Formal Ontology Driven Model Refactoring

Neeraj Kumar Singh, Yamine Aït-Ameur, Dominique Mery

▶ **To cite this version:**

Neeraj Kumar Singh, Yamine Aït-Ameur, Dominique Mery. Formal Ontology Driven Model Refactoring. 2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), Dec 2018, Melbourne, Australia. pp.136-145, 10.1109/ICECCS2018.2018.00022 . hal-02353400

HAL Id: hal-02353400
https://hal.science/hal-02353400v1

Submitted on 7 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in: http://oatao.univ-toulouse.fr/
Eprints ID: 23576

To link to this article : DOI:10.1109/ICECCS2018.2018.00022

URL : https://doi.org/10.1109/ICECCS2018.2018.00022

To cite this version: Singh, Neeraj Kumar and Aït-Ameur, Yamine and Mery, Dominique *Formal Ontology Driven Model Refactoring*. (2018) In: 2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), 12 December 2018 - 14 December 2018 (Melbourne, Australia)

Any correspondence concerning this service should be sent to the repository administrator:
staff-oatao@listes-diff.inp-toulouse.fr

Formal Ontology Driven Model Refactoring

Neeraj Kumar Singh
INPT-ENSEEIHT / IRIT
University of Toulouse, France
Email: nsingh@enseeiht.fr

Yamine Aït-Ameur
INPT-ENSEEIHT / IRIT
University of Toulouse, France
Email: yamine@enseeiht.fr

Dominique Méry
LORIA, Telecom Nancy
Université de Lorraine, France
Email: dominique.mery@loria.fr

***Abstract*—Refactoring, successfully used in the field of programming, can be used in maintenance and restructuring of the large and complex models. In this paper, we present a novel approach for model refactoring and a set of modelling patterns that are applicable for refinement-based formal development. In order to carry out this study, we investigate the previously developed large and complex model and required ontology to develop a domain model and a refactored system model. Further, we use the Rodin tools to check the internal consistency with respect to the desired functional behaviour and the required safety properties. Our main contributions are: to develop a refactoring technique related to the *correct by construction* approach; to use the domain specific knowledge in a system model explicitly; to define a set of modelling patterns; and to define a restructuring mechanism in the formal development. Finally, this proposed approach is evaluated through a complex medical case study: *ECG clinical assessment protocol*.**

***Keywords*-Refactoring, refinement and proofs, ontologies, domain theories, Event-B.**

I. Introduction

In order to reduce the development cost and maintenance cost, the formal modelling of complex systems and adaptation of new changes according to the newly added or modified requirements is a demanding task. In particular, if a system is developed using a *correct by construction* approach then the original structure of the formal model does not accommodate the new changes and newly added requirements without restructuring and modularizing the formal model.

On the other hand, in general, the system development process does not handle the domain knowledge explicitly. Such knowledge is always encoded implicitly during the system development considering some assumptions. For example, in the flight management system, the flight speed can be represented in Kilometres per hour (kph) or Miles per hour (mph) when modelling the collected data from the speed sensors. The sensed speed must be displayed on the pilot screen or the passenger screen according to the selected unit. From this modelling, if any miscalculation results in the unit and thus in the expected value then it can lead to the grave consequences. This situation is due to: (1) the absence, in the designed models, of explicit resource to model units explicitly in the existing modelling languages; (2) the computation runs (e.g. addition of two floats) are formalized in the implicit semantics of the modelling language, without handling the explicit semantics related to units (addition of speed in miles and speed in kilometres). The explicit semantics can be used to associate the unit information with the given numerical values (i.e. speed) by defining it explicitly.

It is highly desirable to define the domain knowledge with a system in an explicit way to improve the quality of the development process and to accommodate the new changes in the system requirements by restructuring the formal model [1]. To consider the domain knowledge in the software engineering practices has been considered an important step in the area of system modelling and analysis. The triptych [2], [3], [4] approach covers three main phases of the software development process: *domain description*, *requirements prescription* and *software design*. $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$ expresses a formal notation, in which $\mathcal{D}$ represents the domain concepts in form of properties, axioms, relations, functions and theories; $\mathcal{S}$ represents a system model; and $\mathcal{R}$ represents the intended system requirements. This notation states that the given domain description ($\mathcal{D}$) and the system model ($\mathcal{S}$) are correct with respect to the given requirements ($\mathcal{R}$). In similar vein, Jackson's structure [2] $\mathcal{E}, \mathcal{S} \vdash \mathcal{R}$ describes the requirements appropriately. In this structure $\mathcal{E}$ is the given environment, $\mathcal{S}$ is the specification that is optative description of a condition over the shared phenomena at the interface between the machine and the environment; and $\mathcal{R}$ is the requirement. The proposed structure must respect the distinction between system and the physical environment, and the environment properties must be achieved by the modelled system [2].

In this paper, we investigate a novel approach for model refactoring that is applicable for refinement based formal development. In order to use the domain knowledge in an explicit manner in a large formal model, we focus on a popular approach refactoring [5]. The proposed approach uses ontologies to define the domain-specific concepts explicitly and redefines the required behaviour and properties in form of patterns that can be reused in the system modelling to describe the required functional behaviour. Note that the proposed refactoring approach restructures the formal model and introduces domain knowledge explicitly in a system model without changing the refinement strategy. In other words, handling the explicit semantics in formal development does not affect the original formal development, in fact it strengthens them by allowing such a facility. This approach has several benefits, such as modularity, integration of domain knowledge, reusability, maintainability, preserving the required safety properties by proving the refactored model, and the development of modelling patterns.

We use the Event-B language for modelling the domain model and system model. Our main contributions are: (1) to develop a refactoring technique related to the *correct by construction* approach; (2) to use the domain specific knowledge in a system model explicitly; (3) to define the modelling patterns; and (4) to define a restructuring mechanism in the formal development.

We demonstrate the usability of the proposed approach through revisiting the formal development of the ECG protocol [6]. In this development, we refactor the whole model by preserving the required safety properties and functional behaviour through integrating the domain knowledge, such as heart and ECG. Moreover, we also demonstrate the other benefits as enumerated above during the model development.

The structure of the article is as follows. In Section II, we review preliminary material: refactoring, ontology and the modelling framework. Section III presents a refactoring methodology for developing the domain model and refactored system model. Section IV illustrates an application of the refactoring methodology: the *ECG clinical assessment protocol*. Section V discusses the paper. Section VI presents the related work and in Section VII, we conclude the paper and discuss the future work.

