Article

The Event-B Modelling with Operators-based Refinement of NoC-based Wireless Sensors Networks

HARICHE Abdelhamid 1, BELARBI Mostefa 2, CHOUARFIA Abdallah 3 and CHAIB Abdelkader 4

1 LIM research laboratory, University of Tiaret, Algeria; abdelhamid.hariche@univ-tiaret.dz
2 LIM research laboratory, University of Tiaret, Algeria; mbelarbi@univ-tiaret.dz
3 University of USTO, Oran; chouarfia@univ-usto.dz
4 LIM research laboratory, University of Tiaret, Algeria; aekshaib@gmail.com

Abstract: The need for high performance, available and reliable embedded systems has made computing systems increasingly complex. Formal methods have the ability to produce critical systems for large industrial projects, and this by creating an original mathematical model that can be formally refined in levels until the final refinement which contains enough of details for an implementation. This work is a first step of VHDL code generation process, it represents how to elaborate formally an embedded system using Abstract data type in a form of theories (NoC, WNoC, colored graph, VHDL) and how to ensure in systematic way all the details and complexity of this system using operators of refinement (Create, Rename, Restrict, Enrich) that are recently proposed for the Event-B method. All the theories are deployed, discharged and used in Event-B models to represent and enhance the performance of this self-organization reliability solution for the wireless sensors network of NoC-based system. This paper summerize the fruit of using new proposed approach for the Event-B formal method that persist the NoC-based ditributed system instead of consuming more than 70% of realization time with any analytic method.

Keywords: Refinement; operators; theories; self-recovering; NoC; Event-B; VHDL

1. Introduction

The complexity (in terms of reduced time and reliability) of embedded systems implementation (just like the FPGA-based embedded systems: the multiprocessor systems on a chip MPSoC based on NoC network architecture) is increasing in term of the algorithms proceeded for this kind of system. Therefore, many bugs in the implementation or by incomplete or incorrect specifications may lead to unintended algorithm. In that reason to provide with only one language an reliable result for the VHDL (Very High Speed Integrated Circuit Hardware Description Language) code generation has still insufficient. Any kind of verification (formal, informal and hybrid verification) is a used to demonstrate well accordance of the development with their specification [1,2], around 70% of the resources employed in a hardware design are used in the functional verification step [1].

Many producers of NoC systems even engineers always wonder what are the real benefits of formal methods and Why should we care about them? When and Where should we expect to use them, and Who should be involved? The answers may not be clearly presented if the problem is so simple that one can write the code straight away, but in it is so useful when the description of the system is too far big for any human purpose. As many claim that using formal methods [3-5] is too challenging because of various intrinsic difficulties. The usage of formal methods has to be incorporated within a development process with guidelines of products in the formation of algorithms that may take a significant time. Some formal methods [6,7] need a transformation process called the refinement to provide a new formulation with no contradiction of the properties in the aim of approaching to an implementation in a programming language. The research presented is about the implementation of a way to automate this transformation, in order to operate systematically this refinement and must ensure the conservation of properties between descriptions.
Event-B is a formal method based on the B method [8] provides the toolset RODIN [9,10] to carry out specifications, refinements and proofs ([11,12]) in Event-B. However, one of the goals of this paper is to make the Event-B may allow executable code generation, where the specifications have implementation details based on the notion of theory (theory Plug-in) to express NoC desired properties of the XY routing algorithm presented by the VHDL code behavior. The fact of operating this phase of refinement [9,11] between two written specifications in Event-B formalism simplify the way of validation of the properties that the architect wish to see such as alertness, safety, reachability or termination. In the classical modeling Every model contains theories in form of different types (integers, sets, relations, functions . . . ), so when systems require additional mathematical structures (e.g: lists, trees, graphs, reals) which might have defined axiomatically in models, theory plug-in with the definitions and proof rules is the new extension of modelling to use. However, our approach of modelling is a real close form of the VHDL code using refinement operators unlike in Event Extension approach had the reason to illustrate the refinement in RODIN tool-set with an Abstract Data Type algebra[13–16], we give illustrations in this paper via the verification examples of the NoC-based Wireless sensors network (WSNoC) system by proposing set of basic theories[17]: starting by the NoC theory [18–20], then the self-recovering mechanism of nodes inside the wireless network using the graph theory [21,22], then to animate and increase the complexity of verification of the auto-organization algorithm we use the colored graph theory. In the final phase, we represented all these properties using VHDL theory. However, all the feasible scenarios for our system following will be checked with an incremental fashion based on special operators to cover different axis and in a methodological way. This paper is organized as follows: Section 2 shows a brief state of art for the formal validation of some algorithm of embedded systems and the operator refinement, the section 3. presents the formalization of the NoC system using the classical modeling using Event-B method, when in section 4 we show the new extension to model this system using refinement operators, then the section 5 illustrates the result of using refinement operators of the Event-B wireless network of NoC sensors and by using the theory plug-in extension. Section 6. concludes this paper along with the future work.

