ISSN 0361-7688, Programming and Computer Software, 2016, Vol. 42, No. 4, pp. 198–205. © Pleiades Publishing, Ltd., 2016.
Original Russian Text © P.N. Devyanin, V.V. Kulyamin, A.K. Petrenko, A.V. Khoroshilov, I.V. Shchepetkov, 2016, published in Programmirovanie, 2016, Vol. 42, No. 4.

Comparison of Specification Decomposition Methods in Event-B

P. N. Devyanin[1], V. V. Kulyamin[2,3,4], A. K. Petrenko[2,3,4], A. V. Khoroshilov[2,5], and I. V. Shchepetkov[2]

[1]*Educational Information Security Community, Moscow, Russia,*
[2]*Institute for System Programming, Russian Academy of Sciences, ul. Solzhenitsyna 25, Moscow, 109004 Russia,*
[3]*Moscow State University, Moscow, 119991 Russia GSP-1, Leninskie Gory, Moscow, 119991 Russia,*
[4]*National Research University Higher School of Economics 1-nd Kozhukhovsky proezd 1/7, Moscow, 101000 Russia,*
[5]*Moscow Institute of Physics and Technology, Institutskii per. 9, Dolgoprudnyi, Moscow oblast, 141700 Russia,*
e-mail: peter_devyanin@hotmail.com, khoroshilov@ispras.ru, kuliamin@ispras.ru, petrenko@ispras.ru, shchepetkov@ispras.ru

Received February 12, 2016

Abstract—Decomposition is an important phase in the design of medium and large-scale systems. Various architectures of software systems and decomposition methods are studied in numerous publications. Presently, formal specifications of software systems are mainly used for experimental purposes; for this reason, their size and complexity are relatively low. As a result, in the development of a nontrivial specification, different approaches to the decomposition should be compared and the most suitable approach should be chosen. In this paper, the experience gained in the deductive verification of the formal specification of the mandatory entity-role model of access and information flows control in Linux (MROSL DP-model) using the formal Event-B method and stepwise refinement technique is analyzed. Two approaches to the refinement-based decomposition of specifications are compared and the sources and features of the complexity of the architecture of the model are investigated.

DOI: 10.1134/S0361768816040022

1. INTRODUCTION

Practically all software systems and complex software models consist of parts, which are often called modules. In concrete programming languages, modules are represented differently; they can be translation units in C and C++, classes in Java, etc. A module includes the description of data types, constants, variables, operations (e.g., functions and methods), and other software entities. The decomposition of a software system assumes its subdivision into modules and the assignment of software entities to modules. The result of decomposition is not only the division of the system into modules but also the organization of the set of relationships between modules. Well-developed programming languages include a rich set of such relationships, such as *type definition–type use* or *function definition–function use*. Object-oriented languages have *class–subclass* inheritance relationships. In formal specification languages, the *definition–refinement* relationship plays a special role.

The software decomposition pursues several purposes, and there are different aspects of its estimation. The main purposes of the decomposition are:

• Reduce the complexity of analysis and system understanding by subdividing it onto parts of which each is simpler than the whole system.

• Ease the requirements for the analysis, compilation, loading, etc. tools by reducing the size of modules to be processed.

• Divide the work on the project by distributing modules between the team members.

• Increase the level of reuse and optimize the development of product lines.

• Simplify the development and maintenance of the system.

By the present time, as a result of studies of software architectures and design patterns, well-developed methods and practices of software decomposition are available. The techniques of software decomposition and capabilities of programming languages and software development platforms have been steadily improved, and they take into account the needs and capabilities of each side of the development process.

However, there is not enough experience in formal specification languages; furthermore, the tools for the analysis of such specifications have significant restrictions with respect to the size and complexity of specifications to be analyzed. The level of maturity of the specification design and analysis methods are also inferior to those available for the software development. For the specifications containing more than

several thousands of code lines, the typical situation is that there is need for decomposition, but no ready to use decomposition methods and criteria for their evaluation that could help select the most appropriate or at least acceptable ones are available.

