HIGH QUALITY IMPLEMENTATION FOR AUTOMATIC GENERATION C# CODE BY EVENT-B PATTERN

Eman K Elsayed[1] and Enas El-Sharawy[2]

[12] Mathimatical and Computer Science Department, Faculty of Science(girls),
Al-Azhar University, Cairo, Egypt

ABSTRACT

In this paper we proposed the logical correct path to implement automatically any algorithm or model in verified C# code. Our proposal depends on using the event-B as a formal method. It is suitable solution for un-experience in programming language and profession in mathematical modeling. Our proposal also integrates requirements, codes and verification in system development life cycle. We suggest also using event-B pattern. Our suggestion is classify into two cases, the algorithm case and the model case. The benefits of our proposal are reducing the prove effort, reusability, increasing the automation degree and generate high quality code. In this paper we applied and discussed the three phases of automatic code generation philosophy on two case studies the first is "minimum algorithm" and the second one is a model for ATM.

KEYWORDS

Formal method, Event-B, Pattern, Code generation, RSM (Resource Standard Metrics) .

1. INTRODUCTION

This paper is not the first step in our work in the formal methods field, but we start from the series of research as in references [1,2], Where we found that formal methods are used to construct models by several advanced theories. There are many formal methods used in various domains for constructing models of complex system together with a several advanced theories and tools [1]. In this paper we automatic generate the C# code for any event-B patterns. That for more generality and distribute on the web. That is to enhance the refinement process.

The introduction of the RODIN [3] platform allows such extensions to be provided by third-party developers, and translation of Event-B to the C# programming language has always been intended. We discuss B2C# as an extension to RODEN and describe its use on two examples. The B2C tool was specifically developed for the generation of source code for MIDAS [4]. MIDAS was developed as a demonstration of a generic Event-B model, capturing the generic properties of binary Instruction Set Architectures (ISAs) in order to provide a reusable template. GCC [3] compilers were developed and used to generate binary executable from hand coded test suites.

Event-B does not specify how to generate executable code from models, and the Rodin platform in its basic form cannot translate models into a programming language without the use

of extensions. It was mainly intended for use as part of a virtual machine project, and supported translation of the most important Event-B constructs. This approach was taken a step further towards a more general-purpose tool, albeit an experimental one in [5].

The model has to be refined according to the Event-B refinement rules (e.g. using the Rodin) until the events only contain concrete constructs that have direct equivalents in C#. The guard of the event is translated into a method returning a Boolean value reflecting enabled-ness, whereas the action results in a separate method containing the C# equivalent of its assignments. C# code generation has a multi-phased translation process. An Event-B pattern is initially restated in an easily translatable sub-set of the notation “Rewrite Phase”. C# code is then automatically generated from the pattern via an appropriate tool “Translation Phase”. Finally “Build Phase” adds support functions and the compiler of the source code language. The paper is organized as the following, after the introduction section two for main concepts. Translation Philosophy is presented in section three. Section four presents the case studies about algorithm and model. Then we present the conclusion and further work directions. Finally we listed the references.

2. MAIN CONCEPTS

This section presents the summary about the main concepts we will build the work on them. We start by overviewing the Event-B formalism and RODIN platform, then we present brief summary about pattern in event-B method.

2.1 Event-B and Rodin

The B Method in reference [6] is a formal approach for the specification and rigorous development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [7]. Event-B [8] is a formal framework derived from the B Method to model and reason about parallel, distributed and reactive systems. Event-B has the associated RODIN platform [9,7], which provides automated tool support for modeling and verification by theorem proving. Then Event-B is an extension of the B-method for specifying and reasoning about complex systems including concurrent and reactive systems. An Event-B model is described in terms of contexts and machines as shown in Figure 1.

