xOWL an Executable Modeling Language for Domain Experts

Laurent Wouters, Marie-Pierre Gervais

2011 15th IEEE International Enterprise Distributed Object Computing Conference

xOWL
an Executable Modeling Language for Domain Experts

Laurent Wouters
EADS Innovation Works
European Aeronautic Defence and Space Company
Suresnes, France
laurent.wouters@eads.net

Marie-Pierre Gervais
Université Paris Ouest
Laboratoire d'Informatique de Paris 6 (LIP6)
Paris, France
marie-pierre.gervais@lip6.fr

***Abstract*—Nowadays, modeling complex domains such those involving the description of human behaviors is still a challenge. An answer is to apply the Domain Specific Languages principle, which advocates that Domain Experts should model themselves their knowledge in order to avoid misunderstanding or loss of information during the knowledge elicitation phase. But Domain Experts must then be provided a modeling language enabling them to describe such complex domains. Moreover, in order to help them build models, immediate feedbacks would have to be available so that they can revise their modeling choices in earlier steps. Model execution is a way to address this issue.**

We provide xOWL, a language that can be used as a backend for multiple domain-specific syntaxes enabling Domain Experts to model themselves the structural as well as behavioral knowledge of their domain. xOWL comes with an interpreter integrated in an environment offering models executability in such way that Domain Experts can work in an iterative and incremental way using a trial and error approach. The implemented prototype is currently in use at EADS.

***Keywords*-Model-Driven Development; Semantic Web; Executable Models**

I. Introduction

These last years, Domain Specific Languages (DSL) are gaining in importance in the Model-Driven Development (MDD) community. They enable Domain Experts to model themselves their knowledge, rather than transmitting this task to a software engineer. This lowers the risk of misunderstanding, or loss of information during the elicitation phase. But this requires making available a visual concrete syntax intuitive enough and close to the experts' domain.

EADS is strongly interested in the modeling of human behavior and user-system interaction. Experts in these areas need a modeling language not only to represent their knowledge but also to perform analyses on the resulting models.

Knowledge representation is a well-known activity for which the Knowledge Management community has designed powerful languages. The Web Ontology Language 2 (OWL2), one of these languages, is supported by tools that first allow Domain Experts to represent their knowledge using familiar concepts [1] and then to reason about it thanks to inference mechanisms.

In addition, OWL2 natively supports the expression of multi-level models through a mechanism called "punning", enabling a model element representing a domain entity to be a class and an instance at the same time. This makes it suitable for the modeling of domains as those of our interests, i.e., human and user-system interaction [1], [2].

However, OWL2 does not provide behavioral modeling concepts but only structural modeling ones. This prevents Domain Experts from automatically executing their models. Modeling and analyses on the resulting model cannot be done in a round-trip way. Working in an iterative and incremental way using a trial and error approach is therefore not facilitated.

To overcome this limitation, we propose xOWL, a modeling language built upon the OWL2 standard, extending it with concepts enabling the expression of domain entities' behaviors. xOWL comes with an interpreter integrated in an environment offering models executability as well as inference facilities. In an approach similar to the design of DSLs, a domain-specific visual syntax for the general purpose xOWL can be developed for each of the targeted class of Domain Experts, thus allowing them to manipulate concepts of their domain. Thanks to the coupling of these two techniques, Domain Experts have immediate feedbacks on the model they built using a syntax having familiar concepts and can therefore improve it step by step.

This paper is structured as follows. In the next section, we provide an example we use in this paper. Section III details xOWL, the language we defined as an extension of OWL2. The technical implementation is described in Sect. IV. We present related works in Sect. V, and conclude in Sect. VI.

II. Example

Organizing and maintaining the security of an event such as the Olympics Games is a challenging issue. Nowadays it is addressed by building a network of information systems interconnecting several emergency services and surveillance

1541-7719/11 $26.00 © 2011 IEEE
DOI 10.1109/EDOC.2011.13

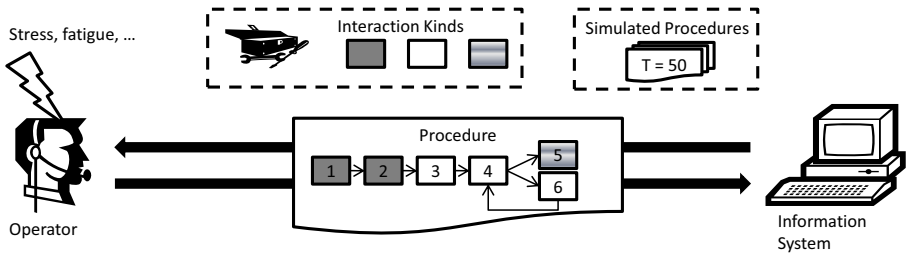

Figure 1. Representation of operators' procedures by UI experts

systems. In order to ensure a coherent use of security and emergency resources during a crisis, specific procedures have to be followed by information systems' operators. These information systems, as well as the procedures for interacting with them are security-critical. Thus, they have to be thoroughly tested. Although the information systems themselves may be developed using formal methods in order to minimize the risk of failure, the operator/system couple is still error-prone due to possible misinterpretations from either part.

In order to efficiently design security-critical systems, human interactions have to be taken into account from the very beginning. In a model-based testing approach[1], preliminary models of the information systems are used to test preliminary models of the procedures. These tests realized in advance, before the complete system is available, can give useful feedbacks to designers in order to cover for potential flaws. In this context, we aim at building a system that will allow User Interaction (UI) Experts to test operators' procedures.

