Towards Integrated Modelling of Dynamic Access Control with UML and Event-B

Inna Vistbakka Elena Troubitsyna
Åbo Akademi University,
Turku, Finland
inna.vistbakka@abo.fi elena.troubitsyna@abo.fi

Role-Based Access Control (RBAC) is a popular authorization model used to manage data-access constraints in a wide range of systems. RBAC usually defines the static view on the access rights. However, to ensure dependability of a system, it is often necessary to model and verify state-dependent access rights. Such a modelling allows us to explicitly define the dependencies between the system states and permissions to access and modify certain data. In this paper, we present a work-in-progress on combining graphical and formal modelling to specify and verify dynamic access control. The approach is illustrated by a case study – a reporting management system.

1 Introduction

The Role-Based Access Control (RBAC) is a de-facto standard mechanism for specifying the data access policies in a large variety of computer-based systems [6]. A role represents a job function in the context of an organization and has associated authorities. The RBAC policy of a system specifies the authorities granted to each role. Usually RBAC gives a static view on the access rights associated with each role, i.e., it defines the permissions to manipulate certain data without referring to the system state. However, such a static view is often insufficient for ensuring system dependability because it might undermine dynamic data integrity properties. In this paper, we present a work-in-progress aiming at combining graphical and formal modelling to represent the dynamic access policy.

A graphical representation is often used as a front-end of a formal model [23, 24, 29, 30]. It facilitates a transition from an informal, natural language, requirements description to their formal representation. In our approach, graphical modelling is used to represent the system functions and business logic associated with them. We rely on the use case model to represent the system roles and functions. The activity diagram shows the workflow associated with the defined functions. From the graphical representation we make a transition to a formal model. We use Event-B [2] as our formal modelling framework. Event-B is a state-based modelling notation. It supports a top-down development approach to correct-by-construction system development. The system development consists of a sequence of correctness-preserving model transformations – refinements. Correctness of models and refinements can be verified by proofs. Rodin platform [22] automates modelling and verification in Event-B.

While creating Event-B specifications, we consult the graphical models to define the roles and the corresponding functions. The dynamic access rights are modelled as enabling or disabling certain operations on data, which depend on the system state. Formal modelling allows us to ensure that the dynamic access rights are preserved throughout the system workflow.

The paper is structured as follows: in Section 2, we define the main concepts of dynamic access policy and present our case study – reporting management system. We define the corresponding graphical models modelling roles, use cases and the workflow. In Section 3, we give a brief introduction into Event-B. In Section 4, we present the formal models that have been constructed on the basis of the graphical

R. Laleau, D. Méry, S. Nakajima, E. Troubitsyna (Eds): Joint Workshop on Handling IMPlicit and EXplicit knowledge in formal system development and Formal & Model-Driven Techniques for Developing Trustworthy Systems.
EPTCS 271, 2018, pp. 105–116, doi:10.4204/EPTCS.271.8

models. We discuss how to represent the dynamic access rights in Event-B. Finally, in Section 5, we overview the related work and discuss the proposed approach.

2 Towards Reasoning About Dynamic Access Policy

In this section, we will define the main notions and primitives required to reason about behaviour of a system using the dynamic access control model. We will discuss the notions of users and their roles, data and the rights to manipulate data. We will also introduce some functions and relations to define the required inter-relationships between these notions.

Let $USERS = \{u_1, u_2, ..., u_n\}$ be a set of users. The users are core elements of the system. In general, a user may stand for a person in the organisation, an administrative entity or a non-person entity, such as a computing (sub)system. We use the term user to cover all the cases.

Let $ROLES = \{r_1, r_2, ..., r_k\}$ be a set of possible user roles within the system. A role is usually seen as a job function performed by a user within an organisation. For example, in a security model, a role is used to indicate the job-related access rights to the data.

Let $RIGHTS = \{ri_1, ri_2, ..., ri_m\}$ be a set of basic access rights that can be defined over the data, e.g., *create, read, write* rights. Moreover, $DATA = \{d_1, d_2, ..., d_l\}$ denotes a set of data entities. The users can access data only by executing operations on a data entity that are regulated by corresponding basic rights. Operations and data are predefined by the underlying system for which RBAC is specified.

The users can access data based on the set of assigned roles. A user authorisation list can be defined as the mapping between users and roles:

$$UR\_Rel : USERS \rightarrow \mathbb{P}(ROLES),$$

which assigns a set of user roles to a given user. The notation $\mathbb{P}(ROLES)$ stands for the powerset (set of all subsets) type over elements of the type *ROLES*.