In this paper, we study formal specification decomposition techniques. We consider various approaches to decomposition and formulate the problem of comparing these approaches with respect to different estimation aspects. The work is based on the experience gained in the formal analysis of the mandatory entity-role model of access and information flows control in Linux (MROSL DP-model [1, 2]). The access control system based on this model was implemented in Astra Linux Special Edition operating system [3] by the research and production association RPA RusBITech in 2014. For the specification and verification of the MROSL DP-model, we chose (see [4, 5]) the formal Event-B method and the Rodin platform [6, 7]. The decomposition was performed using the stepwise refinement technique supported by Event-B and Rodin. On the whole, this work can be classified as a medium size medium complexity project—the work took about two man-years.

We consider only the decomposition of Event-B specifications; moreover, we consider only the methods supported by the Rodin platform, which are various stepwise refinement techniques. For this reason, the conclusions drawn in this paper are not universal. However, they can be useful for those who start the adoption of formal methods in the development of software for critical systems.

In the next section, we describe features of the MROSL DP-model. In Sections 3 and 4, we give a brief description of Event-B and the stepwise refinement technique used for the decomposition of specifications. The difficulties arising in decomposition are discussed in Section 5. Section 6 describes the development of three formal specifications of the model using different refinement techniques and gives their evaluation.

2. MROSL DP-MODEL

The MROSL DP-model deals with the following concepts: *entities, sessions, roles,* and *user accounts.* The entities are the elements of the operating system for which access rights must be assigned and controlled; for example, these are files, folders, or sockets. Roles can be considered as containers of access rights to entities and other roles. Sessions are the operating system processes of which each operates under a user account. If a session has an access right to a role that has an access right to an entity, then it can get the read or write access for this entity.

The MROSL DP-model describes restrictions on the access to entities and roles using three data protection mechanisms. Due to the *mandatory integrity control,* each entity, session, role, and user account has an integrity level that can take two values—low or high. The aim of this protection mechanism is to prevent the modification of trusted (with the high integrity level) entities by untrusted (with the low integrity level) sessions.

The *mandatory access control* is based on partially ordered confidentiality labels of various entities and subjects of the system; the access to entities and roles is granted or denied based on these labels. To be granted access, the label of a subject (session) must be at least comparable with the label of the corresponding entity or role.

The *role-based access control* makes it possible to group access rights by certain criteria. As a result, the access rights used by each specific session are determined exclusively by the set of roles to which it has access; in turn, this set is easier to maintain and modify if required.

The MROSL DP-model defines 44 operations in the form of contracts [8]. Each contract includes a precondition—a set of obligations that must be fulfilled before the operation is called—and a postcondition—a set properties that must be fulfilled after the operation execution. Thirty four of these operations determine the order of creation or deletion of entities, sessions, user accounts, roles, and modifications of the integrity and confidentiality labels. The other ten operations are used for a more precise analysis of the security model in terms of information flows by memory and time and de facto ownership.

3. EVENT-B AND THE RODIN PLATFORM

The formal Event-B method allows one to carry out interactive proofs and use various automatic provers. It has a simple notation and has been successfully used in a number of projects. In this paper, we design and verify specifications using Event-B and the Rodin platform.

In Event-B, each specification consists of *contexts* and *machines.* The contexts contain static (invariable) part of the specification—definitions of *constants* and *axioms.* The machines contain the dynamic part—*variables, invariants,* and *events.* The variables and constants can be sets, relations, functions, numbers, or take Boolean values. The values of variables form the current state of the specification and invariants restrict this state.

Events are used for the modification of the current state in a certain way under certain constraints. Each event consists of *parameters, guard conditions,* and *actions.* The guard conditions restrict the values of parameters of the operation and variables of the machine thus decreasing the number of states in which the event can happen. Actions modify the current state by assigning new values to the variables.

For each case that requires a proof — one-valuedness of expressions, conservation of invariants under state changes, and the correctness of stepwise refinement — Rodin generates corresponding statements to be proved. In order to completely verify the model, all the generated statements must be proved.

4. STEPWISE REFINEMENT

A typical *monolithic* specification in Event-B consists of one context and one machine; however, in the case of large and nontrivial specifications, such a structure complicates the specification understanding, development, and maintenance. In this case, it is reasonable to decompose the specification. Event-B proposes to do this using the stepwise refinement technique.