In the following fictive example[2], UI Experts want to represent (model) their domain in order to simulate (test) it. This domain is a time-critical `Procedure`: the reaction to a medical emergency call. The purpose of the test is to determine whether the `Procedure` can be safely accomplished by the `Operator` under a certain time threshold, even under stressful circumstances. In order to do this, UI Experts have identified the key operator/system `Interactions` composing the `Procedure`:

1) Get the emergency nature
2) Get the emergency location
3) Query nearest medical outpost
4) Query outpost for availability
5) If available: Dispatch emergency call
6) Else: Query next nearest medical outpost, go to 4

For each of these `Interaction` elements, UI Experts have attached a time limit. Then, they have organized the `Interactions` into broad classes. For instance, the gathering of information from a distressed person over the phone (1 and 2) is an `Interaction Kind` that requires specific skills. Then, determining the location and availability of medical teams is another `Interaction Kind` involving the information system. Each of these `Interaction Kind` has also been given an estimated `difficulty` index in order to represent that some `Interaction` are more prone to errors than others. Finally, UI Experts also represent results of `Procedure` simulations with information such as the timestamps for start and end in order to draw meaningful results. Figure 1 illustrates this example.

For the simulation of the represented `Procedure` to be realistic, UI Experts have to link their `Procedure` models to the information system model, as well as to a model of the `Operator` behavior. This allows testing the domain `Procedure` against multiple profiles of `Operator`, such as novice or experimented, under normal or stressful conditions, etc. For the time being, UI Experts have identified the following rules determining the success and time consumption of `Procedure`, depending on the interaction's `difficulty` and the operator `skill` and `stress`:

1) difficulty $\leq$ skill $\Rightarrow$ success $\wedge$ in time
2) difficulty $>$ skill $\wedge$ stress $<$ 5 $\Rightarrow$ success $\wedge$ in time
3) difficulty $>$ skill $\wedge$ stress $\geq$ 5 $\Rightarrow$ success $\wedge$ time * 1.5
4) difficulty $>$ skill $\wedge$ stress $\geq$ 10 $\Rightarrow$ failure $\wedge$ time * 3

Other rules of this domain can be written in a similar fashion.

Due to the high specificity of this knowledge, it has to be represented by UI Experts themselves for being the most accurate and complete. UI Experts work in an iterative fashion, constantly testing their domain models in order to ensure that chosen values like the `skill` in our example are still relevant.

To summarize, this example highlights the following needs:

1) Structural knowledge representation, e.g. the `Operator` concept has a `skill` attribute.
2) Behavioral knowledge representation, e.g. the `EmergencyCall` procedure, which specifies the behavior of the `Operator`.
3) Inference rules representation, e.g. the success rules stated above.
4) Execution of the behavioral knowledge, e.g. the simu-

[1]Here, model-based testing is used in contrast to testing with real systems
[2]Inspired from the EADS involvement in the Beijing Olympics Games.

lation of the `Procedure`.

Multiple technologies described in the litterature address some of the needs expressed above. On one hand, the expression of structural knowledge in conjunction with inference rules is a problem addressed by the Semantic Web community, which produced languages such as OWL2 and the Semantic Web Rule Language (SWRL) [1], [3]. Reasoning capabilities are needed because much of the targeted audience will express their knowledge as (domain) rules, especially in human sciences. In our opinion, the best way to leverage this kind of knowledge is to use inference rules. However, these technologies do not enable their users expressing executable domain behavior, such as `Procedure` in our example. On the other hand, the representation of structural and behavioral knowledge supported by an execution engine is addressed by technologies from the Model-Driven Engineering community, such as fUML [4] and Kermeta [5]. Technologies emanating from the OMG world (with UML and MOF as flagships) can also be connected to reasoning capabilities [6]. However, this often comes down to providing a mapping to Description Logic (or similar) whereas OWL is readily based on the said DL. Furthermore, OMG Ontology Definition Metamodel standard strive to fill in the gap, but reasoning capabilities are still stronger in the semantic technologies ecosystem.

In any case, we cannot expect Domain Experts such as the UI Experts to learn a general purpose modeling language that is therefore not from their domain. To remedy this situation, a DSL-like approach is to be taken, where a concrete visual syntax for each class of Domain Experts (such as the UI Experts) will have to be provided. Taking inspiration from the behavior representation and execution capabilities of technologies such as fUML, we chose to base our work on semantic technologies in order to readily benefit from the strong support for inferences. Consequently, to overcome behavior representation and execution limitations we propose xOWL, described hereafter, along with an application case showing an example of concrete visual syntax for UI Experts.

III. xOWL: an Executable Multi-Level Modeling Language

The following paragraph expands on OWL2 in order to give a better view on how it has been extended. Then, we give details about the proposed extensions, leading to the construction of xOWL, an executable OWL2-based language.

A. OWL2 Basics

OWL2 ontologies are formally defined as sets of axioms [7], where an `Axiom` is a basic unit of information stating what is true in the domain described by the ontology. The OWL2 language defines multiple kinds of axiom; a few are represented in Fig. 2 as examples. For instance, the `Declaration` axiom expresses the existence of a particular entity within the domain. OWL2 entities include classes, individuals and properties. Each `Entity` is identified by an `IRI` (International Resource Identifier). An `IRI` is basically a name to which can be attached different interpretations represented by the OWL2 entities. From the Sect. II example, the `Declaration` axiom would state the existence of the `Procedure` class. Also, the `Procedure` concept is at the same time a classifier for the `EmergencyCall` concept and an ontological instance of the `InteractionKind` concept. This kind of modeling approach, coined "Multi-Level Modeling" is needed to accommodate knowledge from fields such as human sciences. The `ClassAssertion` axiom is used to state the ontological instantiation between these three concepts:

Listing 1. OWL2 axioms

```
1 Declaration(Class(:Procedure))
2 ClassAssertion(:InteractionKind :Procedure)
3 ClassAssertion(:Procedure :EmergencyCall)
```

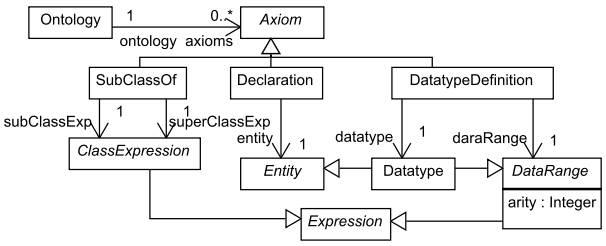

Figure 2. Excerpt of the OWL2 meta-model (from [1]) expressed in OWL2

These simple concepts are extended by the xOWL language, which is built using an architecture presented in the following paragraph.