To formally define access rights to the data provided by the system to different user roles, we define a function *RR_Rel* that maps each user role to a set of the allowed rights:

$$RR\_Rel : ROLES \rightarrow \mathbb{P}(RIGHTS).$$

The above definition describes what a user with a specific role is allowed to do with data in general. However, it does not take into account the system workflow, i.e., it abstracts away from the fact that the access rights over a data entity for each role also depend on the system state. To demonstrate how this issue can be addressed via formal modelling and verification in Event-B, let us now consider an example of a system with a dynamic access control. Then we will show how we can create a specification of such a system and prove the correctness of its behaviour.

A Case Study – Reporting Management System. To illustrate modelling opportunities of Event-B with combination of UML, we use a simple case study – a reporting management system.

The Reporting Management System (RMS) is used by different kinds of employees in an organisation to send periodic (e.g., monthly) work reports. The system has the following requirements:

- *The system has three roles* – reporter, controller and administrator;
  - Every reporter is associated with (supervised by) some controller;
  - Every controller can have more than one associated reporter;

  - Every controller is associated with (supervised by) one administrator;
  - Every administrator can have more than one associated controller.

- *Functionality associated with the roles*:
  - A reporter can create a new report, modify an existing report or delete a not-approved report, as well as submit a report to its associated controller;
  - A controller can read submitted report received from one of the associated reporters, and can either approve or disapprove it;
  - An administrator has an access to all the reports of all her/his associated controllers, and it is her/his responsibility to register reports approved by the controllers.

- *Report access policies*:
  - Until a report is submitted, the reporter can modify or delete it.
  - As soon as a report is submitted, it cannot be altered or deleted by the reporter any longer.
  - Upon controller's approval, the report is registered by the administrator.
  - In case of disapproval, the report is returned back to the reporter and can be further modified or deleted.

The use case diagram of RMS is presented in Figure 1. It shows the actors, their roles in the system and also their possible interactions with the system. The activity diagram for RMS's workflow is presented in Figure 2.

Let us note that each actor function requires certain basic access rights. For instance, a reporter, to be able to execute *Modify* function, should have *read* and *write* access rights to the report file. In its turn, a reporter's supervisor – controller – should have *read* and *write* access rights to the same file to execute *Approval* operation. However, as soon as a reporter submit a report to a controller, she/he can have only *read* access right to the report file.

In Section 4, we will present a refinement-based development of RMS. We will discuss how to represent dynamic access rights in Event-B, formally specify all actors' operations over the data and prove the correctness of behaviour of RMS. Before that, we will briefly overview Event-B modelling framework and its refinement approach.

