An Incremental Refinement Approach to a Development of the Remote Authentication Dial-In User Service Protocol

Sanae El Mimouni, Rajaa Filali, Anas Amamou, Bahija Boulamaat and Mohamed Bouhdadi

***Abstract*—** This paper presents an incremental formal modeling of the Remote Authentication Dial-In User Service (RADIUS) using event B method. The RADIUS protocol is a distributed client/server protocol that protects networks against unauthorized access. We model the protocol step by step by using refinement, a technique of event B. The first step will be the modeling of the most abstract specification of the protocol. Then by the second refinement more details of the protocol specification will be added to the model. By this approach, the model will be a more explicit representation of the target protocol by each refinement. Through a refinement approach, we prove that the abstract goals concerning message exchange of the RADIUS protocol are satisfied. In the developed Event-B models of the RADIUS protocol described in this paper, all proofs are generated and discharged by the Rodin tool. Our Specification is very general and contains basic message exchange process of RADIUS Client/server.

***Keywords*—**Event-b,Formal specification, RADIUS, Refinement.

I. Introduction

Basically created by Livingston Enterprise which was later acquired by Lucent , and as defined by IETF‘s RFC 2865 (RADIUS authentication and authorization) and RFC 2866 (RADIUS accounting), RADIUS is based on the client-server model and message exchanges takes place over User Datagram Protocol (UDP). The Network Access Server (NAS) acts as a RADIUS client which passes on the user request to the RADIUS server. The other RADIUS clients may be wireless access points, routers, and switches. The RADIUS server performs authentication, authorization, and accounting (AAA) for users after it receives requests from the client. The communication between the client and the server is encrypted using a private key which is never sent over the network. Both the client and server are configured with this secret before communication can take place, and it fails if the secret does not match at both ends.

This article is an extended version of a conference paper that appeared as [1].

Even with the practical significance of RADIUS protocol, unfortunately there isn’t a formal specification for it like as done to CSMA/CD Protocol using model checking [2] or petri Nets [3] or even using formal design patterns [4] .So we try to present a formal approach for the protocol. We developed our model specification in Event-B[5][6].We liberally used refinements, both of machines and of contexts. We give a great deal of attention to proofs. Consequently, we now have a specification of RADIUS protocol where all proof-obligations have been discharged.

The RADIUS protocol was first defined in RFC 2058 [7], in January 1997, this RFC contains proposed standard. Also in January 1997 RADIUS accounting was introduced in RFC 2059 [8], status of which is informational. Later in April 1997 these RFCs were obsolete by RFC 2138 [9] and RFC 2139 [10]. Former of these is proposed standard and latter informational. Then in June 2000 RFC 2865 [11] defined RADIUS draft standard and obsoleted RFC 2138. In same month informational RFC 2866 [12] RADIUS accounting obsoleted RFC 2139.For our paper we based on the RFC 2865.

This paper is organized as follows. In Section 2 we will give an informal introduction to the RADIUS protocol, and a brief description of the event B method, then, we introduce Rodin, which is the tool support for Event-B. The main part of this paper, Section 3 describes our strategy of refinement, Moreover we will specify our protocol using event B. Section 4 summarizes the results and draws a conclusion.

II. Basic Concepts

In this section, we provide some background information on the RADIUS protocol, the Event-B formal method, and then present Rodin platform.

A. *RADIUS protocol*

The Remote Authentication Dial-in User Service (RADIUS) [11] is an IETF-defined Client/server protocol and software that enables remote access servers to communicate with a

Sanae El Mimouni is with LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat, Morocco (e-mail: sanae.elm@ gmail.com).

Rajaa Filali is with LMPHE laboratory, University of Mohammed V, Faculty of sciences ,Rabat ,Morocco(e-mail: rajaafilali@gmail.com).

Anas Amamou is with LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat, Morocco (e-mail: amamou.anas@yahoo.fr).

Bahija Boulamaat is with LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat, Morocco (e-mail: boulamaatbahija@gmail.com).

Mohamed Bouhdadi is with LMPHE laboratory, University of Mohammed V, Faculty of sciences, Rabat, Morocco (e-mail: bouhdadi@fsr.ac.ma).

central server to authenticate dial-in users and authorize their access to the requested system or service [11]. It is commonly used to provide centralized Authentication, Authorization, and Accounting (AAA) for dial-up, virtual private network, and, wireless network access.

