ISBN 978-83-60810-51-4

Using Action Reports for Testing Meta-models, Models, Generators and Target Interpreter in Domain-Specific Modeling

Verislav Djukić
Djukić – Software Solutions
Nürnberg, Germany
Email: info@dvdocgen.com

Ivan Luković
University of Novi Sad
Faculty of Technical Sciences
Novi Sad, Serbia
Email: ivan@uns.ac.rs

Aleksandar Popović
University of Montenegro
Faculty of Sciences,
Podgorica, Montenegro
Email:
aleksandarp@rc.pmf.ac.me

Vladimir Ivančević
University of Novi Sad
Faculty of Technical Sciences
Novi Sad, Serbia
Email: dragoman@uns.ac.rs

Abstract— In this paper, we present an approach to testing of models and generated code as well as of target interpreters that relies on the use of modeling tools and model transformation languages. When compared to the existing Model Driven Development (MDD) approaches and tools supporting Domain Specific Modeling (DSM), contributions of our research include: (i) introduction of action reports, which allow semantic actions on elements of a graphical interface for modeling; (ii) creation of recommendations and of the interface for integrating modeling tools with applications; and (iii) construction of a language for the description of the structure of user controls as well as construction of a component for embedding such controls into modeling and meta-modeling tools. The basic idea behind the approach is to use a transformation language to construct complex objects and applications as well as specify operations on complex objects and the interface. In this manner, we not only generate the target platform code from the select domain-specific graphical language (DSGL) models but also directly use these models and appropriate tools as client applications. The applicability of action reports is demonstrated in the examples concerning validation of document models and their generators.

I. Introduction

IN the past few years, there have been increased efforts to improve software engineering through the application of software models. In numerous papers, there are remarks that the adoption of Model Driven Software Development (MDSD) and Unified Modeling Language (UML) as its main language has only partially achieved the proclaimed goals related to development productivity and software quality [1], [2]. Some authors consider the unfitness of UML for domain specific problems to be the main reason for this failure. Expecting that an average software engineer uses or thinks in domain independent abstractions might have been unrealistic. Several approaches, including Domain Specific Modeling (DSM), Model Driven Architecture (MDA) and Model Driven Software Development, still focus on software models, which are sufficiently formal but also understandable to both machines and humans. These models are not only part of the specification but also of the implementation of the corresponding systems. In general, there are two types of models: models that do not contain information about the implementation platform (Platform Independent Model, PIM) and models that contain platform specific information (Platform Specific Model, PSM). Abstract specifications are transformed into models using Model-to-Model (M2M) transformations, or into executable specification, such as source code, by using Model-to-Code (M2C) transformations. This model transformation approach has been known for decades. However, it is becoming more and more popular nowadays because of the positive results obtained by using DSM in software development for various embedded systems.

In our approach, the application of MDD, DSM, and model transformation principles is related to complex problems in document engineering, previously presented in [3]-[9]. Positive experience with the construction and application of domain specific languages (DSLs), together with the problems related to the development of client applications for measurement and control systems, points to the need the following transformation types to be introduced (further explained in Section III):

- (sub)model to application (M2A);
- application to (sub)model (A2M); and
- (sub)model to document (M2D).

By employing these transformations, we intend to make possible the use of modeling tools as client applications, at least in the prototype development phase. Notwithstanding the fact that current techniques for code generation from models have great capabilities, we demonstrate herein the practical value brought by:

- introduction of the submodel concept and high level submodel operations in addition to repository operations;
- introduction of the transaction concept in the context of (sub)models; and
- use of action reports (generators) as synchronization units during the testing of models, client applications, and target interpreters.

A part of the research presented in this paper was supported by Ministry of Education and Science of Republic of Serbia, Grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models.

The action report concept is used as a paradigm for semantic actions in M2A, A2M, and M2D transformations. Semantic actions are operations that synchronize model property values between the DSM tool, client applications, and target interpreter.

In the academic community, much of the model transformation research relies on the OMG's specification Query/View/Transformation (QVT) [10]. The specification consists of three interrelated languages: (i) Relations, (ii) Core and (iii) Operational Mapping. Atlas Transformation Language (ATL) [11] by the Eclipse Foundation [12] is an example of a model-to-model (M2M) transformation language in accordance with the QVT standard. Among the commercial tools, the best known transformation language is MetaEdit+ Reporting Language (MERL) [13]. It is a language mainly focused on model-to-text (M2T) transformations. The approach we present herein relies on the use of action reports. It features transformations that conduct synchronization between the model, client applications, and target interpreter. In this manner, model testing and execution are made both quicker and simpler. By using a generic component for action report parsing and interpretation, it is possible to synchronize applications that feature disparate user interfaces and interpreters [4], [14]-[18].