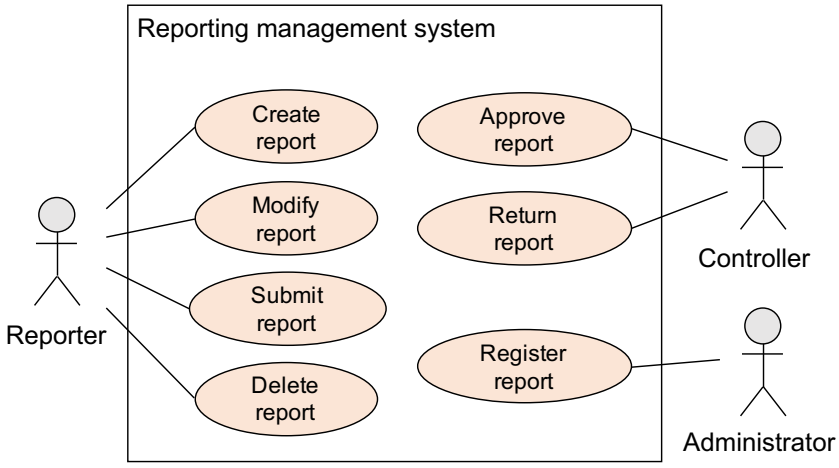

Figure 1: Use Case Diagram – Reporting Management System

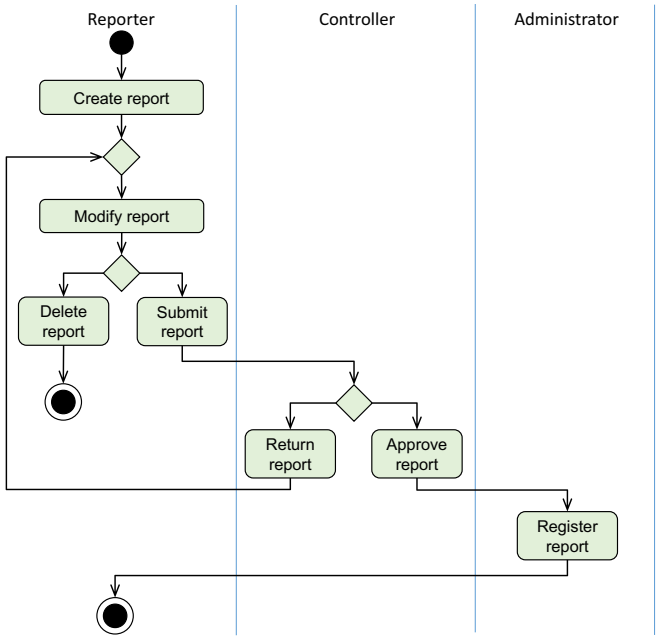

Figure 2: Activity Diagram – Reporting Management System

3 Modelling and Refinement in Event-B

Event-B [2] is a state-based framework that promotes the correct-by-construction approach to system development and formal verification by theorem proving. In Event-B, a system model is specified using the notion of an *abstract state machine* [2]. An abstract state machine encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes the dynamic behaviour of a modelled system. A machine also has an accompanying component, called *context*, which includes user-defined sets, constants and their properties given as model axioms.

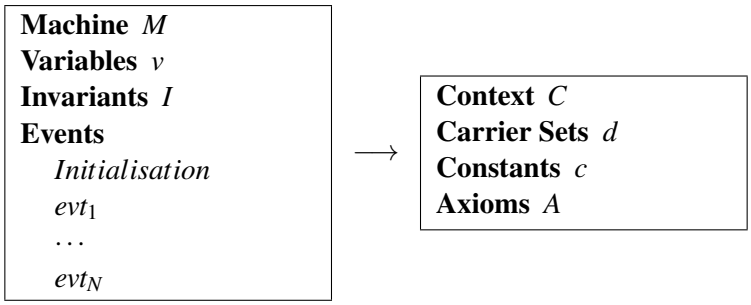

Figure 3: Event-B machine and context

A general form for Event-B models is given in Figure 3. The machine is uniquely identified by its name M. The state variables, v, are declared in the **Variables** clause and initialised in the *Initialisation* event. The variables are strongly typed by the constraining predicates I given in the **Invariants** clause. The invariant clause might also contain other predicates defining essential properties (e.g., safety invariants) that should be preserved during system execution.

The dynamic behaviour of the system is defined by a set of atomic *events*. In general, an event has the following form:

$$e \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G_e \textbf{ then } R_e \textbf{ end},$$

where e is the event's name, a is the list of local variables, the *guard* G_e is a predicate over the local variables of the event and the state variables of the system. The body of an event is defined by a *multiple* (possibly nondeterministic) assignment over the system variables. In Event-B, an assignment represents a corresponding next-state relation R_e. Later on, using the concrete syntax in our Event-B models, we will rely on two kinds of assignment statements: deterministic ones, expressed in the standard form $x := E(x,y)$, and non-deterministic ones, represented as $x :| \mathit{some\_condition}(x,y,x')$. In the latter case, the state variable x gets non-deterministically updated by the value x', which may depend on the initial values of the variables x and y.

The guard defines the conditions under which the event is *enabled*, i.e., its body can be executed. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

If an event does not have local variables, it can be described simply as:

$$e \mathrel{\widehat{=}} \textbf{when } G_e \textbf{ then } R_e \textbf{ end}.$$

Event-B employs a top-down refinement-based approach to system development. Development typically starts from an abstract specification that nondeterministically models the most essential functional requirements. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions. In particular, we can add new events, split events as well as replace abstract variables by their concrete counterparts, i.e., perform *data refinement*.

The consistency of Event-B models, i.e., verification of well-formedness and invariant preservation as well as correctness of refinement steps, is demonstrated by discharging a number of verification conditions – proof obligations. For instance, to verify *invariant preservation*, we should prove the following logical formula:

$$A(d,c),\ I(d,c,v),\ G_e(d,c,x,v),\ R_e(d,c,x,v,v') \vdash I(d,c,v'), \tag{INV}$$

where A are the model axioms, I are the model invariants, d and c are the model constants and sets respectively, x are the event's local variables and v, v' are the variable values before and after event execution. The full definitions of all the proof obligations are given in [2].

