On Implicit and Explicit Semantics: Integration Issues in Proof-Based Development of Systems*

Version to Read

Yamine Ait-Ameur[1], J. Paul Gibson[2], and Dominique Méry[3]

[1] IRIT - ENSEEIHT. Institut de Recherche en Informatique de Toulouse - École Nationale Supérieure d' Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications (ENSEEIHT)
yamine@enseeiht.fr
[2] Département Logiciels-Réseaux, IT-SudParis, Évry, France
paul.gibson@it-sudparis.eu
[3] Université de Lorraine, LORIA CNRS UMR 7503, Vandœuvre-lès-Nancy, France
mery@loria.fr

Abstract. All software systems execute within an environment or context. Reasoning about the correct behavior of such systems is a ternary relation linking the requirements, system and context models. Formal methods are concerned with providing tool (automated) support for the synthesis and analysis of such models. These methods have quite successfully focused on binary relationships, for example: validation of a formal model against an informal one, verification of one formal model against another formal model, generation of code from a design, and generation of tests from requirements. The contexts of the systems in these cases are treated as second-class citizens: in general, the modelling is implicit and usually distributed between the requirements model and the system model. This paper is concerned with the explicit modelling of contexts as first-class citizens and illustrates concepts related to implicit and explicit semantics on an example using the Event B language.

Keywords: Verification, modelling, Contexts, Domains.

1 Introduction: Implicit versus Explicit — The Need for Formality

In general usage, "explicit" means clearly expressed or readily observable whilst "implicit" means implied or expressed indirectly. However, there is some inconsistency regarding the precise meaning of these adjectives. For example, in logic and belief models [1] "a sentence is explicitly believed when it is actively held to be true by an agent and implicitly believed when it follows from what is believed."

* This work was supported by grant ANR-13-INSE-0001 (The IMPEX Project http://impex.loria.fr) from the Agence Nationale de la Recherche (ANR).

However, in the semantic web [2] "Semantics can be implicit, existing only in the minds of the humans [...]. They can also be explicit and informal, or they can be formal." The requirements engineering community use the terms to distinguish between declarative (descriptive) and operational (prescriptive) requirements [3] where they acknowledge the need for "a formal method for generating explicit, declarative, type-level requirements from operational, instance-level scenarios in which such requirements are implicit". We propose a formal treatment of the adjectives implicit and explicit when engineering *software and/or systems*.

Nowadays, several research approaches aim at formalizing mathematical theories for the formal development of systems. Usually, these theories are defined within contexts, that are imported and and/or instantiated. They usually represent the implicit semantics of the systems and are expressed by types, logics, algebras, etc. based approaches. To our knowledge, no work adequately addresses the formal description of domains expressing the semantics of the universe in which the developed systems run and their integration in the formal development process. This domain information is usually defined in an "ontology" [4].

Several relevant properties are checked by the formal methods. These properties are defined on the implicit semantics associated to the formal technique being used. When considering these properties in their context with the associated explicit semantics, these properties may be no longer respected. Without a more formal system engineering development approach, based on separation of implicit and explicit, the composition of software and/or system components in common contexts risks compromising correct operation of the resulting system. This is a significant problem when software and/or systems are constructed from heterogeneous components [5] that must be reliable in unreliable contexts [6].

To clarify, this paper is concerned with the separation of concerns when reasoning about properties of models. Although the concerns need to be cleanly separated, the models need to be tightly integrated: achieving both is a significant challenge.

2 Integrating Implicit and Explicit: Formal Methods and Ontologies

Allowing formal methods users and developers to integrate — in a flexible and modular manner — both the implicit semantics, offered by the formal method semantics, and the explicit semantics, provided by external formal knowledge models like ontologies, is a major challenge. Indeed, the formal models should be defined in the formal modelling language being used, and explicit reference and/or annotation mechanisms must be provided for associating explicit semantics to the formal modelling concepts. Once this integration is realized, the formalisation and verification of several properties related to the heterogeneous models' integration becomes possible. The most important properties that need to be addressed relate to interoperability, adaptability, dissimilarity, reconfigurability and identification of degraded modes. Refinement/instantiation and composition/decomposition could play a major role for specifying and

verifying these properties. Currently, no formal method or formal technique provides explicit means for handling such an integration.

In the context of formal methods, it is well known that several formal methods for system design and verification have been proposed. These techniques are well established on a solid formal basis and their domain of efficiency, their strengths and weaknesses are well acknowledged by the formal methods community. Although, some ad-hoc formalisation of domain knowledge [7] within formal methods is possible, none of these techniques offers a built-in mechanism for handling explicit semantics.