B. xOWL Architecture

In order to extend OWL2, we reuse the OMG Model-Driven Architecture (MDA), which is a layer-oriented architecture. It enables the expression of a model at a layer i using the language defined at the layer $i+1$ [8]. In the OMG terminology, a layer i is said an instance of its upper layer $i+1$. Some works call this instantiation relation a *linguistic instantiation* [9].

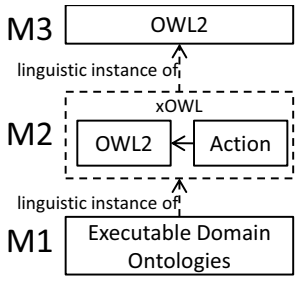

Figure 3. xOWL architecture

As shown in Fig. 3, the OWL2 language itself has been used to express the abstract syntax of xOWL, which includes

two ontologies (packages): the **OWL2** package and the **Action** package. The **OWL2** package redefines the abstract syntax of OWL2 within xOWL. Then, the **Action** package defines the abstract syntax for behaviors, in relation to the **OWL2** package. Thus, domain ontologies expressed using OWL2 can also be considered as expressed in xOWL. Therefore, xOWL is immediately backward compatible with existing ontologies.

The next paragraphs detail the **Action** package containing concepts related to the modeling of behaviors and xOWL expressions, as well as their relations with the **OWL2** package.

C. Behavior Modeling

Behaviors can be represented using a wide variety of ways, such as state machines, activities, processes and algorithms. In xOWL, behaviors are represented as algorithmic structures containing actions executed on xOWL ontologies. Since these basic structures are not suitable for Domain Experts, they are encapsulated into domain-specific high-level constructs defined in xOWL libraries. These libraries also contain the xOWL algorithms stating how to execute the high-level constructs. In the example from Sect. II, UI Experts build their behavioral models using a xOWL library developed by IT Professionals. This library contains a small set of foundation concepts allowing UI Experts to easily express themselves procedures such as `EmergencyCall`. The library also contain a xOWL algorithm stating how to execute procedures.

Algorithms themselves can be expressed using many languages implementing different paradigms. In xOWL, inspired from the OMG fUML language [4], we chose an imperative paradigm enhanced with some features borrowed from the functional paradigm: The language is also able to manipulate lambda expressions, as will be shown in the next paragraphs. Hence, as shown in Fig. 4, classical imperative control structures are provided, such as loops (*For* and *While*), conditional statements (*If*), blocks, invocation of sub-behaviors (*Invokation*), etc. xOWL implements a lexical scoping paradigm for variables.

In the example from Sect. II, IT Professionals write an algorithm for the execution of the corresponding `Procedure` high-level constructs. An excerpt is shown in Listing 2. It simply creates a new instance of the current `Procedure` and loops through the sub-`Interaction` (steps) invoking another function for executing them. Finally, it returns the created instance.

Listing 2. Excerpt from the procedures' execution xOWL algorithm

```
1   //Parameter of the algorithm
2  Param procedure
3   //Instantiate the procedure
4  Var instance = NewIndividual(procedure)
5  Var steps =
6    Query(ObjectPropertyAssertion(:contains procedure ?))
7  For(Var i=0, i!=LengthOf(steps), i++)
8     //Invoke exec_elem
9   Var sub = Invokation(:exec_elem steps[i])
10     //Get the procedure->sub object property
11   Var link = ...
12   Add(ObjectPropertyAssertion(link instance sub))
13 Return instance
```

These algorithmic structures can express the modification of xOWL ontologies using specific statements called *Action* in Fig. 4. Actions allow the addition and removal of axioms from the xOWL ontology. This is coherent with the OWL2 philosophy because axioms are the basic unit of information within an ontology (see Sect. III-A) and actions only offer to add or remove some of these units of information. An example can be seen in the previous listing where a new `ObjectPropertyAssertion` axiom is added (line 12). In order to express axioms such as this one using variables we need more complex expressions shown in the next paragraph.

D. xOWL Expressions as OWL2 Extensions

Initially, OWL2 provides concepts for class expressions, individuals, dataranges, literal expressions, etc. xOWL extends them by adding new kinds of expression, allowing the use of variables, queries and invocations amongst others (see Fig. 5). The extension of the expressions' grammar allows stating "parametric" axioms. For example, in Listing 2, line 12:

```
Add(ObjectPropertyAssertion(link instance sub))
```

This statement adds the axiom asserting that the two individuals in the 'instance' and 'sub' variables are related by the property contained in the 'link' variable (defined in Listing 2, line 11). This is equivalent to "instance . link = sub" in an object-oriented languages, except that 'metalink' is itself a variable and not the name of a property.