This technique makes it possible to refine the existing machines and contexts. If the machine B refines the machine A, then it can extend the state of the machine A by adding new variables and events that will modify these variables. The events included in the machine A can be refined by strengthening their preconditions and adding new postconditions[1]. As a result, a chain of machines beginning from the first (abstract) version and ending with the last most specific version is obtained. The refinement of contexts is done similarly. The chains of contexts and machines together form a hierarchical decomposed specification of the system. Basically, the correctness of each refinement step must be proved; however, it can be correct by construction if it is performed using special transformation rules.

There are two basic kinds of stepwise refinement [9]. The first one is called horizontal refinement. It is used for extending the specification state by adding new variables and invariants, for strengthening guard conditions of events or for adding new actions to them, and for creating new events. The second kind is the vertical refinement. It is used after all possible horizontal refinement steps have been made. It allows one to transform data structures to simplify the implementation of the specification in the source code. The concepts defined in the MROSL DP-model have a too high level and the majority of them have no simple analogues in the implementation code. In this paper, we consider only the horizontal refinement.

5. FEATURES OF THE TASK OF DECOMPOSITION OF FORMAL SPECIFICATIONS

The conventional aims of decomposition and the aspects of the evaluation of a decomposition scheme using refinement are also important for the choice of the method for constructing the Event-B specification. However, the development of specifications differs from the typical software development process; therefore some difficulties can arise that can be easier overcome by choosing an appropriate decomposition method. We consider three of these difficulties:

• The initial unclearness of the nature of the system to be specified and of the specification itself (because the task of specification arises when the developers do not have obvious and conventional solutions.

• Nontriviality of predicting the complexity of verification (deductive proof).

• Discrepancy between the structures of the specification and implementation and, in particular, differences between the set of entities of the specification and the set of entities of the target system (there is no clear correspondence between them).

The main consequence of these difficulties is that the specification should be designed by gradually extending the code base with a considerable number of experimental improvements with rollbacks to find an acceptable solution.

In the next section, we consider two versions of decomposition of the MROSL DP-model and evaluate these versions with account for various aspects of the development of the corresponding specifications.

6. THE DEVELOPMENT AND EVALUATION OF SPECIFICATIONS OF THE MROSL DP-MODEL

We proposed two different methods for the decomposition of the MROSL DP-model specification using stepwise refinement. The first method is the decomposition *based on the data types*[2] of the system to be specified. The aim of this method is to produce a detailed decomposition to maximally facilitate the operation of the verification tools. Under this approach, one data type is added at each refinement level; for example, sessions and operations of their creation, deletion, and modification are first described; then user accounts are described; etc. The second method—the *feature-based* decomposition—uses high-level features of the system as decomposition units. For the MROSL DP-model, we chose as such features the data protection mechanisms described in the system.

To examine the specific features of the domain, we decided to first develop a simpler monolithic specification without decomposition. This specification turned out to be very useful even though it had certain drawbacks. The experience gained in the course of its development helped create and evaluate two specifications obtained using different decomposition methods.

[1] The concept of refinement used in Event-B differs from the classical version used in VDM and RAISE.

[2] Data types are considered together with the corresponding operations.

6.1. Monolithic Specification

For the development of the monolithic specification, we used the approach that requires the periodic proof of the correctness of the next formalized part of the model. This approach enabled us to promptly detect and correct bugs appearing due to the incorrect interpretation of the model details. The specification was completed and proved approximately within a year, and a number of errors in the initial textual description of the MROSL DP-model were detected and eliminated.

Since there was no need in planning the stepwise refinement, the specification was developed faster, but its monolithic structure caused a number of difficulties. First, the specification turned out to be difficult to maintain and improve because changes in its code required its partial reproof. Furthermore, due to the size and complexity of the specification concentrated within a single machine, the automatic provers could hardly do the job, and the proofs had to be done interactively. The size and complexity also negatively affected the readability of the specification code. This can be demonstrated by an example.

Figure 1 shows a part of the event *create_session* from the monolithic specification. This event is an analog of the corresponding operation in the MROSL DP-model; it models the creation of a new session. As all the other events in Event-B, it has a set of parameters, a block of guard conditions (the event preconditions), and a block of actions (postconditions). The guard conditions and actions are implemented in the rows beginning with the labels *@grd* and *@act,* respectively, and the parameters are defined in the block *any*.