II. Preliminaries

A. Refactoring

Refactoring is one of the popular approaches in the field of programming that allows us to restructure the source code without modifying the functional behaviour of a system. This technique helps to clean up the developed code systematically by replacing the complex instructions with simple instructions, minimising the risks of introducing bugs, introducing modularity, and improving the readability and maintainability of the code [5], [7], [8]. In our work, we plan to use the refactoring techniques to the developed complex formal model [9], in which the formal model is developed using a *correct by construction* approach. Our main motivation to use the refactoring approach is to introduce the domain knowledge explicitly in a system model, minimising the complexity of proof structures, improving the maintainability of the developed formal model and improving the readability of the developed model. Note that the model refactoring allows us to restructure the developed formal model without modifying the functional behaviour and the characterization of the system/environment state at the abstract level and the refined levels. It means that all the defined safety properties for the given model must be proved. In addition, this approach has some other benefits that allow extracting the modelling design patterns, proof patterns, expose of an existing bug, separation of the domain model and system model, simplifying the proof strategies and to increase the proof automation. Moroever, all these benefits are derived from the proposed refactoring approach that allows us to refactor a system model applying modelling patterns and to introduce domain concepts explicitly. A set of modelling design patterns and restructuring of models related to refactoring can help to reduce the proof efforts. This reduction results from the factorization, at the *domain model DM* or context of domain properties proved many times at the *system model SM* level. Developing a domain model separately helps to identify any possible bugs of the old model due to the underspecification hidden by the implicit semantics carried by the modelling language.

B. Ontology

Ontology - "science of being" - is originated in philosophy, which is defined as "*hierarchal structuring of knowledge about concepts by sub-classing them according to their properties and qualities*" [10]. Alternatively, It is also defined as "*a declarative model of a domain that defines and represents the concepts existing in that domain, their attributes and the relationships between them*" [10], [11].

Another definition relies on the notion of a dictionary. [12] considers a domain ontology as a *formal and consensual dictionary of categories and properties of entities of a domain and the relationships that hold among them*. Here, an entity represents any concept belonging to the considered domain. *dictionary* entails two major concepts. First, it makes explicit the existence, through a constructive definition or declaration, of entities in the domain under consideration and second any entity or relationship described in this ontology is directly referenceable independently of other entities or relationships. Reference is carried by a symbol defining an identifier. This identification symbol may be either a language-independent identifier, or a language-specific set of words. However, whatever this symbol is, and unlike in linguistic dictionary, it directly denotes a domain entity or relationship. Each *description* of each entity or relationship is formally *stated* using an *ontology modelling language* equipped with a formal semantics. It allows automatic reasoning and consistency checking.

In our work, we use the ontology to model the domain-specific knowledge explicitly. In fact, the construction of a domain model allows us to refactor the previously developed system model. The development of domain model has several benefits: (1) to share knowledge in the same domain; (2) to reuse the existing domain model for any other system model; (3) to provide an explicit list of domain assumptions; (4) to separate the domain knowledge from the operational knowledge; and (5) to perform domain-specific methodical analyses.

C. The Modelling Framework: Event-B

This section describes the essential components of the modelling framework. In particular, we will use the Event-B modelling language [13] for modelling a complex system in a progressive way. There are two main components of Event-B: *context* and *machine*. A *context* is a formal static structure that is composed of several other components, such as *carrier sets*, *constants*, *axioms* and *theorems*. A *machine* is a formal dynamic structure that is composed of *variables, invariants, theorems, variants* and *events* (see Table I). A machine and a context can be connected with *sees* relationship.

CONTEXT	MACHINE
ctxt_id_2	*machine_id_2*
EXTENDS	**REFINES**
ctxt_id_1	*machine_id_1*
SETS	**SEES**
s	*ctxt_id_2*
CONSTANTS	**VARIABLES**
c	v
AXIOMS	**INVARIANTS**
$A(s,c)$	$I(s,c,v)$
THEOREMS	**THEOREMS**
$T_c(s,c)$	$T_m(s,c,v)$
END	**VARIANT**
	$V(s,c,v)$
	EVENTS
	Event evt
	any x
	where $G(s,c,v,x)$
	then
	$v :\mid BA(s,c,v,x,v')$
	end
	END

TABLE I: Model structure

Theorems	$A(s,c) \Rightarrow T_c(s,c)$ $A(s,c) \wedge I(s,c,v)$ $\Rightarrow T_m(s,c,v)$
Invariant preservation	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow I(s,c,v')$
Event feasibility	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant progress	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow V(s,c,v') < V(s,c,v)$

TABLE II: Proof obligations

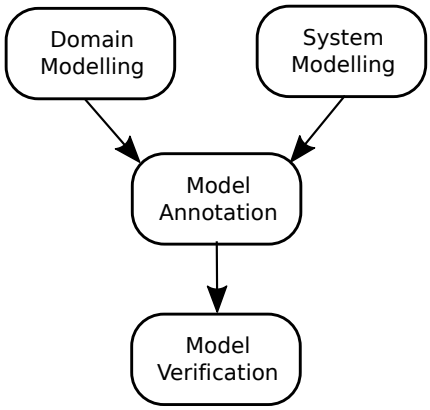

Fig. 1: Four step modelling methodology

An Event-B model is characterized by a list of *state variables* possibly modified by a list of *events*. Events play an important role for modelling the functional behaviour of a system. An event is state transition that contains two main components: *guard* and *action*. A *guard* is a predicate based on the state variables that defines a necessary condition for enabling the event. An *action* is also a predicate that allows modifying the state variables when the given guard becomes true. A set of invariants defines required safety properties that must be satisfied by all the defined state variables. There are several proof obligations, such as invariant preservation, non-deterministic action feasibility, guard strengthening in refinements, simulation, variant, well-definedness, that must be checked during the modelling and verification process (see Table II).

Event-B modelling language allows us modelling a complex system gradually using refinement. The refinement enables us to introduce more detailed behaviour and the required safety properties by transforming an abstract model to a concrete version. At each refinement step, the events can be refined by: (1) keeping the event as it is; (2) splitting an event into several events; or (3) refining by introducing another event to maintain state variables. Note that the refinement always preserves a relation between an abstract model and its corresponding concrete model. The newly generated proof obligations related to refinement ensures that the given abstract model is correctly refined by its concrete version. Note that the refined version of the model always reduces the degree of non-determinism by strengthening the guards and/or predicates. The modelling framework has a very good tool support (Rodin [14]) for project management, model development, conducting proofs, model checking and animation, and automatic code generation. There are numerous publications and books available for an introduction to Event-B and related refinement strategies [13].

D. OntoEventB Plug-in

In [15], the OntoEventB plug-in tool is developed to generate Event-B domain models from ontologies models, such as OWL [16] or PLIB [17]. This tool is integrated in the Rodin [14] platform that allows to generate the domain models applying *shallow* or *deep* approach on the ontology description file. Note that this tool is successfully applied in our work to generate the domain models from the given ontologies.