2. Related Works

This section introduces some current works that had a relation with the verification of mechanism and algorithms of embedded systems in general and the wireless network sensors especially the strategy of self-recovering of faulty nodes starting our search in the literature for the survey of systems using formal method it must be mentioned the one for the component of the FireWire system [23] which is the network leader election protocol, or Tree Identify Protocol[24]. This protocol has been specified and analyzed in a variety of formalisms, including I/O automata, E-LOTOS, and $\mu$CRL. In other way there are no enough data on the length of time it takes within the system development life-cycle (accompanying descriptions in other formal, semiformal, or informal notations). The brief study for the projects issued with our particular wireless network systems starts the ESL (Embedded System Level) tools [25–27] even if there are some conditions about this approach [28].

More of challenges when a transformation from a POOSL model into an UPPAAL model has made to enable formal verification [29] and match the estimation results obtained from the POOSL model to ensure that reconfigurations algorithm follow the intended rules without any invalid configurations. While expressivity is needed for mastering heterogeneity and complexity such the clock synchronization protocol running within a distributed heterogeneous communication system (HCS). SBIP the stochastic extension of the BIP formalism and toolset allows specifying systems with both non-deterministic and stochastic aspects [30]. However, from the user point of view, dealing with SBIP is as easy as dealing with BIP.

In our particular system self-recovering or self-reconfiguration is one important factor during the communication inside a network of sensors. Many works focused on this strategy by attribute some consideration of the functional verification of the routing algorithms using ESL tools [31,32]. The Ant Colony Optimization, with the ability to search for paths, the NoC algorithms were called
REAS (routing based on EAS [33]) and RACS (routing based on ACS 40) are tested and verified using simulations [35], with the ant algorithms, nodes are being able to find routes with latency less than that obtained with XY and OE algorithms, but simulations still take significant time to implement applications mapped on the network, showing its use in examples closer to real world NoCs. Besides those works, UML (Unified Modeling Language) and especially MARTE (Modeling and Analysis of Real-Time Embedded systems) is used to analyse the three-axis classification scheme that used to classify a reconfigurable approach by the community [28, 29, 36] for the reconfiguration (exo-reconfigurable or endo-reconfigurable, dynamic or static, full or partial). Final mention of the formal verification for the algorithm of recovering in the networks by proposing new rules using formal methods such as using Switches algebra [37].

All these previous works miss the details during the coverage of embedded systems properties and constraints. Therefore it must mentioned that some current works used RODIN tool-set to formalize the different actions inside this self-recovering network [38] when others try to use Event-B models to validate the distributed systems [10, 19, 39]. In this context, Various studies have focused on the high-level architecture descriptions ([40–51]). This suite of refinements must be formally described. Some software architecture description languages, such SADL [52] or RAPIDE [53], take into account the sophistication on several levels of abstraction, but they fail to complete the full development of complex software systems because they do not offer support for code generation.

These precedent works always are unlike our formal approach that do care about the architecture and the algorithms delivered by an embedded system with the way of running for its application even the constraint of programming language and depend on the power of proposed theories and the proven polymorphic to guarantee the right preservation of all proofs for any embedded system and especially NoC-based systems, this kind of distributed systems lead us to use set of combined theories (closure, coloured graph, NoC and its extension WNoC, VHDL) in the aim to cover the strategy of reconfiguration inside the wireless network of a set of NoC-based switches, where every level described this embedded system using Event-B theories need a mechanism (operators or refinement) to enhance the validation in a methodological way.

3. Classical modelling with Event-B

As we proceed previously, the method B was the subject of many different works ([54, 55] …). The Event-B [56, 57] for example, is an extension of classical language B with extra syntactic sugar, using post-conditions, terms, an event based syntax and explicit refinement constructions.
3.1. Modelling of NoC-reconfiguration algorithm

The network transmission is realized through routers constituting the network (usually a Mesh topology), and by using switching techniques of data packets constituted of messages and routing rules [17,19,20] (the packets direction’s (whether N, E, S or W) policy is based on the rules of right priority policy). Knowing that The NoC structure (input register, semi crossbar, out-put buffer and a finite state machine) is FPGA-based distributed system two separated levels of modelling had taken the first was for the NoC systems (the routing algorithm) when the second was for the reconfiguration of NoC-based distributed systems (mechanism for software error detection and its correction).

