A State-based Refinement Technique for Event-B

Alexey Khoroshilov
ISP RAS
NRU Higher School of Economics
Lomonosov Moscow State University
Moscow Institute of Physics and Technology
Moscow, Russia
khoroshilov@ispras.ru

Victor Kuliamin
ISP RAS
NRU Higher School of Economics
Lomonosov Moscow State University
Moscow, Russia
kuliamin@ispras.ru

Alexander K. Petrenko
ISP RAS
NRU Higher School of Economics
Lomonosov Moscow State University
Moscow, Russia
petrenko@ispras.ru

Ilya Shchepetkov
ISP RAS
Moscow, Russia
shchepetkov@ispras.ru

***Abstract*—Formal models can be used to describe and reason about the behavior and properties of a given system. In some cases, it is even possible to prove that the system satisfies the given properties. This allows detecting design errors and inconsistencies early and fixing them before starting development. Such models are usually created using *stepwise refinement*: starting with the simple, abstract model of the system, and then incrementally refining it adding more details at each subsequent level of refinement. Top levels of the model usually describe the high-level design or purpose of the system, while the lower levels are more directly comparable with the implementation code. In this paper, we present a new, alternative refinement technique for Event-B which can simplify the development of complicated models with a large gap between high-level design and implementation.**

***Index Terms*—formal models, refinement, event-b**

I. Introduction

Formal methods are a group of tools, notations, and techniques for developing and analyzing software and hardware systems. Application of formal methods is usually hard and time-consuming, it requires special training and resource investments, and is often based on assumptions that are not always reasonable or achievable in reality [1]. However, they can help to find subtle errors [2], [3], missed by other tools and techniques, or to formally prove the absence of errors and thus the correctness of the system. Due to this, formal methods are often used during the development of systems, where the cost of error is exceptionally high: safety-critical systems [4], [5], avionics [6], in the fields of information security [7]–[9] and smart contracts [10], etc.

Formal methods are used to create a *rigorous* mathematical model of a target system at a certain *abstraction* level, which usually consists of:

- state of the system;
- properties of the system formulated as requirements on its state;
- the behavior of the system in the form of operations that transition the system from one state to another.

Formal models are written in languages with precisely defined syntax and semantics. This enables the usage of various automated tools to analyze models in order to find errors or to prove their absence. Another important aspect of formal models is abstraction [11]: omitting non-essential parts of the system from the model that are not relevant to the checked properties (or requirements). It simplifies the development and analysis since large and complex models are both hard to create and reason about.

Abstraction can be used even if the goal is to model the system in its entirety, with all details and features accurately represented. In this case, it is used in conjunction with *stepwise refinement* [12]. Refinement suggests starting with the model that contains either description of a small simple part of the system, or an abstract, not fully detailed, description of the whole system. Then the description of the other parts of the system, or missing details, should be added into separate *levels* of the model. This way, each subsequent level refines the previous one and becomes a more accurate representation of the whole system. The final level of the model should describe the system at the desired level of abstraction, which can be as low as required: it can even directly correspond to the implementation code.

The correctness of the model should be checked at each refinement level using available automated or interactive tools. Since each level is usually far simpler than the whole model, the task of finding errors (or proving their absence) becomes significantly more manageable. But, in addition to checking levels separately, it is also required to demonstrate that refinement itself was performed correctly and following established rules. If some property or requirement was established to be true on some refinement level, and the refinement itself was correct, then it is guaranteed that this property will also be true on all the following levels.

Particular details depend on the chosen formal method, but these rules are generally quite restrictive. It is often possible

The research was carried out with funding from the Ministry of Science and Higher Education of the Russian Federation (the project unique identifier is RFMEFI60719X0295).

to create a viable abstract model of the system and then not being able to refine it to the desired level of detail. So, a refinement plan needs to be thought out well far in advance, and even then it is not guaranteed that you will not get stuck at a certain level of refinement and will not be forced back to do over some parts of the model.