The actions and conditions are connected by arrows in Fig. 1; the arrows also show to which part of the model the actions and conditions belong to. It is seen that the connections are fairly complicated—many interdependent entities are distributed across the event code. For example, *@grd2*–*@grd4*, *@grd7*–*@grd9*, *@grd25*, *@act6,* and *@act7* are responsible for the executable files and session profiles. Moreover, parts of the same event can be connected not only to each other but also to invariants included in the specification and to parts of other events. Event-B does not provide tools for revealing such relationships without stepwise refinement; therefore, these relations are not visible in the monolithic model.

6.2. Decomposition Based on Data Types

The experience gained in the development of the monolithic specification helped us develop and verify a specification using the decomposition based on data types. Even though the MROSL DP-model has already been well studied by that time, insufficiently thorough planning of the refinement steps has several times required the specification to be completely rewritten. All such cases were caused by the fact that an event that simultaneously modified not only the state of the current machine but also the state of a machine preceding the current machine in the sequence, which contradicts the stepwise refinement technique, was added. It is difficult to predict such cases in a nontrivial system due to a large number of relationships between its data types.

As a result, we were able to decompose the model into 15 closely related parts and establish the order in which they should be added to the specification in the form of a chain of machines of which each refines the preceding machine. The stepwise refinement method was correct by construction; therefore, no additional statements to be proved were generated.

Compared with the monolithic version, the decomposed specification has a simple and clear structure. New details are easy to add, and if the model changes, the stepwise refinement makes it possible to naturally reflect those changes in the specification because the decomposition plan takes such situations into account. Such changes affect a less number of proofs, which facilitates the maintenance and development of the specification.

Figure 2 illustrates one of the 15 refinement steps—the machine that describes a small part of the model (more precisely, the executable files and profiles). It is seen how this part affects the existing events *create_user*, *set_user_labels,* and *create_session*, and which conditions it must satisfy. The other machines look similarly.

Despite the advantages described above, the majority of decisions (such as which data structures should be used, how to describe invariants, etc.) are the same in the monolithic and decomposed specifications based on data types. Even the complexity of proofs of correctness is comparable in both cases—the refinement did not significantly facilitate the work for the automatic provers. This implies that even though we were able to facilitate the maintenance and development of the specification and simplify its code, the principle underlying this method of decomposition did not pay off.

6.3. Feature-Based Decomposition

The plan of the feature-based decomposition relies on the modified hierarchical version of the MROSL DP-model .

The original textual description of the MROSL DP-model is fairly large and monolithic; i.e., the elements of the model are presented in the order that is convenient for the description of the model as a whole. First, the elements of states of the abstract system specified within the model are described; then the requirements for the implementation of the mandatory and role control and the mandatory integrity control are presented; then the rules of transforming the system states are given; and finally the system security