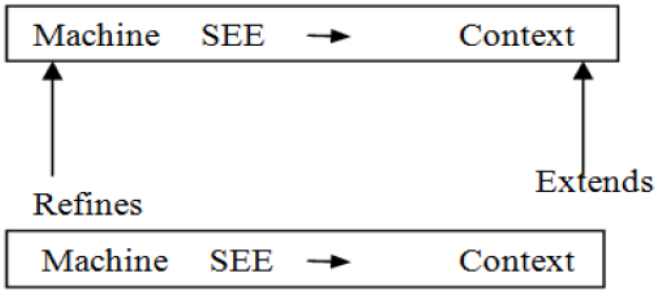


Figure1. The relation between Event-B Machine and context

Reference [11] describes the model components in RODIN as the following:

The ***contexts*** contain the static parts. Each has sets, constants, axioms and theorems. Axioms are to describe the properties of the sets and constants. Theorems are which it must be proved. Moreover, contexts extended by others and seen by more than one machine.

The ***Machines*** contain the dynamic parts were used to provide the behavioral properties of the model. It consists of states and events. The state is defined by means of variables. Each event is composed of three elements: an event name, actions and guards (conditions). Any machines can refine by other.

The Rodin platform is an open and extensible tool for Event-B specification and verification [6,12]. RODIN have various useful plug-ins such as a proof-obligation generator, provers, model-checkers, UML transformers, etc .

2.2 Design Pattern

In our reference [1] we presented more details about design pattern using event-B. That to have approach to reuse the final refinement model in a new development. The proofs of the patterns are saved also. the generation of a refinement of the problem at hand is correct by construction and no proof obligation needs to be generated again. The correctness of the construction relies on a correct matching of the pattern with the problem at hand.

3. C# Translation Philosophy

Automatic generation of source code may be regarded as an “open-loop” refinement step. When there is not static equivalence checking against the previous refinement is possible. Minimize the complexity and minimize the risk of bugs are aims in transformation within RODIN environment. For enforce and discharge of resulting proof obligations, convert of event B notation to C# form.

3.1 The Rewrite Phase

After final refinement step and converted the final model to be a pattern, then using an easily translatable subset of event-b. Table 1was show the general translation from event-B syntax to C# code.

Event-B	C#	Comment
n..m	int	Integer type
x ∈ Y	Y x;	Scalar declaration
x ∈ n..m ⟶ Y	Y x[m+1];	Array declaration
x :∈ Y	/* No action */	Indeterminate initialization
x = y	if(x==y) {	Conditional
x ≠ y	if(x!=y) {	Conditional
x < y	if(x<y) {	Conditional
x ≤ y	if(x<=y) {	Conditional
x > y	if(x>y) {	Conditional
x ≥ y	if(x>=y) {	Conditional
x = y + z	x = y + z;	Arithmetic assignment
x = y - z	x = y - z;	Arithmetic assignment
x = F(y)	F(y,&x);	Function assignment
(a↦b) = F(x↦y)	F(x,y,&a,&b);	Function assignment
x = a(y)	x = a(y);	Array assignment
x ≔ y	x = y;	Scalar action
a ≔ a ⩤ {x↦y}	a(x) = y;	Array action
a ≔ a ⩤ {x↦y} ⩤ {i↦j}	a(x)=y;a(i)=j;	Array actions

Table 1 The general translation from event-B syntax to C# code.

First, in the context each costant converts to its literal value. Gurds range convert to comparison statements.Abstract sets convert to its numerical meanings via mapping functions. Also the global variables are disallowed from the right of assignments and must be restated as intermediate local variables. But logical OR is not supported. So we solve this by divided events with this type of guards into two events. Also we can merge events. This convert requirement does not contribute to the C# translation process,but rnforces division of events into simplest form.

3.2 The Translation Phase

Once pattern has been converting manually, all selected events- without explicit- was translated automatically. A single C# file is produced for each leafmachine. B2C# translation is invoked by a single user action in figure 3. This phase classified into two subsections: first is the event translation and the second is calling function generation.

3.2.1 Event Translation

It is the first translation part. Each event translates in an individual C# function. We must note that null event (i.e. with false guard) is not translated. The comment inserted automatically. This automatic reduction is performed to avoid generation of unreachable run-time code. Events run in the same order defined in event-B model.