Besides Introduction and Conclusion, the paper contains six sections. In Section II, we describe the action report concept and show how it differs from code generators in popular meta-modeling tools. In Section III, we describe M2A, A2M, and M2D transformations with respect to application generation. In Section IV, we describe usage of submodels and transactions in the testing of a language, model, and target framework or interpreter. In Section V, we describe high level domain-specific operations that are implemented through action reports and executed on models. In Section VI, we give an example of the synchronization between a client application and modeling tool. Section VII contains an overview of the current state of technology in testing of meta-models, models, code generators, and a target framework or interpreter.

II. A Concept of Action Report

Domain-specific modeling involves the use of generators, also known as reports. They specify how to utilize information from abstract models to generate code in accordance with a particular concrete syntax [8], [13], [19], [20]. Generator (report) is a program whose interpretation yields a textual representation of the semantics expressed in a model. Since transformation languages support model filtering (selection of objects and relations according to a criterion), submodel or model view is implicitly defined through a generator. In order to have a more precise definition and interpretation of operations on a submodel, we require that the pertaining objects and relations be explicitly declared.

The purpose of introducing action reports is to extend the functions of model transformation programs (generators) to include the synchronization between a modeling tool, target interpreter and client applications that are not defined by a meta-model. With regard to this purpose, we define action report as a report that is used to transform a model into an application as well as operations from external applications into operations on visual representations of the model. Other relevant action reports characteristics include:

- action report is defined in the context of a submodel;
- action reports may execute operations (and be referenced) in the context of both concepts forming a meta-model (modeling language) and objects that are not part of the meta-model, i.e., any user control;
- all the communication between modeling tools and external applications is in the form of textual commands specified in the syntax of a generator language;
- action report is executed inside an optimized transaction whose beginning and end are tied to valid model states;
- in addition to validation being carried out in accordance with the specification of a domain-specific language, i.e., its meta-model, there are target environments that support model interpretation during specification time; this introduces the need for an operation that would calculate specification increment between two model states;
- action reports feature frequent model view changes, i.e., frequent submodel redefinitions;
- when using models to manage business processes, action reports may be used to synchronize business activities prior to a switch to a new management model as well as to incrementally generate documentation and applications;
- action reports are closely related to target interpreter environments, which may vary greatly;
- action reports may be called both synchronously and asynchronously; this requires that, at the meta-level, there be a specification of calling rules, which may define order, frequency, and logical conditions related to the call; and
- if the target interpreter does not support code modification during interpretation time, the problem is reduced to the recompilation of the generated code and the use of appropriate debugging tools, which are often part of the Integrated Development Environments (IDEs).

The role of action reports is illustrated in Fig. 1. They act as an interface between the modeling tool and target interpreter (or the debugging environment for the generated code). The objective is to allow various user groups like meta-modelers, modelers, testers, etc., to use an existing DSM tool as a means of testing generated code and target interpreter, in addition to model and DSL testing. Action reports are not intended to be used for the description of dynamic characteristics of a system. These characteristics may be completely formally specified through UML state diagrams or equivalent DSLs. Action reports may also be used to allow direct use of the existing graphical interface in debugging or testing of the generated code (or target interpreter).

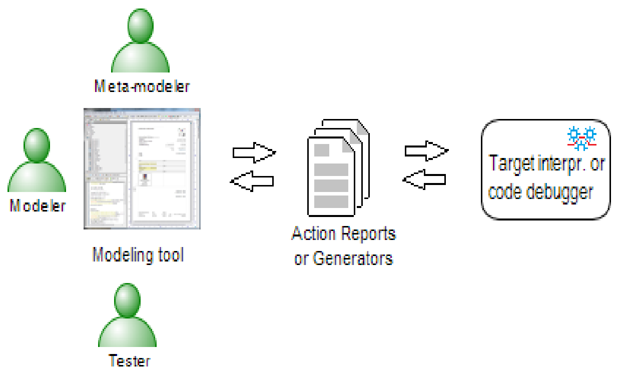

Fig. 1 Role of action reports

In [20] and [21], the authors present ideas and solutions for domain-specific model debugging and transformations. Our consideration of the role of code generator differs slightly from the one presented in [20], in which the generator is treated as a means for the definition of model semantics. In certain cases, when the modeling language is not sufficiently semantically rich, generators may be temporarily used to describe semantics, i.e., surpass problems caused by the conceptual limitations of the DSL. This scenario is typical particularly for the DSL construction phase.

We close the action reports introductory section with a remark that the importance of action reports as defined herein may significantly differ depending on the actual context. In some business domains, the feedback that action reports may provide to modeling tools has no relevance. However, in the specification of measurement and control systems through DSLs, action reports are essential and their use brings numerous advantages [22], e.g., the modeling tool may be used as a Human machine interface (HMI) by exploiting the feedback from the target interpreter and there may be different visual representations of a single language concept.