- The different levels of classical modelling for of the proposed fault tolerant routing process (see Figure 1 (a). And Figure 1(b)) are composed of two main structures available in the classical Event B are: contexts (to express static information about the model) and machines (to express dynamic information about the model, invariants, safety properties, and events).
- The main property of this kind of NoC-based systems is the reconfiguration. Therefore, a new separate modelling is lunch as shown in Figure 2(a). And Figure 2(b).) during this separated validation the re-use of variables (such as rcvd and sent that allows to perform the events SEND and RECEIVE) in the purpose of checking the strategy of reconfiguration of NoC-based distributed systems.
- Modelling the NoC-reconfiguration using vertex algorithm where the graph is a set of nodes and the set of colors (Red_Color, Green_Color, Blue_Color) are used to give a label for each node in the graph following its ability to send and receive data.

3.2. The classical refinement properties

The different formal methodologies offer as much refinement relations. Three basic criteria sufficient to distinguish the main categories of refinement relations (as shown in Table 1.):

- **The placement of refinement in the engineering cycle**, each approach may introduce a refinement step during the design phase to change an abstract specification (architectural or not) into a more concrete specification or lead directly to an implementation, thus confining the implementation or programming stage.
- **The type of refinement process**, refinement is sometimes can be built from the abstract specification, based on specific buildings, and sometimes it is not checked after the creation.
- **The refinement relation criterion**, Refinement may need to concrete as possible as it can the abstract behavior (thus leading to a more detailed specification of the same problem), or whether to simplify behavior corresponding to a system to more generalise the specific abstract problem, or even to make the behavior remains unchanged.
Table 1. The principal properties of refinement for formal methods

<table>
<thead>
<tr>
<th>Formal method</th>
<th>Placement of refinement</th>
<th>Type of refinement</th>
<th>Refinement relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darwin, MetaH, UniCon&amp;Weaves</td>
<td>Implementation</td>
<td>Compiler (except Weaves)</td>
<td>Compilation</td>
</tr>
<tr>
<td>Refinement calculus</td>
<td>Design/implementation</td>
<td>Processing and refinement of behavior or justification of programming</td>
<td>Refinement rules (strengthening postconditions ...) and invariants</td>
</tr>
<tr>
<td>Rapide</td>
<td>Design</td>
<td>Comparison of traces in posteri or</td>
<td>Inclusion of abstract mark on the concrete track</td>
</tr>
<tr>
<td>UML-based Catalysis</td>
<td>Design</td>
<td>Documentation in posteri or</td>
<td>Documentation</td>
</tr>
<tr>
<td>Classical B</td>
<td>Design</td>
<td>Differential</td>
<td>Weakening of preconditions and strengthening post-conditions (over determinism) possible behavior of abstraction</td>
</tr>
<tr>
<td>Event-B</td>
<td>Design</td>
<td>Differential (extension of the classical B)</td>
<td>Weakening of preconditions and strengthening post-conditions (over determinism)</td>
</tr>
</tbody>
</table>

In addition to these main criteria of discrimination, many others (see table 2. and appendix A.) can be used to specify more precisely the different aspects of the refinement relation. The previous tables (Table1.A and Table2. A) describes several methods of software engineering, representing different vertical refinement relations. We present initially by “families” in order to be able to compare, subsequently, their respective approaches refinement.

Generally there are some approaches use the refinement at the end of the software system development process (during the implementation phase), when others provide better support for the architectural refinement during the design phase. in the third category methodologies provide specific buildings to model a concrete specification from the abstract specification (during the constructive way design phase): refinement does not have to be justified in posteriori, it constitutes in itself a development phase.

3.3. Comments and discussion

The classical way modeling of this complex system was force to handle separately each mechanism with its different properties in a set of models with the set of relationships for the process of refinement, then go to the last step of code generation after satisfying all the requirement validation for the complex system (for example the wireless nets based on self-organized Network on chip), this is why we try to show the benefit of using Event-B refinement during this modeling.

The conclusion drawn from all previous refinement study is the need of our approach of the refinement operators in Event-B that follows more criteria in due to better audit investigation of the properties of complex algorithms for a required system. The process algebras constitute formal methods to model interactions between processes of modelling in the reason to ensure model consistency and compliance program. The approach that we propose is to treat vertical refinement of an architectural description into a specification in a the development of formal method, which will eventually lead to code generation.
Table 2. B. The properties of refinement for some formal methods