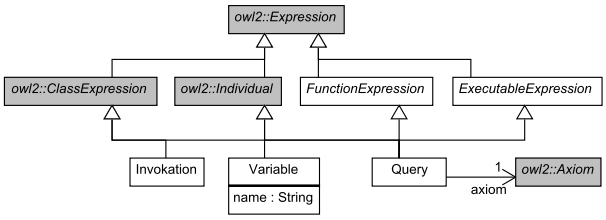

Figure 5. xOWL expressions (excerpt of the **Action** package) expressed in OWL2

For the xOWL algorithms being able to manipulate information, xOWL must provide a way for them to retrieve information from the ontologies. To this end, xOWL provides the *Query* language element expressing a query for information stated within the ontologies. The common task of getting the value of a property for a given object is expressed using a simple *Query*. In our example, a *Query* is used to retrieve the sub-`Interaction` contained by the `Procedure`:

```
Var steps =
    Query(ObjectPropertyAssertion(:contains procedure ?))
```

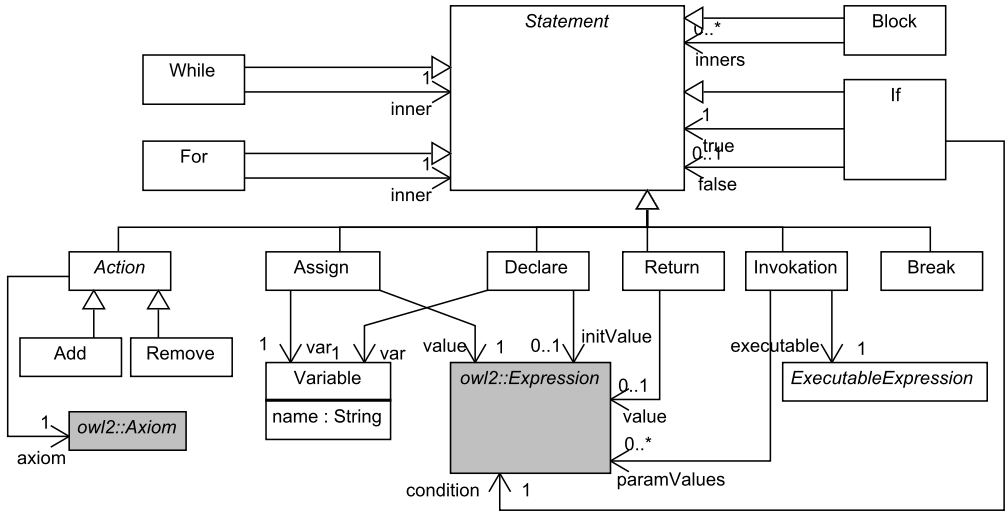

Figure 4. Statements (excerpt of the **Action** package) expressed in OWL2

Here the *Query* will return all values of the free variable '?' that enable the pattern to be matched to an axiom within the ontology. This allows a uniform access to information within the xOWL ontologies without discriminating between ontological levels, or the kind of information queried. In addition, queries are to be interpreted by an inference engine, thus providing powerful capabilities. For example, querying for the subclasses of a given `Class` would also return the inferred subclasses and not only the asserted subclasses (subclass relationships explicitly stated using an axiom).

Finally, xOWL provides language elements for representing common literal expressions, such as mathematical and logical operations (arithmetic operators, Boolean operators, comparators, etc.). The next paragraph will present how the behavior modeling concepts in this **Action** package are related to the plain OWL2 concepts.

E. Relation with the OWL2 Package

xOWL separates the syntax for structural knowledge (**OWL2** package) from the syntax for behavioral knowledge (**Action** package). However, we still need to provide a "glue" that will remain in the OWL2 philosophy. In the previous paragraph, we demonstrated how to express algorithms using statements (*Statement*). Hereafter is explained how algorithms integrate with OWL2 concepts.

As shown in Fig. 6, we encapsulated algorithms within lambda expressions (*Lambda*). In this context, lambdas represent anonymous pieces of "parametric" behaviors, having parameters (*Variable*) and being defined using a *Statement*. However, lambda expressions cannot be used directly because they depend on the lexical context in which they have been defined. Consequently, we use the *Closure* concept to represent a lambda expression within a certain lexical context represented by a collection of *Upvalue*, associating variables to values.

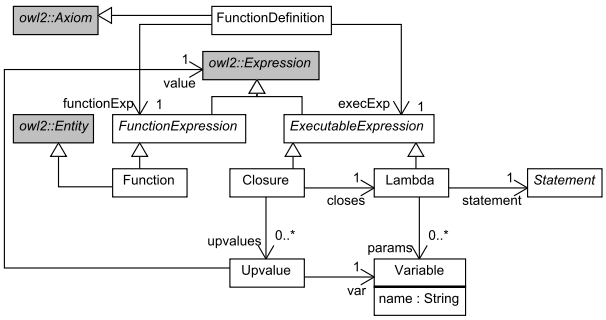

Figure 6. Binding **Action** package to **OWL2** package

From this, we introduce the *Function* concept. A *Function*, as shown in Fig. 6 is a new kind of ontological entity (`Entity`). *Function* are also a kind of *FunctionExpression*, which are expressions (`Expression`). In this way, it is possible for a variable to hold a *Function* as a value. Because a *Function* is an `Entity`, it is referenced by an `IRI`. This makes a *Function* a new possible interpretation for `IRI`. Consequently, a single concept (`IRI`) can now be a `Class`, an `Individual` and a *Function* at the same time.

Finally, *Function* are associated to behaviors (*Closure*) through the use of an axiom. This new kind of axiom, called *FunctionDefinition*, is also shown in Fig. 6. It is similar to the `DatatypeDefinition` axiom shown in Fig. 2 in the way it associates an ontological `Entity` to its definition. Doing so, xOWL is respectful of the OWL2 philosophy, in which the `Axiom` is the unit of information. In our example, this new axiom is used to define the execution function of the `Procedure`:

```
FunctionDefinition(:exec_procedure
 Lambda(Var procedure ... )
)
```

When applying the axiom, the *Lambda* containing the content of the Listing 2 will be evaluated as a *Closure*. Consequently, behaviors can be attached to ontological entities, thus providing a multi-level modeling language supporting the representation of behaviors.