The Rodin platform [22] provides an automated support for formal modelling and verification in Event-B. In particular, it automatically generates the required proof obligations and attempts to discharge them. The remaining unproven conditions can be dealt with by using the provided interactive provers.

4 Formal Development of the Reporting Management System in Event-B

In this section, we will present a refinement-based development of the reporting management system discussed in Section 2. We will model all actors' functions as Event-B events. In general, the semantics of each Event-B event is unambiguously defined by a binary relation between all possible *pre-* and *post-states* of the event. Therefore, modelling RMS in Event-B would allow us to define pre- and post-states for every operation with the data, and model the overall system behaviour as a state transition system.

Moreover, our development will incorporate the essential concepts and relationships between system elements described in Section 2. We will separately discuss both static and dynamic system aspects, represented by Event-B contexts and machines respectively.

Reporting Management System: Abstract Model. Let us note that each actor's function changes the state of a certain report. Hence, the overall behaviour of the system, for each particular report, can be considered as a set of transitions between all the possible states of the report. The corresponding state diagram is represented in Figure 4.

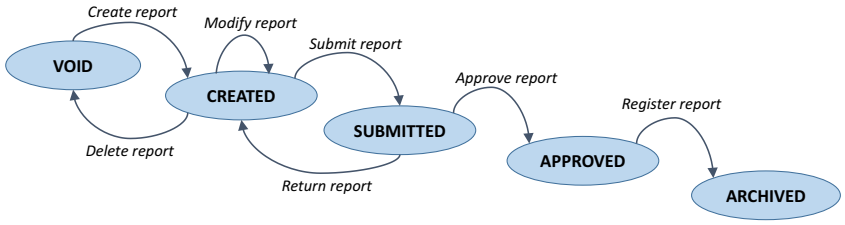

Figure 4: State diagram – Reporting Management System

In our initial Event-B specification, we abstractly specify changing of states of reports in our system. We model the set of reports in the system as a function *report_state*. Initially each report has the state $VOID$. The actual report creation is modelled by the event CreateReport that changes the state of a single report rp to $CREATED$, i.e., $report\_state(rp) := CREATED$. Then the events ModifyReport, DeleteReport and SubmitReport become enabled. When the report is submitted, its state changes to $SUBMITTED$. Upon report approval, its state is changed to $APPROVED$, otherwise, if the report is rejected, it returns back to the state $CREATED$. Finally, once the administrator registers already approved report, the report goes to its final state $ARCHIVED$.

The structure of our initial model – machine RMS_abs – is given in Figure 5. This machine essentially covers all the use cases presented in Figure 1. All required sets and constants are defined in the static part of the model – context RMS_c0 (not presented in the paper).

First Refinement: Introducing Roles. The purpose of this refinement step is to elaborate on the initial system specification and introduce user roles. We will link each role with the set of functions that correspond to it. Moreover, for each role, we will define the required basic access rights – *create, read, write, delete*.

In the context part of Event-B specification, we define a set of actor roles $ROLES = \{Reporter, Controller, Administrator\}$. $RIGHTS$ is the set of basic access rights, where $RIGHTS = \{C, R, W, D\}$.

To specify dynamic access rights for the introduced roles, we define a variable *permissions* with the following properties:

$$permissions \in ROLES \times REPORTS \rightarrow \mathbb{P}(RIGHTS),$$

$$\forall r \in REPORTS \cdot permissions(Reporter, r) \subseteq \{C, W, R, D\} \wedge$$
$$permissions(Controller, r) \subseteq \{R, W\} \wedge permissions(Administrator, r) \subseteq \{R, W\}.$$

The variable *permissions* is a function that assigns to each role and a report a number of possible access rights that can be associated with the role.

Obviously, for each role, the set of available access rights to a report depends on the current state of this report. For instance, a controller can have read (R) and write (W) rights only to the submitted reports. Moreover, if the report that has been submitted for approval, it cannot be further modified by the

Machine RMS_m0
Sees RMS_c0
Variables *reports*, *report_state*
Invariants
$reports \subseteq REPORTS \wedge$
$report\_state \in REPORTS \rightarrow REPORT\_STATES \wedge$
$report\_state[reports] \subseteq REPORT\_STATES \setminus \{VOID\}...$
Events
Initialisation $\widehat{=}$ ...

