M3D: A Tool for the Model Driven Development of Web Applications

Mario L. Bernardi  
Department of Engineering - RCOST, University of Sannio, Benevento, Italy  
mlbernar@unisannio.it

Marta Cimitile  
Faculty of Jurisprudence, Unitelma Sapienza University, Roma, Italy  
marta.cimitile@unitelma.it

Giuseppe Di Lucca  
Department of Engineering - RCOST, University of Sannio, Benevento, Italy  
dilucca@unisannio.it

Fabrizio M. Maggi  
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands  
f.m.maggi@tue.nl

ABSTRACT

Nowadays, Web Applications (WAs) are complex software systems, used by multiple users with different roles and often developed to support and manage business processes. Due to the changing nature of the supported processes, WAs need to be easily and quickly modified, to adapt and align them to the processes they support. In recent years, Model Driven Engineering (MDE) approaches have been proposed and used to develop and evolve WAs. However, the definition of appropriate MDE approaches for the development of flexible process-centric WAs is still limited. In particular, (flexible) workflow models have never been integrated with the models (e.g., presentation, information models) used in MDE approaches to develop this type of applications. In this paper, we present M3D (Model Driven Development with Declare), a tool for developing WAs that integrates three MDE metamodels used to represent the main components of a WA with the metamodel of Declare, a declarative language to model business processes. The tool exploits and combines the declarative nature of Declare and the advantages of MDE to get an efficient roundtrip engineering support to develop and evolve flexible process-centric WAs.

Keywords
Web Applications Development and Evolution, Model Driven Engineering, Declarative Processes, Domain Specific Languages, Code Generation

Categories and Subject Descriptors
D.2.2 [Design Tools and Techniques]: Evolutionary prototyping

1. INTRODUCTION

Web Application (WA) development and evolution are becoming more and more critical due to the changeability of the market requests and the increasing necessity for rapid customizations. This awareness is encouraging the introduction of Model Driven Engineering (MDE) approaches to support the development of WAs [10, 7]. However, even if MDE principles are already largely applied to the development of traditional WAs, their adoption in the context of process-centric WAs still suffers from a lack of uniformity in terms of languages, technologies and models.

One of the major limits of the existing approaches for the development of process-centric WAs is that the generation of the process components is not fully automated. Still several manual customizations are required thus forcing the developers to iterate the generation steps several times.

This aspect is even more critical when MDE is applied for the evolution of this type of applications, since also small changes in the process structure require the regeneration of the process-related components. These components often contain customizations that must be correctly merged and weaved with the incoming changes. This lack of adequate round-trip engineering support reduces, in fact, the benefits deriving from the application of MDE approaches in the WA development and maintenance.

These issues are also connected to the nature of MDE [12, 6]. As a matter of fact, MDE methods were born with the aim of modeling Web information systems [1], focusing on information storing, retrieving, transformation and presentation. However, nowadays, real world WAs can be considered as complex information systems involving complex business processes and centered on goals, resources administration and business rules used by multiple users with different roles [9]. For this reason, it is necessary the integration of standard MDE approaches with adequate metamodels for process definition and representation.

However, the most limiting aspect for a widespread adoption of MDE in industrial contexts is the lack of flexibility in the processes definition. In fact, the existing process-centric MDE approaches are based on procedural process modeling languages. This reduces the flexibility of the process model underlying a process-centric WA and adds more complexity when users need more autonomy in their interactions with the application. Indeed, in a procedural model, this variability would require the explicit representation of many alternative execution paths, thus making the model itself
quickly complex and difficult to manage. For this reason, it is necessary to integrate MDE approaches with metamodels for process definition and representation able to improve the flexibility of the process model underlying a process-centric WA.

In this paper, we present M3D (Model Driven Development with Declare), a tool for developing WAs based on the integration of three MDE metamodels used to represent the structure of information, service and presentation layers of a WA with the metamodel of Declare [13, 14, 17], a declarative language to represent business processes. M3D enables the automatic generation of a fully working prototype of a process-centric WA. The usage of MDE technologies in conjunction with a declarative process modeling language allows the developers to represent the behavior of a WA with a high level of flexibility through a compact set of constraints. Furthermore, M3D allows the developers and the designers to work on the process only at the model level without spending any effort for customizing and maintaining the generated code. When changes are in the underlying process (e.g., due to modifications in the process requirements), just the constraints representing the process at the model level are to be modified. The new constraints are automatically translated and forwarded to both the process engine and the presentation layer that, at run-time, will behave according to them. Developers are not required to make any customization of the generated code.