Regarding ontologies and domain modelling, most of the work has been achieved in the large semantic web research community. There, the problem consists of annotating web pages and documents with semantic information belonging to ontologies. Thus, ontologies have mainly been used for assigning meanings and semantics to terms occurring in documents. Once, these meanings are assigned, formal reasoning can be performed due to the ontologies being based on descriptive logic. In general, however, the documents to be annotated do not conform to any model and the domain associated to the documents is not fixed. Therefore, ontologies behave like a model associated to the resources that are annotated.

We propose an integration of both worlds. On the one hand, formal methods facilitate prescriptive modelling whereas, on the other hand, ontologies provide mechanisms for explicit descriptive semantics. We conclude by noting that, in most cases, the formal models are usually defined in a fixed and limited application domain well understood by the developers.

3 A Simple Example

The illustration of the addressed problem and the underlying ideas are given in this section through a simple case study. As a first step, we demonstrate a typical development involving solely a formal model; and in a second step we show how formalized explicit knowledge contributes to identifying relevant problems related to heterogeneity.

Let us consider a simple system issued from avionic system design. We identify two sub-systems: the first one is part of the flight management system acting in the closed world (heart of the avionic systems), it produces flight information like altitude and speed; and the second is the display part of a passenger information system (open world). It displays, to the passengers, information issued from the closed world, here altitude and speed. The information is transmitted from the closed world to the open world within a communication bus. Communications are unidirectional from the closed world to the open world only.

The development of this system considers a formally expressed specification which is refined twice. Figure 1 shows the structure of the development for this case study. The next two subsections show the two proposed formal developments expressed within the Event B formal method.

We note that this example is intended only as a proof-of-concept. Its goal is not to demonstrate the power of our approach, it shows only that there is

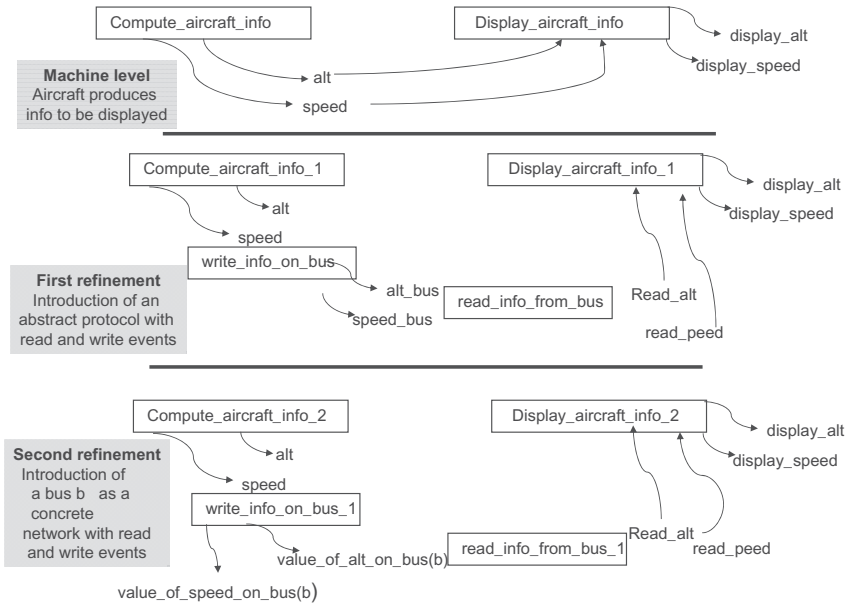

Fig. 1. A global view of the formal development

utility in separating the implicit and explicit semantics and that there is at least one such way of doing this separation in Event-B. This demonstrates that a fully formal and automated approach is feasible. Further work — on a range of case studies — will examine and compare different mechanisms for implementing the approach, with particular emphasis on scaleability and universality: can we model much larger, heterogeneous, domains of knowledge?

3.1 Formal Model with Implicit Semantics

In the implicit semantics, the models are constructed within the modelling capabilities offered by the modelling language.

The Machine Level: Specification. The first formal specification of the problem expresses that the system should communicate a computed value to be displayed. Variables, described in the invariant clauses, are *speed* (recording the effective speed), *alt* (recording the effective altitude), *display_speed* (recording the displayed speed), *display_alt* (recording the displayed altitude) and *consumed* (recording the control and the synchronisation between the produced and the consumed values. Two events are defined, one producing the information to be displayed and a second displaying this information.