Additional extensions have been provided for usability, although not detailed in this paper. These usability features include:

- Native support for the expression of arrays and associated expressions including array concatenation and slicing.
- Native support for interoperability with a host language. As shown in the next section describing the implemented prototype, Java has been used as a host language. Therefore, algorithms expressed using xOWL can manipulate Java objects and invoke Java operations. This gives access from xOWL ontologies to the entire Java ecosystem. This host language may not have formal declarative semantics. However, one can still either forbid the use of such constructs or limit them to particular ontologies in order to leverage the formal declarative semantics of xOWL for verification purposes.

Finally, OWL2 is a formally defined language, thus bearing precise semantics. Then, in order for xOWL to be consistent, it also had to provide a formal definition based on the one provided by OWL2. Consequently, the operational semantics of xOWL have been formally defined using a transition system, based on the OWL2 direct semantics [7][3]. These formally defined operational semantics have then been implemented in a proof of concept prototype.

IV. The xOWL Implementation and Application

This section will first present some general aspects of the implemented interpreter. Then, we apply the language key features to the example presented in Sect. II.

A. Prototype Interpreter

The implemented prototype has been designed in a modular way, reusing existing libraries, in order to limit development efforts while maximizing functionalities. Figure 7 shows the general architecture of our proposal.

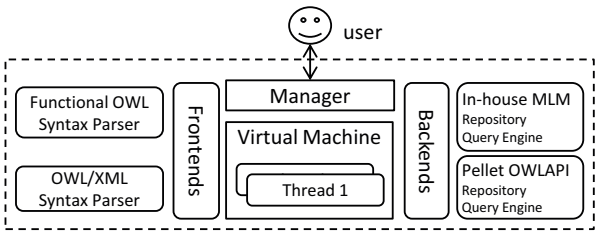

Figure 7. Architecture of the xOWL interpreter

[3]See http://www.xowl.info/

The prototype can be split into three main parts. The first and central one encompasses the `Virtual Machine` for the language interpreter, as well as the `Management` layer. The second part contains what are called `Frontends`, i.e. sets of components allowing the loading of extended ontologies. Finally, the last part contains `Backends`, i.e. sets of components for the in-memory representation of extended ontologies and querying capabilities, etc.

The `Virtual Machine` is responsible for the implementation of the language operational semantics. It contains the logic for the interpretation of expressed algorithms. On top of the `Virtual Machine`, the `Management` layer acts as an interface layer with users, providing a clean and simple API facade.

`Backends` are the parts of the prototype that take care of the in-memory representation of extended ontologies. Multiple `Backends` mean that multiple methods of representation are available. Two `Backends` are currently implemented in the prototype and as shown in Fig. 7. The first one is an experimental in-house multi-level modeling backend. The second relies on OWLAPI, a free set of libraries for OWL2 and Pellet [10], an OWL2 reasoner. Doing so we take advantage of the inference and querying capabilities of the reasoner. Finally, one can also implement `Backends` based on the Jena API [11] or CORESE [12], a reasoner developed by the INRIA for RDF [13].

In order for the interpreter to be easily and efficiently used, some additional features have been included. The interpreter then natively provides support for debugging through specific interfaces. Debugging features include runtime breakpoint injection and removal, step-by-step execution, stack and context (variable values) exploration and on-the-fly expression parsing and evaluation.

B. Usage and Concrete Syntax

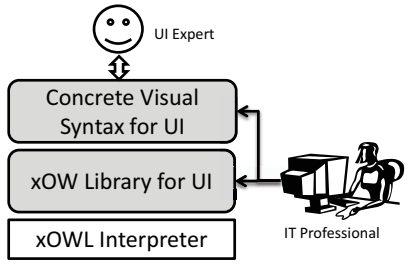

Figure 8. Architecture of the solution for UI Experts

As explained in previous sections, we aim at endowing Domain Experts that have no software-related skills with the xOWL language. The abstract syntax of which is described in Sect. III. In order to maximize Domain Experts' engagement regarding their modeling task, domain-specific representation metaphors are preferable. Then, the challenge that needs to be addressed is the elaboration of concrete syntaxes, one for

each class of Domain Experts. The approach chosen here, summarized in Fig. 8, is two-fold.

First, IT Professionals have to build a minimal xOWL library for the targeted class of Domain Experts. This xOWL library for one particular domain will typically package the domain-specific concepts expressed in the form of xOWL ontologies. It will also contain the algorithms (expressed in xOWL) that specify how to interpret these concepts in term of behaviors. Consequently, user models built using this library can be executed.

Then, a domain-specific concrete syntax has to be implemented. This concrete syntax provides terms from the domain-specific xOWL library. Consequently, Domain Experts can use the xOWL language transparently through a concrete syntax depending on a domain-specific xOWL library.

C. Application to the Example

This section illustrates how UI experts use xOWL to describe their structural and behavioral knowledge and test the resulting model. We come back to the example of Sect. II. Figure 9 is a structural representation of what UI Experts *want* to model, namely the behavior of the `EmergencyCall` procedure. As shown in the figure,