During each level, it is required to correctly refine both the state of the model and its operations (which describe the behavior of the target system). Refinement of operations usually possesses the biggest challenge, since its rules are the most restrictive. Due to this, it is usually very difficult to create a model in which operations differ greatly between refinement levels [13], even if ultimately they describe the same behavior. Such models require various workarounds, which make it hard to understand and to use verification tools to find errors.

In this paper, we present a new, alternative refinement technique aimed at facilitating the development of formal models when drastically different levels of abstraction are needed. It is solely state-based: no refinement of operations is performed at any stage. Despite this, the technique is still sound and supports the usage of automated verification tools to check the correctness of the model.

The proposed technique is intended to be used with the formal method Event-B, which will be outlined in the next section. Section 3 describes the particular rules of the refinement in the prism of its Event-B implementation. Section 4 presents the state-based refinement technique, including both its inner theoretical details and its practical implementation as a part of the Event-B framework. Section 5 provides a practical evaluation of the suggested technique. Related work is reviewed in Section 6. The last section concludes the paper and considers future work.

II. Event-B

Event-B [14] is a formal method and associated language of creating formal models. It is mathematical in nature and is based on predicate logic and set theory. Event-B is used to describe the behavior and properties of systems that can be modeled as *discrete transition systems*.

Each Event-B model consists of *contexts* and *machines*. Contexts describe the static properties of the system: elements that are not changed in time. These properties are expressed in the form of *carrier sets*, *constants*, and *axioms*.

Carrier sets are used to denote the basic types of the model. The values of different types are completely independent and can not be compared with each other. Each constant must have a type, which can be fairly complex: it can be an element of some carrier set, a subset, an ordered pair, a relation between sets, and also all possible combinations of them. Axioms are predicates that are used to specify both the type of constants and additional properties that constants may posses. So, axioms can reference both constants and carrier sets.

Machines describe the behavior of a modeled system in the form of states, properties of the states, and transitions between states. States are modeled using *variables* and are formed by their values. Variables must have types: they are declared using *invariants*, which, just like axioms, are predicates. To do so they can reference variables, constants, and carrier sets. Invariants are also used to describe important properties of the modeled system and are expected to be true in every state of the model.

Transitions between states are expressed with *events*. Events may contain *parameters*, *guard conditions* and *actions*. Guard conditions are predicates on state variables and event parameters that must be true for the event to be enabled. Actions are atomic assignments that modify the values of state variables. They can be predetermined, can depend on the current state of the model, and can depend on the values of event parameters. One event can't contain multiple actions that change the same variable. Events are supposed to preserve invariants: if guard conditions are true, then from a safe state (where all invariants are true) event must transition the system to another safe state. This is one of the main things that are checked when the correctness of an Event-B model is investigated.

Event-B models are developed using the Rodin platform [15]. It provides means to create and edit models, as well as to verify their correctness. Rodin is capable of finding syntax and type errors; more serious design errors or inconsistencies can be found fully automatically using integration with the ProB model checker [16].

Model checking is very good at finding errors, but due to the state explosion problem, it is not very applicable to complex models. Rodin also supports *deductive verification*, which consists of generating from a model and its properties (primarily expressed as invariants) a collection of *proof obligations* and proving that they are true. The truth of such obligations implies that the model conforms to its properties. Rodin can use several theorem provers and SMT solvers [17] able to automatically discharge prove obligations. Users can also discharge proof obligations interactively, to assist automated tools with difficult cases. Such interactive proofs are then automatically checked for soundness.

III. Refinement Rules

Detailed models of complex systems can become too difficult to reason about, even for various automated tools. Like many other formal methods, Event-B supports stepwise refinement to decompose models into smaller and easier pieces. But these pieces are not just separate files: refinement is more like a transformation of one model into another through the series of refinement levels, bounded together by precisely defined rules. In Event-B there are the following main rules of refinement:

1) Each new refinement level may add additional carrier sets, constants, axioms, state variables, events, and invariants to the model.
2) The behavior of the new refinement level, represented with its events and bounded by its invariants, must correspond to the behavior of all previous levels. It must not produce state transitions that are impossible and explicitly forbidden by the previous levels.
3) A new refinement level may replace some old abstract variables with the new concrete ones: for example, an

unordered set can be replaced by an ordered array, which may more accurately represent a particular detail of the modeled system. In this case, some special *gluing invariants* must be added, which will relate the values of replaced variables and their replacements.