3.2.2 Calling function generation

The second translation part is inserted calling functions to attempt execution of each event function until triggering is detected. We note that the calling function implicitly introduces determinism into models containing non-deterministic event triggering.

3.3 The Build Phase

Once automatic translation of the Event-B model is complete, an execution environment must be provided and compiled by a suitable C# development tool chain. Implementing functions must be provided for all un-interpreted functions, the instrumentation functions, and deadlock handling functions if any have been generated. The C# file generated after define file "EventbIncludes.h" whose inclusion has been automatically inserted. A top-level C# main function must be provided to call the generated functions "INITIALISATION" and "Iterate".

4. CASE STUDIES

We apply the previous phases to generate C# code on the pattern for the minimum algorithm in section 4.1 and section 4.2 present the three phases on ATM model.

4.1 Automatic C# Code Generation from Minimum Search Algorithm

4.1.1 Rewrite Phase of Minimum Search Algorithm

Rewrite phase contains rewriting of the final refinement step of minimum search algorithm to a pattern to make the event-B model in a more appropriate translatable form for C#, using Table 1 we can rewrite event-B expressions according to C# language rules.

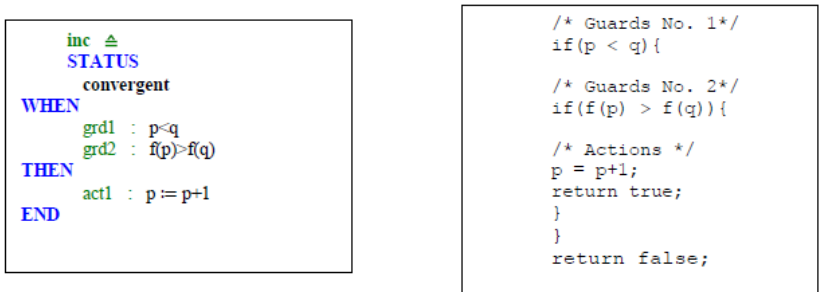


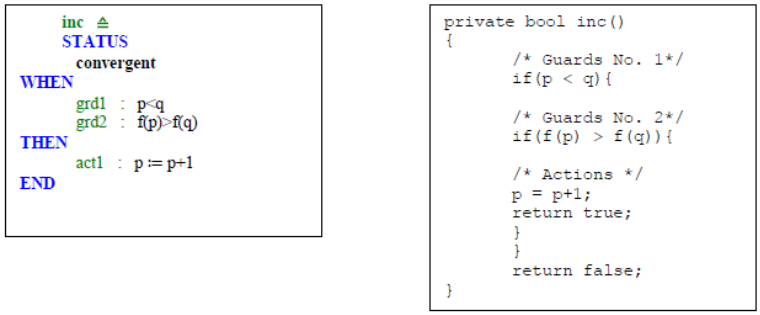
Figure 2. Example of rewritten event-B event (inc)

4.1.2 Translation Phase of Minimum Search Algorithm

This phase classified into two subsections: event translation and calling function generation.

4.1.2.1 Event Translation of Minimum Search Algorithm

Translation phase contains first event translation that is will be done by converting each event of event-B algorithm to an individual C# function. See the following example:



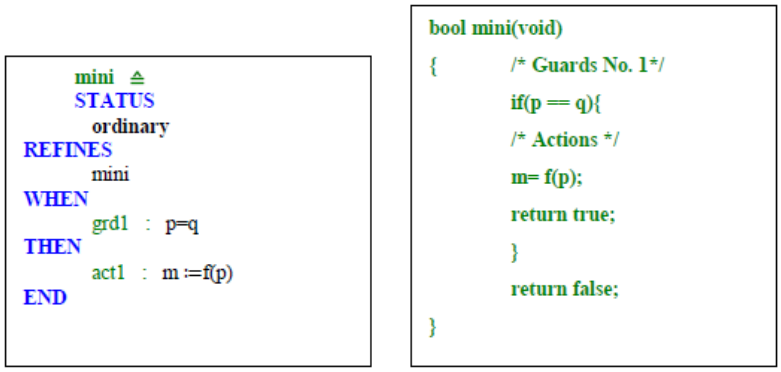
```
inc ≙
STATUS
  convergent
WHEN
  grd1 : p<q
  grd2 : f(p)>f(q)
THEN
  act1 : p := p+1
END
```

```
private bool inc()
{
        /* Guards No. 1*/
        if(p < q){

        /* Guards No. 2*/
        if(f(p) > f(q)){

        /* Actions */
        p = p+1;
        return true;
        }
        }
        return false;
}
```