The paper is structured as follows. Section 2 presents some related work. Section 3 describes the approach underlying M3D [3, 2, 1]. Section 4 describes the tool. Section 5 discusses the application of M3D to a case study, while Section 6 provides some conclusive notes and remarks.

2. RELATED WORK

The topic of the integration of process modeling techniques with MDE approaches for the development of WAs is not new in literature. In [12], the authors use a workflow conceptual model supporting the interaction between the hypertext and an underlying workflow management system. In [9], an approach is proposed based on the integration of the Object-Oriented Hypermedia method (OO-H) and the UML-based Web Engineering (UWE). In [15], the content and the navigation models are extended with activity entities and activity nodes respectively, represented through UML primitives. In [16], Business Process Management (BPM) [18] and Object Oriented Web Solution (OWS) [8] are combined. In particular, in this work, model-to-model transformations are used to generate a navigational model from the BPM definition and model-to-text transformations are used to produce an executable process definition in BPEL. Recently, in [4], the authors have presented an integrated approach and a toolset supporting the specification, the design and the implementation of complex, multi-party business processes. This approach is based on BPEL for the process representation, on the MDE methodology and on code generation techniques to produce WAs. Finally, in [5], BPMN is used to describe process actors, tasks and business constraints. These are transformed into a Web user interface and into a Web service orchestration model encoded in a DSL called WebML.

From the literature here illustrated, it is clear that the integration of MDE with process modeling languages is always based on procedural notations. M3D integrates the MDE methodology with Declare and exploits the advantages deriving from the adoption of our MDE methodology in combination with a declarative language.

3. THE UNDERLYING APPROACH

The approach underlying M3D [3, 2] is centered on four integrated metamodels. Each metamodel defines a corresponding DSL to describe a particular aspect of the WA: (i) an information layer metamodel used to represent domain elements and relationships; (ii) a service layer metamodel, used to express system and business logic operations; (iii) a presentation layer metamodel describing the UI by using the concepts of Views, Sub-views, Canvas and Widgets; and (iv) a process layer, based on the Declare metamodel, to represent the business process enacted by the WA users as a set of constraints.

### 3.1 Information Layer

The definition of the WA information layer metamodel is centered upon a common set of modeling concepts that can be effectively used to define sound and rich information models. The adopted constructs and modeling primitives are selected from the well known object-oriented standard, i.e., from the semantics and notation provided by the UML class diagrams. An excerpt of this model is shown in Fig. 1. Here, the main concept is the Classifier that models a generic domain object specialized as an Entity (representing the super-class of unique and persistent objects in the WA domain model). Therefore, entities are characterized by a unique identifier and by a set of properties, and can be linked by binary relationships. A relationship specification is represented as an Association in the model. It specifies the cardinality of relationship roles, the mappings to underlying relational model and any other properties useful to support the generation of the source code and of the other WA artifacts. In addition to an Entity, the model also defines the ValueObject metaclass that can represent a TransferObject (needed to handle values), a PrimitiveType (mapped to a language type) or a UserDefinedType, that refers to types that need special treatment (e.g., binary objects, images, internationalized text). Finally, the metamodel introduces the Operation concept representing behaviors that refer to the elements of the information layer metamodel.

### 3.2 Service Layer

An excerpt of the service layer is depicted in Fig. 2a. The ServiceDefinition is the main concept, defined as an externally accessible entry point in the WA and specifying a set of behaviors adhering to a well known interface. This interface and its properties, specified through the ServiceOperation metaclass, allows the designer to customize the service behavior in several ways. Each ServiceOperation can specify a behavior directly, but can also delegate it to a DataAccessObject (DAO). Each DAO is responsible for handling data of an Entity by implementing CRUD operations (or custom retrieval methods) on it.