The RADIUS protocol is based on a Client/server model. A Network Access Server (NAS) operates as a client of RADIUS. The client is responsible for passing user information to designated RADIUS servers, and then acting on the response which is returned.

RADIUS servers are responsible for receiving user connection requests, authenticating the user, and then returning all configuration information necessary for the client to deliver service to the user.

A RADIUS server can act as a proxy client to other RADIUS servers or other kinds of authentication servers.

The operation of the RADIUS protocol involves six types of message exchanges between the client and the server, as described in the following sections and a simple procedure of RADIUS communication is shown in the figure 1:

- *Access-Request:* Sent by a RADIUS Client to request authentication and authorization for a network access connection attempt. It determines whether a user is allowed access to a specific NAS, and any other specific service.
- *Access-Accept:* Sent by a RADIUS server in response to an Access-Request message when all conditions are met. The message informs the RADIUS Client that the connection attempt is authenticated and authorized and it contains the list of configuration values for the user.
- *Access-Reject:* Sent by a RADIUS server in response to an Access-Request message if any condition is not met. This message informs the RADIUS Client that the connection attempt is rejected. A RADIUS server sends this message if either the credentials are not authentic or the connection attempt is not authorized.
- *Access-Challenge:* Sent by a RADIUS server in response to an Access-Request message if all conditions are met and RADIUS server wishes to issue a challenge to which the user must respond. The Client in response resubmits its original Access-Request with a new request ID, response (encrypted), and including the Attribute from the Access-challenge.
- *Accounting-Request:* Sent by a RADIUS Client to specify accounting information for a connection that was accepted.
- *Accounting-Response:* Sent by the RADIUS server in response to the Accounting-Request message. This message acknowledges the successful receipt and processing of the Accounting-Request message.

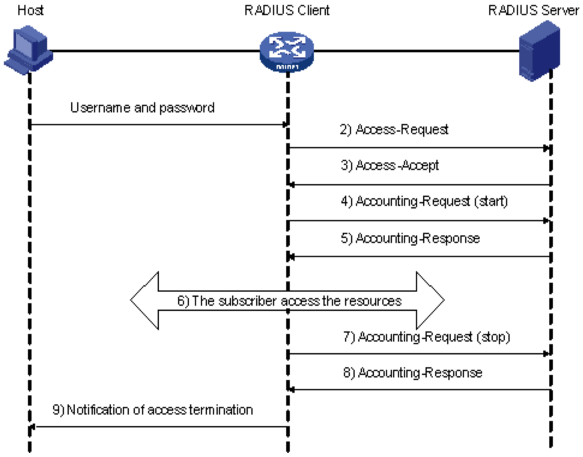
Host
RADIUS Client
RADIUS Server
Username and password
2) Access-Request
3) Access-Accept
4) Accounting-Request (start)
5) Accounting-Response
6) The subscriber access the resources
7) Accounting-Request (stop)
8) Accounting-Response
9) Notification of access termination

Fig. 1 Basic message exchange process of RADIUS

The following shows how RADIUS operates as shown in the figure above:

1. The user enters the username and password.

2. Having received the username and password, the RADIUS client sends an authentication request (Access-Request) to the RADIUS server.

3. The RADIUS server compares the received user information with that in the Users database. If the authentication succeeds, it sends back an Access-Accept message containing the information of user's right. If the authentication fails, it returns an Access-Reject message.

4. The RADIUS client accepts or denies the user according to the returned authentication result. If it accepts the user, it sends an accounting start request (Accounting-Request) to the RADIUS server, with the value of Status-Type being "start".

5. The RADIUS server returns a start-accounting response (Accounting-Response).

6. The subscriber accesses the network resources.

7. The RADIUS client sends a stop-accounting request (Accounting-Request) to the RADIUS server, with the value of Status-Type being "stop".

8. The RADIUS server returns a stop-accounting response (Accounting-Response).

9. The subscriber stops network resource accessing.

In this paper we model a simple RADIUS procedure of communication without considering accounting messages.

B. Event B method

Formal methods are mathematical based techniques which are used for describing the properties of a system.

They provide a systematic approach for the specification, development and verification of software and hardware systems and because of the mathematical basis we can prove that a specification is satisfied by an implementation [13].