Figure 3. Example of Event translation for inc. event

4.2.1.1 Calling Function Generation of Minimum Search Algorithm

Translation phase contains second a calling function of minimum search algorithm that is will be inserted after all events have been translated, when an event is detected the inserted calling function will be trigger.



```
mini ≙
STATUS
  ordinary
REFINES
  mini
WHEN
  grd1 : p=q
THEN
  act1 : m :=f(p)
END
```

```
bool mini(void)
{       /* Guards No. 1*/
        if(p == q){
        /* Actions */
        m= f(p);
        return true;
        }
        return false;
}
```

Figure 4. Example of mini event and derived C# function

```
/* Iterate function to call every events except INITIALISATION event*/

 public bool Iterate()

{
       if( mini()==true ) return true;
       if( inc()==true ) return true;
       if( dec()==true ) return true;

       /* Signal deadlock */
       return false;
}
                      /* main() function to start execution */

public static void Main()
 {
mini_1 objmini_1 = new mini_1();
       if( objmini_1.INITIALISATION()==true ){
       for(int n=0;n<=1000;n++){
       objmini_1.Iterate();
       } } }
};//class close
```

Figure 5. Example of generated event-calling function for minimum search algorithm

4.1.3 The Build Phase of Minimum Search Algorithm

A top-level C# main function must be provided to call the generated functions "INITIALISATION" and "Iterate". The calling of INITIALISATION function of minimum search algorithm must be called before Iterate.

```
/*Source code generated from RODIN project [mini] file [mini_1]*/
/* Generated [18.22] on [27/6/2013] */
/* Translation Begins Here */
using System;
using System.Collections.Generic;
public class mini_1{
/* Global Constants defined of [mini_ctx] */

const long[] f=new long[1000];
/* C# array declaration when constant is given in range of predicate */
/* No translatable type found for [n] */
/* Global variables defined in [mini_1.mch] */
ulong m; /* Integer in range undefined */
int p; /* Integer in range */
int q; /* Integer in range */

/* Event1 [INITIALISATION] */

public bool INITIALISATION()
{
/* No guards in this event */

/* Actions */
m = 0;
p = 1;
q = n; return true;}
```

Figure 6. Example of translation header of minimum C# derived code

```
                    /* Translation Begins Here */
using System;
using System.Collections.Generic;
public class mini_1{
/* Global Constants defined of [mini_ctx] */
const long[] f=new long[1000]; /* C# array declaration when constant is
given in range of predicate */
/* Global variables defined in [mini_1.mch] */
ulong m; /* Integer in range undefined */
int p; /* Integer in range */
int q; /* Integer in range */
public bool INITIALISATION()
public bool Iterate()
/* main() function to start execution */
public static void Main(){
 mini_1      objmini_1 = new mini_1();
      if( objmini_1.INITIALISATION()==true ){
      for(int n=0;n<=1000;n++){
      objmini_1.Iterate(); }}}}
};//class close
                          /* End of The Translation */
```

Figure 7. Example of calling function environment for minimum search algorithm

4.2 Automatic C# Code Generation from ATM Pattern Model

An auto teller machine ATM is a machine that allows bank customers to do some of the banking transactions 24 hours per day. We create the ATM model in Event-B. But in the model case we must sure that the automatic Proof is 100% so we used SMT prover as we present in reference [13].

4.2.1 Rewrite Phase of ATM Pattern Model

Rewrite phase contains rewriting of the final refinement step of ATM model to a pattern to make the event-B model in a more appropriate translatable form for C#, using Table 1 we can rewrite event-B expressions according to C# language rules.