### 3.3 Presentation Layer

Figure 2b describes the presentation layer. This model is centered on the Activity concept representing either a user activity that requires the interaction of the user or a batch activity that can be...
executed in background by the process execution engine. The Activity metaclass is the root of a complex hierarchy comprised of different kinds of activities such as BatchActivity, CRUDActivity (used by the code generator to generate CRUD services for entities), ListActivity, TableActivity, GridActivity, ViewActivity just to name few. These activities gather data from the elements of the information model using different kinds of layouts (i.e., lists, tables, views). Each Activity is composed of a set of ActivityProperty instances pointing to external data (such as files, DBMS, remote objects or services) or referencing attributes of the information model (by means of the metaclass AttributeActivityProperty).

Another key aspect of the presentation layer metamodel is the navigability. Activities can specify transitions to other activities and the post-conditions that must be satisfied on the output data. This is important to ensure that data is exchanged among activities instances as specified by the underlying process description.

### 3.4 Process Layer

The process underlying a process-centric WA is described through Declare, a user-friendly constraint-based language first introduced by Pesic and van der Aalst in [14, 17]. Declare constraints have a formal semantics based on LTL and the language is equipped with facilities to execute and verify process models. In this way, it is possible to easily check for possible inconsistencies in the model and to understand whether the modeled process behaves as expected.

Moreover, Declare provides a simple interface that is understandable for users with different backgrounds. A detailed description of Declare is in [13].

The process layer metamodel is reported in Fig. 3. In this metamodel, a Process is composed of events and constraints on these events. Constraint is an abstract metaclass for a rich set of constraints (e.g., Precedence, NotCoexistence). An Event combines a ProcessActivity instance with an ActivityState representing a valid state for that activity. Each ProcessActivity is linked to the service layer metamodel through the mapping of the ProcessActivity attributes with the input/output parameters of a ServiceOperation that specifies the behavior associated to that ProcessActivity. Each ProcessActivity is also linked to the presentation layer being associated to a single UIRoot containing several inner activities. These activities can be created by using the Attribute information thus allowing the user to interact with the process. The process instances are managed by a dedicated component called ProcessManager. The code generator customizes part of the ProcessManager code that dynamically injects ActivityTransition instances into ProcessActivity objects (this requires querying the Declare engine). In particular, when an activity is selected to be executed, the ProcessManager collects the referenced input instances and process variables by wrapping them into a context object, and expose this object to the execute operation of the destination activity.
3.5 Code Generation

Our proposed approach is based on MDE technologies used to enable the automatic generation of a fully working workflow-enabled WA, starting from design models. At the heart of our solution there is a generation approach that takes as input the four instances of the four defined metamodels to generate the implementation for a target platform adopting different well-known architectural patterns. In our case, a J2EE spring-based platform has been selected as target; it exploits the Command-based Model-View-Controller design to build the UI, an aspect-orientation approach to modularize persistence and logging, and dependency injection to increase flexibility and reuse. The approach has been designed to fulfill two goals: first, the use of DSL editors supports the designers with precise specifications driving the modeling of all the aspects of the WA (i.e., information, service, presentation and process points of view); second, it allows designers to link the models of the WA, written using the DSL editor, to the code automatically generated from the models by the target platform.

The generation process of the J2EE based executable prototype produces the following resources:

- source code of Java classes (POJOs, Beans, UI modules, Process steps);
- HTML and CSS resources, such as page headers and footers;
- Platform-specific Descriptors, such as J2EE and JSF descriptors (e.g., web.xml and metadata files to refer to the required components and libraries);
- DBMS scripts, used to implement the database of the application by means of a DBMS (i.e., generating valid SQL code);
- metadata, needed to configure the generated application prototype as an Eclipse Web Dynamic Module project;
- Declare engine integration code, generated by using a Declare model and responsible to query the Declare engine about the next possible activities that can be executed by the user (given the current process instance and the set of constraints).

The model-to-text transformations are implemented as a set of templates written with the Xpand language. To illustrate the generation approach in more detail, an excerpt of an Xpand template, reported in Fig. 4, is discussed below. This Xpand template is used to generate the Java classes implementing an Entity, its Operation classes and the associated DAO and Service classes. This template defines, for each Entity of the information layer metamodel, the specialization of a generic DAO providing CRUD operations and useful persistence methods (by means of the CRUD Operations and DB_Operations expansions). In addition, for each ServiceDefinition specified in the model, a Service class is generated. In turn, a Service class contains an external operation for each ServiceOperation specified in the model (generated by means of the Service-Operation template expansion). Each ServiceOperation is generated as an empty stub in which, by adopting the Generation Gap Operation template expansion). Each ServiceOperation is generated as an empty stub in which, by adopting the Generation Gap Operation template expansion). Each ServiceOperation is generated as an empty stub in which, by adopting the Generation Gap Operation template expansion). Each ServiceOperation is generated as an empty stub in which, by adopting the Generation Gap Operation template expansion). Each ServiceOperation is generated as an empty stub in which, by adopting the Generation Gap Operation template expansion). Each Service is realized by a Delegate service (given the current process instance and the set of constraints).