<table>
<thead>
<tr>
<th>Refinement property</th>
<th>Event-B</th>
<th>Classical method B</th>
<th>UML-based Catalysis</th>
<th>Rapide</th>
<th>Refinement calculus</th>
<th>Darwin, MetaH, UniCon &amp; Weaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR, VR Distinction</td>
<td>No</td>
<td>-</td>
<td>No</td>
<td>-</td>
<td>Decomposition &amp; style refinement</td>
<td>No</td>
</tr>
<tr>
<td>Horizontal Refinement (HR)</td>
<td>Yes (but not in all)</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>static composite components</td>
<td>-</td>
</tr>
<tr>
<td>Vertical Refinement (VR)</td>
<td>Yes (but limited)</td>
<td>Yes (but limited)</td>
<td>Yes</td>
<td>-</td>
<td>change of whole architectural styles</td>
<td>Yes</td>
</tr>
<tr>
<td>Data refinement</td>
<td>Limited</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td>For what may be defined in styles</td>
<td>-</td>
</tr>
<tr>
<td>Functional refinement</td>
<td>Yes</td>
<td>YES</td>
<td>Yes</td>
<td>No</td>
<td>For what may be defined in styles</td>
<td>-</td>
</tr>
<tr>
<td>Behavioral refinement</td>
<td>Limited</td>
<td>No</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Compositional refinement</td>
<td>Yes (Except Implementations)</td>
<td>Yes (Except Implementations)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Code generation</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiples abstraction levels</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>if the styles are considered as well</td>
<td>No</td>
</tr>
<tr>
<td>Resetting</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes bosses</td>
<td>No</td>
</tr>
<tr>
<td>Preserving the inherent properties</td>
<td>Yes (proof obligations)</td>
<td>Yes (proof obligations)</td>
<td>No</td>
<td>Yes but only in level 1</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Preservation of properties defined by the user</td>
<td>NO (not automatic)</td>
<td>NO (not automatic)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multi-languages</td>
<td>reserved keywords</td>
<td>reserved keywords</td>
<td>-</td>
<td>several sub-languages</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>
The refinement operators allow us both to bring the different constructions of an architecture description language to approach a level of abstraction implementation while ensuring compliance refinements, and afford a formal method for developing a design stage more "ergonomic".

4. New operators-based refinement in Event-B

The fact of operating this refinement phase between two written specifications in Event-B formalism [58–62] is proceeding with two types of mechanisms: The decomposition, or horizontal refinement (no change of level of abstraction by adding new components), The transformation, or vertical refinement (decreases the level of abstraction for example to say that the set of the port in for a node in a VHDL code must be denoted "std_logic"). Thus, the vertical refinement mechanisms have put in place in order of our consideration, to turn the attention during this paper to the control structure of an architectural description (routing and recovering algorithms of NoC-based systems) to explain in understandable terms proposed into the Event-B modelling development.

4.1. The current refinement approaches

The research problem is the implementation of a way to automate the transformation from a given abstract model AbM to a new model ConcM is introduced by using refinement. According to R. Burstall every language can be seeing like an Abstract Data Type model [13][15,16], we can clearly see that Event-B model has this ADT structure:

\[
\begin{align*}
\text{Model} & : \text{Context} \times \text{Machine} \\
\text{Machine} & : \text{Variant} \times \text{Invariants} \times \text{Events} \\
\text{Invariant} & : \text{variable} \times \text{inv\_predicate} \\
\text{Events} & : \text{Guards} \times \text{Actions} \\
\text{Context} & : \text{sets} \times \text{Constants} \times \text{Axioms} \times \text{Theorem}
\end{align*}
\]

For each model we can formalize that the Machine is seeing from a context with this approach:

\[
\text{See} : (M : \text{Model}, \text{Mach} : \text{Machine}) \rightarrow \text{Cont} : \text{Context}
\]

\[
\forall M, \text{Mach}, \text{Cont} \cdot \text{Mach} \in M \land \text{Cont} \in M \Rightarrow \text{See}(M, \text{Mach}) := \text{Cont}
\]

So the refinement was an operation between Concrete and abstract model:

\[
\text{Refine} : (\text{AbsM} : \text{Model}) \rightarrow \text{ConcM} : \text{Model}
\]

\[
\forall \text{AbsM}, \text{ConcM} \cdot \text{AbsM} \neq \text{ConcM} \Rightarrow \text{Refine(AbsM)} := \text{ConcM}
\]

In the classical method of refinement users may go systematically from an abstract level to a concrete one using: the atomicity decomposition (or AD) or the event extension approach.

- The AD proposed by M. Butler [62] has an algorithm and translation rules (the dashed line for new events and the solid line for the abstract Event and its new introduced one) to generate automatically an Event-B model using its plug-in in the tool set RODIN. Therefore, the way to break the weakness of refining process by addressing the ADL(Figure 4) as a new operation make an order to the relationships between Event.

  The rules to know about which part is the parameter \text{Prm} introduced are intuitive and of course the Atomicity decomposition plug-in in RODIN tool set with the proof obligation will allow to do refinement but always this refinement is not that explicit because we handle it with the Refine operation when this concept is always bordered with context, machine.