III. A2M, M2A, AND M2D TRANSFORMATIONS

M2A/A2M transformations are basically M2T/T2M transformations whose purpose and syntax variations have been described in various papers. These transformations have been applied also in numerous tools for code generation from models [8], [12], [13], [20]. The motivation for introducing M2A/A2M transformations is to allow us to differentiate in code generation between: (i) procedures that generate the code for the communication between modeling tools and a target interpreter and (ii) procedures that generate the code to be interpreted in the target interpreter. In this context, the target interpreter is important as a component that gives feedback for the refinement of both the model and meta-model, i.e., DSL refinement. The reason for introducing the notion of a M2D transformation is a need to isolate the procedures for the generation of documentation concerning the results of testing of models, meta-models, or a target interpreter. Activities of testing and documenting of the testing results for meta-models, models, and an interpreter, are henceforth referred to as **Me**ta-modeling, **M**odeling, **I**nterpreting and **D**ocumenting (MeMID) activities.

The most important characteristics of M2A/A2M transformations include:

- target text is a code in a general purpose language (GPL), DSL, or any textual format interpretable by a modeling tool or a target interpreter;
- target text is focused on operations involving reading and value modification of repository properties as well as on operations done on elements representing DSL concepts (graphical interface, symbols that represent objects, relations, etc.);
- these transformations may include operations on external elements of the presentation that are not part of the modeling tool (Fig. 2);
- these transformations do not modify the meta-model, however it might be useful to support a semiautomatic inclusion of user controls that graphically represent language concepts; and
- when there is a disparity between the concepts directly supported by the interpreter and those of the DSL, these transformations provide an interface for the communication between the relatively incompatible units.

In Fig. 2, we illustrate the scope area of M2A/A2M and M2D transformations labeled “Action reports”. Action reports are interpreted in the context of a modeling tool, a repository, and client applications. They may be exchanged between these contexts as well as updated in any of them.

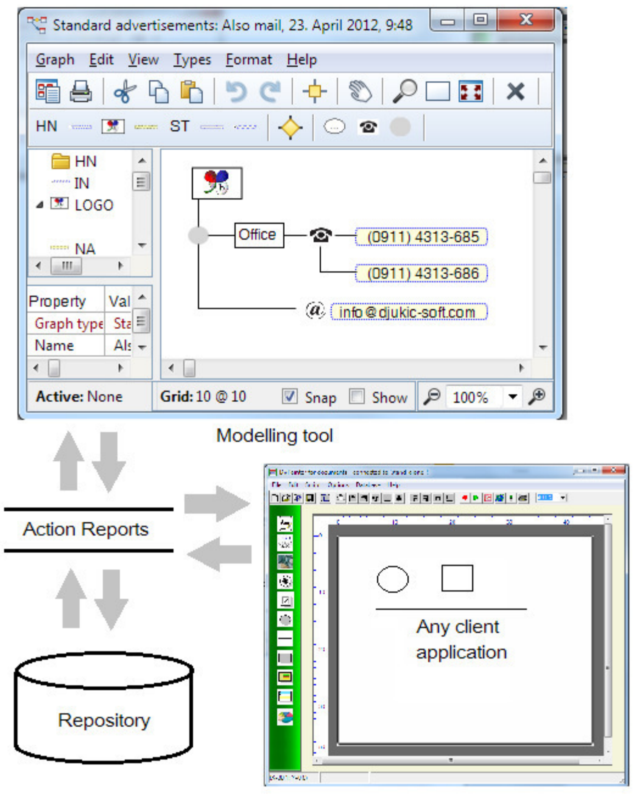

Fig. 2 Scope area of transformations

By introducing these transformations we satisfy some of the user requirements related to the more agile testing of DSLs, models, and target interpreters. Our approach has limited use in situations in which the target environment is not present in the form of a generated code interpreter. In that case, after each model modification, the code is generated, then compiled, and the application is rerun. On the other hand, it is not important at which abstraction level operations executable by the target interpreter are. The matching is done at the transformation level. The ideal environment for MeMID activities is the one that supposes the existence of the "universal interpreter" and does not require termination of the interpretation in order to switch to the interpretation of the modified model. These "hot" switches to a new version of the model are known as incremental updates. Universal interpreters that are independent of the application domain do not exist. Any generalization necessarily leads to a greater separation of the language used to describe the problem from the language interpretable by the interpreter. In practice, there is a compromise to solve the widest possible class of problems by using an existing interpreter of the similar purpose. In this manner, at least in the system prototype development phase, it is possible to have full parallelism in the refinement of the meta-model (DSL), concrete models, code generator, and interpreter.