EVENT Compute_aircraft_info
WHEN
$grd1 : consumed = 1$
THEN
$act1 : alt :\in \mathbb{N}_1$
$act2 : speed :\in \mathbb{N}_1$
$act3 : consumed := 0$
END

$inv1 : speed \in \mathbb{N}_1$
$inv2 : alt \in \mathbb{N}_1$
$inv3 : consumed \in \{0, 1\}$
$inv4 : display\_speed \in \mathbb{N}_1$
$inv5 : display\_alt \in \mathbb{N}_1$

EVENT Display_aircraft_info
WHEN
$grd1 : consumed = 0$
THEN
$act1 : display\_alt := alt$
$act2 : display\_speed := speed$
END

Event Compute_aircraft_info models the update step for the altitude and the speed; the *consumed* variable is set to 0 and the event Display_aircraft_info is triggered when the variable $Display\_aircraft\_info$ is updated by the sent value.

The invariant types and constrains the variables within numerical bounds; it does not take into account the fact that the produced values and consumed values belong to different domains. The *ontological context* is to provide information for relating these two domains. The main idea is to *annotate* the model by expressing the knowledge domain using the context models in figure 2.

First Refinement: Introducing an Abstract Communication Protocol. The new model First_Refinement extends the state by new variables recording the traffic of messages through a bus. It specifies that we have to manage the transmission of messages with the addition of new control variables (*written*, *read*, *displayable*). Two new events model the reading and the writing to and from the bus. The two abstract events are refined by strengthening guards with respect to the new control variables (*read*, *written*, *displayable*). The new model introduces an abstract protocol for the bus.

EVENT compute_aircraft_info_1
REFINES compute_aircraft_info
WHEN
$grd1 : consumed = 1$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : alt :\in \mathbb{N}_1$
$act2 : speed :\in \mathbb{N}_1$
$act3 : consumed := 0$
END

EVENT Display_aircraft_info_1
REFINES Display_aircraft_info
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 0$
$grd4 : displayable = 1$
THEN
$act1 : display\_alt := read\_alt$
$act2 : display\_speed := read\_speed$
$act3 : displayable := 0$
END

The two next events model the abstract protocol for exchanging the data. They describe the fact that a value is written to and then read from an abstract bus.

EVENT write_info_on_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : alt\_bus := alt$
$act2 : speed\_bus := speed$
$act3 : written := 0$
END

EVENT read_info_from_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : read\_alt := alt\_bus$
$act2 : read\_speed := speed\_bus$
$act3 : read := 0$
END

Second Refinement: Concretizing the Bus for Communication. The current system is still abstract and we have to add details concerning the bus. Finally, the four events of the model First_Refinement are refined to concretize actions over the bus b. The two first events are directly related to the computation and display components.

EVENT compute_aircraft_info_2
REFINES compute_aircraft_info_1
WHEN
$grd1 : consumed = 1$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : alt :\in \mathbb{N}_1$
$act2 : speed :\in \mathbb{N}_1$
$act3 : consumed := 0$
END

EVENT Display_aircraft_info_2
REFINES Display_aircraft_info_1
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 0$
$grd4 : displayable = 1$
THEN
$act1 : display\_alt := read\_alt$
$act2 : display\_speed := read\_speed$
$act3 : displayable := 0$
END

The next two events are the operations over the bus. They precise how the *speed* and *altitude* information are written to and read from the bus b

EVENT read_info_from_bus_2
REFINES read_info_from_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : read\_alt := value\_of\_alt\_on\_bus(b)$
$act2 : read\_speed := value\_of\_speed\_on\_bus(b)$
$act3 : read := 0$
END

EVENT write_info_on_bus_2
REFINES write_info_on_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : value\_of\_alt\_on\_bus(b) := alt$
$act2 : value\_of\_speed\_on\_bus(b) := speed$
$act5 : written := 0$
END

3.2 Formal Model with Explicit Semantics

The previous development follows the formal modelling approach provided by the Event B method, focusing on the binary relation referred to in the introduction of this paper.

The second development, presented below, introduces the explicit knowledge carried out by ontologies, it is used for coding the ternary relationship referred to in the introduction. In the case of Event B, it is formalized within *contexts*. The ternary relationship is obtained by *annotation* i.e. linking the model elements, variables in our case, to the explicit knowledge. In the following we illustrate the process of handling explicit domain knowledge in Event B models, using the same aircraft case study as before.

Contexts for Defining Explicit Domain Knowledge. The first step consists of introducing the explicit domain knowledge through a formal model for ontologies. It will be used to annotate the concepts seen in the previous models.

In the simple case we are addressing, this knowledge is defined by contexts (see figure 2). In this case, we are concerned by the description of the units that may be associated to the *altitude* and to the *speed*.