- In the RODIN tool-set using the \text{Event Extension} is to get more information about what it was refined by searching about the changes added in each Event or simply what is the new introduced value in every extended event (see Figure 3).
4.2. The notion of refinement by operators

The main idea for this approach is to make the refinement development very explicit by adding new operator that touch all the Event-B model and basically the events, we introduce four operators: create, Enrich, Restrict and Rename.

\[ \text{Preserved} : (\text{Evt} : \text{Events} \times M : \text{Models}) \rightarrow P_{\text{Invs}} : \text{Invariants} \]
\[ \forall \text{Evt}, n, \text{Invs}, M : \text{Evt} \in M \land \text{Invs} \in M \land n \in 0 \cdots \text{card}(\text{Invs}) \land ((\text{Invs}(n) \subset \text{Guards}) \lor (\text{Invs}(n) \subset \text{Actions})) \Rightarrow (\text{Preserved(Evt)} := P_{\text{Invs}}) \land (P_{\text{Invs}}(n) := P_{\text{Invs}}(n) \cup \text{Invs}(n)) \]

Our \text{OP\_refine} operation take in consideration: Invariant, Events, Context and cares about the state of these three before and after refinement process, knowing that every event must Preserved at least by one invariant, the types of an event and its invariant(s) are Declared from the context (as shown in Figure 5.) so:

\[ \text{Declared} : (\text{Evt} : \text{Events} \times P_{\text{Invs}} : \text{Invariants} \times \text{Cont} : \text{Context}) \rightarrow D_{\text{Cont}} : \text{Context} \]
\[ \forall \text{Evt}, n, P_{\text{Invs}}, D_{\text{Cont}}, l, M : \text{Evt} \in M \land \text{Preserved} (\text{Evt}) := P_{\text{Invs}} \land l \in 0 \cdots \text{card}(P_{\text{Invs}}) \land n \in 0 \cdots \text{card}(\text{Cont}) \land ((P_{\text{Invs}}(l) \subset \text{Cont}) \lor (P_{\text{Invs}}(n) \subset \text{Cont})) \Rightarrow (\text{Declared(Evt}, P_{\text{Invs}}, \text{Cont}) := D_{\text{Cont}}) \land (D_{\text{Cont}}(n) := D_{\text{Cont}}(n) \cup \text{Cont}(n)) \]

Our approach presented during this paper is based on set of operators that represent refinement process which are as follow: The renaming operator, noted by "Rename", The enrichment operator, noted by "Enrich", The restriction operator, noted by "Restrict", The creation operator, noted by "create".
4.2.1. The Renaming operator

We can observe that this operator is similar to the dashedline one in the AD approach it’s about taking the abstract Event with no changes so we can formalize that into:

\[
\text{Rename} : (\text{AbsE} \times \text{AbsP}_{inv} \times \text{AbsD}_{cont}) \rightarrow (\text{ConcE} \times \text{ConcP}_{inv} \times \text{ConcD}_{cont}) \\
\forall \text{AbsE}, \text{ConcE}, \text{AbsC}, \text{AbsM}, \text{ConcM}, \text{AbsP}_{inv}, \text{ConcP}_{inv}, \text{AbsD}_{cont}, \\
\text{ConcD}_{inv} \cdot \text{AbsE} \in \text{ConcM} \land \text{ConcE} \in \text{ConcM} \land \text{Preserved} (\text{AbsE}, \text{AbsM}) = \text{P}_{inv} \\
\land \text{Decalred} (\text{AbsE}, \text{AbsP}_{inv}, \text{AbsC}) = \text{AbsD}_{cont} \\
\Rightarrow \text{Rename} (\text{AbsE}, \text{AbsP}_{inv}, \text{AbsD}_{cont}) := (\text{ConcE}, \text{ConcP}_{inv}, \text{ConcD}_{cont}) \\
\land ((\text{ConcE} := \text{AbsE}) \land (\text{AbsP}_{inv} := \text{ConcP}_{inv}) \land (\text{ConcD}_{cont} := \text{AbsD}_{cont}))
\]

When the model is a continuation of an existed problem in a model Mod0 so the refinement is linked to rename this problem as RMod0 we can say that it is a reuse for the problem of Mod0 and generally we need to change the variables name and ensure the refinement proof between abstract variables Var_Mod0 and concrete variables Var_RMod0. Renaming a model is to take the Event preserved by invariants and declared by context with no change from abstract level (AbsE, AbsP\text{inv}, AbsD\text{cont}) to the concret level (ConcE, ConcP\text{inv}, ConcD\text{cont}).