4) New events can only change the values of variables that were also added on the same refinement level. This is done to guarantee that there will be no way to break old invariants in the following refinement levels.
5) Old events may be *extended* by one or several events of the new refinement level. The extended events may contain additional parameters, guard conditions, and actions. Additional actions may only modify variables that were added on the same refinement level.
6) Old events may be *refined* by one or several events of the new refinement level. Instead of simple extension, where existing guard conditions and actions are left as is, in the refined event they can be rewritten. In this case, it must be explicitly proved that the refined event does only what is allowed to do by the event it refines.
7) If a refined or extended event is enabled (its guard conditions are true) and all the invariants hold, then the guard conditions of the event it refines also must be true.
8) A new refinement level may replace or remove some parameters in its refined events. In this case, some special condition called *witness* must be added to the event, which will relate the values of parameters and their replacements.

These are mainly the rules of machine refinement. Context refinement is far simpler: it is only possible to add new carrier sets, constants, and axioms. In a way, the new context simply extends the old one with more details. Fig. 1 demonstrates the structure of a typical Event-B model. It is a series of contexts and machines, where contexts can extend each other, machines can *see* and use elements defined in contexts, and refine each other.

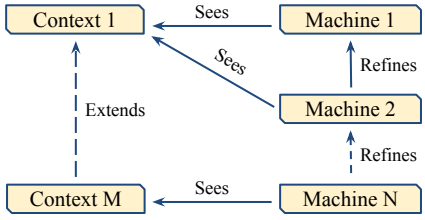

Fig. 1. Structure of a typical Event-B model.

At last, we should note that multiple refinement is not possible. One machine can not refine several other machines; however, it is possible for several machines to separately refine a single one. This will result in several model branches, which will not be possible to converge again together using standard means.

IV. State-based Refinement Technique

Among the mentioned rules those of event refinement are the most constraining. We propose a technique that allows sidestepping them and helps to create more flexible designs. First, we describe the theory of the technique, and then we talk about how it can be used in practice given the current restrictions of Event-B and Rodin.

A. Theory

Typically formal models are developed as a series of refinement levels, from the most abstract level to the most detailed and concrete one. This approach isn't always working as expected if there is too much difference between the behaviors of abstract and concrete levels. To overcome this we suggest splitting such "incompatible" levels into separate formal models, which we call "abstract" and "detailed", and establishing the correspondence relation between their states. Both models can be comprised of a series of conventional refinement levels, but there are additional requirements for their invariants, states, and events.

Conventional refinement allows "reusing" invariants and proofs: once invariant is established to be true on one refinement level, it is guaranteed to be true on all the following levels. With the suggested technique, there are no explicit connections between separate models, so such important invariants need to be replicated in each one of them. This complicates proofs and increases the size of the models, but it is a necessary trade-off that allows achieving the required flexibility.

States of abstract and detailed models should correspond to each other since they describe the same system. However, each model can freely include elements of the state that are not present in the other model. This way the abstract model can, for example, describe the system as a whole, while the detailed model describes only a small part of it (but in far greater detail). This is an improvement over conventional Event-B refinement, which does not allow the state of abstract levels to be bigger than the state of more concrete levels: only the reverse is possible.

Events of each model are not restricted at all: they can describe the behavior of the system completely differently. These events, however, need to preserve all established invariants of the model they belong to. Once it is proved that they correctly change the states, then all possible states of both models created with the state-based refinement technique are considered correct and safe. From this we can demonstrate that abstract and detailed models indeed correspond to each other simply by establishing the refinement relation between their states, completely omitting events from the proof.

Such a refinement relation should map each possible state of the detailed model to a corresponding state from the abstract model. The correspondence here means that both states should describe the same state (or set of states) of the modeled system. If the state of the detailed model is safe (a safe state is a state that satisfies all the invariants of its model), then its corresponding state of the abstract model should also be safe. It must be explicitly proved using the defined relation between states by demonstrating that some parts of the invariants of the

abstract model are a logical consequence of all invariants of the detailed model.