Meters, inches, kilometers per hour, and miles per hour are introduced to define distance speed measures. Conversion functions, that define equivalences

CONTEXT $domain\_knowledge\_for\_units$
CONSTANTS
$inches, meters, mph, kph, inch2meters, mphour2kphour$
AXIOMS
$axm1 : inches \subseteq \mathbb{N}_1$
$axm2 : meters \subseteq \mathbb{N}_1$
$axm3 : mph \subseteq \mathbb{N}_1$
$axm4 : kph \subseteq \mathbb{N}_1$
$axm5 : inches \neq \varnothing$
$axm6 : meters \neq \varnothing$
$axm7 : mph \neq \varnothing$
$axm8 : kph \neq \varnothing$
$axm9 : inch2meters \in inches \rightarrow meters$
$axm10 : mphour2kphour \in mph \rightarrow kph$
END

Fig. 2. The ontological context

in terms of ontology definitions, are described by the functions $inch2meters$ and $mphour2kphour$. We do not detail the defintions of these two functions but they can be made more precise by an implementation step at a later phase in the process.

Annotation: Associating Explicit Knowledge to Model Variables. Once the explicit knowledge has been formalized, it becomes possible to annotate the concepts available in the obtained formal models. In our case, the variables are annotated by explicitly referring to the ontology defined in the context of figure 2. Measurement units are introduced in an explicit way.
The variables are then defined as follows. When the annotations have been specified, the verification of the previous development defined in section 3.1 is no longer correct. Some proof obligations cannot be satisfied due to incoherent assignments.

$inv1 : speed \in mph$
$inv2 : alt \in inches$
$inv3 : consumed \in \{0, 1\}$
$inv4 : display\_speed \in kph$
$inv5 : display\_alt \in meters$

The new invariant defines the ontological constraints that should be satisfied by the events. For example, one of the generated proof obligations for checking the preservation of $inv5 : display\_alt \in meters$ by the event Display_aircraft_info fails to prove that $alt \in meters$. Thus, we should modify the event Display_aircraft_info by removing the previous *act1* and *act2* and by adding the ontological information provided by the two functions $inch2meters$ and $mph2kphour$ in the rewritten actions *nact1* and *nact2* . The example is simple and gives an obvious way to solve the unproved proof obligation: without refinement it may be much more difficult to discover why similar proof obligations are not discharged.

Consequently, the following events — Display_air craft_info and Compute_aircraft _info — require further description. In Particular, Display_aircraft_info has been modified in order to handle converted values issued from Compute_aircraft_info.

EVENT Compute_aircraft_info
WHEN
$grd1 : consumed = 1$
THEN
$act1 alt :\in inches$
$act2 speed :\in mph$
$act3 consumed := 0$
END

EVENT Display_aircraft_info
WHEN
$grd1 : consumed = 0$
THEN
$nact1 display\_alt := inch2meters(alt)$
$nact2 : display\_speed := mphour2kphour(speed)$
END

First Refinement: Introducing an Abstract Communication Protocol. As a next step, we can add new features in the current model Main_exchange by refining it into First_Refinement_Dom.

The new model First_Refinement_Dom performs the same extension of the state as in the previous case using implicit knowledge. This is quite natural since none of these state variables (i.e. *written*, *read*, *displayable*) are annotated. Two new events model the reading to and the writing from the bus. The invariant is extended by sub-invariants $inv6 \ldots inv15$. Notice the introduction of new kinds of invariants, labeled $inv13$ and $inv14$, borrowed from the context where the explicit knowledge is described. They define ontological invariants.

$inv1$ *to* $inv5$ *of last model*
$inv6 : speed\_bus \in kph$
$inv7 : read\_alt \in meters$
$inv8 : read\_speed \in kph$
$inv9 : alt\_bus \in meters$
$inv10 : written \in \{0, 1\}$
$inv11 : read \in \{0, 1\}$
$inv12 : displayable \in \{0, 1\}$
$inv13 : (written = 0) \Rightarrow (alt\_bus = inch2meters(alt) \wedge speed\_bus = mphour2kphour(speed))$
$inv14 : (read = 0) \Rightarrow (read\_alt = alt\_bus \wedge read\_speed = speed\_bus)$
$inv15 : (displayable = 0) \Rightarrow (display\_alt = read\_alt \wedge display\_speed = read\_speed)$

The two abstract events are refined by strengthening guards with respect to the new control variables (*read*, *written*, *displayable*). The new model introduces an abstract protocol for the bus.