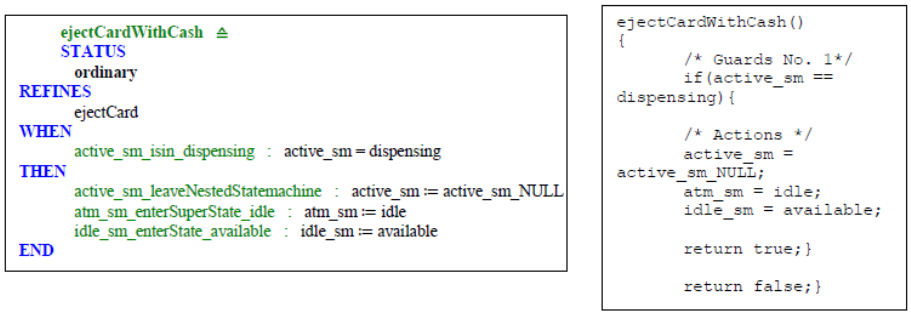


Figure 8: Example of rewritten event-B event (ejectCardWithCash)

4.2.2 Translation Phase of ATM Pattern Model

This phase classified into two subsections: event translation and calling function generation.

4.2.2.1 Event Translation of ATM Pattern Model

Translation phase contains first event translation that is will be done by converting each event of event-B ATM model to an individual C# function. See the following example:

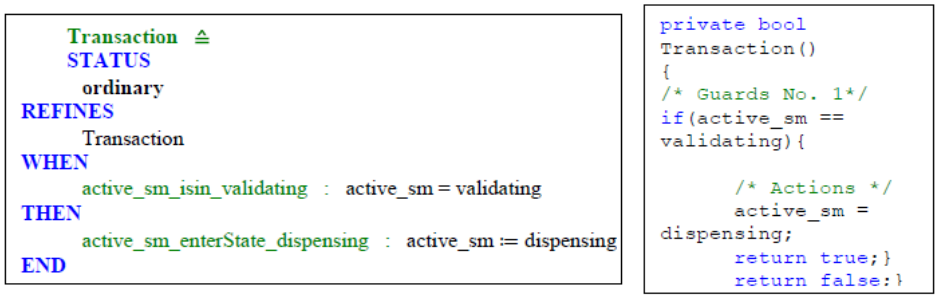


Figure 9. Event translation for Transaction event

4.2.2.2 Calling Function Generation of ATM Pattern Model

Translation phase contains second a calling function of ATM model that is will be inserted after all events have been translated, when an event is detected the inserted calling function will be trigger.

```
/* Iterate function to call every events except INITIALISATION event*/
 public bool Iterate()
{       if( insertCard()==true ) return true;
        if( ejectCardWithCash()==true ) return true;
        if( Transaction()==true ) return true;
        if( ejectCardWithoutCash()==true ) return true;
        if( offline()==true ) return true;
        if( online()==true ) return true;
        if( pinEntered()==true ) return true;
        return false;
}
                        /* main() function to start execution */
public static void Main()
 {
ATM_R  objATM_R = new ATM_R();
        if( objATM_R.INITIALISATION()==true ){
        for(int n=0;n<=1000;n++){
        objATM_R.Iterate();
        } }}
};//class close
```

Figure 10. Example of generated event-calling function for ATM model

4.2.3 The Build Phase of ATM Pattern Model

A top-level C# main function must be provided to call the generated functions "INITIALISATION" and "Iterate". The calling of INITIALISATION function of ATM model must be called before Iterate.

```
/*Source code generated from RODIN project [ATM] file [ATM_E]*/
/* Generated [0.23] on [25/12/2013] */

/* Translation Begins Here */

using System;
using System.Collections.Generic;

public class ATM_R{

                        /* Global Constants defined of [ATM_A_implicitContext] */
```

Figure 11. Part of translation header of ATM derived C# code

5. THE RESULT ANALYSIS

In this section we make comparison between manual and automatically correct C# code. RSM (Resource Standard Metrics) is an application using to measure the quality C, C++, C# and Java programming language. It was created to use on all Windows and UNIX operating systems [13].