To make a more formal definition of the state-based refinement relation, let's represent abstract and detailed models as labeled transition systems, LTS_A and LTS_D. Each LTS is a tuple $< L, S, \rightarrow >$, where L is a set of labels, S is a set of possible states and $\rightarrow \subseteq S \times L \times S$ is a labeled transition relation. Also, there is a subset $Initial \subseteq S$ of initial states and a subset $Safe \subseteq S$ of safe states. $Initial$ and $\rightarrow$ can be used to calculate a subset $Reach \subseteq S$ of reachable states: states that are transitively reachable from some initial state through a transition relation.

Now let's define the binary relation $R \subseteq S_D \times S_A$ between states of the detailed transition system LTS_D and the abstract transition system LTS_A. This relation should consist of pairs of states that are corresponding to each other and both representing the same state (or set of states) of the modeled system. Such a relation is called a state-based refinement relation, if it projects each reachable state of the LTS_D to some safe state of the LTS_A (see Fig. 2). This condition can be expressed as $proj(Reach_D) \subseteq Safe_A$. By proving that the detailed model is correct, we can establish that $Reach_D \subseteq Safe_D$, and consequently that $proj(Reach_D) \subseteq proj(Safe_D)$. So, to demonstrate that a given relation is indeed a state-based refinement relation, it is left to prove that $proj(Safe_D) \subseteq Safe_A$.

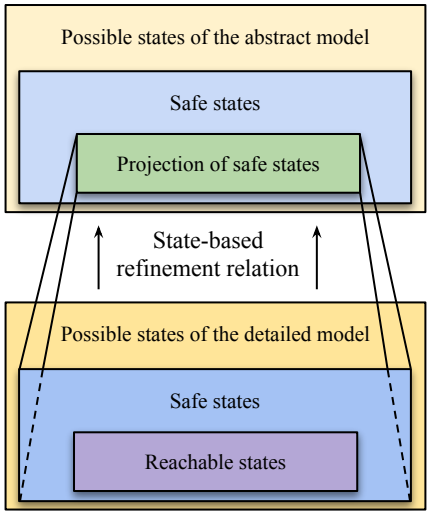

Fig. 2. State-based refinement relation.

The set of safe states $Safe_A$ of the abstract model is shaped by all the invariants that represent the important properties of the system. However, guards of events can also contain such properties. To take them into account with the state-based refinement technique they should be also expressed as invariants. This may require some non-trivial changes to the model, but it should be always possible to accomplish. This way the established refinement relation between models can indeed prove that all important properties are indeed present in both models.

At last, the technique can be naturally used with more than 2 models, if there is a need in several different abstraction levels. In this case, there can be a series of intermediate models, connected by a chain of refinement relations and proofs. Others, more peculiar configurations of models, are also possible.

B. Practice

In this section, we describe how the suggested state-based refinement technique can be used in the Rodin platform with the Event-B language. Here we outline the detailed step by step guide intended primarily for those who are sufficiently familiar with Event-B and want to try the technique. The guide consists of the following steps:

1) Formalize an abstract representation of the system as a separate Event-B model, possibly with the usage of conventional refinement. All required safety properties and requirements should be formalized as invariants, the corresponding behavior should be formalized as events. The correctness of the abstract model should be explicitly proved.
2) Formalize a detailed, concrete representation of the system as a separate Event-B model, possibly with the usage of conventional refinement. All required safety properties and requirements should be again formalized as invariants, the corresponding behavior should be formalized as events. The correctness of the detailed model should be explicitly proved.
3) Add an artificial event to the abstract model, which will describe all possible safe ways of changing its state:
   - Parameters of this event should contain new values for each state variable.
   - Actions of this event should assign parameter values to each corresponding state variable.
   - Guard conditions of this event should mirror the invariants of the model: but instead of using state variables, these guards will use event parameters instead.
4) Formally prove that this artificial event is correct. In most cases, this can be done using automated provers.
5) Add a context to the abstract model, which will contain all the carrier sets, constants, and axioms of the detailed model. At this step there can be possible naming conflicts with the existing carrier sets and constants: they should be resolved.
6) Add a refinement level to the abstract model, which will use previously added context (in addition to all the contexts from the abstract model). At this level the artificial event should be refined the following way:
   - Change type of the artificial event from the "extended" to "not extended".
   - Remove guard conditions that have the corresponding invariants on the detailed model from the event, with an exception of the guard conditions that describe the types of parameters. Other remaining guard conditions should only describe parts of the