Event-B is a formal method for specifying, modeling and reasoning about systems. An evolution of the B-Method developed by Jean-Raymond Abrial [14]. Event-B is now centered on the general notion of events. The formal concepts used in Event-B are by no means new. They were proposed a long time ago in a number of parent formalisms, such as Action Systems [15][16][17], TLA+ [18][19], and UNITY [20].

Event-B is a formal modeling method for developing systems via step-wise refinement [21][22], based on first-order logic. Event-B models are organized in terms of two basic components: contexts and machines. Machines and contexts can be inter-related: a machine can be refined by another one, a context can be extended by another one and a machine can see one or several contexts as shown in figure 2.

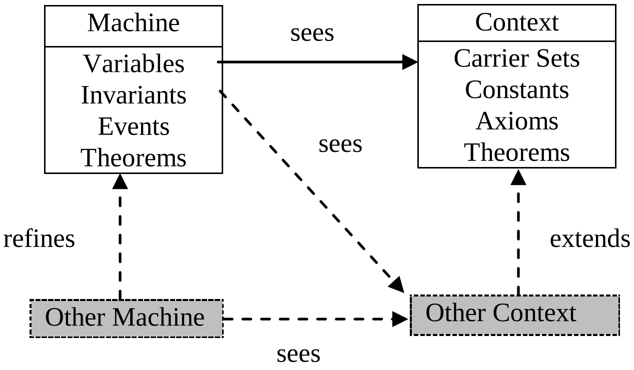

Fig. 2 Event-B Machines and Contexts

- Contexts specify the static part of a model. They may contain carrier sets (similar to types), constants, axioms (containing carrier sets and constants), and theorems (expressing properties derivable from axioms).

-Machines specify behavioral properties of the models. They may contain variables defining the state of a machine, invariants constraining that state, and events (describing possible state changes). Each event is composed of a set of guards and a set of actions. Guard state the necessary conditions under which an event may occur, and actions describe how the state variables evolve when the event occurs.

Contexts/Machines may be refined from more abstract to more concrete contexts/machines. Event-B models are systematically structured in refinement chains.

Building a model usually starts with a very abstract model of the system, and then gradually details are added through several modeling steps in such a way that leads us towards a suitable implementation; this approach is called refinement [21][22].Thus, instead of building a single model in a flat manner, we have a sequence of models, where each of them is supposed to be a refinement of the previous.

From a given model M1, a new model M2 can be built as a refinement of M1. In this case, model M1 is called an abstraction of M2, and model M2 is said to be a concrete version of M1. A concrete model is said to refine its abstraction. Each event of a concrete machine refines an abstract event or refines skip. An event that refines skip is referred to as a new event since it has no counterpart in the abstract model.

A key concept in Event-B is proof-obligation (PO) capturing the necessity to prove some internal property of the model such as typing, invariant preservation by events, and correct refinements. Strong tool support is provided in order to support this proof process.

Event-B is not specific to embedded systems design but it is currently being investigated by several industrial from different sectors (automotive, transportation, space) in the context of the DEPLOY project [23].

In Event-B, an event is defined by the syntax: EVENT e WHEN G THEN S END , Where G is the guard, expressed as a first-order logical formula in the state variables, and S is any number of generalized substitutions, defined by the syntax S ::= x := E(v) | x := z : | P(z). The deterministic substitution, x := E(v), assigns to variable x the value of expression E(v), defined over set of state variables v. In a non-deterministic substitution, x := z : | P(z), it is possible to choose non-deterministically local variables, z, that will render the predicate P(z) true. If this is the case, then the substitution, x := z, can be applied, otherwise nothing happens.

It is also important to indicate that the most important feature provided by Event-B is its ability to stepwise refine specifications. Refinement is a process that transforms an abstract and non-deterministic specification into a concrete and deterministic system that preserves the functionality of the original specification. During the refinement, event descriptions are rewritten to take new variables into account. This is performed by strengthening their guards and adding substitutions on the new variables. New events that only assign the new variables may also be introduced. Proof obligations (POs) are generated to ensure the correctness of the refinement with respect to the abstract model. Event-B is supported by several tools, currently in the form a platform called Rodin.