With respect to the connectedness of meta-models and models, modern tools vary greatly. Some tools support meta-modeling only through textual syntax and feature weak synchronization between meta-models and models [12]. Other tools consistently support abstract graphical models, graphical DSL constructions, and different visual representations for the same language concept as well as full synchronization between the meta-models and models [13]. Different visual representations of a single language concept allow animations, i.e., visual presentations of states during interpretation [22].

The use of M2A/A2M transformations is illustrated with the model debugging examples featured in the fourth and fifth section. That sort of debugging is not equivalent to the debugging inside GPL Integrated Development Environments (GPL IDE). With the GPL-to-assembly transformations, there is a finite, predetermined set of source and target language concepts. On the other hand, in DSM neither the source nor the target language needs to be known in advance. Moreover, in directory publishing neither of them is usually known in advance because the used terminology is specific to a particular sector, region, or book edition. The source language is constructed to meet the domain-specific needs and the target code may substantially depend on the existing libraries and frameworks. One of the approaches to the formation of a stronger logical relationship between debugging environments and modeling tools includes the use of patterns. In this manner, it is generally possible to relate the model to the target code. One disadvantage of the use of patterns is that they need to be created for each combination of a DSL and target platform. The critical issue is how efficient the debugging of the resulting code is when done through a GPL IDE that is logically separated from the meta-modeling tool. This problem is extensively debated and the proving of the language validity is a topic of numerous papers and books [2], [21].

Further discussion of MeMID activities is based upon an assumption that the debugging rules or steps should be defined inside the M2A/A2M transformations in order to provide the feedback from target interpreter toward model.

IV. Submodels, Transactions, and Reports in the Testing of language, Model, and Target Interpreter

Modeling tools usually support the concept of model decomposition, which implies that an object, relation, role, port, or property may be linked to a submodel. This allows for a model to be described and expressed at different logical levels. During model testing, it is necessary to focus on just a subset of elements. For example, in CASE tools providing creation of logical database models, it is possible to define a database view. In our case, a user-defined model view is also known as a submodel. It generally includes at least one relation and two objects having a role in the relation. Submodel is not just a selection of objects in the presentation of the instances of language concepts, but it is similar to database views. It is a complex object with its own structure, operations, and constraints. Although (sub)model operations and constraints are used to express fundamental dynamics of the system described by the model, they are not sufficient to express the rules for the translation of the model from one consistent state to another. For this reason, modeling tools should include support for the transaction concept. Transaction is defined as an operation that validates a sequence of actions on a model and updates the repository. When compared to the database transaction, it also includes validation of the generated code and of the target interpreter, i.e., validation in the context of MeMID activities. Therefore, we expect that modeling tools explicitly support (i) submodels which, in addition to decomposition, include selection of relations and objects; and (ii) MeMID transactions.

The purpose of submodels and transactions is illustrated by an example presented in Fig. 3. It is a typical example of a fully automated MeMID activity. The diagram in the left section of the figure features activities A1-A4 that are part of the production of advertisements and related documents. One of the activities (A2 – Standard ad production) is composite and involves the use of DVAdLang, a domain-specific language for the production of advertisements [6], [23]. In the modeling tool, there is an object-subgraph relation between A2 and an advertisement model. The subgraph is an advertisement model that features a logo, several phone numbers, and an email address (marked with M4). In the upper right section of the figure, there are three abstract advertisement models (M1-M3) in three consistent states (S1-S3), all of them representing the same advertisement. There are two levels of verification: (i) model verification during design time, done by the modeling tool and in