abstract state that have no correspondence in the state of the detailed model.
- Add the corresponding new parameter for each state variable of the detailed model.
- Add guard conditions that mirror the invariants of the detailed model: instead of using state variables, these guards will use new event parameters.
- Add new parameters and guard conditions, which will define the refinement relation between the state of the detailed model and the state of the abstract model by using the event parameters that correspond to them.

7) Prove the correctness of the refined artificial event by discharging all proof obligations generated by Rodin.

The last step is the key one. For each removed guard condition, that corresponds to some invariant of the abstract model, Rodin will generate a proof obligation to prove that this guard condition still holds. It can be proved using the event parameters and guard conditions, that describe the state of the detailed model and the refinement relation between the detailed and abstract models. This in turn would mean that for each safe state of the detailed model (represented as parameters of the artificial event) there is a corresponding safe state from the abstract model (deleted invariants that a still proved to be true).

As can be seen from the guide, this particular implementation of the suggested state-based refinement technique has some weak points:

- There is a lot of manual copying of the invariants and states from one model to another. This is an error-prone process that is better to be avoided.
- The refinement relation between states of both models is defined with event parameters and guard conditions. Currently, it is not possible to prove that there are possible values for these parameters that satisfy the guard conditions, this is just assumed by Rodin.

Potentially the guide can also be improved by removing event parameters that correspond to the state of the abstract model and replacing them with Event-B witnesses that connect their values with the values of the parameters that correspond to the state of the detailed model. This is a large change that we didn't have time to test yet.

V. Practical Evaluation

We have used the suggested state-based refinement technique to develop a model of an access control mechanism in a Linux-based operating system. The system was thoroughly described in the document called "security policy model" and was implemented as a module inside the Linux kernel.

Our task consisted of the following steps:

- To formalize the security policy model in Event-B and to prove its correctness.
- To refine the developed Event-B model to the level of detail of the operating system kernel interface (that is, to the level of the system call interface).
- To prove that the refined model of the system call interface still satisfies all the security properties.

In our previous project [18] (which was quite similar to this one, but used the different security policy model) we tried to use the conventional refinement to achieve the same goals. Due to the strictness of refinement rules, we had to represent each system call as a graph of events connected with a special state variable. This approach proved to be quite difficult and inefficient: it resulted in a large number of additional state variables, invariants, and overly-complicated events.

This time we have introduced the state-based refinement technique and formalized the security policy model and system call interface as separate Event-B models. We have noticed that the model of the system call interface was far easier to understand and develop than before. Though we couldn't reuse some proofs, which is possible with conventional refinement, and had to repeat the security properties in both models, overall the suggested technique turned out to be a success for us and allowed us to achieve the outlined goals.

VI. Related Work

Refinement rules in various formal methods differ depending on the main goal of the method. They can be classified into two kinds of refinement (with nonempty intersection) – *simulation-targeted refinement*, which is mostly used when a detailed model should simulate an abstract one, and *proof-targeted refinement*, which should transfer properties proven in the abstract model into the detailed one with preservation of proofs. The most general simulation-targeted refinement is considered in [19], [20], simulation-targeted refinements are also used in VDM [21] and RAISE [22] methods. Refinements used in Z and Object-Z [23], [24] are simultaneously simulation- and proof-targeted. The refinement proposed in this paper, along with the original Event-B refinement, is proof-targeted. One of the early examples of proof-targeted refinement is considered in [25].