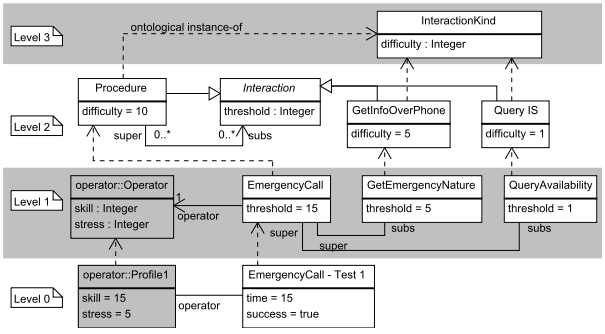

Figure 9. Excerpt from the structural multi-level representation of `EmergencyCall`

the `InteractionKind` concept classifies the different `Interaction`. It declares that each kind of interaction has a `difficulty` index. The `Procedure` concept is a classifier for UI Experts' procedures, for instance the `EmergencyCall`. The `Interaction` concept federates new kinds of interaction created by UI Experts. Using these concepts, UI Experts express the behavior of the `EmergencyCall` procedure. The 6 steps described in Sect. II are then to be represented, although not all of them are shown in Fig. 9 due to lack of space. The `GetEmergencyNature` is then an ontological instance of the `GetInfoOverPhone`, which is an interaction kind defined by UI Experts. Similarly, the `QueryAvailability` interaction is an instance of the `QueryIS` interaction kind. Level 0 contains the result of `Procedure`' simulations as ontological instances of the `Procedure` themselves. Figure 9 also shows that the `Procedure` model is connected to a model of the `Operator`. Albeit not shown here due to space, the `Procedure` model is also connected to the Information System's model.

The kind of representation shown in Fig. 9 akin to a UML class diagram could be a concrete visual syntax for OWL2 ontologies. However, it implicitely refers to terms that are not from the domain of the UI Experts. Actually, in order to efficiently manipulate such a representation, UI Experts would have to understand concepts such as "Class" and "Individual". Consequently, following the approach described in subsection IV-B, IT Professionals have been commisioned for the development of the corresponding xOWL library and a concrete visual syntax.

The xOWL library contains both the core concepts and the algorithm for their execution. Thus `InteractionKind`, `Procedure` and `Interaction` are contained in the library. Others (e.g.: `EmergencyCall`) being all either direct or indirect ontological instances of those ones do not need to be included in the library. The algorithm, a excerpt of which is given in Listing 2 from Sect. III., enabling UI experts to apply analyses to the represented `Procedures` only uses these three concepts. Moreover, as explained in Sect. II, the knowledge about how to execute `Procedures` is complemented by inferences rules. For example, the knowledge about whether the Procedure is successful or not is better expressed using inference rules. In this case, reasoning capabilities are to be directly interwoven with the execution capabilities because the very result of the `Procedure` execution algorithm expressed in xOWL depends on inference rules. These rules can be written using the Semantic Web Rule Language (SWRL). For example, rule 1 cited in Sect. II can be expressed as:

`Operator`(?x, ?o) $\wedge$ `skill`(?o, ?s) $\wedge \cdots \wedge$ `swrlb:lesserThanOrEqual`(?d, ?s)
$\Rightarrow$ `success`(?x, true) $\wedge \cdots$

The inference rules are not included in the xOWL library for UI Experts because they will follow the same rapid prototyping cycle as the `Procedure` data. The rules being loaded by the backend reasoner of the xOWL interpreter, this one is then able to leverage them during the simulation/execution of the represented `Procedure`.

The concrete visual syntax is made available to UI Experts by its implementation in a CAD [4] (Computer-Aided Design) tool (see Fig. 10). The tool provides UI Experts a set of icons corresponding to the UI domain concepts expressed in the xOWL library. As shown in Fig. 10, the concrete visual syntax for UI Experts uses boxes to represent elemenents of the `Procedure`, such as the `GetEmergencyLocation` interaction. Arrows symbolise the flow of `Interactions`.

[4]The CAD designation is preferred over CASE (Computer-Aided Software Engineering) here because the latter focuses on the design of software artifacts whereas the tool shown here targets the UI domain, outside of the Software Engineering field.

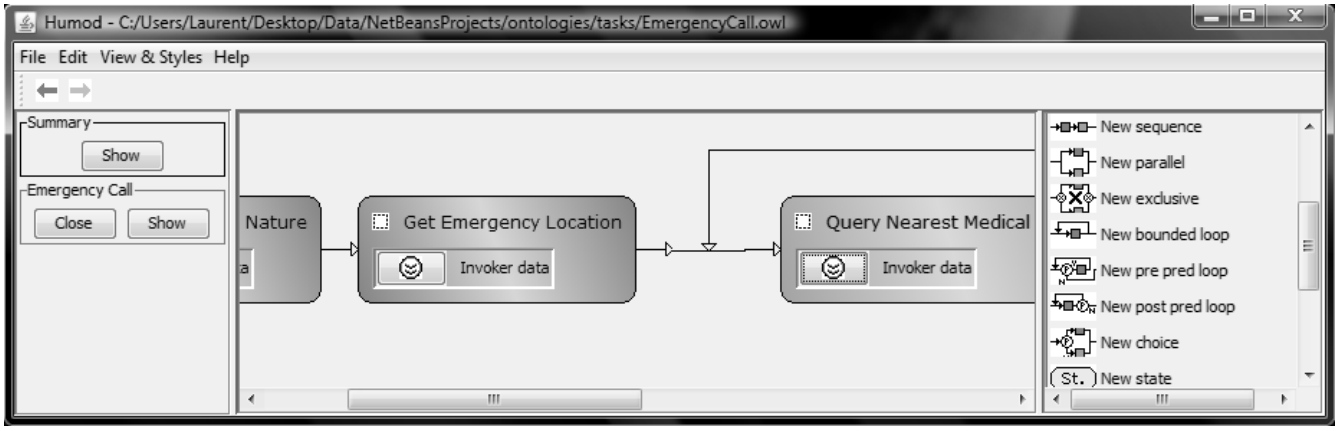

Figure 10. Procedure modeling and testing tool implementing the visual concrete for UI Experts

This concrete visual syntax has been constructed using existing formalisms used by the UI Experts on paper. In addition to enabling UI Experts to model their knowledge, this tool allows them to test their models. To this end, it relies upon the xOWL interpreter for the execution of procedures' simulations (execution) and analyses. The algorithm defining how to execute procedures being also defined in the xOWL library for UI Experts, the interpreter is able able to execute them. Using at the same time inference rules defined in Sect. II, the interpreter simulating the procedures can determine whether these are actually successfully handled by the operator. For instance, the tool will provide as feedbacks to the UI Experts that the procedures cannot be safely handled by the operator because he/she is too much stressed.