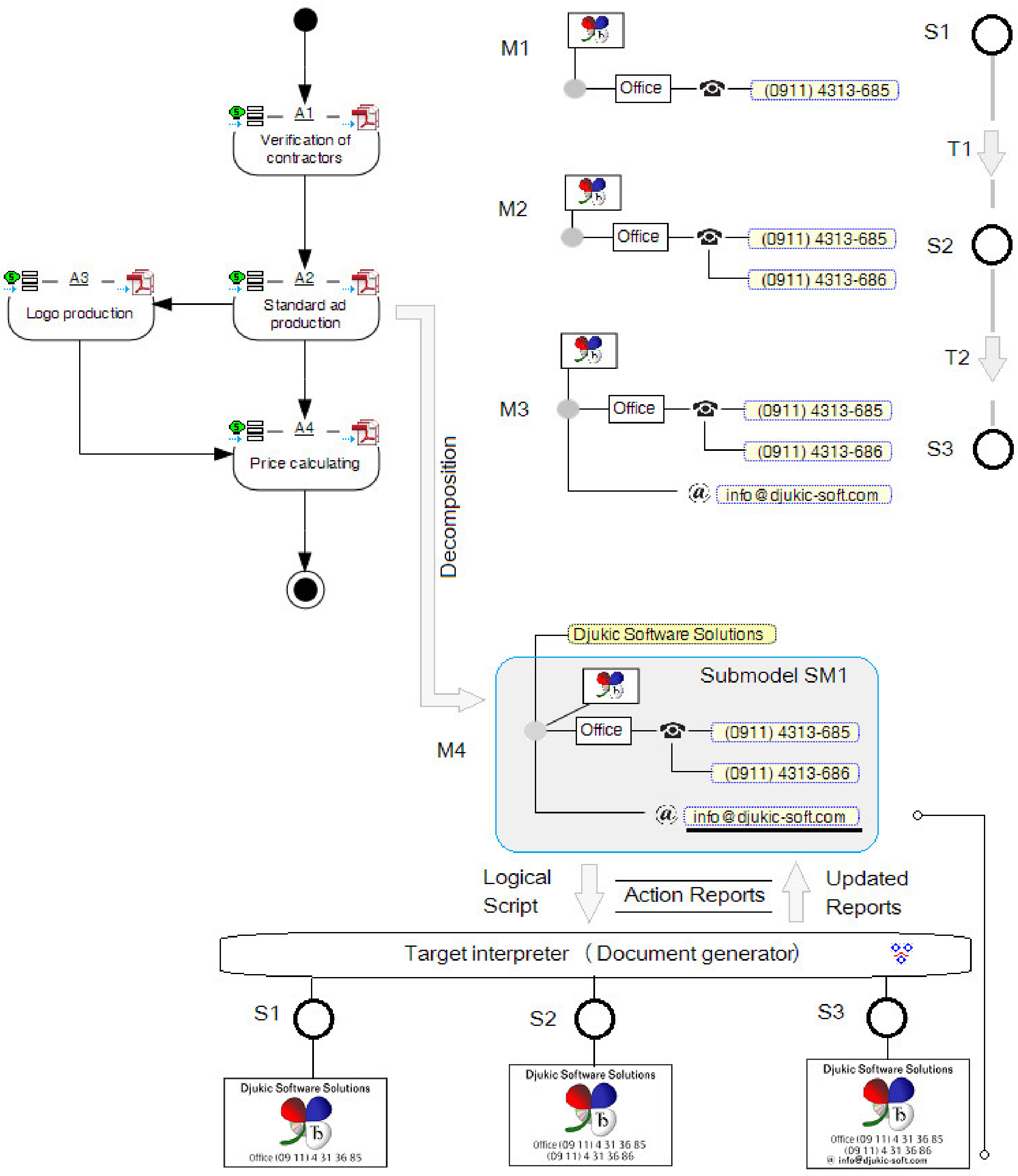

Fig. 3 Submodels, transactions, and testing of models and the target interpreter

accordance with the meta-model; and (ii) on-demand verification of the generated code, usually executed on transaction confirmation calls (in the figure marked by T1 and T2). Successfully completed transactions may represent transitions between two model states or, as in this case, valid advertisement states. In the model M4, there is the submodel SM1 (a shaded rectangle with rounded edges) that includes the following objects: Office (of type Place), two phone numbers (of type Phone), and user's email address (of type eMail). The submodel also features relations In content unit (the gray circle inside the submodel and to the left) and Phone rings in (the telephone symbol in the center).

Transition from one diagram state to another, from submodel S1 to S3, or between valid states, is verified by PDF rendering using our interpreter. In this manner, we obtained advertisement images, which are shown in the lower section of Fig. 3. The rendered image may be: (i) result of the transformation of a model in a valid state or (ii) increment between two consistent model states. As generators (reports) are associated with models, so advertisement models in concrete syntax include the definition of an action report in the form of metadata. The concrete syntax in the given example is a DVAdLang logical script. During the interpretation of the concrete advertisement model, the target interpreter interprets synchronization commands defined through action reports and sets corresponding property values in the report definition. The modeling tool analyzes the modified action report and runs operations on the graphical interface elements. In the example featured in Fig. 3, as a result of the executed operation, the text in the email symbol is underlined. The transfer of action report packets between modeling tool and target interpreter may be: cyclic (when CPU is idle), in intervals, or on a specified condition (event).

Examples 1-3 further explain the contents of Fig. 3 and include: (i) specification of the action report AR1, which sets the text property Font.Underline in the objects in the modeling tool, (ii) general form of a logical script (LS) with metadata (ARMeta) that is used as a basis for document generation, and (iii) concrete logical script that is a result of the transformation done by the action report AR1 for the advertisements shown in Fig. 3.