![Figure 4: An excerpt of the Xpand rule defined to translate an Entity into a DAO java class (the part generating the associated domain class is omitted)](image)

for each Entity defined in the information model, a page containing a CRUDActivity is generated. This CRUDActivity is built by composing a ListActivity (to implement the retrieve operation) and a contextual menu on the Entity instances that allows for the execution of the other CRUD operations. A navigational bar with links to these pages is generated by another template and injected into each view of the application, allowing the user to quickly reach and manage the contents associated to each Entity in the model.

Templates also generate a PendingTasks page, in which process instances (grouped by process definition) are reported, using a TableActivity. On each process instance a contextual menu is available. This menu mixes a static part, containing process management actions (such as start, stop or remove activities), and a dynamic one obtained by querying, at runtime, the Declare engine. The dynamic portion shows the steps that can be executed, given the current state of the process instance and the set of constraints in the reference Declare process model.
4. A TOOL FOR WEB APPLICATION GENERATION

Figure 5 shows the architecture of both the generation platform (on the left side) and the generated application (on the right side) characterizing M3D. The core layer of the generation environment is based on the Eclipse Modeling Platform including the Xtext and the Xpand frameworks, needed respectively to develop the DSL Editors and to drive the generation tasks. On top of it, the code generation layer contains the Xpand Template manager component. It is responsible of applying templates on the user models to generate the needed resources. The IDE integration layer is comprised of four DSL editors (one for each metamodel). They are built using the Xtext framework and integrated into the tool thus allowing the developer to model the application and to start the generation process. The platform also supports the reconciliation of changed artifacts with respect to changes in the models.

A key aspect of the entire generation approach is an effective support of typical round-trip engineering tasks. The generated artifacts are annotated and the generator conservatively ignores the assets not annotated by the @AutoGenerated annotation. This allows developers to preserve their customization across multiple development cycles. The same approach is adopted for other types of artifacts such as descriptors, CSS scripts and HTML activity templates that can be customized by removing them from the list of the autogenerated resources.

After the definitions of all the four models, the developer can interact with the IDE Facet Integration component to start the generation process, driven by the Xpand Workflow component. This acquires the four models and performs the generation by using the services of the middle layer. The tool-chain is implemented as a set of Eclipse Modeling plug-ins, while the EMF/OMOF metamodel is used to represent and manage the four metamodels. The entire generation process (from modeling to code generation and customization) is supported within a single development environment. Xpand is used as transformation language, mainly for its excellent support of EMF metamodels. Other features of this language (aspect orientation, polymorphism, extensibility, type system abstraction and validation) have been exploited for generating the implementation of crosscutting features such as persistence and logging, and to allow for flexible customization of the generated code.

Starting from a model instance, a textual output can be generated by defining and expanding polymorphic macros based on the model elements and their attributes and properties. The elements of the model can be referenced and accessed by the Xpand transformation expressions specified in the templates. The templates can import other templates and model instances and, in addition, also functional Java extensions can be implemented. In our approach, a Java component (called Xpand Design Java Extension) implements several handlers and element mappers needed to drive the generation of the needed resources.

Each element of the model contributes to all transformations. For instance, an Entity of the information model is translated into a Java class and linked to the ServiceDefinition classes in which it is involved. Each ServiceDefinition is, in turn, associated to the DAO accessing the DBMS layer defined by a set of SQL scripts. Similarly, each activity in the process is generated as a Java class linked to the Activity class as specified in the process definition. A generated process manager interacts with the Declare engine updating (when an activity is executed) the state of the entire process enabling and disabling the activities that can be executed, according to the process constraints. When an activity is executed, its linked root Activity is instantiated by the process manager and its UI part is rendered allowing the user to specify input data if needed. The process manager obtains the transitions dependencies from the Declare engine and computes the ActivityProperty input instances to be injected into the executing Activity.