III. Methodology

A. Modelling Methodology

This section presents a modelling methodology (see Fig. 1), which contains the different modelling steps: *domain modelling*, *system modelling*, *model annotation* and *model verification* [1], [18]. These modelling steps are described as follows:

1) **Domain Modelling.** The required information related to a domain may be modelled as a domain ontology through defining concepts, entities, relationships, constraints and rules. In this work for modelling the domain model, we choose the Event-B modelling language to formalize the required domain concepts derived from the domain ontology, which can be described as theories in Event-B. Note that Event-B theory plugin[1] can be used for this purpose.
2) **System Modelling.** For developing a safe system considering all the required functionalities is a challenging problem. A system can be described using *axioms*, *constants*, *variables* and *events*. In this work for modelling the system model, we also choose the Event-B modelling language.
3) **Model Annotation.** Model annotation is used to link the domain model and the system model explicitly by describing the relationships between design model entities and ontology concepts. As a consequence, the annotated design model is enriched by the domain properties expressed in the ontology.
4) **Model Verification.** This last step is performed when the system model is annotated with the domain model. Two verification steps are envisioned. The first one is

[1]http://wiki.event-b.org/index.php/Theory_Plug-in#Standard_Library

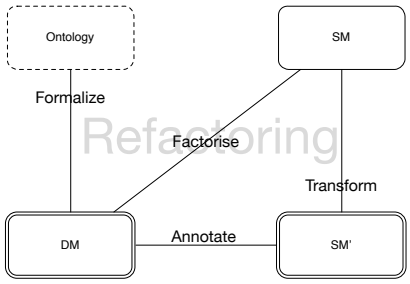

Fig. 2: Generic Refactoring Methodology

conducted on the designed system model before annotation (maybe no longer correct after annotation) to check the consistency and then the second one must be conducted on the designed system model after annotation to check the overall consistency considering the domain knowledge. Note that in the second step, the verification also allows us for checking new emerging properties due to the integration of domain model and system model using annotations.

B. Refactoring Methodology

A common way of restructuring, introducing modularity, minimising the complexity of proof structures, and improving the maintainability and readability of a formal specification is to use the refactoring techniques, which allow changing the structure of a model without changing the system functionalities and the behavioural objectives of the model [5], [7], [8]. In general, two kinds of refactoring techniques are identified: *structural refactoring* and *behavioural refactoring.* The structural refactoring changes the structure of a formal model and preserves its behaviour without changing the reachable states. Note that this structuring mechanism allows a developer to transfer the same safety properties to the new refactored model because this refactoring ensures the observability equivalence. The behavioural refactoring may change the behaviour of a formal model. In fact, applying this approach there are partly reachable states comparing to the formal model before refactoring [19]. Our work sets up the structural refactoring.

We propose to handle explicitly domain knowledge in a formal model using structural refactoring. Fig. 2 depicts a graphical layout of the input and output of the defined structural refactoring moving source models to target ones. The upper part of this figure denotes the source models (system models (SM) and ontology).

We consider that ontology[2] is available but it is not used by the system models SM since they do not explicitly refer to the domain model (ontology). The lower part of the Fig. 2 represents the target models composed of domain models (DM) obtained from the ontology, and the refactored system models (SM'). The horizontal lines define model dependencies (e.g. visibility, extension) between models while the vertical lines describe refactoring operations. For example, the target domain models (DM) are developed by formalising the ontologies. The target system models (SM') are refactored from the source system models SM.

A set of refactoring operations is identified. The approach we propose consists in analysing whether a refactoring operation can be applied to any complex formal model developed progressively using a *correct by construction* approach and the conceptual knowledge related to a domain is formalized implicitly in a system model. Each refactoring operation can be seen as a before-after predicate that preserves the properties of the source models while making explicit domain knowledge in the target models (refactored models).

To support system models SM refactoring, we have identified a set of structural development operations corresponding to specific model mappings. These mappings shall fulfill the characteristics attached to ontologies, in particular, the unique referencing mechanism. We have identified the following operations.

- **Formalize_DM.** The definition of DM consists in producing a domain model by selecting the relevant ontologies associated to the studied system. This DM may be formalized as a context or a theory depending on the used formal method. The consistency of DM shall be ensured (axioms providing definitions of domain concepts shall be inhabited).
- **Factorise_SM_2_DM.** The operation moves from the system model SM, the *definitions of concepts* (e.g. definitions related to variables, invariants or theorems) of SM, to the ontology or DM. If not available, these concepts are raised at ontological level else they are added as redundant concepts (derived concepts using ontology modelling operators).
- **Transform_SM_2_SM'** A target system model SM' is produced from a source system model SM by adding relationships to the DM model, i.e. adding direct references to the DM concepts, or adding mappings between SM and DM concepts. This operation may require rewriting both static (axioms, theorems, etc.) and behavioural concepts (guards, before-after predicates, substitutions, etc.). **Remark.** In this case, it should be noted that the new emerging invariants and theorems may be expressed in the SM' model. They are entailed by the domain knowledge explicitation.

The previous operations mention the notion of mapping between models. Ontology engineering provides with several kinds of mappings like equivalence, subsumption, algebraic mappings which can be formalized in Event-B.

Finally, once refactoring is performed, the SM models shall be submitted to a new verification process.

IV. Development in Medical Domain: Clinical Assessment of ECG

We adopt our generic refactoring methodology for developing the domain model and system model together using

[2]Several ontologies and domain models have proposed by several organizations, standards, companies, etc. The process of building these ontologies is out of scope of this paper.

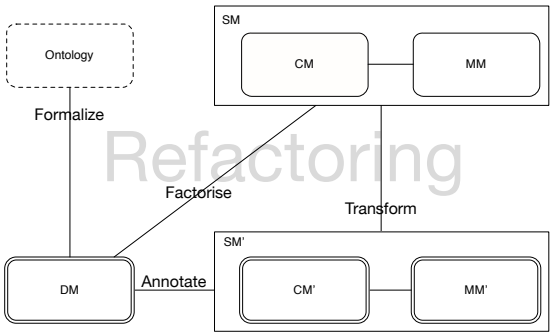

Fig. 3: Refactoring Methodology for Event-B

the Event-B modelling language. Fig. 3 depicts an extended graphical layout of the generic refactoring methodology to show the different components of the Event-B models. In the extended figure, the system model SM is composed of the context model CM and machine model MM. Similarly, in the refactored system model SM' is also composed of the context model CM' and machine model MM'.

The formal model of the ECG clinical protocol [9] is revisited to apply the proposed refactoring approach. Here, we recall the ECG protocol and then we develop the domain model, context model and system model progressively.

A. Electrocardiogram

An electrocardiogram (EKG or ECG)[3] [6] signal presents an electrical activity of the human heart in continuous form to show the depolarisation and repolarisation phenomena. A typical cycle of ECG (see Fig. 4) represents a sequence of waves and intervals, which is denoted as P-QRS-T-U. These waves and intervals are defined as: *P-wave* - a small deflection caused by the depolarisation of atria before contraction to show an electrical wave propagation from the SA node through the atria; *PR interval* - an interval between the beginning of the P-wave to the beginning of the Q-wave; *PR segment*- a flat segment between the end of the P-wave and the start of the QRS interval. *QRS interval* - an interval between the P-wave and T-wave with greater amplitude to show the depolarization of the ventricles; *ST interval* - an interval between the end of the S-wave and the beginning of the T-wave; *ST segment* - a flat segment starts at the end of the S-wave and finishes at the start of the T-wave; *T-wave* - a small deflection caused by the ventricular repolarisation, whereby the cardiac muscle is prepared for the next cycle of ECG; and *U-wave* - a small deflection immediately following the T-wave due to repolarization of the Purkinje fibers.

B. The Medical Domain modelling

A medical domain is characterised by the abundance knowledge of medical science collected from various sources. Ontology has played a significant role in representing medical

[3]The interested reader is referred to [6] for the detailed information on the ECG signal and the ECG clinical assessment protocol.

