOCheckStatusFalse && inResetTimer
8	Perform BIO Security Check	rlVerifyBIOCorrectProceed	prAtBIOCheckStatusNULL && prAtBIOReadStatusTrue && prAtIDCheckStatusTrue && prBIOCorrect && prAtCurrentNumBIOAttemptsSmallerOrEqualThanMax	inSetAtBIOCheckStatusTrue && inGetTimeDifference
9	Perform Timer Validation	rlVerifyTimerAccept	prAtIDCheckStatusTrue && prAtBIOCheckStatusTrue && prAtTimeDifferenceSmallerOrEqualThanMax	inSetAtTimeDifferenceStatusTrue
10	Perform Timer Validation	rlVerifyTimerReject	prAtIDCheckStatusTrue && prAtBIOCheckStatusTrue && prAtTimeDifferenceGreaterThanMax	inSetAtTimeDifferenceStatusFalse
11	Perform Entry Blocker Security	rlVerifyTotalAccessGrantec	prAtIDCheckStatusTrue && prAtBIOCheckStatusTrue && prAtTimeDifferenceStatusTrue	inSetAtSecurityAreaAccessStatusGranted
12	Perform Entry Blocker Security	rlVerifyTotalAccessDenied	prAtIDCheckStatusFalse || prAtBIOCheckStatusFalse || prAtTimeDifferenceStatusFalse	inSetAtSecurityAreaAccessStatusDenied
13	Allow Access to Secure Area	rlOpenSecurityArea	prAtSecurityAreaAccessStatusGranted	inSetAtSecurityAreaAccessStatusOpen &&
14	Allow Access to Secure Area	rlSecurityAreaOpened	prAtSecurityAreaAccessStatusOpen	inNotifyUserAboutEntryBlockerReset && INSTIGATIONS TO RESET ALL
15	Block Access to Secure Area	rlBlockSecurityArea	prAtSecurityAreaAccessStatusDenied	inSetAtNotifyAdministratorTrue && inNotifyUserAboutAccessDenied
16	Notify System Administrator	rlAdministratorNotified	prAtNotifyAdministratorTrue	inSetAtNotifyAdministratorFalse && inNotifySystemAdministrator && inNotifyUserAboutEntryBlockerReset && INSTIGATIONS TO RESET ALL
17	Exit System	rlExitSystem	prAtIDCheckStatusNULL && prAtIDReadStatusTrue && prExitSystem	inExitSystem

In parallel to the elaboration of the ***Rules*** **Table**, the *Component Model* was generated in a creative synthesis activity subdivided in three steps: 1. *Define Rules*; 2. *Define Premises and Instigations*; 3. *Associate Rules to FBEs* [WIECHETECK, 2011].

The existence of NOR modeling facilitated the *Rules* definition and their interdependencies. For example, the requirement modeled in NOR *"Protect Secure Areas"* (figure 3) which is a disjunction (<<*disjunction*>>) between "*ID Fail*", "*Time Fail*", and "*BIO Fail*", was modeled by the NOP *Rule rlVerifyTotalAccessDenied*, as a disjunction between the equivalent corresponding premises *prAtIDCheckStatusFalse*, *prAtBIOCheckStatusFalse,* and *prAtTimeDifferenceStatusFalse* (table 1).

Seventeen (17) *Component Models* were created (one for each *Rule*). The model for the *Rule rlReadID* is illustrated as an example in the figure 8. It is possible to notice the behavior of the *Rule* (*rlReadID*), its *Premises* (*prAtIDReadStatusFalse*), its *Instigations* (*inVerifyEmployeeID*, *inReadID inSetAtIDReadStatusTrue*), and the methods (*mtVerifyEmployeeID*, *mtReadID*, *mtSetIDReadStatusTrue*) that may be triggered in the *FBEs* (*EmployeeController*, *ID_Checker*). Besides that, it is possible to observe the *Attributes* used by the *Rule* (*atIDReadStatus*, *atCurrentReadID, AtEmployeeID*).

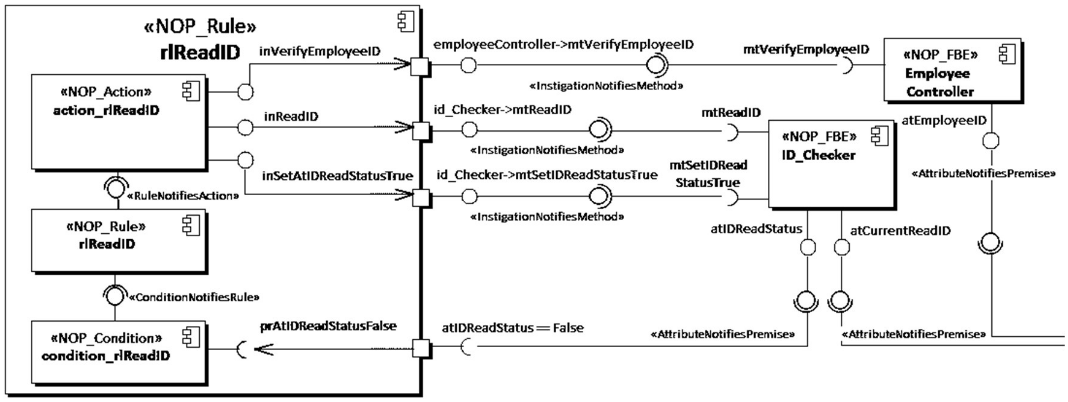

Figure 8. *Rule rlReadID* from Component Model (NOD)