4.2.2. The enriching operator

The context of the extended model EC0 from the abstract context C0 contain replacement properties of sets or expressions even for the extended machine EM0 of the machine M0 which contain variables and events with new elements to add into the actions; generally Enrich operator represent a refinement amelioration with proof obligations. The machine EM0 of the model EMod0 must be a refinement of the machine M0 the dynamic part of the model Mod0 i.e. \( M_0 \supseteq EM_0 \). So to
Enrich a model (in the reason of getting the event ConcE) is to add in actions newAct for the abstract Event AbsE which are preserved by invariants ConcP_Inv declared in the context ConcD_Inv.

\[\text{Enrich} : (AbsE \times AbsP_{\text{Invs}} \times AbsD_{\text{Cont}} \times NewAct \times NewInv \times NewCont) \rightarrow \\
(\text{ConcE} \times ConcP_{\text{Invs}} \times ConcD_{\text{Cont}})
\]

\[\forall AbsE, ConcE, AbsC, AbsM, ConcM, AbsP_{\text{Invs}}, ConcP_{\text{Invs}}, AbsD_{\text{Cont}},
\]

\[\text{ConcD}_{\text{Invs}} \cdot AbsE \in \text{ConcM} \land ConcE \in \text{ConcM} \land \text{Preserved}(AbsE, AbsM) = P_{\text{Invs}}
\]

\[\land \text{Decalred}(AbsE, AbsP_{\text{Invs}}, AbsC) = AbsD_{\text{Cont}} \\
\Rightarrow \text{Enrich}(AbsE, AbsP_{\text{Invs}}, AbsD_{\text{Cont}}) := (\text{ConcE}, ConcP_{\text{Invs}}, ConcD_{\text{Cont}})
\]

\[\land ((\text{ConcE} := AbsE \cup \text{NewGrd}) \land ((\text{ConcP}_{\text{Invs}} := AbsP_{\text{Invs}} \cup \text{NewInv})
\]

\[\lor (\text{ConcD}_{\text{Cont}} := AbsD_{\text{Cont}} \cup \text{newCont})
\]

4.2.3. The restriction operation

\[\text{Restrict} : (AbsE \times AbsP_{\text{Invs}} \times AbsD_{\text{Cont}} \times NewGrd \times NewInv \times NewCont) \rightarrow \\
(\text{ConcE} \times ConcP_{\text{Invs}} \times ConcD_{\text{Cont}})
\]

\[\forall AbsE, ConcE, AbsC, AbsM, ConcM, AbsP_{\text{Invs}}, ConcP_{\text{Invs}}, AbsD_{\text{Cont}},
\]

\[\text{ConcD}_{\text{Invs}} \cdot AbsE \in \text{ConcM} \land ConcE \in \text{ConcM}
\]

\[\land \text{Preserved}(AbsE, AbsM) = P_{\text{Invs}} \land \text{Decalred}(AbsE, AbsP_{\text{Invs}}, AbsC) = AbsD_{\text{Cont}} \\
\Rightarrow \text{Restrict}(AbsE, AbsP_{\text{Invs}}, AbsD_{\text{Cont}}) := (\text{ConcE}, ConcP_{\text{Invs}}, ConcD_{\text{Cont}})
\]

\[\land ((\text{ConcE} := AbsE \cup \text{NewGrd}) \land ((\text{ConcP}_{\text{Invs}} := AbsP_{\text{Invs}} \cup \text{NewInv})
\]

\[\lor (\text{ConcD}_{\text{Cont}} := AbsD_{\text{Cont}} \cup \text{newCont})
\]

In some cases, the model CMod0 is the reuse of the model Mod0 with some particularity in the constraints (guards, invariants, or Axioms) represented by Grd_CMod0 or Inv_CMod0 or even Axm_CMod0. So we can say that the refinement had introduced new constraints as newGrd or NewCont to more clarify an invariant (ConcP_inv or created a new one new_Inv) preserved by the event ConcE.

4.2.4. The creation operation

During the refinement we found a new event created that it didn’t had any relationship with the others introduced before so it is just refinement from the Skip so the concret model will have a new Events with its new preserved invariants that had been declared in the context (or redeclared).

\[\text{Create} : (AbsE \times AbsP_{\text{Invs}} \times AbsD_{\text{Cont}} \times \text{NewEvents} \times \text{NewInv : } \times \text{NewCont}) \rightarrow \\
(\text{ConcE} \times ConcP_{\text{Invs}} \times ConcD_{\text{Cont}})
\]

\[\forall AbsE, ConcE, AbsC, AbsM, ConcM, AbsP_{\text{Invs}}, ConcP_{\text{Invs}}, AbsD_{\text{Cont}}, AbsC,
\]

\[\text{ConcD}_{\text{Invs}} \cdot AbsE \in \text{ConcM} \land ConcE \in \text{ConcM} \land \text{Preserved}((\text{NewEvents}, AbsM) = \emptyset
\]

\[\land ((\text{Decalred}(\text{NewEvents}, \text{Preserved}(\text{NewEvents}, AbsM), AbsC) = AbsD_{\text{Cont}})
\]

\[\lor (\text{Decalred}(\text{NewEvents}, \text{Preserved}(\text{NewEvents}, AbsM), AbsC) = \emptyset))
\]

\[\Rightarrow \text{Create}(AbsE, AbsP_{\text{Invs}}, AbsD_{\text{Cont}}) := (\text{ConcE}, ConcP_{\text{Invs}}, ConcD_{\text{Cont}})
\]

\[\land ((\text{ConcE} := \text{NewEvents}) \land ((\text{ConcP}_{\text{Invs}} := AbsP_{\text{Invs}} \cup \text{NewInv})
\]

\[\lor (\text{ConcD}_{\text{Cont}} := AbsD_{\text{Cont}} \cup \text{newCont})
\]
5. The process of refinement using operators