Example 1. The action report AR1 is defined using DVDocRepLang [8], [24], a language similar to MERL. It is presented in Listing 1. AR1 is applicable to all models that are of the same type as M1-M4 from Fig. 3. It is used to generate, in accordance with the syntax of DVAdLang language, a logical script from the abstract models of advertisements. AR1 contains a section that exports object properties, and a description of semantic actions for synchronizations marked by `CALL_TYPE` and `ACTION` keywords.

```
Report 'AR1'
CALL_TYPE = event; /*interval,cyclic,event*/
foreach >ContentUnit {
do .()
{
'<'type '>'
  if type = 'LOGO' then
    ID  ',' :Alignment; ',' :Height;
  else
    :Value;
  endif
  newline
  dowhile ~Phones in>Phone connections~Phone
rings in.()
  {
  '<' type '>' :Value; newline
    ACTION_BEGIN
      '<STATE>'objID
      :Font.Underline=true;
    ACTION_END
  }
}
```

Listing 1. Action report example

The existing syntax of DVDocRepLang, which is used for M2T transformations, is extended with: (i) `CALL_TYPE` command for the declaration of conditions or intervals for the exchange of action reports with the target interpreter, and (ii) `ACTION_BEGIN` and `ACTION_END` primitives, which mark a report code section related to synchronization. In Listing 1, the new language commands are marked in bold.

Example 2. The general form of a logical script given in DVAdLang syntax, which is generated by using the action report from Example 1, is presented in Listing 2. Global metadata marker `<AR_META>` contains the definition of an action report. This definition is required by the target interpreter during the whole synchronization process done with the modeling tool and client applications.

```
<AR_META>="REPORT AR1..."
<CU>Initial logical script
<STATE>S1
<CU>Increment for S2 (Transaction T1)
<STATE>S2
<CU>Increment for S3 (Transaction T2)
<STATE>S3
```

Listing 2. Embedded definition of an action report in the target language

The `<STATE>objID` commands are used to mark code sections responsible for the specification of document content units, their appearance, and dynamic characteristics. When the interpreter encounters the `<STATE>` command, it interprets it as a request for the call of the transaction that contains property-setting operations marked by `ACTION_BEGIN` and `ACTION_END` (in this case, Font.Underline=true). All the actions, except the current one, are removed from the action report, which is then sent back to the modeling tool. On the reception side, in the simplest case, it just sets the property value.

Example 3. A detailed specification of DVAdLang and DVDocLang languages, together with examples, is given in [3]. An example of a logical script presented in Listing 3 illustrates that the `<STATE>` command is used to: (i) mark increments in the interpretation, e.g., breakpoints during debugging; and (ii) mark, in the concrete model of an advertisement, the point when the original action report gets updated.

```
...
<LOGO>7937,center,10
<PO>Office
<RN>(0911)4313685
<STATE>S1
<RN>(0911)4313686
<STATE>S2
<EM>info@djukic-soft.com
<STATE>S3
```

Listing 3. Example of a logical script in the target language

Semantic action of synchronization through an action report may be arbitrarily complex. It may include incremental specification and rendering of documents inside MeMID activities. In this particular example, since the target interpreter is a document renderer, semantic action represents both a proof of model execution and a rendered documentation about model testing.

V. ACTION REPORTS AND OPERATIONS ON MODEL

The simplified scenario from the previous examples includes interpretation of action reports featuring basic semantic actions that are reduced to setting the value of a single property. A more advanced scenario might include the use of action reports to: (i) construct submodels and carry out all operations on (sub)models without the need for the execution of low-level API functions on the repository, (ii) define transactions, and (iii) conduct synchronization with client applications, e.g., those classified as HMI.

The construction of submodels and corresponding operations is similar to the definition of views in relational databases or the definition of complex objects in object databases. We focus on operations that could significantly improve MeMID activities when the modeling tool is linked to the target interpreter via action reports. Therefore, we give an overview of the select operation set:

- **CreateSubmodel** (listOfElems) – creates a submodel based on the specified list of objects, connections, relations, roles, and properties from an existing model;
- **SetCurrentSubm** (m_ID) – sets one of the defined submodels as the current one;
- **DeleteSubmodel** (m_ID) – deletes the submodel definition;
- **AddModel** (m_1,m_2) – joins two submodels into one without modifying any relations;
- **Subtract** (m_1,m_2) – removes `m_2` from the existing composite model `m_1`;
- **Multiply** (m_1,n) – creates a new model by repeating the model `m_1 n` times;
- **Intersection** (m_1,m_2) – returns a model containing intersecting element from `m_1` and `m_2`;
- **Union** (m_1,n) – joins two models without repeating elements having same identifiers;
- **SimDifference** (m_1,m_2) – finds a symmetric difference between the two models;
- **Remove** (objType|relType) – removes objects or relations of the specified type from the submodel; and
- **Clone** (objType|relType|roleType) – clones the complete model or just object, relations, roles, and properties of the specified type or matching the specified pattern.