To summarize, the xOWL language addresses the identified needs for structural and behavioral modeling. Because it is based on OWL2, is can be used in conjunction with the SWRL language for the expression of inference rules. The xOWL interpreter is able to execute algorithms expressed in xOWL, using inference rules Using a DSL-style approach, a minimal xOWL library has been developed by IT Professional for UI Experts, as well as a concrete visual syntax. Consequently, UI Experts can represent and test their knowledge in an iterative fashion using the provided CAD tool.

V. Related Works

This work relates to multiple fields, each developing different sets of researches. First, the modeling community produced several examples of multi-level modeling languages, and at least one explicitly multi-level programming language [14]. In addition, the knowledge engineering and semantic web communities provided strong logic-based artifacts, OWL2 being one of them, that are related to this work.

The model engineering community has produced several researches proposing multi-level modeling languages. Some languages like Nivel [15] enforces strict multi-level modeling (no relation between concepts from different ontological layers), and provides a formal definition. However, Nivel is not able to express behavior, a feature provided by the Kermeta language [5]. Kermeta is an executable modeling language that is based on EMOF (conceptually) and ECORE (in practice). It allows the expression of behaviors as algorithms, on top of a meta-data modeling language. However, Kermeta does not support the use of inferences during execution.

Behavior modeling is broad subject that has been explored by different communities. From a Model-Driven Engineering point of view, the UML standard provides several ways to represent the behavior of a system [16]. Other technologies such as Kermeta have a more fundamental approach. In addition, other works focus on the formal modeling of behavior [17]. For instance the B formal method uses the formal definition of behavior for the mathematical proofs of programs [18]. However, our motivation is to leverage the formal operational semantics of the language for the definition of domain-specific libraries. With different motivations comes different applications. The B language is not usable in our context due to its mathematical proving orientation. Nevertheless, the formal modeling of behaviors is a necessary step toward reasoning capabilities.

An intermediate approach supporting rules in the context of programs has been implemented in Jess [19]. Jess allows its users to express inference rules along normal Java programs. However, Jess is not a modeling language defined using formal semantics. Consequently, it does not seem suitable for our purpose. Nevertheless, the Jess approach is interesting because the one presented here is somehow symmetric. Jess extends the Java ecosystem (executable) with inference rules, whereas we strive to extend ontologies supporting formal reasoning with executability.

This reasoning feature is indeed provided by technologies from the semantic web community. For instance, the OWL2 language serves as a base for the definition of various languages for the expression of rules, most notably SWRL [3],

which can be used by backends of our xOWL interpreter. The combined use of OWL2 and SWRL enables the expression of behaviors through SWRL rules only. However, some behavioral knowledge such as the procedures of our example in Sect. II is best expressed using a different formalism. To this end, xOWL provides a practical solution because it does not restrict the expression of behaviors to predefined formalisms. In our example implemented in Sect. IV-C, operator's procedures, which are the behavior of the operator, is expressed using a process formalism. Other formalisms such as state-machines can be used by building an adequate domain-specific xOWL framework.

Other works focus on the representation of changes within ontologies. For instance, an ontology of the possible changes within OWL2 ontologies has been produced [20]. Although providing insight on ontology modifications, this work does not allow the representation of changes within algorithmic structures, a feature provided by xOWL.

Another approach is the representation of behaviors using pure ontologies only. In particular, the representation of algorithms using ontologies has been explored in [21]. However, the work of Grassi et al focuses on reasoning about the behavior and not representing it for its execution. Because the OWL2 language elements are used to represent algorithms, it is not possible to differentiate in an ontology the concepts describing structural knowledge and those describing algorithms, thus preventing their execution. In contrast, xOWL provides new language elements for the expression of algorithms, thus enabling an interpreter to distinguish the algorithms for their execution.

Also, an approach focusing on the expression of DSLs has been presented in [22]. The tool called Magic Potion [23] enables IT Professionals to define executable DSLs using ontologies. However, whenever knowledge about the domain change it requires to redesign the DSL, migrate the existing data, etc. In contrast, xOWL is a general purpose language which can be adapted to domains using domain-specific libraries, without losing its ability to express and execute behaviors. A different approach is also provided by the Eclipse EProvide plugin, which enables the definition of executable DSLs with visual concrete syntaxes [24]. EProvide supports the automatic generation of the appropriate tooling for executable DSLs, such as debuggers. We differ from EProvide by defining an general-purpose executable language that is then specialized for specific domains.

Moreover, designing a new Domain-Specific Language from scratch is also a risky approach [25]. For one, IT Professionals has to elicit domain knowledge from the Domain Experts. Misunderstandings are not uncommon during these processes [26]. Our approach strives to mitigate these risks by requiring from the IT Professionals to build only a minimal domain-specific xOWL library and the corresponding concrete syntax. The xOWL language itself is readily available and address the representation and execution needs expressed by Domain Experts (UI Experts in our example). This results in an approach much similar to the UML profiling mechanism, which can be extended to define DSLs, decoupling the abstract and concrete syntaxes [27]. However, our approach priviledges expressivity by combining multi-level modeling capabilities with behavior modeling concepts and inference rules.