C. Rodin Platform

Rodin is an Eclipse-based development environment for Event-B. It is open source and provides an environment for system modeling and analyses, including support for checking specification correctness and for refinement proofs. While constructing an Event-B program, Rodin will automatically generate a set of POs for the program under consideration. Each PO is a logical formula, whose validity implies that certain correctness properties are satisfied by the program under consideration. In Rodin, the correctness properties include:

1. The Event-B program is not in an invalid state (i.e. a state where some invariant might not hold).

2. The behaviour of a concrete Event-B program will correspond to the behavior of its abstract program.

The first property is ensured by proving that the invariant is preserved and by proving the well-definedness of predicates [24]. The second one, i.e. the correspondence between abstract and concrete Event-B programs, is usually called the refinement PO.

There are three kinds of POs which can be generated from Rodin to ensure that the refinement is correct [24]:

– Guard strengthening (GRD)

– Action simulation (SIM)

– Equality of a preserved variable (EQL)

Obligations are proved either automatically or manually. In automatic mode, Rodin uses some predefined proof tactics made up of internal and external provers to discharge the obligations. In interactive mode, the user “guides” the proof attempts by applying some simple proof steps to simplify the obligations before invoking some trusted external provers to finish the proofs. As interactive proofs require manually interventions, it is usually considered as some costs of developing formal models. More teaching materials on Event-B and Rodin can be found at [25].

III. Specifying RADIUS protocol using Event B

A. Refinement strategy

In this short section, we present our strategy for constructing the RADIUS protocol specially the message type exchanges that take place between the Client and the Server, which is shown in the figure below. This will be done by means of an initial model followed by one refinement.

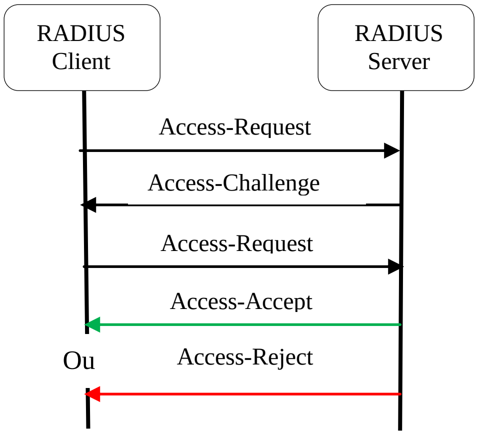

Fig. 3 Simple procedure of RADIUS communication

- The initial model essentially presents message exchange between the client and the server without considering any condition.
- In the first refinement, we introduce the condition that take side the client status and we add a timer.

B. Initial Model

The initial model of RADIUS protocol is presented as follow:

The context is made of two sets Requests and the Responses. These sets represent the message type exchanges that take place between the Client and the Server. Which are Access_Request, Access_Accept, Access_Challenge, and Access_Reject.

```
SETS
    Requests
    Responses
CONSTANTS
    Access_Request
    Access_Accept
    Access_Reject
    Access_Challenge
AXIOMS
    axm1  :  Access_Request ∈ Requests
    axm2  :  Access_Accept ∈ Responses
    axm3  :  Access_Reject ∈ Responses
    axm4  :  Access_Challenge ∈ Responses
END
```

Then we can use two variables to represent the paquets send by the client and the server: paquet_client to denote the request that have been sent, and paquet_server to indicate the response that have been given.

```
VARIABLES
    paquet_client
    paquet_server
INVARIANTS
    inv1  :  paquet_client ⊆ Requests
    inv2  :  paquet_server ⊆ Responses
```

Initially, there are no requests of the client or responses from the server hence both variables are initialed by 0.

```
    INITIALISATION
    act1  :   paquet_client ≔0
    act2  :   paquet_server ≔0
```

When the client chooses to use RADIUS, it creates an "Access_Request" containing some information and sends it to the server side. We do not discuss in this paper the information that is in the message; we just focus about the operation that happened between the client and the server.

```
    clt_access_request
ANY
      msg
WHERE
      grd1  :       msg = Access_Request
      grd2  :       msg   ∉ paquet_client
THEN
      act1  :   paquet_client ≔ paquet_client ∪ {msg}
END
```

Access-Accept packets are sent by the RADIUS server, and provide specific configuration information necessary to begin delivery of service to the user.

```
    srv_access_accept
ANY
      msg
WHERE
      grd1  :     msg = Access_Accept
      grd2  :    msg  ∉  paquet_server
```

```
THEN
        act1  :  paquet_server ≔  paquet_server ∪ {msg}
END
      srv_acces_challenge
ANY
        msg
WHERE
        grd1  :     msg = Access_Challenge
        grd2  :     msg ∉ paquet_server
THEN
        act1  :  paquet_server  ≔ paquet_server ∪ {msg}
END
```

```
      srv_access_reject

ANY
        msg
WHERE
        grd1  :     msg = Access_Reject
        grd2  :     msg ∉ paquet_server
THEN
        act1  :  paquet_server ≔ paquet_server ∪ {msg}
END
```