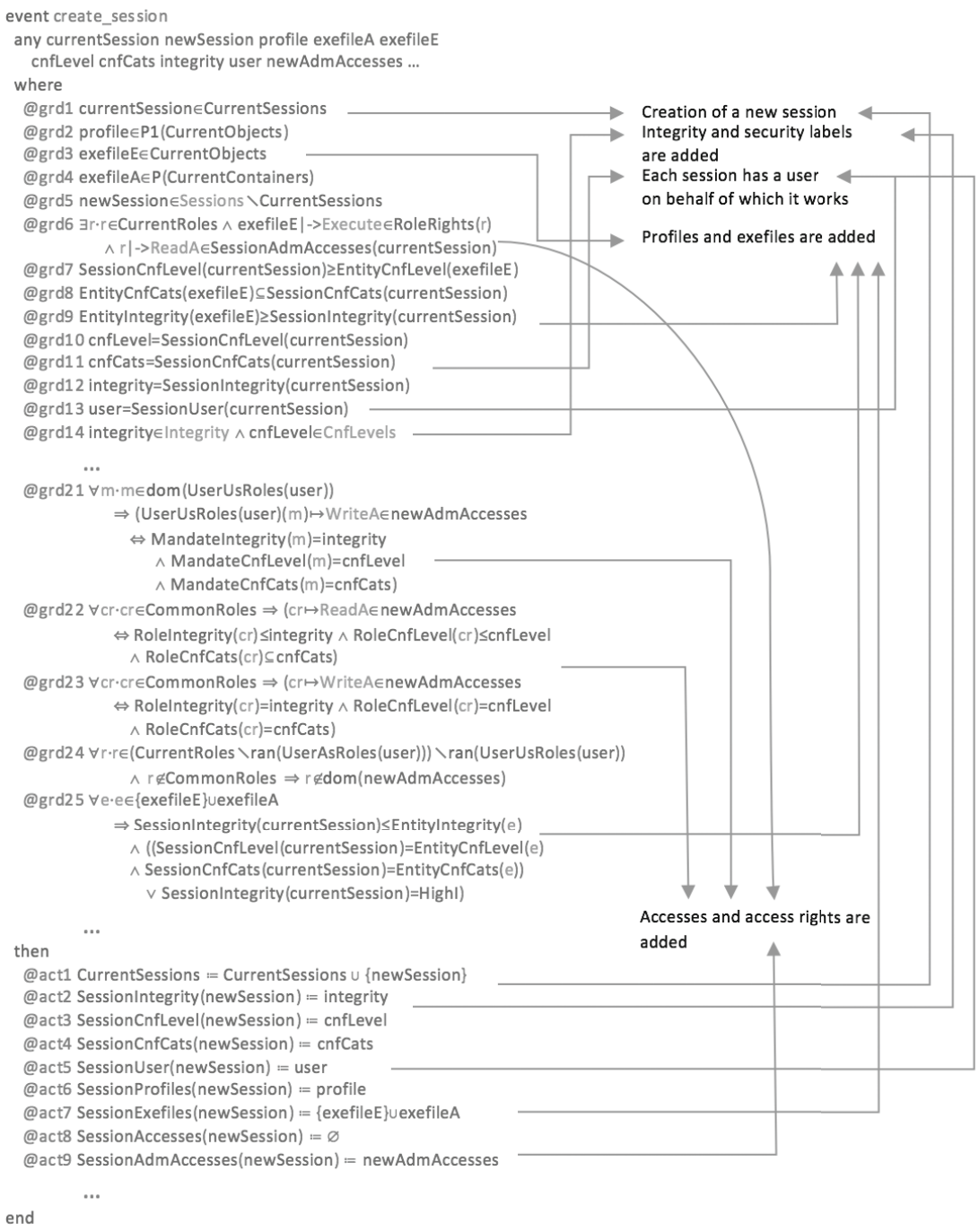

Fig. 1. A part of the event *create_session* from the monolithic specification.

conditions are formulated and justified and approaches to the model application in a special purpose operating system (SPOS) are considered (an example of such an operating system is Astra Linux Special Edition). As a result, the model becomes more and more difficult to modify because each change in any of its elements requires appropriate modifications in all the related elements of the model; furthermore, the validity of the majority of proved statements must be checked again. In addition, due to the large size, the monolithic nature of the model, and the impossibility to implement it step by step in this form, the model is difficult to use by the SPOS developers, and it is difficult to use this model as a basis for other models (e.g., the model of a hypervisor for the SPOS).