EVENT compute_aircraft_info_1
REFINES compute_aircraft_info
WHEN
$grd1 : consumed = 1$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : alt :\in inches$
$act2 : speed :\in mph$
$act3 : consumed := 0$
END

EVENT Display_aircraft_info_1
REFINES Display_aircraft_info
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 0$
$grd4 : displayable = 1$
THEN
$act1 : display\_alt := read\_alt$
$act2 : display\_speed := read\_speed$
$act3 : displayable := 0$
END

The two next events model the abstract protocol for exchanging the data. The abstract protocol manages the relationship between the measurement units. The ontological annotation appears in the invariant $inv13$: the protocol ensures the *correct* communication.

EVENT write_info_on_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : alt\_bus := inch2meters(alt)$
$act2 : speed\_bus := mphour2kphour(speed)$
$act3 : written := 0$
END

EVENT read_info_from_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : read\_alt := alt\_bus$
$act2 : read\_speed := speed\_bus$
$act3 : read := 0$
END

Context Extension: Need of Explicit Knowledge for the Bus. The current system is still abstract and we have to add details concerning the bus. Following good engineering practice, the communication bus should be described independently of any usage in a given model. Here again, an ontology of communication medias is needed. It is defined in a context that extends the one defined for measure units. The bus has specific properties that are expressed in a new context domain_knowledge_for_protocols(in figure 3.2).

CONTEXT $domain\_knowledge\_for\_protocols$
EXTENDS $domain\_knowledge\_for\_units$
SETS
bus, bus_type
CONSTANTS
$unidirectional, bidirectional, type\_of\_bus$
AXIOMS
$axm1 : bus\_type = \{unidirectional, bidirectional\}$
$axm2 : type\_of\_bus \in bus \rightarrow bus\_type$
$axm3 : bus \neq \varnothing$
$axm4 : \exists bb \cdot (bb \in bus \wedge type\_of\_bus(bb) = unidirectional)$
END

Fig.3.Context for the bus

Notice that the definition of explicit knowledge is modular. It uses contexts that import only those ontologies that are needed for a given development. Moreover, it is flexible since contexts can be changed, if the domain knowledge or the nature of the manipulated concepts evolves.
The whole formal development of the system does not need to be rewritten.

Second Refinement: Concretizing the Bus for Communication. The new invariant extends the previous one, whilst integrating the state of the bus. It also asserts that the bus is unidirectional. The invariant $Ninv3 : b \in bus$ is an ontological invariant and the context enriches the description of the domain.

$Ninv1 : value\_of\_speed\_on\_bus \in bus \rightarrow kph$
$Ninv2 : value\_of\_alt\_on\_bus \in bus \rightarrow meters$
$Ninv3 : b \in bus$
$Ninv4 : (written = 0) \Rightarrow (value\_of\_speed\_on\_bus(b) = mphour2kphour(speed))$
$Ninv5 : (written = 0) \Rightarrow (value\_of\_alt\_on\_bus(b) = inch2meters(alt))$
$Ninv51 : type\_of\_bus(b) = unidirectional$
$Ninv6 : (read = 0) \Rightarrow (read\_speed = value\_of\_speed\_on\_bus(b))$
$Ninv7 : (read = 0) \Rightarrow (read\_alt = value\_of\_alt\_on\_bus(b))$
$Ninv8 : alt\_bus = value\_of\_alt\_on\_bus(b)$
$Ninv9 : speed\_bus = value\_of\_speed\_on\_bus(b)$

Finally, the four events of the model First_Refinement_Dom are refined to concretize the actions over the bus b. The two first events are directly related to the computation and display components.

EVENT compute_aircraft_info_2
REFINES compute_aircraft_info_1
WHEN
$grd1 : consumed = 1$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : alt :\in inches$
$act2 : speed :\in mph$
$act3 : consumed := 0$
END

EVENT Display_aircraft_info_2
REFINES Display_aircraft_info_1
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 0$
$grd4 : displayable = 1$
THEN
$act1 : display\_alt := read\_alt$
$act2 : display\_speed := read\_speed$
$act3 : displayable := 0$
END

The two next events — read_info_from_bus_2 and write_info_on_bus_2 — model operations over the bus. They both deal with ontological annotations, where the more detailed characteristics of the bus are necessary for guaranteeing the safety of the global system.

EVENT read_info_from_bus_2
REFINES read_info_from_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 0$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : read\_alt := value\_of\_alt\_on\_bus(b)$
$act2 : \begin{pmatrix} read\_speed := \\ value\_of\_speed\_on\_bus(b) \end{pmatrix}$
$act3 : read := 0$
END

EVENT write_info_on_bus_2
REFINES write_info_on_bus
WHEN
$grd1 : consumed = 0$
$grd2 : written = 1$
$grd3 : read = 1$
$grd4 : displayable = 1$
THEN
$act1 : \begin{pmatrix} value\_of\_alt\_on\_bus(b) := \\ inch2meters(alt) \end{pmatrix}$
$act2 : \begin{pmatrix} value\_of\_speed\_on\_bus(b) := \\ mphour2kphour(speed) \end{pmatrix}$
$act5 : written := 0$
END

The summary of proof obligations tells us that the proof is not complex. In fact, the example is simple and does not require further interaction as the ontological annotations help to automatically derive the proofs.