These operations on models may be specified through action reports in the section marked by `ACTION_BEGIN` and `ACTION_END`. At the level of modeling tools, interpretation is done by the code generator. At the level of target interpreter and client applications, interpretation is done by the DVDocRepLang [8], [24] component, which is similar to the MERL interpreter. An example of parallel interpretations is given in the subsequent section.

VI. An Example of Synchronization between Application and Modeling Tool

Although the target interpreter and external applications are not a part of the modeling tool, it is necessary to allow their simple integration and use in MeMID activities. A generic solution to this problem would be difficult to produce because there is no universal model interpreter. Therefore, we restrict ourselves to the pragmatic approach that utilizes action reports and common properties of visualization elements in the modeling tool and external applications.

In Fig. 4, we illustrate the relationship between the elements that are part of the MeMID activities during: (i) specification of reports and actions; and (ii) interpretation. Firstly, properties of graphical elements in the modeling tool are linked to properties of HMI client application components (Property linking). DVDocRepLang supports automation of this activity to a large extent. The selected subset of common properties is the object of the semantic action marked by ACTION_BEGIN and ACTION_END. An action report may contain more than one action.

In the upper left section of the figure, there is an illustration of the function block object with four input and two output parameters (properties). The lines ending in dots represent disconnected roles in the relations between function blocks. In the upper right section of the figure, there is a user component showing input and output values in the form of a rectangle, except for a logical type property (represented by a circle) whose value is **false**. In the case of the value **true**, the circle to the right is filled. Synchronization during interpretation time is represented by curved lines with arrowheads. The modeling tool generates code based on the generator specification and forwards this specification through the generated code as metadata. During interpretation, at the marked synchronization points, the interpreter sets the values of properties included in the definition of the action. For instance, the operation may be reduced to the modification of a string: from **:in3;** to **:in3=2.54;**. This translates into the modification of the value of the **in3** property to **2.54**.

In comparison with the existing MERL syntax, property referencing is extended with the property value setting.

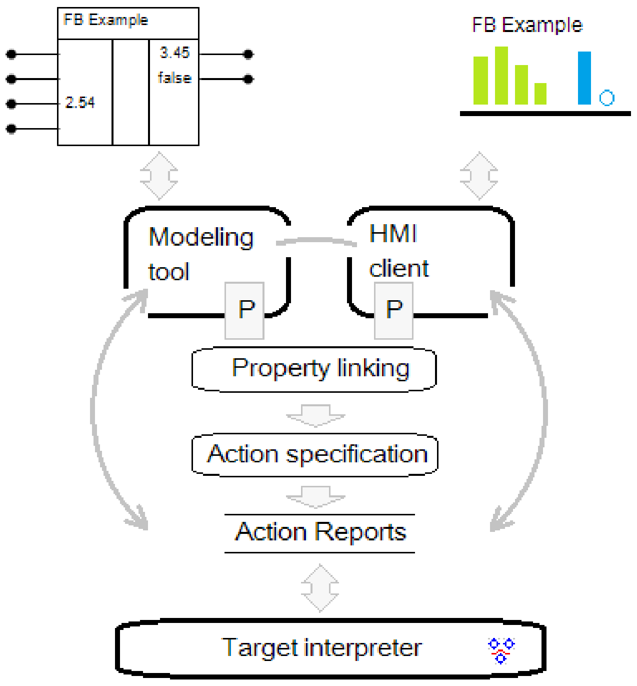

Fig. 4 Editor of common properties, action specifications, and synchronization

The modified report is forwarded to: (i) modeling tool for the purpose of modifying interface properties and (ii) HMI client application for the purpose of setting the values for visualization controls. The action report interpreter resides in both the modeling tool and the client application. Report exchange is performed periodically or on a certain event that is not time dependent.

VII. State of Technology in MeMID

The tracking of model changes presents an important research topic of practical relevance to MDD community. In [22], the authors introduce new features of MetaEdit+ Workbench [13] and present various capabilities for visualizing language concepts of a DSL, including dynamic modification of appearance properties. MetaEdit+ Workbench is a tool that provides support for various development phases including meta-modeling, modeling, code generation, and simulation of the modeled system. In our approach, we borrow two well-established ideas that are implemented in modern database management systems: transactions and views. By relying on transactions, we are able to track object modifications, which are explicitly stated inside action reports.

Any target system may use MetaEdit+ API over web services to perform model manipulation. In this regard, our approach offers similar functionalities concerning (sub)model modification. One of the main advantages is that the deployment of action reports eliminates the need for low-level API functions on the repository side. As a result, the specification of target interpreter feedback is less complex.