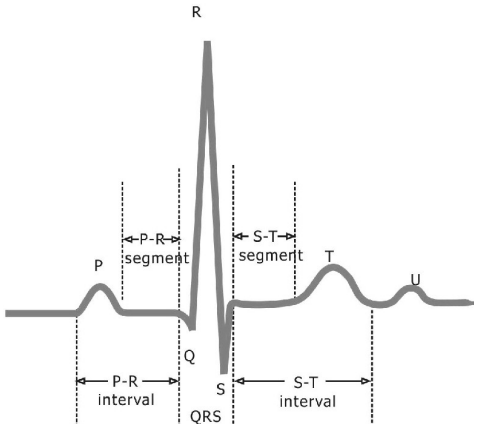

Fig. 4: ECG Deflections

knowledge systematically in an independent format to share and reuse across other biomedical domains. The medical ontology framework provides the common medical concepts, relationships, properties, and axioms related to the biomedical, disease, diagnosis, treatment, anatomy, pharmacology, clinical procedure and so on. There are several medical ontologies, such as GALAN, OpenCyc, WordNet, UMLS, SNOMED-CT and FMA developed by researchers, industries and medical centers.

According to our proposed refactoring methodology, we develop a domain model derived from the available ontologies, and the existing system model. To our knowledge, there are several databases and ontologies to represent the ECG. For describing the conceptual knowledge of biological process of the ECG, we use the OBO (Open Biomedical Ontologies) Process Ontology [20], classified as the fundamental relation, spatial relation, temporal relation and participation relation [20]. In the current work, the two main fundamental used relations are is_a and $part\_of$.

$$\begin{array}{l} A\ is\_a\ B = \forall x[\mathbf{inst}(x, A) \Rightarrow \mathbf{inst}(x, B)] \\ A\ part\_of\ B = \forall x[\mathbf{inst}(x, A) \Rightarrow \exists y(\mathbf{inst}(y, B)\ \&\ x\ \mathbf{part\_of\_inst}\ y)] \end{array}$$

The is_a relation states that every instance of class A is an instance of class B and the second relation states that *A part_of B* holds if and only if: for every individual x, if x instantiates A then there is some individual y such that y instantiates B and x is a part of y. In the previous definitions, **inst** is a relation between a class instance and a class which it instantiates and the **part_of_inst** is a relation between two class instances. Other relations are defined in ontology modelling languages. All of them are rigorously defined.

The whole concepts of the ontology modelling language need to be formalized in the used formal modelling language, Event-B in our case. The fundamental relations, ***is_a*** and ***part_of***, are defined in an Event-B context using axioms ($axm1$ - $axm5$). $axm2$ and $axm3$ define ***is_a*** relation and ***part_of*** relation, respectively. Other axioms ($axm1$, $axm4$ and $axm5$) are used to support the formal definition of the defined

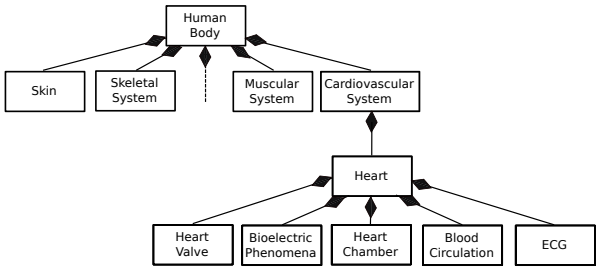

Fig. 5: Overview of a medical domain ontology

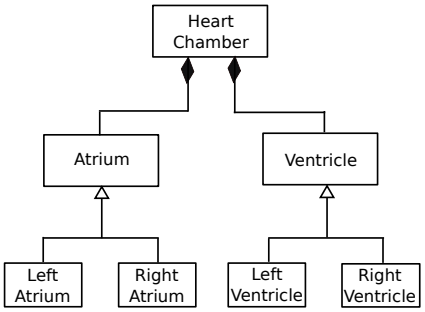

Fig. 6: Human heart chamber ontology

relations. This axiomatization defines the semantics of the ontology modelling operators. Other definitions are possible according to the chosen semantics.

$$
\begin{array}{l}
axm1 : HAS\_INSTANCES = CLASS \leftrightarrow INSTANCE \\
axm2 : IS\_A = \{IsA | IsA \in CLASS \leftrightarrow CLASS \wedge (\forall x, y \cdot (x \in CLASS \wedge \\
\quad y \in CLASS \wedge x \mapsto y \in IsA \Leftrightarrow \\
\quad union(\{r \cdot r \in HAS\_INSTANCES | ran(\{x\} \lhd r)\}) \subseteq \\
\quad union(\{r \cdot r \in HAS\_INSTANCES | ran(\{y\} \lhd r)\}))) \} \\
axm3 : PART\_OF = \{PartOf | PartOf \in CLASS \leftrightarrow CLASS \wedge \\
\quad (\forall x, y \cdot (x \in CLASS \wedge y \in CLASS \wedge \\
\quad x \mapsto y \in PartOf \Leftrightarrow \forall p \cdot p \in union(\{r \cdot r \in \\
\quad HAS\_INSTANCES | ran(\{x\} \lhd r)\}) \Rightarrow \\
\quad (\exists q \cdot q \in union(\{r \cdot r \in HAS\_INSTANCES | ran(\{y\} \lhd r)\}) \wedge \\
\quad p \mapsto q \in PartOf\_Inst)))) \} \\
axm4 : PartOf\_Inst \in INSTANCE \leftrightarrow INSTANCE \\
axm5 : (\forall p \cdot p \in INSTANCE \Rightarrow p \mapsto p \in PartOf\_Inst) \wedge \\
\quad (\forall p, q \cdot p \in INSTANCE \wedge q \in INSTANCE \wedge \\
\quad p \mapsto q \in PartOf\_Inst \wedge q \mapsto p \in PartOf\_Inst \Rightarrow \\
\quad p = q) \wedge (\forall p, q, r \cdot p \in INSTANCE \wedge q \in INSTANCE \wedge \\
\quad r \in INSTANCE \wedge p \mapsto q \in PartOf\_Inst \wedge \\
\quad q \mapsto r \in PartOf\_Inst \Rightarrow p \mapsto r \in PartOf\_Inst)
\end{array}
$$

These defined relations are used to model the domain knowledge for developing the ECG protocol using a correct by construction approach. The domain model of the ECG relies on existing work [6], [21], [22], [23]. It is based on several sub ontologies, organised in a subsumption hierarchy, such as human heart, blood circulation, bioelectric phenomena and ECG (see Fig. 5). The initial set of axioms are defined by applying the *Formalize_DM* refactoring operation (see Fig. 2 and Fig. 3).