For this reason, we are now proceeding from the monolithic description of the model to the hierarchical one that makes it possible to represent the model

```
machine N11 refines N10 sees C3

...
invariants
  @UserProfiles_type UserProfiles∈CurrentUserAccounts→P1(CurrentEntities)
  @SessionProfiles_type SessionProfiles∈CurrentSessions→P(Entities)
  @SessionExefiles_type SessionExefiles∈CurrentSessions→P(Entities)
  @UserProfilesIntIsCorrect
    ∀u,p·u∈CurrentUserAccounts ∧ p∈UserProfiles(u) ⇒ UserIntegrity(u)=EntityIntegrity(p)
  @UserProfilesCnfIsCorrect
    ∀u,p·u∈CurrentUserAccounts ∧ p∈UserProfiles(u) ⇒ UserCnfLevel(u)=EntityCnfLevel(p)

...
events

...
event create_user extends create_user
 any profiles
 where
  @grd33 profiles∈P1(CurrentEntities)
  @grd34 ∀p·p∈profiles ⇒ EntityIntegrity(p)=integrity
  @grd35 ∀p·p∈profiles ⇒ EntityCnfLevel(p)=cnfLevel
 then
  @act16 UserProfiles(user) ≔ profiles
end

event set_user_labels extends set_user_labels
 any profiles
 where
  @grd58 profiles∈P1(CurrentEntities)
  @grd59 ∀p·p∈profiles ⇒ EntityIntegrity(p)=integrity
  @grd60 ∀p·p∈profiles ⇒ EntityCnfLevel(p)=cnfLevel
 then
  @act16 UserProfiles(user) ≔ profiles
end

...
event create_session extends create_session
 any exefileA exefileE profile
 where
  @grd12 profile∈P1(CurrentObjects)
  @grd13 exefileE∈CurrentObjects
  @grd14 exefileA∈P(CurrentContainers)
  @grd15 EntityCnfLevel(exefileE)⊆SessionCnfLevel(currentSession)
  @grd16 EntityCnfLevel(exefileE)⊆UserCnfLevel(user)
  @grd17 EntityIntLevel(exefileE)≥integrity
then
  @act7 SessionProfiles(newSession) ≔ profile
  @act8 SessionExefiles(newSession) ≔ {exefileE}∪exefileA
end
end
```

Fig. 2. The part of the decomposed specification describing the executable files and profiles.

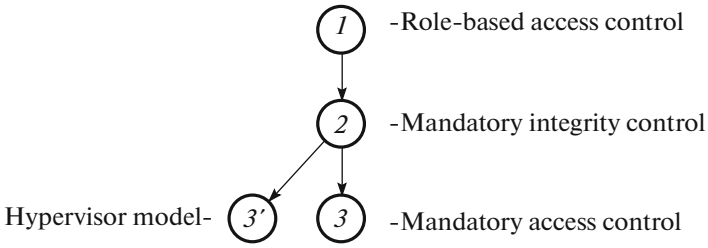

Fig. 3. Layers of the hierarchical representation of the MROSL DP-model of the hypervisor for the SPOS.

layer by layer. Each lower layer of the model should be an abstract system whose elements are independent of the new elements belonging to the higher layer; the higher layer inherits and modifies or (if required) augments the elements of the lower layer. For example, one can define the first layer as a model of the role-based access control, the second layer as a model of the role-based access control and mandatory integrity control, the third layer as a model of the role-based access control, mandatory integrity control, and mandatory access control with only information flows by memory, etc. In such a hierarchical description, the model of hypervisor for the SPOS can be considered as an alternative (additional) third layer (the model of the role-based access control, mandatory integrity control, and hypervisor) because it may be assumed that the hypervisor for the SPOS must ensure the correct operation of its mandatory integrity control and the mandatory access control must not be implemented by the hypervisor. This idea is illustrated in Fig. 3.

The three presently available layers of the hierarchical representation of the MROSL DP-model were formalized in an Event-B specification using the stepwise refinement technique. Even though the formalization was completed quickly and without significant problems, we found out that the stepwise refinement does not make it possible to describe the abstraction method used in the hierarchical representation in a natural way. Consider this situation using an example.

In the MROSL DP-model, each user account has a set of individual roles, and their number and properties directly depend on the integrity and confidentiality labels of the user account. The individual roles are created simultaneously with the creation of the user account itself. The problem is that neither mandatory integrity control nor mandatory access control exist on the base abstract layer; therefore, user accounts have no corresponding labels; hence, it is sufficient to create two individual roles for each user account. As new access control mechanisms are added on the next two layers, the number of created roles will grow, which contradicts the stepwise refinement technique because it requires that each event modifies the variables defined on the abstract layer in the same way as they were modified by the corresponding abstract event. In the case under consideration, this means that the number of created individual roles must coincide with the number of the originally created roles on the base layer; only a more detailed description of their properties may be made.