In [21], the authors report the lack of support for model debugging in DSL tools. While most of GPL IDEs support model debugging because language syntax and semantics are known in advance (and because there is a compiler), the situation concerning DSLs is substantially more complex. Standard debugging scenario is conceptually restricted by operating systems, target frameworks, and libraries. Therefore, any pragmatic approach featuring even minor improvements related to MeMID activities is going to represent a significant contribution to the testing of domain-specific models.

VIII. Conclusion

In this paper, we present the first practical results and foundations of an approach aimed at further improvement of DSM tools. Our objective is to automate to a greater extent: (i) MeMID activities; (ii) testing of models, generated code, and interpreter; and (iii) generation of documentation about test cases. In the areas of document engineering and development of measuring and control systems, the action report approach gives good results, especially when combined with DSM tools that, instead of relying on patterns, conduct M2T transformations by using a dedicated language and interpreter. Our further research is directed at the implementation of additional operations on submodels and testing of the approach in different application domains.

References

[1] Steven Kelly, Juha-Pekka Tolvanen, "Domain-Specific Modeling: Enabling Full Code Generation", ISBN: 978-0-470-03666-2, March 2008, Wiley-IEEE Computer Society Press.

[2] Anneke Klippe, Software Language Engineering: Creating Domain-Specific Languages Using Metamodels, Addison-Weslay 2008, ISBN: 0-321-55345-4

[3] Verislav Djukic, "DVDocLang Language Reference", Accessed: March, 2012 www.dvdocgen.com/Framework/DVDocLang.pdf

[4] Ivan Lukovic, Verislav Djukic, DVDocLang vs. XSL-FO, www.dvdocgen.com/Framework/DVDocLang_XSL-FO.pdf

[5] Kosar T., Oliveira N., Mernik M., Pereira M. J. V., Črepinšek M., Cruz D., Henriques P. R., "Comparing General-Purpose and Domain-Specific Languages: An Empirical Study", Computer Science and Information Systems (ComSIS), ISSN: 1820-0214, Vol. 7, No. 2, May 2010, pp 247-264.

[6] Verislav Djukić, Ivan Luković, Aleksandar Popović, "Domain-Specific Modeling in Document Engineering", Proceedings of the Federated Conference on Computer Science and Information Systems, Poland, 2011

[7] Ivan Lukovic, Verislav Djukic, "DVQL Language Specification", www.dvdocgen.com/Framework/DVQL.pdf, Accessed: March, 2012

[8] Verislav Djukić, Aleksandar Popović, "DVDocRepLang grammar specification", www.dvdocgen.com/Framework/ DVDocRepLang.pdf, Accessed: March, 2012

[9] Ivan Lukovic, Pavle Mogin, Jelena Pavicevic, Sonja Ristic, "An Approach to Developing Complex Database Schemas Using Form Types", Software: Practice and Experience, ISSN: 0038-0644, Vol. 37, No. 15, 2007, pp. 1621-1656.

[10] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, http://www.omg.org/spec/QVT/1.0/,.

[11] ATL - a model transformation technology, http://www.eclipse.org/atl/

[12] Eclipse Modeling Framework Project (EMF) , http://www.eclipse.org/modeling/emf/

[13] MetaEdit+ Workbench, MetaCase, www.metacase.com

[14] Apache Software Foundation: "FOP", http://xmlgraphics.apache.org/fop/0.95/index.html

[15] Microsoft Extensible Application Markup Language (XAML), http://xml.coverpages.org/ms-xaml.html

[16] User Interface Markup Language (UIML), http://www.uiml.org/

[17] Verislav Djukic, "DVDoc Renderer Benchmak", Accessed: March, 2012 http://www.dvdocgen.com/Framework/DVDocRenderBench.pdf

[18] Verislav Djukic, DVDocGen Framework, application interface, http://www.dvdocgen.com/Framework/DVDocFramework.pdf, Accessed: March, 2012

[19] Olivier Beaudoux, Arnaud Blouin, "Using Model Driven Engineering technologies for building authoring applications", Proceedings of ACM Symposium on Document Engineering, 2010

[20] Benjamin Klatt, "A Closer Look at the model2text Transformation Language", http://wiki.eclipse.org/Model2Text_using_Xpand_and_QVT_for_Query

[21] Raphael Mannadiar, Hans Vangheluwe, "Debugging in Domain-Specific Modelling", SLE'10 Proceedings of the Third international conference on Software language engineering

[22] MetaEdit+ 5.0 Beta Primer, Accessed: May, 2012 http://www.metacase.com/download/metaedit/MetaEdit+ 5.0 Beta Primer.pdf

[23] Verislav Djukić, DVDocFlowLang demo , video, Accessed: March, 2012 http://www.dvdocgen.com/Framework/DVDocFlow.wmv

[24] Verislav Djukić, DVDocRepLang demo, video, Accessed: March, 2012 http://www.dvdocgen.com/Framework/ModelTransformation.wmv