Fig. 6 describes a very high level description that contains four chambers: left atrium, right atrium, left ventricle and right ventricle organised using the ***part_of*** and ***is_a*** relationships. The OBO relationships are defined in axioms ($axm1$-$axm4$) according to Fig. 6. The next axiom ($axm5$) defines a set of possible physical units, which can be associated with variables and constants ($axm6$ - $axm8$). The next remaining axioms ($axm9$ - $axm11$) are used to define the normal and abnormal heart rate. All these axioms are defined by applying the refactoring operations using *Formalize_DM* and *Factorize_SM_2_DM* (see Fig. 2 and Fig. 3).

$$
\begin{array}{l}
axm1 : partition(CLASS, \{Heart\}, \{Heart\_Chamber\}, \{Atrium\}, \\
\quad \{Ventricle\}, \{Left\_Atrium\}, \{Right\_Atrium\}, \{Left\_Ventricle\}, \\
\quad \{Right\_Ventricle\}) \\
axm2 : \{Atrium \mapsto Heart\_Chamber\} \in PART\_OF \wedge \\
\quad \{Ventricle \mapsto Heart\_Chamber\} \in PART\_OF \\
axm3 : \{Left\_Atrium \mapsto Atrium\} \in IS\_A \wedge \\
\quad \{Right\_Atrium \mapsto Atrium\} \in IS\_A \\
axm4 : \{Left\_Ventricle \mapsto Ventricle\} \in IS\_A \wedge \\
\quad \{Right\_Ventricle \mapsto Ventricle\} \in IS\_A \\
axm5 : partition(UNIT, bpm, mm, cm, mu\_m) \\
axm6 : F\_UNIT \in UNIT \rightarrow \mathbb{P}(\mathbb{Z}) \\
axm7 : HEART\_RATE \in \{Heart\} \leftrightarrow F\_UNIT \\
axm8 : HEART\_RATE = \{Heart \mapsto (bpm \mapsto 1 \,..\, 300)\} \\
axm9 : NORMAL\_HEART\_RATE \in \{Heart\} \leftrightarrow F\_UNIT \\
axm10 : NORMAL\_HEART\_RATE = \{Heart \mapsto (bpm \mapsto 60 \,..\, 100)\} \\
axm11 : ABNORMAL\_HEART\_RATE = \\
\quad HEART\_RATE \setminus NORMAL\_HEART\_RATE
\end{array}
$$

Fig. 7 presents a high level description of the ECG using the OBO relations for deflections known as waves and segments. The elementary concepts are represented using the ***is_a*** and ***part_of*** relationships (axioms $axm1$-$axm13$). The remaining axioms ($axm14$-$axm52$) characterise the ECG signal. They are derived from the existing ontologies and the previously developed ECG model [9] by applying the refactoring operations *Formalize_DM* and *Factorise_SM_2_DM* (see Fig. 2 and Fig. 3).

$$
\begin{array}{l}
axm1 : partition(CLASS, \{ElementaryForm\}, \{Waveform\}, \{Wave\}, \\
\quad \{Segment\}, \{Cycle\}, \{P\_Wave\}, \{QRS\_Wave\}, \{T\_Wave\}, \{U\_Wave\}, \\
\quad \{PQ\_Segment\}, \{ST\_Segment\}, \{Q\_Wave\}, \{R\_Wave\}, \{S\_Wave\}) \\
axm2 : \{Wave \mapsto ElementaryForm\} \in IS\_A \\
axm3 : \{Segment \mapsto ElementaryForm\} \in IS\_A \\
axm4 : \{Cycle \mapsto Waveform\} \in PART\_OF \\
axm5 : \{P\_Wave \mapsto Wave\} \in IS\_A \wedge \{P\_Wave \mapsto Cycle\} \in PART\_OF \\
axm6 : \{QRS\_Wave \mapsto Wave\} \in IS\_A \\
axm7 : \{T\_Wave \mapsto Wave\} \in IS\_A \wedge \{T\_Wave \mapsto Cycle\} \in PART\_OF \\
axm8 : \{U\_Wave \mapsto Wave\} \in IS\_A \wedge \{U\_Wave \mapsto Cycle\} \in PART\_OF \\
axm9 : \{PQ\_Segment \mapsto Segment\} \in IS\_A \wedge \\
\quad \{PQ\_Segment \mapsto Cycle\} \in PART\_OF \\
axm10 : \{ST\_Segment \mapsto Segment\} \in IS\_A \wedge \\
\quad \{ST\_Segment \mapsto Cycle\} \in PART\_OF \\
axm11 : \{Q\_Wave \mapsto QRS\_Wave\} \in PART\_OF \wedge \\
\quad \{Q\_Wave \mapsto Cycle\} \in PART\_OF \\
axm12 : \{R\_Wave \mapsto QRS\_Wave\} \in PART\_OF \wedge \\
\quad \{R\_Wave \mapsto Cycle\} \in PART\_OF \\
axm13 : \{S\_Wave \mapsto QRS\_Wave\} \in PART\_OF \wedge \\
\quad \{S\_Wave \mapsto Cycle\} \in PART\_OF \\
\\
axm14 : RR\_Int\_equidistant \in \{Cycle\} \times LEADS \rightarrow BOOL \\
axm15 : P\_Positive \in \{P\_Wave\} \times LEADS \rightarrow BOOL \\
\ldots \\
\ldots \\
\ldots \\
axm52 : QRS\_Axis_S tate \in \{QRS\_Wave\} \times LEADS \rightarrow QRS\_direcstions
\end{array}
$$

C. The Context Refactored Model (ECG Protocol)

In this section, we revisit the developed ECG protocol [9] to apply the proposed refactoring approach. Fig 8 describes the stepwise development of the domain model and system model covering the given requirements. This is a generic development where the domain model and system model evolve progressively.

$$
\begin{array}{l}
axm1 : partition(State, \{OK\}, \{KO\}) \\
axm2 : partition(SState, \{Yes\}, \{No\}) \\
axm3 : HState \in \{Heart\} \rightarrow State \\
axm4 : HSState \in \{Heart\} \rightarrow SState \\
\\
\mathbf{CS1} : \mathbf{ClinicalProp1} = (\lambda x \mapsto y \cdot x = Cycle \wedge y = P\_Wave \wedge \\
\quad ((\exists l \cdot l \in \{II, V1, V2\} \wedge PP\_Int\_equidistant(x \mapsto l) = TRUE \wedge \\
\quad RR\_Int\_equidistant(x \mapsto l) = TRUE \wedge RR\_Interval(x \mapsto l) = \\
\quad PP\_Interval(x \mapsto l)) \wedge P\_Positive(y \mapsto II) = TRUE) | TRUE) \\
\\
\mathbf{CS2} : \mathbf{ClinicalProp2} = (\lambda x \mapsto y \cdot x = Cycle \wedge y = P\_Wave \wedge \\
\quad ((\forall l \cdot l \in \{II, V1, V2\} \Rightarrow PP\_Int\_equidistant(x \mapsto l) = FALSE) \vee \\
\quad RR\_Int\_equidistant(x \mapsto l) = FALSE \vee RR\_Interval(x \mapsto l) \neq \\
\quad PP\_Interval(x \mapsto l)) \vee P\_Positive(y \mapsto II) = FALSE) | TRUE)
\end{array}
$$

Fig. 9 depicts a standard clinical procedure for analysing the ECG. The initial assessment step allows us to check the sinus rhythm and the heart rate (state of the heart), formally defined in the abstract model using domain knowledge and