C. *First refinement*

We are going to refine our abstract model to a more concrete one, by adding new variables and modifying our existing events. For this we introduce the client status and a timer. We define a carrier set named STATUS. It is made of three distinct elements: valid, invalid, moreinfo, which present the RADIUS client status.

```
SETS
      Statut
CONSTANTS
      valid
      invalid
      moreinfo
AXIOMS
      axm1  :  Statut ={valid, invalid, moreinfo}
      axm2  :  valid≠ invalid
      axm3  :  moreinfo ≠ invalid
      axm4  :  moreinfo ≠ valid
END
```

We can use in this refinement three variables to represent client status, which can be valid, invalid or moreinfo, the variables T and Time to indicate the timing.

```
VARIABLES
      client_st
      T
      Time
INVARIANTS
```

```
        inv1  :  client_st ∈ Statut
        inv2  :  T ∈ ℕ
        inv3  :  Time ∈ BOOL

        inv4  :  client_st=valid ⇒( ∀ m·m∈ Responses ∧ m
=Access_Accept)
         inv5  :  client_st=moreinfo ⇒( ∀ m·m∈ Responses ∧
m =Access_Challenge)
         inv6  :   ∀ m·m∈ Responses ∧ m =Access_Reject⇒
client_st=invalid
```

The Access-Request is submitted to the RADIUS server via the network. If no response is returned within a length of time, the request is re-sent a number of times.

Upon receipt of an Access-Request from a valid client, an appropriate reply must be transmitted

```
       clt_access_request
REFINES
        clt_access_request
ANY
        msg
WHERE
        grd1  :       msg = Access_Request
        grd2  :       msg ∉ paquet_client
        grd3  :       Time = FALSE
THEN
        act1  :   paquet_client ≔  paquet_client ∪ {msg}
        act2  :   Time ≔ TRUE
END
```

If the client is valid then the RADIUS server sends Access-Accept response to the client.

```
              srv_access_accept
REFINES
        srv_access_accept
ANY
        msg
WHERE
        grd1  :    msg = Access_Accept
        grd2  :    msg ∉ paquet_server
        grd3  :    client_st = valid
THEN
        act1  :   paquet_server ≔ paquet_server ∪ {msg}
END
```

If any condition is not met, the RADIUS server sends an "Access-Reject" response indicating that this user request is invalid.

```
       srv_acces_challenge
REFINES
        srv_acces_challenge
ANY
        msg
WHERE
```

```
        grd1  :      msg = Access_Challenge
        grd2  :      msg ∉  paquet_server
        grd3  :      client_st = moreinfo
THEN
        act1  :   paquet_server ≔ paquet_server ∪ {msg}
END
```

```
      srv_access_reject
REFINES
        srv_access_reject
ANY
        msg
WHERE
        grd1  :     msg = Access_Reject
        grd2  :     msg ∉ paquet_server
        grd3  :     client_st = invalid
THEN
        act1  :
                 paquet_server ≔  paquet_server ∪ {msg}
END
```

The server can respond to this new Access- Request with either an Access-Accept, an Access-Reject, or another Access-Challenge.

The last event in our model is the event of timing.

```
    time
WHEN
      grd1  :      Time = TRUE
THEN
      act1  :        T  ≔  T+1
END
```

IV. Conclusion

In this paper we have presented formal modeling of the RADIUS protocol using Event B.

In this approach the modeling process starts with an abstraction of the protocol which specifies the goals of the protocol. In our case study, presents message exchange between the client and the server without considering any condition are the main protocol goals. The abstract level of our Event-B model shows these goals in a very general way, and then during refinement level, features of the protocol are modeled and the goals are achieved in a detailed way.

The use of Event-B and Rodin as a formal modeling environment has several advantages. Firstly, the model can be gradually developed by step-wise refinements, which allows hierarchical design exploration at different abstraction levels. Secondly, the obligation to discharge POs ensures full model consistency throughout all levels.

For our future work, we would like to develop a Diameter protocol which is similar to RADIUS and compare them in event B method.