CreateReport $\widehat{=}$
any rp
where $rp \in DATA$
$report\_state(rp) = VOID$
then
$reports := reports \cup \{rp\}$
$report\_state(rp) := CREATED$
end

ModifyReport $\widehat{=}$
any rp
where $rp \in reports$
$report\_state(rp) := CREATED$
then
$skip$
end

DeleteReport $\widehat{=}$ ...
any rp
where $rp \in reports$
$report\_state(rp) = CREATED$
then
$report\_state(rp) := VOID$
$reports := reports \setminus \{rp\}$
end

SubmitReport $\widehat{=}$
any rp
where $rp \in data$
$report\_state(rp) = CREATED$
then
$report\_state(rp) := SUBMITTED$
end

ApproveReport $\widehat{=}$
any rp
where $rp \in reports$
$report\_state(rp) = SUBMITTED$
then
$report\_state(rp) := APPROVED$
end

ReturnReport $\widehat{=}$ ...

RegisterReport $\widehat{=}$
any rp
where $rp \in reports$
$report\_state(rp) = APPROVED$
then
$report\_state(rp) := ARCHIVED$
end
end

Figure 5: The machine RMS_abs

reporter until the end of the approval period. Therefore, during the approval period, the reporter has only read (R) right to this particular report. Hence, we should restrict the set of enabled rights depending on a report's state. To address this new behaviour, we refine the corresponding events of the abstract model. Some of the refined CreateReport and SubmitReport events are presented in Figure 6.

The model invariants describe the dynamic access policies depending on a report state. These invariants ensure access rights conformity to avoid possible conflicts between the roles. Thereby, it allows us to prove the dynamic data integrity properties within the model. Some invariants are presented in Figure 6.

Further Refinements. In the subsequent refinement steps, we augment the specification with further details. In particular, we introduce for each report a *time window* to model certain periods when a periodic report can be created. We also elaborate on the event ModifyReport to model the possible changes of a

Machine RMS_ref1
Sees RMS_c1
Variables *reports*, *report_state*, *permissions*, ...
Invariants ...

$\forall rp \cdot rp \in REPORTS \wedge report\_state(rp) = VOID \Rightarrow (permissions(Reporter \mapsto rp) = \{C\} \wedge permissions(Controller \mapsto rp) = \varnothing \wedge permissions(Administrator \mapsto rp) = \varnothing)$

$\forall rp \cdot rp \in reports \wedge report\_state(rp) = CREATED \Rightarrow (permissions(Reporter \mapsto rp) = \{R,W,D\} \wedge permissions(Controller \mapsto rp) = \varnothing \wedge permissions(Administrator \mapsto rp) = \varnothing)$

$\forall rp \cdot rp \in reports \wedge report\_state(rp) = SUBMITTED \Rightarrow (permissions(Reporter \mapsto rp) = \{R\} \wedge permissions(Controller \mapsto rp) = \{R,W\} \wedge permissions(Administrator \mapsto rp) = \varnothing)$

$\forall rp \cdot rp \in reports \wedge report\_state(rp) = APPROVED \Rightarrow (permissions(Reporter \mapsto rp) = \{R\} \wedge permissions(Controller \mapsto rp) = \{R\} \wedge permissions(Administrator \mapsto rp) = \{R,W\})$

...
Events...
CreateReport **refines** CreateReport $\widehat{=}$
any *rp*
where ...
$C \in permissions(Reporter, rp)$
then
$reports := reports \cup \{rp\}$
$report\_state(rp) := CREATED$
$permissions(Reporter, rp) := \{R,W,D\}$
end
SubmitReport **refines** SubmitReport $\widehat{=}$
any *rp*
where ...
$R \in permissions(Reporter, rp)$
then
$report\_state(rp) := SUBMITTED$
$permissions := (permissions \lhd (\{Reporter \mapsto rp \mapsto \{R\}\} \cup \{Controller \mapsto rp \mapsto \{R,W\}\}))$
end
...
end

Figure 6: The machine RMS_ref1

report before its submission. In particular, we define the notion of report timestamps to keep track on time.

Next, we populate our model with the users and introduce a number of inter-relationships between the system users and their roles as well as relationships between the users. For instance, a reporter sends a report for approval to her/his associated controller while the controller sends the approved report to the associated administrator.

As a result of the described refinement chain, we arrive at a final model of RMS. We specify and verify dynamic access control via allowed rights on data according to the system policies.