RSM is program analyses source code for quality issues but the compiler will not enforce. Use as prerequisites to code peer review or analyze subcontracted source code. By using RSM quality application we obtain an understandable report of quality analysis notices with easy regular expressions for user required quality parameters.

The following figures show RSM output analysis reports for the manual and event–B automatic C# code generation for the minimum algorithm in figures 12,13, and for ATM model in figures 14,15.

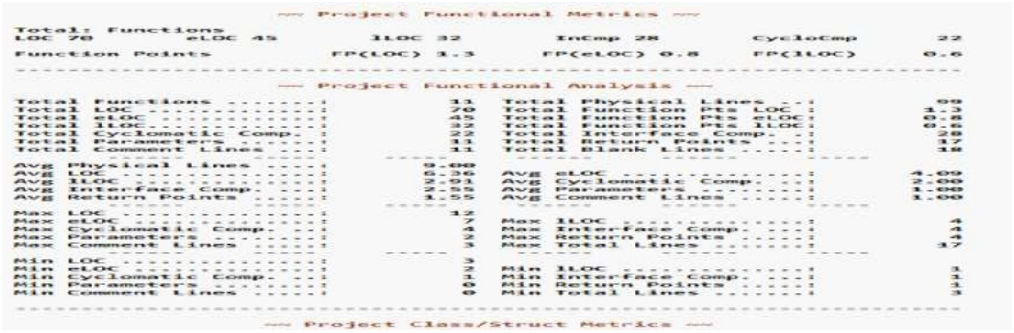


Figure 12. RSM report screen shot for automatically generated C# code of minimum algorithm

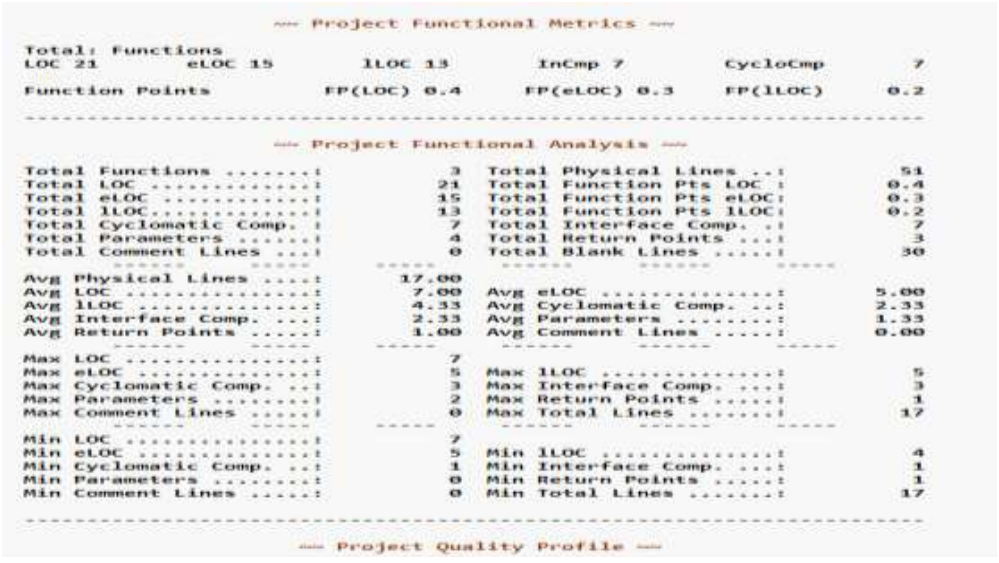


Figure 13. RSM report screen shot for C# minimum algorithm's code wrote by hand

Figure 14. RSM report screen shot for automatically generated C# code of ATM model

Figure 15. RSM report screen shot for C# ATM code wrote by hand

In the case of generation code for minimum algorithm, the result is translated to C# code then we measure the quality of the derived C# code and compare with manual programming in C# language. The result obtained from our approach have suitable comments automatically and don`t depend on user. The generated call functions are defined in the file “EventbIncludes.h”, whose inclusion has been automatically inserted. These results proved that using event-B pattern with SMT prover achieve the improvement of software quality.