The *Petri Net Model* used in NOD method demonstrates the dynamics between NOP elements. Petri Nets (PN) allows to model, simulate and even verify concurrency and synchronization of resources in systems [CARDOSO and VALETTE, 1997].

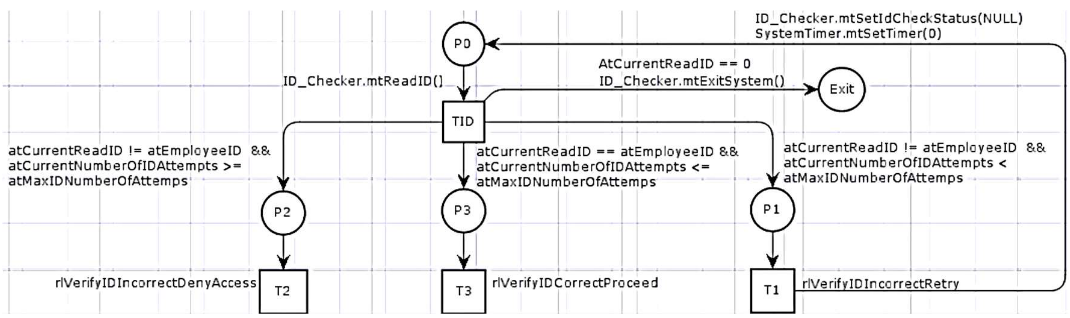

Figure 9. Part of the modeled Petri Net Model (NOD)

In figure 9, part of the **Petri Net** modeled for this study is shown, in which the NOP *Rules* were mapped as PN **transitions:** *rlReadID* (TID)*, rlVerifyIDIncorrectDenyAccess* (T2), *rlVerifyIDCorrectProceed* (T3) and *rlVerifyIDIncorrectRetry* (T1). The PN **places** (P1, P2, P3, etc.) are representing the *Premises* of each *Rule*. In this model it is possible to notice the concurrency between the *Rules* (T1, T2, T3), which is a characteristic of several NOP applications. In the current study, the *Petri Net Model* was not executed for validation, which is a possibility for future works.

6. NOP Framework C++ 2.0 Implementation

The system implementation was performed in *Visual Studio 2017®* (*Microsoft*) tool, according to established development standards for *NOP Framework C++ 2.0* [RONSZCKA et al., 2017].

Figure 10 (a) shows the NOP C++ codes for the *Rules rlVerifyTotalAccessGranted* and *rlVerifyTotalAccessDenied*, in which is possible to observe the *Premises* and *Instigations* of each Rule. The *Rule rlVerifyTotalAccessGranted* is the same as previously shown in figure 1 and table 1.

Figure 10 (b) shows an execution *prompt command* regarding the NOP application, in which is possible to notice some approved *Rules* (e.g.: *rlReadID, rlVerifyIDCorrectProceed, rlReadBIO*) that executed the corresponding *Methods* (e.g.: *mtSetIDReadStatusTrue*, *mtSetIDCheckStatusTrue*).

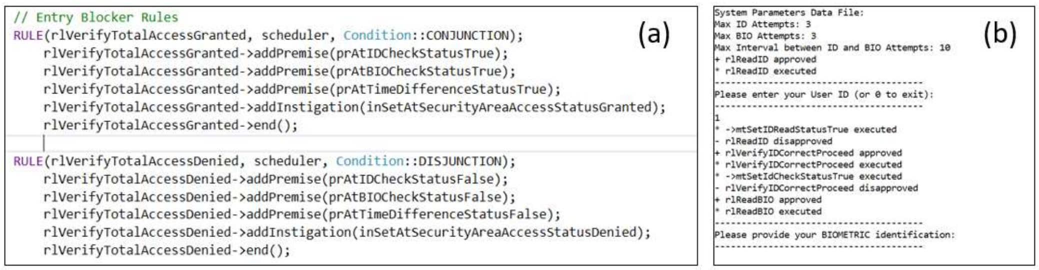

Figure 10. (a) Part of NOP Rules Code (b) NOP Application Execution

Similarly to what was reported in [MENDONÇA et al., 2015] and [WIECHETECK, 2011], it is noted that the creation of a well-structured NOD project allows a near-mechanical implementation in the NOP *C ++ 2.0 Framework*, as may be observed in the corresponding artifacts and codes. Likewise, the NOR project facilitated the project in NOD, due to the visual inputs provided by the requirements model.