The process of Refinement with operators using theories (see [17]) (NoC, WNoC, graph, coloured graph, VHDL) to design a distributed system where every node is NoC-based element that can execute a recovering algorithm during any failure, these theories could be combined the following operators:

5.1. Refinement by creation

In the beginning of modeling using NoC theory NoC_th to present new Events Sending_data and receiving_data (model0) also we can see a creation of new set color (model4) in the moment that we introduce the new extension of graph theory ColoredV_th.

5.2. Refinement by renaming

In the second model there is no such a difference between sending data in model0 that describes the NoC properties using NoC_th theory and model1 (the model describes reconfiguration properties by theory WSNoC_th) so we rename the event we can say the same thing about the event receiving_data. Another example of renaming is when we still renaming the sets used in the level of model3 presenting coloured graph properties (ColoredV_th theory) to elaborate VHDL properties using VHDL_th theory in model4.

5.3. Refinement by enrichment

After introducing the graph theory into model1 (this model represent WNoC_th theory of recovering NoC-switches properties) we add actions to the events sending_data and sending_flagdata to obtain events on model2, taking another example presented in the last level of model4 (that used VHDL_th theory), all the events are enriched from their abstract version of model3 (using ColoredV_th theory).

5.4. Refinement by restriction

The example of using restrict operator presented after the introduction of graph theory in model2 to ensure that every new link still cover the closure property so we add more guaré for the event Relink of the model1 (that cover WNoC_th theory).

6. Results and discussions

The start point is always with the NoC theory where the node can send, receive and forward packets some POs in this theory are discharged manually (see the Fig.6.) but the most of rules related for the NoC system are convenient for the RODIN prover and by the result the NoC properties could be applied. According to the NoC theory a node can process inside a network of NoC-based
switches applying all the graph properties using the closure theory, in this level the number of manual discharged POs are increasing due to the complexity of network properties combined with the NoC properties (see Figure 7).

The NoC-based network may become into the failure state, so the coloration theory (extended from Closure theory) manage the failure zone (a number of faulty nodes) and colored it by the four available colors or adding to a pending list to be colored, here the number of manual Discharged POs still big according to the new strategy of handling a failure (as shown in the Fig.8.).

The WNoC theory express the emission of the reconfiguration data to reconfig_node and receiving bistream and forward packets till a faulty node returns to the initial state and the system with its all nodes are correct, it still has to color the failed nodes that are waiting to be recovered, the POs are all proved automatically which means that the coloring rules and NoC properties that are introduced previously made the system more suitable (as shown in the Figure 9.).

After combining all theories to express the properties of this system, VHDL variables are introduced to represented the VHDL behavior of this system and it’s clear that all POs are automatically discharged which means that the system is ready to the code generation step (Fig.10.).

In the term of implementation for the exchanged data (see Figure 11.) this kind of formal verification helped to create a new structure where the flag bit express the state of node (1: means faulty, 0: means healthy) and the phase of emission (1: flag-data sending and reconfiguration data, 0: correct emission).

Our refinement approach (see Table 3.) in Event-B is applied during the architectural design phase and ensures the translation of constructions of a language to another which does not manipulate the same concepts. To do this, each “elementary transformation” is represented by one or more rules in the form of polymorphic theories.
Figure 9. Statistic proofs of the fourth level of modelling using WNoC theory.

Figure 10. Statistic proofs of the fifth level of modelling using VHDL theory.

Figure 11. The new generated structure of NoC exchanged data after new modelling.
Table 3. The principal refinement properties for our approach in Event-B

<table>
<thead>
<tr>
<th>Formal method</th>
<th>Placement of refinement</th>
<th>Type of refinement</th>
<th>Refinement relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator-based Event-B</td>
<td>Architectural design</td>
<td>Transformation by generic operators using polymorphic theories</td>
<td>Weakening of preconditions and strengthening post-conditions (over determinism) with rules for handling events by the state of its invariants preserved and preserved their definitions in contexts (axioms, sets, etc.).</td>
</tr>
</tbody>
</table>