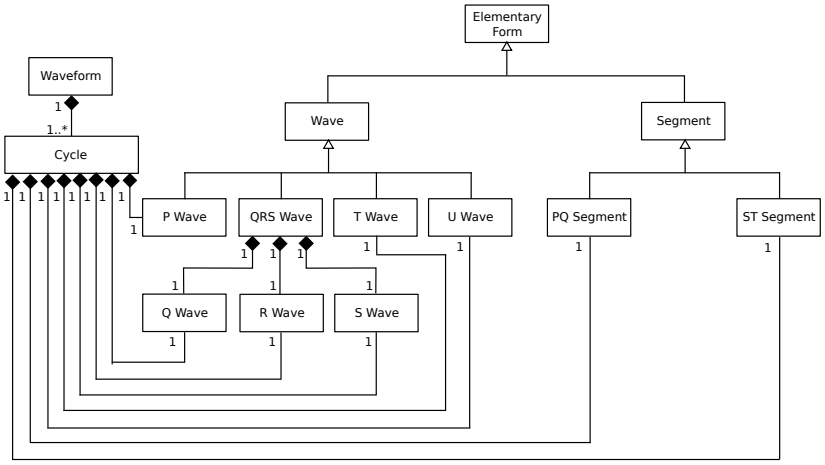

Fig. 7: ECG ontology

Domain	System	Requirements
D_1	S_1	R_{Set_1}
$\downarrow extends$	$\downarrow refines$	$\downarrow added$
D_2	S_2	R_{Set_2}
$\downarrow extends$	$\downarrow refines$	$\downarrow added$
$\vdots$	$\vdots$	$\vdots$
$\downarrow extends$	$\downarrow refines$	$\downarrow added$
D_n	S_n	R_{Set_n}

Fig. 8: Development of Event-B models

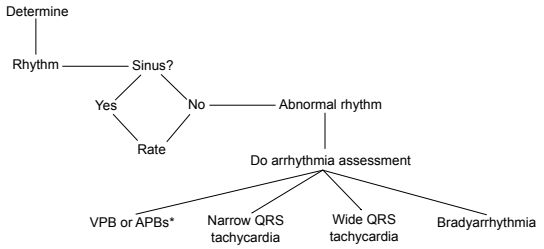

Fig. 9: Basic Diagram of Assessing Rhythm and Rate [6]

some new axioms ($axm1$-$axm4$). These axioms are defined to specify the heart state ($HState$) and the sinus state of the heart ($HSState$). For developing the ECG protocol, the abstract model defines two clinical properties $CS1$ and $CS2$. These clinical properties use the domain knowledge to specify the ECG assessment protocol. These clinical properties are introduced in the context model using the refactoring operations *Transform_SM_2_SM'* (see Fig. 2 and Fig. 3). *Note that these properties were introduced implicitly in the previous model of the ECG protocol [9].*

D. The Machine Refactored Model (ECG Protocol)

In the abstract model, we define three variables, *Sinus* - sinus state of the heart, *Heart_Rate* - heart rate limit, and *Heart_State* - normal or abnormal heart state, in $inv1$-$inv3$. A set of safety properties ($saf1$-$saf6$) is introduced: $saf1$ - if the positive visualization of P-wave in lead II is $FALSE$, then there is no sinus rhythm; $saf2$ - if the sinus is yes than the clinical property $ClinicalProp1$ must be $TRUE$; $saf3$ - if the clinical property $ClinicalProp2$ is $TRUE$ then there is no sinus rhythm; $saf4$ - if the heart rate belongs to the range of the normal heart rate and the sinus rhythm is yes then the heart state is *OK*; $saf5$ - if the heart rate belongs to the abnormal heart rate and the sinus rhythm is yes then the heart state is *KO*; and $saf6$ - if the heart rate belongs to the normal heart rate and there is no sinus rhythm then the heart state is *KO*. These set of properties are modified by using the refactoring operation *Transform_SM_2_SM'* (see Fig. 2 and Fig. 3). Note that the defined clinical properties (*ClinicalProp1* and *ClinicalProp2*) are used to state the safety properties.

$$
\begin{array}{l}
inv1 : Sinus \in HSState \\
inv2 : Heart\_Rate \in HEART\_RATE \\
inv3 : Heart\_State \in HState \\
saf1 : P\_Positive(P\_Wave \mapsto II) = FALSE \Rightarrow Sinus = Heart \mapsto No \\
saf2 : Sinus = Heart \mapsto Yes \Rightarrow \mathbf{ClinicalProp1}(Cycle \mapsto P\_Wave) = TRUE \\
saf3 : \mathbf{ClinicalProp2}(Cycle \mapsto P\_Wave) = TRUE \Rightarrow Sinus = Heart \mapsto No \\
saf4 : Heart\_Rate \in NORMAL\_HEART\_RATE \wedge Sinus = Heart \mapsto Yes \\
\qquad \Rightarrow Heart\_State = Heart \mapsto OK \\
saf5 : Heart\_Rate \in ABNORMAL\_HEART\_RATE \wedge Sinus = Heart \mapsto Yes \\
\qquad \Rightarrow Heart\_State = Heart \mapsto KO \\
saf6 : Heart\_Rate \in NORMAL\_HEART\_RATE \wedge Sinus = Heart \mapsto No \\
\qquad \Rightarrow Heart\_State = Heart \mapsto KO
\end{array}
$$

The abstract model of the ECG protocol contains three events, *Rhythm_test_TRUE*, *Rhythm_test_FALSE* and *Rhythm_test_TRUE_abRate* to assess the heart state by analysing the heart rhythm and the normal or abnormal heart rate. These events have used the domain model knowledge and the given clinical properties to specify the required behaviour. In these events, the guards are modified by applying the refactoring operation *Transform_SM_2_SM'* (see Fig. 2 and Fig. 3). The first guard of these events annotated with clinical

properties is refactored in the context model as CS1 or CS2. The second guard of these events annotates the heart rate that is explicitly defined in the domain model.

EVENT Rhythm_test_TRUE
ANY $rate$
WHEN
grd1 : $\mathbf{ClinicalProp1}(Cycle \mapsto P\_Wave) = TRUE$
grd2 : $rate \in NORMAL\_HEART\_RATE$
THEN
act1 : $Sinus := Heart \mapsto Yes$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := Heart \mapsto OK$
END

EVENT Rhythm_test_FALSE
ANY $rate$
WHEN
grd1 : $\mathbf{ClinicalProp2}(Cycle \mapsto P\_Wave) = TRUE$
grd2 : $rate \in HEART\_RATE$
THEN
act1 : $Sinus := Heart \mapsto No$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := Heart \mapsto KO$
END

EVENT Rhythm_test_TRUE_abRate
ANY $rate$
WHEN
grd1 : $\mathbf{ClinicalProp1}(Cycle \mapsto P\_Wave) = TRUE$
grd2 : $rate \in ABNORMAL\_HEART\_RATE$
THEN
act1 : $Sinus := Heart \mapsto Yes$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := Heart \mapsto KO$
END

The abstract model is further enriched by introducing the essential assessment steps progressively in a sequence of refinements, which corresponds to the standard analysis step of the ECG protocol [6].

Note that the new refactored models are different from the old models. For example, below we show an event equivalent to the refactored event *Rhythm_test_TRUE*. In the old event, the guards and other domain properties are defined implicitly, while in the new refactored model the required properties are defined only once in the domain model (such as *ClinicalProp1*) and it is used in the system model. Note that such refactoring approach has increased the proof automation.