The right side of Fig. 5 shows the interaction at the process level between the UI manager that is responsible of handling the user interaction and the Declare engine acting on the basis of the modeled process. At each interaction event, the UI manager issues queries to the Declare engine and enables the correct ProcessActivity asking the user how to proceed.

5. THE CASE STUDY

In order to validate the feasibility and the effectiveness of our approach (from the definition of the models to the execution of the code generation tasks), we show an example application consisting of the development of a typical WA for ordering a product in an online shop. In the remaining of this section, all key aspects of this case (i.e., information, service, and presentation models) are described and the results are discussed.

In Fig. 6, we show an excerpt of the DSL instance representing the information model of the WA. This model includes entities (Order, ItemLine, Invoice and a small Item hierarchy) to represent the main concepts of the application domain and uses the Association class to link them. PrimitiveType, UserDefinedType and Enum metaclasses are used to define some secondary concepts such as Address, Country and CustomerName that do not require an object identity. Finally, the model also includes generic collections repre-
senting sets of objects of given types. These collections express an
association with several roles involving multiple objects (i.e.,
the aggregation between Order and Item or the composition between
Item and Property).

The excerpt of the DSL instance, describing the service model and
reported in Fig. 7, defines the operations implementing the
business logic of the WA. For instance, the Order Entity is defined
by specifying its Attribute and Operation classes modeling data and
behaviors of the concept “order” of the application domain (such as
the list of ordered items and the method to compute the total price
of the order). The model defines a DAO for each Entity and in-
declaring the service model of the case study

ontact between Choose Item and Send Order indicates
that users must choose at least an item for sending an order. The
label 0..1 associated to Send Order (absence constraint) means that
for each process instance, Send Order can be executed at most once.
Furthermore, the not succession between Send Order and Choose
Item specifies that, when an order has been sent, it is not possible to
choose further items anymore. It is possible for the user to register
an account at any point in time in the process and more than once.
However, it is possible to pay only after the registration and after
having sent an order (as indicated by the precedence constraints
between Register and Pay Order and between Send Order and Pay
Order respectively). As seen for Send Order, also Pay Order can be
executed at most once in a process instance. In addition, it is possible to ask for a receipt more than once in a process instance,
but only after the payment (specified by the precedence constraint
between Pay Order and Ask for Receipt). It is also possible to can-
cel an order. The precedence constraint between Send Order and
Cancel Order specifies that the users can cancel the order in any
point of the process, but only after having sent the order. Finally,
the not succession between Cancel Order and Pay Order) indicates
that after having canceled an order it is not possible to pay for it
anymore. The resulting process model is made by 6 events and 9
constraints.

Each ProcessActivity in the process model should also specify
the attributes of the entities, defined in the information model, on
which it depends. At run-time, this allows the creation of the UI ac-
tivities that, in turn, collect (by interacting with the users) the input
data needed to execute the logic associated to that ProcessActivity.

Figure 8b shows a small excerpt (due to space limitations) of the
Declare DSL model for our case study. The model shows the inte-
gration of the Declare DSL model and the information model.
Each Event defines its inputs and outputs as scoped variables (that
are bounded to the current instance of the Process or the Process-
Activity) or as attributes (of instances of persistent entities accessed
through DAOs).

The online shop example has been defined with the underlying
goal of evaluating the richness and the completeness of the pro-
posed metamodeling approach (to assess that it is able to produce
The Declare process model

(a) The Declare process model

Figure 8: Declare model of the online shopping example

(b) An excerpt of the DSL Declare model

WebApplication OnlineShopping {
  Process processOrder {
    states { READY,STARTED, ... }
    ... 
    Event SendOrder {
      inputs { Key Order . id
                       String Order . name
                       List<Item> Order . items }
      outputs { Key Order . id
                     Status Order . status }
      ActivityOperation execute
      delegates OrderDAO . updateStatus
      Precedence CancelOrder , PayOrder ;
      NotSuccession ChooseItem ; }
    ... 
  }
}

A screenshot of the resulting online shop WA is reported in Fig. 9. It shows a CRUDActivity view for the Item entity. The Items List gives access to Item instances allowing the user to manage them.