7. Discussion and Conclusion

In this case study, the integration between the NOR model into the NOD method was performed during the development of a NOP application. By presenting system requirements graphically, NOR aids to the development of NOP software in three steps:

1. In the **Class Model** creation (system structure).
2. In the **High-Level States Model** creation (system behavior).
3. In the ***Rules* Table** and **Component Model** creation (specific tailored behavior of NOP application *Rules*)

Therefore, it can be concluded that the NOR modeling can be harmoniously integrated to the NOD method, facilitating the development of Software Engineering design for NOP applications. This would lead to future works on engineering-oriented requirements models such as SysML. It is also suggested the refinement of NOR modeling techniques through a specific study of interrelationships between requirements.

Acknowledgements

The authors would like to thank CAPES/CNPq from Brazil by the financial support and to UTFPR by all the support and infrastructure provided.

Referências

[CARDOSO and VALETTE, 1997] Janette Cardoso e Robert Valette. *Petri Nets.* Original title: Redes de Petri. Florianópolis, Ed. da UFSC, p. 212, 1997.

[FERREIRA, 2015]. Cleverson A. Ferreira. *Language and Compiler for the Notification Oriented Paradigm*: *Advances and Comparisons*. Original title: Linguagem e

Compilador para o Paradigma Orientado a Notificações: Avanços e Comparações. Dissertação de Mestrado. PPGCA/UTFPR, Curitiba, Brasil, 2015.

[FRIEDENTHAL et al., 2014] Sanford Friedenthal, Alan Moore, Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language.* The Morgan Kaufmann / OMG Press, 3rd Ed., 2014.

[GABBRIELLI and MARTINI, 2010] Maurizio Gabbrielli e Simone Martini. *Programming Languages: Principles and Paradigms. Series: Undergraduate Topics in Computer Science*. 1st Edition, XIX, 440 p., Softcover, 2010.

[INCOSE, 2006] INCOSE *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. International Council on Systems Engineering, C. Haskins. (Ed.), Version 3, 2006.

[MENDONÇA et al., 2015] Igor T. M. Mendonça, Jean M. Simão, Luciana V. B. Wiecheteck, Paulo C. Stadzisz. *Development Method for Rule-Based Systems using the Notification Oriented Paradigm.* Original title: Método para Desenvolvimento de Sistemas Orientados a Regras utilizando o Paradigma Orientado a Notificações. Cong. Bras. Inteligência Computacional, (12), 1–6., 1 CD–ROM. Curitiba, Brasil, 2015.

[RONSZCKA et al., 2017] Adriano F. Ronszcka, Glauber Z. Valença, Robson R. Linhares, João A. Fabro, Paulo C. Stadzisz, Jean M. Simão *Notification-Oriented Paradigm Framework 2.0: An Implementation Based on Design Patterns. IEEE Latin America Transactions*, Volume 15, Issue 11, pp. 2221-2232, Nov 2017.

[SANTOS, 2017] Leonardo A. Santos. *Language and Compiler for Notification Oriented Paradigm*: *advances in coding and validation for a Robotic Soccer application.* Original title: Linguagem e compilador para o paradigma orientado a notificações: avanços para facilitar a codificação e sua validação em uma aplicação de controle de futebol de robôs. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, Brasil, 2017.

[SIMÃO and STADZISZ, 2008] Jean M. Simão e Paulo C. Stadzisz. *Notification Oriented Paradigm – A Technique for Notification Oriented Software Composition and Execution.* Original title: Paradigma Orientado a Notificações – Uma Técnica de Composição e Execução de Software Orientado a Notificações. Patente. INPI, 2008.

[SIMÃO et al., 2016] Jean M. Simão, Hervé Panetto, Yongxin Liao, Paulo C. Stadzisz. *A Notification-Oriented Approach for Systems Requirements Engineering*. 23rd IPSE International Conference on Transdisciplinary Engineering, Curitiba, Brazil. IOS Press, 4, pp.229-238, Oct 2016.

[WIECHETECK et al., 2011] Luciana V. B. Wiecheteck, Jean M. Simão, Paulo C. Stadzisz. *A UML Profile for Notification Oriented Paradigm.* Original title: Um Perfil UML para o Paradigma Orientado a Notificações. *Anais do III Congreso Internacional de Computación y Telecomunicaciones*, pp. 1-16., Peru, 2011.

[WIECHETECK, 2011] Luciana V. B. Wiecheteck. *Software Development Method using the Notification Oriented Paradigm.* Original title: Método para Projetos de Software usando o Paradigma Orientado a Notificações. *Master Thesis*, CPGEI/UTFPR. Curitiba, Brasil, 2011.

[YOUNG, 2004] R. R. Young. *The Requirements Engineering Handbook*. 1st Ed., Artech House, Boston, 2004.