Summary. A development of RMS discussed above follows the certain strategy. The proposed approach can be summarised as follows:

1. Define main roles and their functions associated with the system. Represent actors as roles, functions as use cases and create a use case model of the system.
2. Create an activity diagram representing the intended workflow.
3. For each use case define the basic access rights required to execute this particular function.
4. Create a state diagram representing how execution of each function changes the state of the data.
5. Using the created state diagram, create an abstract specification in Event-B that defines the state of data and the corresponding state transitions.
6. Using the use case diagram, refine the abstract specification to define roles and the corresponding access rights.
7. Using the activity diagram, refine the specification to represent the workflow and time-dependant properties of dynamic state-based RBAC.

5 Conclusions

Related Work. Significant amount of work has been done on integrating graphical and formal specification techniques to support system development (see e.g., [5, 11, 13, 27]). In particular, the combination UML with Petri Nets is discussed in [5]. The paper shows a method for translating UML sequence diagrams to Petri nets and verifying deadlock freeness, reachability, safety and liveness properties. The work [11] presents a framework for integration UML with Object-Z. Various kinds of UML diagrams are used to specify the system from different concerns during the requirements elicitation and analysis stage. Then the captured information is used to develop a complete Object-Z specification. In our work, we follow the same idea and consult the graphical models while creating Event-B specifications. In [26, 27] UML-B – a graphical formal modelling notation – has been proposed to support class diagrams and state machines concepts within the Event-B development. However, the goal of our research is to consider possible combination UML and Event-B modelling in the context of access control model and RBAC policies.

The topic of combining graphical domain-specific notation with formal Event-B models has also been explored in a number of works focusing on modelling dependable systems [18, 19, 23, 24, 29, 30]. The overall goal pursued in these work were to facilitate construction of a formal model by relying on a suitable graphical notation.

The importance of RBAC visualization has been recognized by Jaeger and Tidswell [10]. A body of research done on applying UML to describe and analyse RBAC policies [9, 21, 28]. A number of works uses UML and OCL based domain specific language to design and validate the access control model. For instance, in the work [9] the authors applied UML and OCL to discover and eliminate undesired policy properties, which do not meet the security requirements. In [28] UML is also used to describe security properties. In contrast to our work, here the authors transform UML models to Alloy for analysis purpose.

There is a number of works that address the policy analysis and verification issues related to RBAC model. For instance, the problem of inconsistent access control specifications is studied in [25]. To verify the correctness of event-driven RBAC policies a Petri-Net based framework has been applied. Similar to our work, Event-B has been applied to model and analyse access control policies [3, 8]. Akeel et al.

[3] have studied the problem of data leakage threats in the access control model. They used Event-B and associated refinement approach to formalise the requirements over the specific policy elements that satisfy Confidentiality, Privacy and Trust properties. In our work we use Event-B to formulate and prove the dynamic data integrity properties. The challenge of integrating RBAC into systems modelling and verification has been addressed also by Benaïssa et al. in [4]. They start from a security policy description given in a Prolog-like formalism (OrBAC), and refine the description into an Event-B model capturing the system-specific activities to be verified under the policy.

The basic RBAC model has been extended in a variety of ways [1, 7, 20]. For instance, the problem of spatio-temporal RBAC model is discussed in [1]. The authors considered role-based access control policies under time and location constraints. Moreover, they demonstrated how the proposed model can be represented and analysed using UML and OCL. Ray et al. [20] proposed location-aware RBAC model that incorporates location constraints in user-role activation. In our work we consider dynamic, state-dependent constraints within the access control model. In [12] the interactions between agents have been studied using goal-oriented perspective. In this work, the roles were defined as agent capabilities to perform certain tasks.

In our previous work [14] to facilitate development in Event-B we relied on the Event Refinement Structure approach (ERS). This approach augments Event-B refinement with a graphical notation that allows the designers to explicitly represent the relationships between the events at different abstraction levels as well as define the required event sequences in a model. In current work we used graphical models as a middle hand to construct Event-B specifications.

Verification of data integrity and data consistency properties in Event-B framework has also been investigated in [15] in the context of cloud data base. Moreover, in [16, 17] we verified by proofs correctness and safety of consistent updates of patient data records. However, the current work is mainly focused on possible combination UML and Event-B to specify and verify dynamic access control.

Discussion. In this paper, we have discussed a problem of an integrated modelling of dynamic access rights. Our approach aimed at combining graphical modelling and formal specification in Event-B. The used graphical models to define in a structured way the roles associated with the system, use cases and workflow. The graphical models were used as a middle hand to construct formal models in Event-B.