The “Main Controls” navigational bar contains a button for each Entity of the information model (shown in upper left part of figure). This “Main Controls” navigational bar is automatically generated from the information model and injected into each UI activity allowing the application administrator to execute CRUD operations on each Entity and to manage processes. This bar also contains a PendingTasks button that gives access to the list of the active process instances for a ProcessOrder. This list is shown in Fig. 10 and reports, for each process instance (ProcessOrder), the process instance id, the activity name and the history of the activities already executed (Activity Path), the associated items (quantity and ID) and the current status. The customization class in this case was created to indicate quantity and ID of the selected items. In the figure, the process order number 274266 has one item chosen and the status is valid and running. Moreover, for this order we can visualize the contextual menu. The static part of the menu includes the actions for process management (stop, edit and remove). The dynamic part (queried using the Declare engine) shows that in this process instance it is possible to execute Choose Item again, Send Order to buy the items or Register a new account.

To verify the flexibility and effectiveness of the approach when a change is required in the process, the process model was modified in order to avoid the possibility of canceling an order after that it has been payed. To fulfill this requirement just the Declare model was modified by adding a new not succession constraint between Pay Order and Cancel Order. Starting from the new model the WA

a typical process-centric WA). To this aim the application uses almost all language concepts used in the metamodels here illustrated including:

- inheritance, composition and association between entities;
- the core concepts of the Declare language (covered by the process shown in Fig. 8a);
- the generation of a non-trivial hierarchy of UI activities allowing the user to interact with each step of the process.

The application was made up of 13kLOC of code (of which less than 1kLOC was manually customized), and included 9 entities, 9 DAOs and 4 services. For each Entity, a CRUDActivity was automatically generated with its gap classes for the customization. At the end of the generation, the entire application consisted of 54 classes and 12 interfaces, 22 of these classes were generated as prescribed by the Generation Gap pattern allowing for the customization of data objects, services and activity behaviors. The effort to develop the WA was of one day/man, mainly spent to develop the models (about 90%). Only 10% of the time was spent to manually customize code. This effort is significantly lower than the effort spent for developing the same application with a traditional (non-MDE) approach, that required more than two days.

A screenshot of the resulting online shop WA is reported in Fig. 9. It shows a CRUDActivity view for the Item entity. The Items List gives access to Item instances allowing the user to manage them.

The “Main Controls” navigational bar contains a button for each Entity of the information model (shown in upper left part of figure). This “Main Controls” navigational bar is automatically generated from the information model and injected into each UI activity allowing the application administrator to execute CRUD operations on each Entity and to manage processes. This bar also contains a PendingTasks button that gives access to the list of the active process instances for a ProcessOrder. This list is shown in Fig. 10 and reports, for each process instance (ProcessOrder), the process instance id, the activity name and the history of the activities already executed (Activity Path), the associated items (quantity and ID) and the current status. The customization class in this case was created to indicate quantity and ID of the selected items. In the figure, the process order number 274266 has one item chosen and the status is valid and running. Moreover, for this order we can visualize the contextual menu. The static part of the menu includes the actions for process management (stop, edit and remove). The dynamic part (queried using the Declare engine) shows that in this process instance it is possible to execute Choose Item again, Send Order to buy the items or Register a new account.

To verify the flexibility and effectiveness of the approach when a change is required in the process, the process model was modified in order to avoid the possibility of canceling an order after that it has been payed. To fulfill this requirement just the Declare model was modified by adding a new not succession constraint between Pay Order and Cancel Order. Starting from the new model the WA
can be automatically regenerated. The effort needed was, therefore, sensibly lower than the effort usually required to do such a modification with a traditional approach.

6. CONCLUSIONS

In this paper, we have presented M3D, a tool based on the integration of three MDE metamodels (representing the main components of a WA) with the declarative process modeling language Declare. The declarative model allows us to model in a simple and compact way business processes whose activities can be performed in different orders, i.e., processes whose workflow can branch into several alternative paths. The declarative model is integrated with three MDE metamodels representing the main components (information, presentation, and service) of a WA. A prototype of the WA is automatically generated by the M3D tool. The generated code requires a very low degree of manual customizing operations. The case study showed the effectiveness of the tool and highlighted the flexibility deriving from its adoption also when changing the process underlying the WA. Indeed, changes in the process, just required changes in the Declare model to re-generate the WA code. Future work will be devoted to verify the goodness of the approach in the development of larger and more complex WAs with the aim of exploring and improving it.

7. REFERENCES