A natural solution would be to create a greater number of roles in advance than are necessary on the base layer and then refine their number and properties. However, one cannot simply create extra roles—one must check that they satisfy the invariants included in the model. Since we know nothing about these additional roles on the base layer, a great number of preconditions describing them must be added, and a part of them will be surely redundant on the next layers as they will be replaced by preconditions based on the integrity and confidentiality labels of these roles.

We tried this approach, and it complicated the proof of the conservation of invariants and increased the specification size. Then, we developed another solution. The MROSL DP-model was modified in such a way that the individual user roles were not created together with the creation of the corresponding user account but were taken from a set of roles that are already available in the model. Since no new roles are created in this case, there are no problems with the stepwise refinement when going from the base layer of the model to the other layers. In addition, the number of statements to be proved that are generated for the creation of the user account decreases, and the size of the final specification is also decreased because the number of preconditions describing the created roles is decreased.

6.4. Comparison of the Decomposition Methods

Consider the following criteria for the evaluation of the specification decomposition methods described above: simplification of the analysis and understanding of the system, weakening of requirements for the verification tools, increasing the reuse level, and simplification of maintenance and development. Both decomposition methods when applied to specifications of the MROSL DP-model using the stepwise refinement technique could not significantly improve the operation of the automatic provers. At the same time, the decomposition based on data types and the

feature-based decomposition helped simplify the specification code and improved its structure; they also made it possible to modify the specification and add new details in a natural way. Finally, the feature-based decomposition outperforms the decomposition based on data types in terms of reuse: it allows one to develop specifications of similar systems on the basis of existing systems (e.g., to develop the specification of the access control model for the hypervisor based on the existing model for the operating system).

7. CONCLUSIONS

Two different decomposition methods of formal specifications based on the experience gained in the formal analysis of the mandatory entity-role model of access and information flows control in Linux (MROSL DP-model) are evaluated. The decomposition was performed using the formal Event-B method and the stepwise refinement technique supported by it. The stepwise refinement was used to decompose specifications of the model in two different ways—with the focus on the data types of the system being specified and on the higher level features of this system or, more precisely, on the data protection mechanisms described in the system.

Both decomposition methods have similar advantages and disadvantages; however, the feature-based decomposition outperforms the decomposition based on data types in terms of reuse because it makes it possible to develop specifications of similar systems on the basis of the existing specifications.

ACKNOWLEDGMENTS

This study was supported by the Ministry for Science and Education of the Russian Federation, project no. RFMEFI60414X0051.

REFERENCES

1. Devyanin, P.N., *Models of Security of Computer Systems: Access control and Data Flows,* Moscow: Hot line Telecom, Moscow, 2013.
2. Devyanin, P.N., Security Conditions for Information Flows by Memory in the MROSL DP-model, Prikl. Diskr. Mat, Appendix, 2014, vol. **7**, pp. 82–85.
3. Astra Linux. http://www.astra-linux.com
4. Devyanin, P., Khoroshilov, A., Kuliamin, V., et al., Formal Verification of OS Security Model with Alloy and Event-B, Proc. of the Fourth Int. Conf. on *Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ-2014),* Toulouse 2014, pp. 309–313. https://www.springer.com/us/book/9783662436516
5. Devyanin, P.N., Kulyamin V.V., Petrenko, A.K., et al., On the Representation of the MROSL DP-model in the Formalized Event-B Notation (Rodin Platform*), Konf. RusKripto-2014* (Proc. of the Conf, RusKripto-2014), Moscow, 2014. http://www.ruscrypto.ru/resource/ summary/rc2014/05_devyanin.pdf
6. Abrial, J.-R., *Modeling in Event-B: System and Software Engineering,* Cambridge: Cambridge University Press, 2010.
7. Abrial, J.-R., M. Butler, S. Hallerstede, et al., Rodin: An Open Toolset for Modelling and Reasoning in Event-B, Int. J. on Software Tools for Technol. Transfer, 2010, vol. **12**, no. 6, pp. 447–466.
8. Kulyamin V.V., Methods of Software Verification, Competition of Reviews on Information and Telecommunication Systems, 2008.
9. Damchoom, K., An Incremental Refinement Approach to a Development of a Flash-Based File System in Event-B, Ph. D. thesis, University of Southampton, School of Electronics and Computer Science, 2010.

Translated by A.V. Klimontovich