Formal modelling in Event-B allowed us to rigorously define and verify the dynamic access rights and formulate and prove the dynamic data integrity properties. In this paper, we merely consulted the graphical models to create the corresponding specifications. Currently, we are working on defining graphical and formal specification patterns for representing the dynamic access rights. Such patterns would allow us to automatically translate graphical models into the corresponding formal models and consequently enable the formal verification of dynamic data integrity properties.

Moreover, as a part of our future work, we are also planing to further extend the current work and consider a more complex setting of access model. In particular, it would be interesting to investigate the situation when the users can get simultaneous or partial access to some parts of a data entity depending on their roles and resource states.

References

[1] Ramadan Abdunabi, Mustafa Al-Lail, Indrakshi Ray & Robert B. France (2013): *Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model*. IEEE Systems Journal 7(3), pp. 501–515, doi:10.1109/JSYST.2013.2242751.

[2] Jean-Raymond Abrial (2010): *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, doi:10.1017/CBO9781139195881. Available at `http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521895569`.

[3] Fatimah Y. Akeel, Asieh Salehi Fathabadi, Federica Paci, Andrew M. Gravell & Gary Wills (2016): *Formal Modelling of Data Integration Systems Security Policies*. *Data Science and Engineering* 1(3), pp. 139–148, doi:10.1007/s41019-016-0016-y.

[4] Nazim Benaïssa, Dominique Cansell & Dominique Méry (2007): *Integration of Security Policy into System Modeling*. In: *B 2007: Formal Specification and Development in B, 7th International Conference of B Users, Proceedings*, *Lecture Notes in Computer Science* 4355, Springer, pp. 232–247, doi:10.1007/11955757_19.

[5] E. Cunha, Marcelo Custódio, Herbert Rocha & Raimundo S. Barreto (2011): *Formal Verification of UML Sequence Diagrams in the Embedded Systems Context*. In: *Brazilian Symposium on Computing System Engineering, SBESC 2011*, IEEE Computer Society, pp. 39–45, doi:10.1109/SBESC.2011.18.

[6] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn & Ramaswamy Chandramouli (2001): *Proposed NIST standard for role-based access control*. *ACM Trans. Inf. Syst. Secur.* 4(3), pp. 224–274, doi:10.1145/501978.501980.

[7] Ludwig Fuchs, Günther Pernul & Ravi S. Sandhu (2011): *Roles in information security - A survey and classification of the research area*. *Computers & Security* 30(8), pp. 748–769, doi:10.1016/j.cose.2011.08.002.

[8] Thai Son Hoang, David Basin & Jean-Raymond Abrial (2009): *Specifying Access Control in Event-B*. *Technical report* 624, doi:10.3929/ethz-a-006733720.

[9] Oliver Hofrichter, Martin Gogolla & Karsten Sohr (2013): *UML/OCL based Design and Analysis of Role-Based Access Control Policies*. In: *(GEMOC 2013) and (AMINO 2013) Co-located with (MODELS 2013)*, *CEUR Workshop Proceedings* 1102, CEUR-WS.org, pp. 33–42. Available at `http://ceur-ws.org/Vol-1102/amino2013_submission_2.pdf`.

[10] Trent Jaeger & Jonathon Tidswell (2001): *Practical safety in flexible access control models*. *ACM Trans. Inf. Syst. Secur.* 4(2), pp. 158–190, doi:10.1145/501963.501966.

[11] Soon-Kyeong Kim & David A. Carrington (2000): *An integrated framework with UML and Object-Z for developing a precise and understandable specification: the light control case study*. In: *7th Asia-Pacific Software Engineering Conference (APSEC 2000)*, IEEE Computer Society, pp. 240–248, doi:10.1109/APSEC.2000.896705.

[12] Linas Laibinis, Inna Pereverzeva & Elena Troubitsyna (2017): *Formal reasoning about resilient goal-oriented multi-agent systems*. *Science of Computer Programming* 148, pp. 66–87, doi:10.1016/j.scico.2017.05.008.

[13] Hung Ledang & Jeanine Souquières (2002): *Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B*. In: *9th Asia-Pacific Software Engineering Conference (APSEC 2002)*, IEEE Computer Society, p. 495, doi:10.1109/APSEC.2002.1183053.

[14] Inna Pereverzeva, Michael J. Butler, Asieh Salehi Fathabadi, Linas Laibinis & Elena Troubitsyna (2014): *Formal Derivation of Distributed MapReduce*. In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014. Proceedings*, *Lecture Notes in Computer Science* 8477, Springer, pp. 238–254, doi:10.1007/978-3-662-43652-3_21.