References

[1] S. El Mimouni,R. Filali,A. Amamou,B. Boulamat and M. Bouhdahi, “A Mechanically and Incremental Development of the Remote Authentication Dial-In User Service Protocol “ ,Proceedings of the 1st International Conference on Mathematical Methods & Computational Techniques in Science & Engineering (MMCTSE 2014), pp.199-203. 2014

[2] M. Sirjani, M.M. Jaghoori, S. Forghanizadeh, M. Mojdeh, and A. Movaghar. “Model Checking CSMA/CD Protocol using an Actor-Based Language”, in the Proceedings of the International Conference on Software Engineering, WSEAS, February 2004.

[3] E. Antonidakis, “Conferencing protocols and petri net analysis”, WSEAS Transactions on Computers, vol. 5, no 12, pp. 3112-3118, 2006.

[4] X. B. Li, F.X ZHAO , “Formal development of a washing machine controller by using formal design patterns,” Proceedings of the 3rd WSEAS International Conference on COMPUTER ENGINEERING and APPLICATIONS (CEA'09) , pp. 127–132, 2009.

[5] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.

[6] M.,Butler, “Incremental Design of Distributed Systems with Event-B”. In: Marktoberdorf Summer School 2008 Lecture Notes. IoS (November 2008)

[7] C. Rigney, A. Rubens, W. Simpson and S. Willens, RFC 2058: Remote Authentication Dial In User Service (RADIUS). [Online].Available: www.ietf.org/rfc/rfc2058.txt, January 1997.

[8] C. Rigney, RFC 2059: Radius Accounting [Online].Available: www.ietf.org/rfc/rfc2059.txt , January 1997.

[9] C. Rigney, A. Rubens, W. Simpson and S. Willens, RFC 2138: Remote Authentication Dial In User Service (RADIUS). [Online] Available: www.ietf.org/rfc/rfc2138.txt, April 1997.

[10] C. Rigney, RFC 2139: Radius Accounting [Online].Available: www.ietf.org/rfc/rfc2139.txt, April 1997.

[11] C. Rigney, A. Rubens, W. Simpson and S. Willens, RFC 2865: Remote Authentication Dial In User Service (RADIUS). [Online]. Available: www.ietf.org/rfc/rfc2865.txt , June 2000.

[12] C. Rigney, RFC 2866: Radius Accounting [Online].Available: www.ietf.org/rfc/rfc2866.txt, June 2000

[13] M. Jeannette, A. Wing. “specifier’s introduction to formal methods”. IEEE Computer, 23(9):8–24, 1990

[14] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[15] R.-J. Back,” Decentralization of process nets with centralized control”.2nd ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing, 1983.

[16] R.-J. Back, Refinement Calculus II: Parallel and Reactive Programs. In: de Bakker J. W., de Roever W. P., Rozenberg G. (eds.), Lecture Notes in Computer Science, Springer, vol 430, pp. 67-93, 1990.

[17] R. J. Back and R. Kurki-Suonio. “Distributed cooperation with action systems”. ACM Transactions on Programming Languages and Systems. 10(4): 513–554, 1988.

[18] L. Lamport. Specifying Systems:” The TLA+ Language and Tools for Hardware and Software Engineers”. Addison-Wesley, 1999.

[19] L.Lamport, “The temporal logic of actions,” Transactions on Programming Languages and Systems (TOPLAS), vol.16 no.3, pp. 872-923, 1994.

[20] K. Chandy, J. Misra, “Parallel Program Design: a Foundation”, Addison-Wesley, 1989.

[21] W.-P. de Roever and Kai Engelhardt “Data Refinement: Model-oriented Proof Theories and their Comparison” Cambridge Tracts in Theoretical Computer Science, vol. 46. Cambridge University Press, Cambridge (1998)

[22] A. Rezazadeh, M. Butler, N. Evans.: “Redevelopment of an Industrial Case Study Using Event-B and Rodin.” In: BCS- ACS Christmas 2007 Meeting – Formal Method. In: Industry (2007)

[23] DEPLOY FP7 Project, [Online].Available: http://www.deploy-project.eu , January 2014.

[24] J.R Abrial.: Summary of Event-B proof obligations (2008), [Online]. Available: http://www.docstoc.com/docs/7055755/

[25] Event-B and RODIN. Available: http://wiki.event-b.org, April 2011.