EVENT Rhythm_test_TRUE
ANY $rate$
WHEN
grd1 : $(\exists l \cdot l \in \{II, V1, V2\} \land$
$PP\_Int\_equidistant(l) = TRUE \land$
$RR\_Int\_equidistant(l) = TRUE \land$
$RR\_Interval(l) = PP\_Interval(l)) \land$
$P\_Positive(II) = TRUE$
grd2 : $rate \in 60 \mathinner{..} 100$
THEN
act1 : $Sinus := Yes$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := OK$
END

Fig. 10: Taken from the old ECG model [9]

Due to space constraints, we omit the rest of the development. A detailed formal development of this ECG protocol is available on the website[4].

E. Model Verification

In this section, we describe the proof statistics of the developed model using refactoring approach. As we know that this development is based on the Event-B modelling language, which allows us to check the *consistency checking* and *refinement checking*. Table III presents the proof statistics of the progressive development of the old ECG model and the refactored ECG model. In this development applying the proposed refactoring approach, we achieve 543 (100%) proof obligations, in which 391 (73%) POs are proved automatically, and the remaining 152 (27%) are proved interactively using the Rodin provers, while the old development has more POs. Note that the generated POs of new refactored model also include other possible POs related to refactoring operations. The old model has more POs, including more manual interactions, due to the complex predicates and implicit domain knowledge. This refactoring approach has simplified the modelling constructs and development process that allows us to automate the several proof strategies of the refactored model. Moreover, the interactive proof obligations are also very simple that are proved with the help of SMT solver.

First, we mention that our approach has been deployed on a non trivial development issued from the medical domain. Note that the obtained new refactored model of ECG is simple compared to the old model of ECG. Some of the states and behavioural properties, previously defined implicitly in the old model of ECG, are defined explicitly in the new refactored model of ECG. Shared ontological definitions are referenced in the new obtained model.

Moreover, according to the Table III, the proof efforts have been decreased comparing with the old model [9]. In particular, the number of interactive proofs is significantly decreased. Indeed, the domain model properties are proved once for all in DM and are used as hypotheses to prove the properties of the system model SM.

The results shown in Table III indicate that the use of refactoring approach with explicit domain knowledge has improved significantly the process of formal development and has produced the new simplified proof strategies.

V. Assessment

The explicitation of domain knowledge in system modelling leads to the expression of properties absent in the old system model due to implicit or lack of domain knowledge. For example, the heart rate is represented in a pair of unit (bpm) and value that must comply whenever heart rate is modified. Note that in our old model, there was no unit (bpm) for representing the heart rate, and thus this property was absent.

The proposed refactoring approach restructures the formal model and introduces domain knowledge explicitly in the system model. The approach reduces the system complexity, proof efforts and improves the model consistency. For example, the clinical properties (CS1 and CS2) are defined once in the context model using the domain concepts borrowed from DM. These properties have been used later in the SM' to define safety properties (see $saf2$ and $saf3$) and guards ($grd1$ in all three events).

Due to model refactoring, the developed DMs can be reused for any other system model SM in relation with this domain. Moreover, due to separation of concerns, these refactored models are easily maintainable and can be used for further designs or analyses. For example, we have developed the DM

[4]http://singh.perso.enseeiht.fr/Conference/ICECCS2018/ECGModels.zip

Model	Old Model			Refactored Model		
	Total number of POs	Automatic Proof	Interactive Proof	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	41	33(80%)	8(20%)	43	22(52%)	21(48%)
First Refinement	61	54(88%)	7(12%)	49	36(74%)	13(26%)
Second Refinement	41	38(92%)	3(8%)	39	32(82%)	7(18%)
Third Refinement	51	36(70%)	15(30%)	47	39(83%)	8(17%)
Fourth Refinement	60	35(58%)	25(42%)	50	36(72%)	14(28%)
Fifth Refinement	43	22(51%)	21(49%)	36	29(81%)	7(19%)
Sixth Refinement	38	14(36%)	24(64%)	30	22(74%)	8(26%)
Seventh Refinement	124	29(23%)	95(77%)	114	74(65%)	40(35%)
Eighth Refinement	52	30(57%)	22(43%)	53	33(63%)	20(37%)
Ninth Refinement	21	9(42%)	12(52%)	15	12(80%)	3(20%)
Tenth Refinement	67	43(64%)	24(36%)	65	54(84%)	11(16%)
Total	599	343(58%)	256(42%)	543	391(73%)	152(27%)

TABLE III: Proof Statistics

of ECG used here for analysing the medical protocol. The same ECG DM can be used for developing any other medical system.

Note that the total number of refinements for both the old model and the newly refactored model *is identical*. The use of refactoring has great impact on reducing the proof efforts by restructuring and decomposing the complex model. The number of automated proofs has been increased. For example, the old model of ECG [9] has 58% automated POs while the new model of ECG has 73% of automated POs due to the availability of new hypotheses in the DM. Moreover, the interactive POs of the new model is simpler than the old ECG model.

VI. Related Work

The use of ontology in software engineering for designing a complex system has great interest by several researchers to consider the domain knowledge explicitly. Zayas et al. [24] proposed a methodology to interoperate existing heterogeneous design models to make them complete and precise with shared domain knowledge. In [1], [25], authors proposed a new approach for handling domain knowledge in design models. In this work, the domain models are developed using ontologies that can be used further during the system development applying annotation mechanism. Hacid et. al. [18] have used the similar approach to develop a domain model based on ontology for developing a system model using stepwise development. In [15], authors proposed a generic approach to integrate the domain description formalized by ontologies into an Event-B development process.

The initial idea of refactoring was proposed by Opdyke [7] and Griswold [26] in their dissertations. Fowler et al. [5] described the code refactoring approaches, methods and associated tools. Refactoring is defined as, *"Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure"*. Mens and Tourwé [27] compared and discussed different criteria related to the refactoring activities, specific techniques and formalism that can be used for reducing software complexity using restructuring. Bois et al. [8] proposed quality matrix to describe the impact of refactoring. They developed a set of guidelines used to improve the coupling and cohesion characteristics. Several tools, such as Eclipse[5], IntelliJ IDEA[6], have been developed for source code refactoring.

Refactoring approaches are not limited to programming languages. They have been adopted by the formal specification modelling languages: [28] for Event-B, [29] for ASM, [30] for Alloy, and [31] for Object-Z. Whiteside et al. [32] proposed a proof script refactoring approach for constructing, restructuring, and maintaining the development of formal proofs to support complex proofs. Kobayashi et al. [28] proposed the refactoring approach to restructure the refinements in Event-B. The main contribution is refinement decomposition based on a slicing strategy of a large model. The objective of our proposed refactoring approach is different from others. The main objective is to steer refactoring by modelling explicitly domain knowledge in a system model, minimising the complexity of proof structures, improving the maintainability of the developed formal model, exposing of any existing bug and improving the readability of the developed model.