6 CONCLUSIONS

In this paper we conclude that using event-B pattern for automatic translated web language as C# source code, provided that sufficient refinement has been performed to introduce full determinism and use an easily translatable sub-set of the notation. The RODIN tool supports development of an appropriate plug-in translation tool well, by provision of all necessary model information via supported interfaces. Automatic generation of a pattern to a language outside the Formal Methods environment precludes static equivalence checking of this final “refinement” step. Also we measure the quality of this code to assess the event-b model.

The benefits of our proposal in this paper are to reduce the proving effort, to reuse a model and to increase the degree of automation [1]. Also avoid un-interpreted functions, the instrumentation functions, and deadlock handling functions if any have been generated.

When we obtained an executable C# code from formal method like event-b, we could achieve the integration between requirements and coding. Addition to the completely logically correct and generate automatically.

Finally, we think that series of steps from create event-B model with then create event-B pattern then generate C# code is a new generation in high quality software engineering for mathematician.

As a future work Automatic Rewrite Phase will be improved to make it automatic instead of user interaction edition. The translation process will be completely automatic translation and adding more guided phases to learn unexperienced user.

7. REFERENCES

[1] Eman Elsayed, Gaber El-Sharawy and Enas El-Sharawy(2013) “Integration Of Automatic Theorem Provers In Event-B Patterns” IJSEA. Vol. 4 No 1 http://www.airccse.org/journal/ijsea/current2013.html.

[2] Elsayed,E. ,El-Sharawy,G. and El-Sharawy,E.(2013)”Enhancing Event-B Pattern” LAMBERD academic publishing, Germany.

[3] Abrial, J.-R.; Butler, M. ; Hallerstede, S.; Hoang, T.S. ; Mehta, F. & Voisin, L. , (2010) ”Rodin: an open toolset for modelling and reasoning in Event-B”, International Journal on Software Tools for Technology Transfer (STTT) (6) 447–466.

[4] Butler, M., (2009) “Decomposition Structures for Event-B”. In: Integrated Formal Methods. Lecture Notes in Computer Science, vol.5423,pp.20–38.Springer,Berlin. http://www.springerlink.com

[5] Abrial, J.-R. ,(2007) “A system development process with Event-B and the RODIN platform”, in: Proceedings of International Conference on Formal Engineering Methods, ICFEM’07, in: Lecture Notes in Computer Science, vol. 4789, Springer-Verlag, pp. 1–3.

[6] Voisin,L.(2006) “A Description of the RODIN Prototype”, http//rodin.cs.ncl.ac.uk.

[7] Stallman,R. (2001)“Using and Porting the GNU Compiler Collection”, Free Software Foundation

[8] Abbasi, K. ; Akkaya, M. & Younis, A., (2007) “A distributed connectivity restoration algorithm in wireless sensor and actor networks”, in: Proceedings of 32nd IEEE Conference on Local Computer Networks, LCN’07, IEEE, pp. 496–503.

[9] Wright, S. (2008) “Using EventB to Create a Virtual Machine Instruction Set Architecture”, Abstract State Machines, B and Z, SpringerLink,

[10] Butler,M Hallerstede,S(2007) "The Rodin Formal Modelling Tool” BCS-FACS Christmas 2007 Meeting –Formal Methods In Industry

[11] Hallerstede,S(2008)"On the Purpose of Event-B Proof Obligations“ Abstract State Machines, B and Z, SpringerLink.

[12] Abrial, J.R. ; Butler, M. ; Hallerstede, S.; Hoang T.S., Mehta, F. &Voisin, L.(2010) “RODIN: an open toolset for modelling and reasoning in event-B”, Int. J. Softw. Tools Technol. Transf. 12(6), 447–466

[13] El-Sharawy,E.(2011(“Design of Software Quality System”,Master Thesis, Cairo, Egypt, pp76.