VI. Conclusion

We are interested in the modeling of complex domains as illustrated in this paper. A way to tackle this complexity is to provide Domain Experts with tooling to model themselves their knowledge and validate their models. To this end, we have developed xOWL, a language that extends the OWL2 standard language, complementing it with behavioral concepts. xOWL is supported by an interpreter enabling models executability. In addition, using minimal development IT Professional can produce a modeling environment targeting a specific field of expertise, as it has been shown in Sect. IV A strong advantage of our language is the combination of executability and inference mechanism. This provides Domain Experts (UI Experts for instance) meaningful feedbacks from the simulation, thanks to the inference rules.

Although the proof of concept tools have been handed over to UI Experts at EADS, validation is an ongoing work. Outside the example we present, xOWL is applied to industrial-grade data coming from the aeronautics and human science fields at EADS, i.e., in a diversity of application fields (a glimpse of which is given in the paper). Future works investigate several issues. We are studying a full-fledged integration with the SPARQL language (SPARQL Protocol and RDF Query Language) [28]. Indeed, current xOWL *Query* is a simplified version of SPARQL query. This integration would allow more expressiveness. Additionally, taking advantage of the modular architecture of the xOWL interpreter, new backends supporting other reasoners and inference engines should be developed. Having different backends will allow xOWL users to choose the one best fitting their needs, as each reasoner has its own limitations. An important issue is the evaluation of performance. Although the current implementation is satisfactory regarding this particular application example, a performance study must be conducted. We are investigating both empirical studies based on the UI Experts and complexity theoretical studies.

References

[1] W3C, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax," Oct. 2009. [Online]. Available: http://www.w3.org/TR/owl2-syntax/

[2] B. Neumayr and M. Schrefl, "Multi-level Conceptual Modeling and OWL," in *ER 2009 Workshops on Advances in Conceptual Modeling - Challenging Perspectives*. Springer-Verlag, 2009.

[3] W3C, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," May 2004. [Online]. Available: http://www.w3.org/Submission/SWRL/

[4] OMG, *Semantics of a Foundational Subset for Executable UML Models*, OMG Std., Rev. Version 1.0 Beta 3, 2010.

[5] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, "Weaving Executability into Object-Oriented Meta-languages," in *MoDELS*, ser. LNCS. Springer Berlin / Heidelberg, 2005.

[6] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML Class Diagrams," *Artificial Intelligence*, vol. 168, pp. 70 – 118, 2005.

[7] W3C, "OWL 2 Web Ontology Language Direct Semantics," Oct. 2009. [Online]. Available: http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/

[8] OMG, "MDA Guide Version 1.0.1," 2003.

[9] C. Atkinson, M. Gutheil, and B. Kennel, "A Flexible Infrastructure for Multilevel Language Engineering," *IEEE Trans. Softw. Eng.*, vol. 35, 2009.

[10] B. Parsia and E. Sirin, "Pellet: An OWL DL Reasoner," in *International Workshop on Description Logics*, 2004.

[11] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: Implementing the Semantic Web Recommendations," in *13th International World Wide Web conference on Alternate track papers & posters*. ACM, 2004.

[12] O. Corby, R. Dieng-kuntz, and C. Faron-zucker, "Querying the Semantic Web with the CORESE Search Engine," in *16th European Conference on Artificial Intelligence*. IOS Press, 2004.

[13] W3C, "Resource Description Framework (RDF)," Feb. 2004. [Online]. Available: http://www.w3.org/RDF/

[14] T. Kühne and D. Schreiber, "Can Programming Be Liberated From the Two-level Style: Multi-level Programming with DeepJava," in *OOPSLA '07*. ACM, 2007.

[15] T. Asikainen and T. Männistö, "Nivel: A Metamodelling Language with a Formal Semantics," *Software and Systems Modeling*, vol. 8, 2009.

[16] OMG, *UML Version 2.1.2*, OMG Std., 2007.

[17] S. Wang, J. Ma, Q. He, and J. Wan, "Formal Behavior Modeling and Effective Automatic Refinement," *Inf. Sci.*, vol. 180, pp. 3894–3913, 2010.

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

[19] E. F. Hill, *Jess in Action: Java Rule-Based Systems*, E. F. Hill, Ed. Manning Publications Co., 2003.

[20] R. Palma, P. Haase, O. Corcho, and A. Gómez-Pérez, "Change Representation For OWL 2 Ontologies," in *7th Internation Workshop OWL: Experiences and Directions*, 2009.

[21] S. Grassi, S. Barrett, and F. Sordillo, "Ontology Based Algorithm Modeling: Obtaining Adaptation for SOA Environment," in *2nd workshop on Middleware for Service Oriented Computing*. ACM, 2007.

[22] D. Djuric, J. Jovanovic, V. Devadzic, and R. Sendelj, "Modeling Ontologies as Executable Domain Specific Languages," in *3rd India Software Engineering Conference*. ACM, 2010.

[23] D. Djuric and V. Devedzic, "Magic Potion: Incorporating New Development Paradigms through Metaprogramming," *Software, IEEE*, vol. 27, 2010.

[24] D. Sadilek and G. Wachsmuth, "Prototyping Visual Interpreters and Debuggers for Domain-Specific Modelling Languages," in *Model Driven Architecture Foundations and Applications*, vol. 5095. Springer, 2008, pp. 63–78.

[25] P. Laird and S. Barrett, "Towards Context Sensitive Domain Specific Languages," in *Proceedings of the 1st International Workshop on Context-Aware Middleware and Services*, 2009.

[26] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-Specific Languages," *ACM Comput. Surv.*, vol. 37, pp. 316–344, 2005.

[27] J. Pardillo and C. Cachero, "Domain-Specific Language Modelling with UML Profiles by Decoupling Abstract and Concrete Syntaxes," *J. Syst. Softw.*, vol. 83, pp. 2591 – 2606, 2010.

[28] W3C, "SPARQL Query Language for RDF," Jan. 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/