VII. Conclusion

This paper has presented a refactoring approach that allows us to refactor a complex formal model, where the formal model is developed using a *correct by construction* approach and the domain concepts have been modelled implicitly. We have proposed a set of operations that allows us to refactor a system model considering domain specific knowledge in form of ontology to produce the domain model and system model through preserving the correctness of functional behaviour of the system. We have highlighted these refactoring operations for automation. In the current work, we have applied all these operations manually. The semantical description of the operations is beyond the scope of this paper. Due to the modelling complexity and variety of refinement laws, we do not claim completeness of the refactoring operations at this stage. Our results showed that the proposed refactoring

[5]http://www.eclipse.org
[6]https://www.jetbrains.com

operations can largely be automated to produce a simplified formal model with an explicit domain model.

In order to apply the refactoring approach, we selected the Event-B modelling language, which allows incremental refinement based on a *correct-by-construction* approach, for generating formal models. Further, the Rodin tools have been used to verify formally the produced refactored model. To assess the effectiveness of our proposed refactoring approach, we have revisited the formal development of the ECG protocol. We have developed the domain model and refactored system model in Event-B by using ontology and revisiting the developed formal model of ECG. In this development, we have refactored the whole model by preserving the required safety properties and functional behaviour through integrating the domain knowledge, such as the ECG. In order to guarantee the 'correctness' of the system behaviour, we have used a list of safety properties in the refactored model. Each refactored model was proven to guarantee the preservation of those safety properties.

Our future goal is to provide a semantical description and formalisation of the proposed refactoring operations ($Transform\_X\_2\_X$ and $Factorise\_X\_2\_X$). In the current work the total number of refinements is identical for both the old model and the newly refactored model. Thus, our new challenge in future will be to modify the refinement strategy for restructuring the formal development and optimising the refinement levels. In addition, studying the reusability of the proofs performed on the source models for the refactored target models is not addressed in this paper. We also plan to investigate this point in our future work. Moreover, we want to develop a tool based on the proposed refactoring operations to automate the process of refactoring the complex formal models.

ACKNOWLEDGMENT

This study was undertaken as part of the IMPEX (IMplicit and EXPlicit semantics Integration in proof based developments of discrete systems) research programme (ANR-13-INSE-0001).

REFERENCES

[1] Y. Ait-Ameur and D. Méry, "Making explicit domain knowledge in formal system development," *Sci. Comput. Program.*, vol. 121, no. C, pp. 100–127, Jun. 2016.

[2] M. Jackson and P. Zave, "Domain descriptions," in *Proceedings of IEEE International Symposium on Requirements Engineering, RE 1993,*, pp. 56–64.

[3] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 1, pp. 1–30, 1997.

[4] D. Bjørner, "Manifest domains: analysis and description," *Formal Asp. Comput.*, vol. 29, no. 2, pp. 175–225, 2017.

[5] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[6] M. G. Khan, *Rapid ECG Interpretation*. Humana Press, 2008.

[7] W. F. Opdyke, "Refactoring object-oriented frameworks," Ph.D. dissertation, Champaign, IL, USA, 1992, uMI Order No. GAX93-05645.

[8] B. Du Bois, S. Demeyer, and J. Verelst, "Refactoring " improving coupling and cohesion of existing code," in *Proceedings of the 11th Working Conference on Reverse Engineering*, ser. WCRE '04. IEEE Computer Society, 2004, pp. 144–151.

[9] D. Méry and N. K. Singh, "Medical protocol diagnosis using formal methods," in *Foundations of Health Informatics Engineering and Systems - First International Symposium, FHIES 2011, Johannesburg, South Africa, August 29-30, 2011. Revised Selected Papers*, 2011, pp. 1–20.

[10] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *Int. J. Hum.-Comput. Stud.*, vol. 43, no. 5-6, pp. 907–928, Dec. 1995.

[11] N. Guarino, *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, Trento, Italy*, 1st ed. Amsterdam, The Netherlands: IOS Press, 1998.

[12] S. Jean, G. Pierra, and Y. Aït-Ameur, "Domain Ontologies: A Database-Oriented Analysis," in *Web Information Systems and Technologies, International Conferences WEBIST*, ser. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2007, pp. 238–254.

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

[14] Project RODIN, "Rigorous open development environment for complex systems," http://rodin-b-sharp.sourceforge.net/, 2004.

[15] L. Mohand-Oussaid and I. Ait-Sadoune, *Formal modelling of domain constraints in Event-B*. Springer International Publishing, 2017.

[16] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein *et al.*, "Owl web ontology language reference," *W3C recommendation*, vol. 10, 2004.

[17] G. Pierra and H. Wiedmer, "Industrial automation systems and integrationparts librarypart 42: methodology for structuring part families," Technical Report ISO DIS 13584-42, International Organization for Standardization, 30 May 1996. ISO/TC 184/SC4/WG2, Tech. Rep., 1996.

[18] K. Hacid and Y. Ait-Ameur, *Annotation of Engineering Models by References to Domain Ontologies*. Springer International Publishing, 2016, pp. 234–244.

[19] S. Mitsch, J.-D. Quesel, and A. Platzer, *Refactoring, Refinement, and Reasoning*. Cham: Springer International Publishing, 2014, pp. 481–496.

[20] T. Bittner and M. Donnelly, "Logical properties of foundational relations in bio-ontologies," *Artif. Intell. Med.*, vol. 39, no. 3, pp. 197–216, Mar. 2007.

[21] "https://bioportal.bioontology.org/ontologies/ECG."

[22] B. Gonçalves, G. Guizzardi, and J. G. Pereira Filho, "Using an ecg reference ontology for semantic interoperability of ecg data," *J. of Biomedical Informatics*, vol. 44, no. 1, pp. 126–136, Feb. 2011.

[23] "http://aber-owl.net/ontology/ECG."

[24] D. S. Zayas, A. Monceaux, and Y. A. Ameur, "Knowledge models to reduce the gap between heterogeneous models: Application to aircraft systems engineering," in *15th IEEE ICECCS*, 2010, pp. 355–360.

[25] Y. Ait-Ameur, J. P. Gibson, and D. Méry, *On Implicit and Explicit Semantics: Integration Issues in Proof-Based Development of Systems*. Springer Berlin Heidelberg, 2014, pp. 604–618.

[26] W. G. Griswold, "Program restructuring as an aid to software maintenance," Ph.D. dissertation, Seattle, WA, USA, 1992, uMI Order No. GAX92-03258.

[27] T. Mens and T. Tourwe, "A survey of software refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, Feb 2004.

[28] T. Kobayashi, F. Ishikawa, and S. Honiden, *Refactoring Refinement Structure of Event-B Machines*. Cham: Springer International Publishing, 2016, pp. 444–459.

[29] H. Yaghoubi Shahir, R. Farahbod, and U. Glässer, *Refactoring Abstract State Machine Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 345–348.

[30] R. Gheyi and P. Borba, "Refactoring alloy specifications," *Electr. Notes Theor. Comput. Sci.*, vol. 95, pp. 227–243, 2004. [Online]. Available: https://doi.org/10.1016/j.entcs.2004.04.014

[31] T. McComb, *Refactoring Object-Z Specifications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 69–83.

[32] I. Whiteside, D. Aspinall, L. Dixon, and G. Grov, *Towards Formal Proof Script Refactoring*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 260–275.