4 Discussion

4.1 Proof-Based Development Methods for Safe and Secure Models and Systems: The Importance of Refinement

Deductive verification for program correctness has outstanding challenges. Formal methods toolsets assist the developer who is trying to check a set of proof obligations using a proof assistant. In contrast to these semi-automatic proof techniques, model checking [8] appears to be a better solution when the developer does not want to interact with the proof tool and, although model checking is addressing specific systems with a reasonable size or is applied on abstractions of systems to facilitate the proof, there are limits to the use of model checking-based techniques. Finally, another solution is to play with abstractions and to apply the abstract interpretation [9] engine by defining appropriate abstractions and domains of abstractions for analysing a program. Deductive verification techniques, model checking and abstract interpretation analyse programs or systems which are already built and we call this the *a posteriori* approach where the process of analysis tries to extract semantic information from the text of the program or the system.

The correct-by-construction approach [10] advocates the development of a program using a process which is proof-guided or proof-checked and which leads to a correct program. This is an *a-priori* verification approach. These proof-based development methods [11] integrate formal proof techniques in the development of software and/or systems. The main idea is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete ones. The relationship between two successive models in this sequence is that of refinement [11]

The essence of the refinement relationship is that it preserves already proven system properties including safety properties and termination. At the most abstract level it is obligatory to describe the static properties of a model's data by means of an invariant predicate. This gives rise to proof obligations relating to the consistency of the model. These are required to ensure that data properties which are claimed to be invariant are preserved by the events or operations of the model.

The Event B Method [12] is a refinement-based, correctness-by-construction approach for the development of event-based systems (or, more generally, event-based models). Several case studies [13] show that the Event B method provides a flexible framework for developing complex systems in an incremental and proof-based style. Since refinement necessitates checking proof obligations, an idea is to introduce concepts of reusability and instantiation of Event B models [14,15]: making it possible to re-apply already developed and proven models.

Refinement is a critical step in formal design: as we move from the abstract to the concrete we transform our requirements into an operational solution. Without refinement, the correctness of non-trivial design steps is usually a computational intensive (often intractable) problem. Refinement allows us to split the design phase into a sequence of refinement steps, each of which is proven correct through the discharging of proof obligations. When the sequence is well-engineered, this can often be done in an automated fashion. The role of the software engineer is to "find" such a sequence. The purpose of this research is to aid the engineer in this task.

4.2 Explicit Semantics of Modelling Domains and Domain Ontologies

According to Gruber [4], an ontology is a specification of a conceptualisation. An ontology can be considered as the modelling of domain knowledge. Nowadays, ontologies are used in many diverse research fields and several proposals for ontology models and languages and corresponding operational systems have been developed in the last decade. The main characteristics of an ontology are: being formal and consensual and offering referencing capabilities.

As both an ontology and a conceptual model define a conceptualization of a part of the world, we must clarify their similarities and differences. Conceptual models respect the formal criterion. Indeed, a conceptual model is based on a rigorously formalized logical theory and reasoning is provided by view mechanisms. However, a conceptual model is application requirement driven: it **prescribes**

and **imposes** which information will be represented in a particular application (logical model). Thus, conceptual models do not fulfil the consensual criterion. Moreover, an identifier of a conceptual model defined concept is a name that can be referenced only inside the context of an Information System. Thus, conceptual models also do not fulfil the capability to be referenced criterion [16].

Several ontology models — like OWL [17] and KAON [18] for description logic, and PLIB [19] and MADS [20] for database design — are based on constructors provided by conceptual models based on either database or knowledge base models. These models add other constructors that facilitate satisfaction of the consensual criterion (context definition, multi-instantiation) and the capability to be referenced criterion.

Within our approach, it is clear that re-usable domain knowledge can, and should, be integrated into the modelling of a system's environment. A problem with current modelling approaches is that this knowledge is often distributed between the inside and the outside of a system in an ad-hoc fashion. By formalising the notion of ontology we can encourage (perhaps oblige) system engineers to be more methodological in how they structure and re-use their ontologies. We are currently investigating whether this can be done through a better integration of existing ontology models into our Event-B framework (through annotations) or whether we need to build a re-usable ontological framework in Event-B, motivated by the aspects of existing ontology models that we have found useful in our case studies.