Others also noticed the strictness of refinement rules. [26] discusses the limitations of refinement rules: it is stated that they are not always applicable, and when they do, they are difficult to apply effectively in practice. The authors noted that these rules are not sufficient to guarantee that the refined model will satisfy the user's real requirements even if it satisfies the abstract model. To change this, they propose a new software development model, which operates with modified refinement rules that explicitly take into account the user's requirements. [27] presents an alternative refinement notion for B-method called retrenchment, intending to increase the number of different systems and designs that can be efficiently formalized. Retrenchment allows the interface of the model to be changed between the abstract and the detailed levels, providing that the different interfaces satisfy some additional constraints. The integration of retrenchment into Event-B is proposed in [28], however, it requires some additions to the language and changes to the Rodin platform.

VII. Conclusion

We have presented a new state-based refinement technique for Event-B that can be used to simplify the development of formal models when drastically different levels of abstraction are needed to be combined. Such models can be hard or even impossible to develop using conventional refinement due to the strictness of its rules and restrictions. The suggested technique allows to sidestep such rules by splitting different levels of abstraction into separate formal models, which are connected with a refinement relation between their states. The correctness of separate models and established state-based refinement relation between them can be proved using automated verification tools.

The technique was evaluated during the development of a model of an access control mechanism in a Linux-based operating system. The evaluation showed that the technique indeed simplifies the development of formal models with different abstraction levels and allows achieving more flexible and easier to understand designs than with conventional refinement, though at expense of requiring additional proofs and repeating of security properties in different models.

Albeit the technique itself is sound, its usage with Event-B and the Rodin platform can be further improved, mainly by automating some of its steps to eliminate possible user-induced mistakes. This can be achieved by developing a plug-in for Rodin and can be considered as possible future work.

References

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

[2] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How Amazon web services uses formal methods," *Communications of the ACM*, vol. 58, no. 4, pp. 66–73, Mar. 2015.

[3] E. M. Clarke and J. M. Wing, "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, Dec. 1996.

[4] T. Lecomte, T. Servat, and G. Pouzancre, "Formal Methods in Safety-Critical Railway Systems," in *10th Brasilian Symposium on Formal Methods*, 2007, p. 10.

[5] E. Troubitsyna, L. Laibinis, I. Pereverzeva, T. Kuismin, D. Ilic, and T. Latvala, "Towards Security-Explicit Formal Modelling of Safety-Critical Systems," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, A. Skavhaug, J. Guiochet, and F. Bitsch, Eds. Cham: Springer International Publishing, 2016, pp. 213–225.

[6] O. Laurent, "Using Formal Methods and Testability Concepts in the Avionics Systems Validation and Verification (V&V) Process," in *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ser. ICST '10. USA: IEEE Computer Society, Apr. 2010, pp. 1–10.

[7] C. Danmin, S. Yue, and C. Zhiguo, "A Formal Specification in B of an Operating System," *The Open Cybernetics & Systemics Journal*, vol. 9, no. 1, pp. 1125–1129, Sep. 2015.

[8] P. N. Devyanin, A. V. Khoroshilov, V. V. Kuliamin, A. K. Petrenko, and I. V. Shchepetkov, "Formal Verification of OS Security Model with Alloy and Event-B," in *Proceedings of the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z - Volume 8477*, ser. ABZ 2014. New York, NY, USA: Springer-Verlag New York, Inc., 2014, pp. 309–313.

[9] V. Kuliamin, A. Khoroshilov, and D. Medveded, "Formal Modeling of Multi-Level Security and Integrity Control Implemented with SELinux," in *2019 Actual Problems of Systems and Software Engineering (APSSE)*, Moscow, Russia, Nov. 2019, pp. 131–136.

[10] A. Lahbib, A. Ait Wakrime, A. Laouiti, K. Toumi, and S. Martin, "An Event-B Based Approach for Formal Modelling and Verification of Smart Contracts," in *Advanced Information Networking and Applications*, ser. Advances in Intelligent Systems and Computing, L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, Eds. Cham: Springer International Publishing, 2020, pp. 1303–1318.