Table 4. The auxiliary properties of refinement for our approach in Event-B

<table>
<thead>
<tr>
<th>Refinement property</th>
<th>Operator-based refinement for Event-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR , VR Distinction</td>
<td>Yes</td>
</tr>
<tr>
<td>Horizontal Refinement (HR)</td>
<td>Yes</td>
</tr>
<tr>
<td>Vertical Refinement (VR)</td>
<td>Yes</td>
</tr>
<tr>
<td>Data refinement</td>
<td>Yes</td>
</tr>
<tr>
<td>Functional refinement</td>
<td>Yes</td>
</tr>
<tr>
<td>Behavioral refinement</td>
<td>Yes</td>
</tr>
<tr>
<td>Compositional refinement</td>
<td>Yes</td>
</tr>
<tr>
<td>Code generation</td>
<td>Yes (external code generator)</td>
</tr>
<tr>
<td>Multiples abstraction levels</td>
<td>Yes</td>
</tr>
<tr>
<td>Resetting</td>
<td>Yes</td>
</tr>
<tr>
<td>Preserving the inherent properties</td>
<td>Yes</td>
</tr>
<tr>
<td>Preservations of properties defined by the user</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-languages</td>
<td>Yes</td>
</tr>
</tbody>
</table>

We perform, moreover, distinguish between refinement supported by Event-B (see Table 4.) the architectural decomposition and our aim to make Event-B as presented in the next table.

7. Conclusions and Future Work

Our approach is founded after a useful recall of characteristics of various existing formalisms upon which we rely. While the previously presented state of the art on different refinement of methodologies in a fairly general framework, we focus on a more specific problem.

We propose to support us, upstream of a classic formal development of an architectural description in the formal language Event-B. Initially, an architectural description of the system comes from the requirement analysis phase in a form of theories. Rather that refine this architectural description to implementation in a purely architectural framework. Furthermore, we must also define how to switch from one language to the other, this is why we need to formally describe this series of refinements by a process presented by four operators: create, enrich, rename, restrict.

The software architecture description language that interests us is based on an Abstract Data type structured in the shape of polymorphic theories. This gives Event-B an important expressive power, which notably allows the expression of dynamic architectures.

Our study is illustrated by the example of the code generation and development of a network composed of NoC-based wireless Self-Organized nodes, this approach used many theories just like the VHDL theory, the great advantage of our approach is to have tools to help the formal development and VHDL
code generation just like in the work of RODIN group\cite{63,64} on other programming languages. The following describes the areas in which further research can be carried out as an extension of our work. Items are prioritized according to their immediate importance: robustness of the operator based refinement approach\cite{65} (Make the user able to refine using operators in Event-B in a significant number of systems), creating a library of theories\cite{66} (make the formalism established as Isabelle have a rich set of libraries from simple mastering of theory Complex Continuing mathematics). improved Support for Data-Types (make the plugin Theory supports data types in the logic of Event-B), The Event-B2VHDL\cite{67} (The phase of the automatic generation give more primary value to be integrated as a plug-in RODIN which is closer to the semantic system modeled the tool EHDL \cite{68}).

**Acknowledgments:** This research is a part of research projects supported by the thematic research agency (ATRST) of research direction (Algerian Ministry of High Education and Scientific Research.), so it is pretty convenient to take a chance to address our gratitude for their partnership and funding during the realization of this humble work.

### Appendix A  Classical refinement auxiliary properties of formal methods

Refinement auxiliary properties may be a horizontal refinement (decomposition) or vertical (with change in level of abstraction) and possibly a clear distinction between the two. It is also possible to have specific structures for data refinement, functional, behavioural, or compositional refinement. In addition, not all methods lead from an abstract specification to the code generation, as they could not afford the distinction between different levels of abstraction. Despite these action aspects of refining, some additional criteria concern the reusability of some parts of the process of refinement and preservation of properties (inherent or defined by the user). Finally, multiple languages or vocabularies can be used to identify different levels of abstraction or refinement steps. Therefore, these criteria can be used to organize the different approaches into categories according to their refinement relations.

### References


14. Hariche, A.; Belarbi, M.; Daoud, H.; Based B Extraction of QNoC architecture properties; international workshop on Mathematic and computing; December 17-18; 2012, Tiaret; MOMA Journal; volume 02; pp. 7–13.


40. Garlan, D.; Allen, R.; Ockerbloom, J.; Architectural Mismatch or, Why it’s hard to build systems out of existing parts; Proceedings of the 17th International Conference on Software Engineering (ICSE-17); Avril 1995.


58. Clements, P.C.; A Survey of Architecture Description Languages; Eighth International Workshop on Software Specification and Design; Germany; march 1996.
64. Edmunds, A.; Butler, M.; Maamria, I.; Silva, R.; Lovell, C.; Event-B Code Generation: Type Extension with Theories; Abstract State Machines, Alloy, B, VDM, and Z, Lecture Notes in Computer Science; 2012; volume 7316; pp. 365-368.

© 2017 by the authors. Licensee Preprints, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).