4.3 Properties and Methodology

The separation of implicit from explicit gives rise to many difficulties with respect to the methodological aspects of developing software and/or systems, but it opens up many opportunities with respect to the types of properties that can be handled more elegantly. Building on the notion of functional correctness — where a software and/or a system must be verified to meet its functional requirements when executing in a well-behaved environment — we must consider the issue of system reliability being compromised. In such circumstances we would like the behaviour to degrade in a controllable, continuous, manner rather than having a non-controllable abrupt crash. One of the advantages of the proposed approach is that we can automatically distinguish between problems due to an environment which is not behaving as expected (where the system makes an assumption about its environment which is false some time during execution) and an internal fault (where the environment makes some assumption about the system which is false some time during execution). We can also automatically execute some self-healing mechanism that is guaranteed — through formal verification — to return the system and its environment to a safe, stable state.

The key to our methodology is the integration of the implicit and explicit modelling, which is shown in figure 3. The architecture should be generally applicable to the development of software and/or systems in a wide range of problem domains. This needs to be validated through application of the tools and techniques in a range of case studies that go beyond the case study presented in this paper.

The development approach may also be considered generic with respect to its application using a variety of formal techniques rather than specific to a single (set of) method(s).

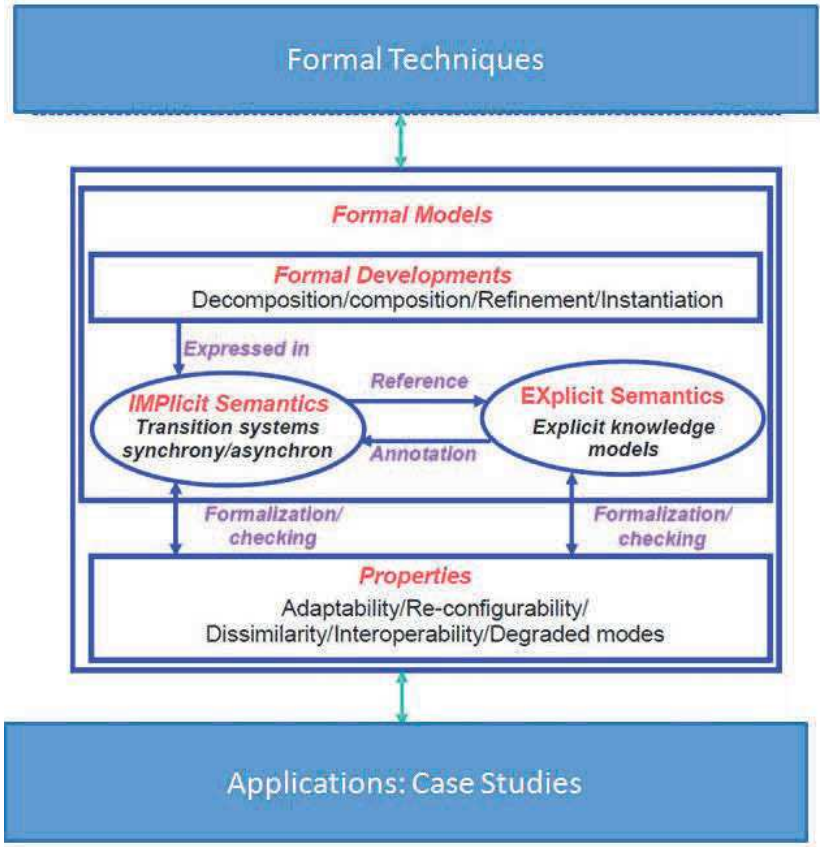

Fig. 3. A research architecture

As a minimum, the formal models must offer mechanisms for (de)composing systems, as well as for refinement and instantiation. The separation of implicit and explicit semantics is critical to independent development and verification. Furthermore, as illustrated in the case study of this paper, their integration must be directly supported by the development architecture: the implicit models will reference the explicit semantics which will provide annotations for the operational models. The best means of supporting this integration require further work: especially when we consider that combining top-down and bottom-up approaches in a formal development process is already very challenging. Initial work has shown us that the combination of different software development techniques (in particular, refinement with composition) is a major challenge. This is normally done in an ad-hoc manner, where the many different composition mechanisms lead to sate explosion problems when analysing behaviour. We propose working in an algebraic manner, which will constrain the ways in which composition can be performed: making our methodology simpler and more amenable to automated verification. A simplification of the semantics of composition risks

reducing the expressiveness of our language, but we argue that finding the balance is a key part of making our approach both sound, in theory, and applicable, in practice. This balance is necessarily different when considering implicit and explicit aspects of modelling. Without a separation of these concerns, we risk a compromise which helps in the development of neither. Through separation, we can better balance the modelling of each.