[15] Inna Pereverzeva, Linas Laibinis, Elena Troubitsyna, Markus Holmberg & Mikko Pöri (2013): *Formal Modelling of Resilient Data Storage in Cloud*. In: *Formal Methods and Software Engineering - 15th International Conference on Formal Engineering Methods, ICFEM 2013*, *Lecture Notes in Computer Science* 8144, Springer, pp. 363–379, doi:10.1007/978-3-642-41202-8_24.

[16] Inna Pereverzeva, Elena Troubitsyna & Linas Laibinis (2012): *Formal Development of Critical Multi-agent Systems: A Refinement Approach*. In: *2012 Ninth European Dependable Computing Conference, 2012*, IEEE Computer Society, pp. 156–161, doi:10.1109/EDCC.2012.24.

[17] Inna Pereverzeva, Elena Troubitsyna & Linas Laibinis (2013): *A refinement-based approach to developing critical multi-agent systems*. *IJCCBS* 4(1), pp. 69–91, doi:10.1504/IJCCBS.2013.053743.

[18] Yuliya Prokhorova, Linas Laibinis & Elena Troubitsyna (2015): *Facilitating construction of safety cases from formal models in Event-B*. Information & Software Technology 60, pp. 51–76, doi:10.1016/j.infsof.2015.01.001.

[19] Yuliya Prokhorova & Elena Troubitsyna (2012): *Linking Modelling in Event-B with Safety Cases*. In: Software Engineering for Resilient Systems - 4th International Workshop, SERENE 2012. Proceedings, Lecture Notes in Computer Science 7527, Springer, pp. 47–62, doi:10.1007/978-3-642-33176-3_4.

[20] Indrakshi Ray, Mahendra Kumar & Lijun Yu (2006): *LRBAC: A Location-Aware Role-Based Access Control Model*. In: Information Systems Security, Second International Conference, ICISS 2006, Proceedings, Lecture Notes in Computer Science 4332, Springer, pp. 147–161, doi:10.1007/11961635_10.

[21] Indrakshi Ray, Na Li, Robert B. France & Dae-Kyoo Kim (2004): *Using uml to visualize role-based access control constraints*. In: 9th ACM Symposium on Access Control Models and Technologies, SACMAT, ACM, pp. 115–124, doi:10.1145/990036.990054.

[22] Rodin: *Event-B platform*. Available at `http://www.event-b.org/`.

[23] Kaisa Sere & Elena Troubitsyna (1999): *Hazard Analysis in Formal Specification*. Lecture Notes in Computer Science 1698, Springer, pp. 350–360, doi:10.1007/3-540-48249-0_30.

[24] Kaisa Sere & Elena Troubitsyna (1999): *Safety Analysis in Formal Specification*. In: FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Proceedings, Volume II, Lecture Notes in Computer Science 1709, Springer, pp. 1564–1583, doi:10.1007/3-540-48118-4_33.

[25] Basit Shafiq, Ammar Masood, James Joshi & Arif Ghafoor (2005): *A Role-Based Access Control Policy Verification Framework for Real-Time Systems*. In: 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2005), IEEE Computer Society, pp. 13–20, doi:10.1109/WORDS.2005.11.

[26] Colin F. Snook & Michael J. Butler (2006): *UML-B: Formal modeling and design aided by UML*. ACM Trans. Softw. Eng. Methodol. 15(1), pp. 92–122, doi:10.1145/1125808.1125811.

[27] Colin F. Snook & Michael J. Butler (2008): *UML-B: A Plug-in for the Event-B Tool Set*. In: Abstract State Machines, B and Z, First International Conference, ABZ 2008. Proceedings, p. 344, doi:10.1007/978-3-540-87603-8_32.

[28] Wuliang Sun, Robert B. France & Indrakshi Ray (2011): *Rigorous Analysis of UML Access Control Policy Models*. In: POLICY 2011, IEEE International Symposium on Policies for Distributed Systems and Networks, IEEE Computer Society, pp. 9–16, doi:10.1109/POLICY.2011.30.

[29] Elena Troubitsyna (2003): *Integrating Safety Analysis into Formal Specification of Dependable Systems*. In: 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), CD-ROM/Abstracts Proceedings, IEEE Computer Society, p. 215, doi:10.1109/IPDPS.2003.1213394.

[30] Elena Troubitsyna (2008): *Elicitation and Specification of Safety Requirements*. In: The Third International Conference on Systems, ICONS 2008, IEEE Computer Society, pp. 202–207, doi:10.1109/ICONS.2008.56.