[11] D. Bjørner, *Software Engineering 1: Abstraction and Modelling*, ser. Texts in Theoretical Computer Science. An EATCS Series, Software Engineering. Berlin Heidelberg: Springer-Verlag, 2006.

[12] N. Wirth, "Program Development by Stepwise Refinement," in *Pioneers and Their Contributions to Software Engineering: sd&m Conference on Software Pioneers, Bonn, June 28/29, 2001, Original Historic Contributions*, M. Broy and E. Denert, Eds. Berlin, Heidelberg: Springer, 2001, pp. 545–569.

[13] D. Efremov and I. Shchepetkov, "Runtime Verification of Linux Kernel Security Module," in *Formal Methods. FM 2019 International Workshops*, ser. Lecture Notes in Computer Science, E. Sekerinski, N. Moreira, J. N. Oliveira, D. Ratiu, R. Guidotti, M. Farrell, M. Luckcuck, D. Marmsoler, J. Campos, T. Astarte, L. Gonnord, A. Cerone, L. Couto, B. Dongol, M. Kutrib, P. Monteiro, and D. Delmas, Eds. Cham: Springer International Publishing, 2020, pp. 185–199.

[14] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*, 1st ed. New York, NY, USA: Cambridge University Press, 2010.

[15] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 6, pp. 447–466, Nov. 2010.

[16] M. Leuschel and M. Butler, "ProB: A Model Checker for B," in *FME 2003: Formal Methods*, ser. Lecture Notes in Computer Science, K. Araki, S. Gnesi, and D. Mandrioli, Eds. Berlin, Heidelberg: Springer, 2003, pp. 855–874.

[17] D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin, "SMT Solvers for Rodin," in *Abstract State Machines, Alloy, B, VDM, and Z*, J. Derrick, J. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 194–207.

[18] P. Devyanin, D. Efremov, A. Khoroshilov, V. Kuliamin, A. K. Petrenko, and I. Shchepetkov, *"Modeling and verification of access control policies in operating systems"*. Moscow: Goryachaya Liniya-Telecom, 2019, in Russian.

[19] G. Schellhorn, "Verification of ASM Refinements Using Generalized Forward Simulation," *Journal of Universal Computer Science*, vol. 7, no. 11, pp. 952–979, 2001.

[20] E. Börger, "The ASM Refinement Method," *Formal Aspects of Computing*, vol. 15, no. 2, pp. 237–257, Nov. 2003.

[21] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object-oriented Systems*. London: Springer-Verlag, 2005.

[22] C. George, "The RAISE specification language. A tutorial," in *VDM '91 Formal Software Development Methods*, ser. Lecture Notes in Computer Science, S. Prehn and H. Toetenel, Eds. Berlin, Heidelberg: Springer, 1991, pp. 238–319.

[23] J. Derrick and E. A. Boiten, *Refinement in Z and Object-Z: Foundations and Advanced Applications*, 2nd ed. London: Springer-Verlag, 2014.

[24] C. Bolton and J. Davies, "Refinement in Object-Z and CSP," in *Integrated Formal Methods*, ser. Lecture Notes in Computer Science, M. Butler, L. Petre, and K. Sere, Eds. Berlin, Heidelberg: Springer, 2002, pp. 225–244.

[25] W.-P. d. Roever and K. Engelhardt, *Data Refinement: Model-Oriented Proof Methods and Their Comparison*. Cambridge University Press, Dec. 1998.

[26] S. Liu and R. Adams, "Limitations of formal methods and an approach to improvement," in *Proceedings 1995 Asia Pacific Software Engineering Conference*, Dec. 1995, pp. 498–507.

[27] R. Banach and M. Poppleton, "Retrenchment: An engineering variation on refinement," in *B'98: Recent Advances in the Development and Use of the B Method*, ser. Lecture Notes in Computer Science, D. Bert, Ed. Berlin, Heidelberg: Springer, 1998, pp. 129–147.

[28] R. Banach, "Retrenchment for Event-B: UseCase-wise development and Rodin integration," *Formal Aspects of Computing*, vol. 23, no. 1, pp. 113–131, Jan. 2011.