5 Conclusions

We have argued that many problems in the development of correct software and/or systems could be better addressed through the separation of implicit and explicit semantics. The key idea is to re-formalize correctness as a ternary (rather than binary) relation.

We have proposed that traditional formal methods need to be better integrated with ontology models, in order to support a clearer separation of concerns.

Through a simple example, we have illustrated how ontological semantics can be specified using Event B contexts and that this information can be integrated with the behavioral requirements — in an incremental fashion — through refinement. The simple example addresses the simple problem of information interchange. (A good example of the consequences of not modelling this formally can be found in the report on the Mars Climate Orbiter [21] where confusing imperial and metric measurements caused a critical failure.)

A main contribution of the paper is to place the implicit-explicit structure within the context of the relevant state-of-the-art. We emphasise the importance of building on well-established formal methods, treating domain/context ontology models as first-class citizens during development, and the need for a pragmatic approach that integrates into existing methods in a unified and coherent fashion.

Finally, we give an overview of where we think further research needs to be done, formulating the goals as the need for an architecture of inter-related research tasks.

References

1. Levesque, H.J.: A logic of implicit and explicit belief. In: Brachman, R.J. (ed.) AAAI, pp. 198–202. AAAI Press (1984)
2. Uschold, M.: Where are the semantics in the semantic web? AI Mag. 24, 25–36 (2003)
3. van Lamsweerde, A., Willemet, L.: Inferring declarative requirements specifications from operational scenarios. IEEE Trans. Softw. Eng. 24, 1089–1114 (1998)
4. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5(2), 199–220 (1993)
5. Ait-Ameur, Y., Méry, D.: Handling heterogeneity in formal developments of hardware and software systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 327–328. Springer, Heidelberg (2012)

6. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: Proceedings of the First Workshop on Self-healing Systems, WOSS 2002, pp. 27–32. ACM, New York (2002)
7. Bjorner, D.: Software Engineering 1 Abstraction and Modelling; Software Engineering 2 Specification of Systems and Languages, Software Engineering 3 Domains, Requirements, and Software Design. Texts in Theoretical Computer Science. An EATCS Series. Springer (2006)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
9. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)
10. Leavens, G.T., Abrial, J.R., Batory, D., Butler, M., Coglio, A., Fisler, K., Hehner, E., Jones, C., Miller, D., Peyton-Jones, S., Sitaraman, M., Smith, D.R., Stump, A.: Roadmap for enhanced languages and methods to aid verification. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE 2006, pp. 221–236. ACM, New York (2006)
11. Back, R.J.R.: On correct refinement of programs. Journal of Computer and Systems Sciences 23(1), 49–68 (1981)
12. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
13. Abrial, J.R., Cansell, D., Méry, D.: A mechanically proved and incremental development of ieee 1394 tree identify protocol. Formal Asp. Comput. 14(3), 215–227 (2003)
14. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to event-b. Fundam. Inf. 77(1-2), 1–28 (2007)
15. Cansell, D., Gibson, J.P., Méry, D.: Refinement: A constructive approach to formal software design for a secure e-voting interface. Electr. Notes Theor. Comput. Sci. 183, 39–55 (2007)
16. Jean, S., Pierra, G., Aït-Ameur, Y.: Domain ontologies: A database-oriented analysis. In: Cordeiro, J.A.M., Pedrosa, V., Encarnação, B., Filipe, J. (eds.) WEBIST (1), pp. 341–351. INSTICC Press (2006)
17. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L., et al.: Owl web ontology language reference. W3C recommendation 10, 2006-01 (2004)
18. Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., et al.: Kaon—towards a large scale semantic web. E-Commerce and Web Technologies, 231–248 (2002)
19. Pierra, G.: Context-explication in conceptual ontologies: the plib approach. In: Proceedings of the 10th ISPE International Conference on Concurrent Engineering (CE 2003). Enhanced Interoperable Systems, vol. 26, p. 2003 (2003)
20. Parent, C., Spaccapietra, S., Zimányi, E.: Spatio-temporal conceptual models: data structures + space + time. In: Proceedings of the 7th ACM International Symposium on Advances in Geographic Information Systems, GIS 1999, pp. 26–33. ACM, New York (1999)
21. Stephenson, A., Mulville, D., Bauer, F., Dukeman, G., Norvig, P., LaPiana, L., Rutledge, P., Folta, D., Sackheim, R.: Mars climate orbiter mishap investigation board phase I report. Technical report, NASA, Washington, DC (1999)